ANASTACIA

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

# D2.5

## Policy-based Definition and Policy for Orchestration Final Report

This deliverable presents the results of ANASTACIA Task 2.1, which aims to analyse and define the security policy models for the ANASTACIA framework. The document includes the policy models to deal with the main security requirements of the identified use cases

| | |
|---|---|
| **Distribution level** | PU |
| **Contractual date** | 31.12.2018 [M24] |
| **Delivery date** | 27.12.2018 [M24] |
| **WP / Task** | WP2 / T2.5 |
| **WP Leader** | UMU |
| **Authors** | Alejandro Molina Zarca, Jorge Bernal Bernabé, Jordi Ortíz, Antonio Skarmeta (UMU). |
| **EC Project Officer** | Carmen Ifrim<br>carmen.ifrim@ec.europa.eu |
| **Project Coordinator** | Softeco Sismat SpA<br>Stefano Bianchi<br>Via De Marini 1, 16149 Genova – Italy<br>+39 0106026368<br>stefano.bianchi@softeco.it |
| **Project website** | www.anastacia-h2020.eu |

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

# Table of contents

# Index of figures

# Index of tables

# PUBLIC SUMMARY

This deliverable describes the design and definitions of the security policies for the ANASTACIA framework. The design and modelling start from a study of the state of art about current techniques, technologies and policy languages as well as the relationship between them. That study serves as baseline to gather a set of base policy models and feasible ideas applicable over the framework. This study can be consulted at D2.1. After that, the document exposes a short overview of the ANASTACIA architecture in order to provide the reader a global vision of the policy-based framework, facilitating the comprehension of the following sections, which are focused on the definition of the security policy models.

Regarding the policy model's definition, this document exposes how ANASTACIA will adopt the two-level policy definition approach; the first high-level policy language aims to simplify the task of non-technical users, whereas the second one is a richer and more powerful policy language, but still independent of the subjacent layers. For each level, it has been exposed the main components of the design and how it is possible to use the models to specify the security policies. The main security policies applicable over the main identified IoT and MEC scenarios have been identified too. In this regard, the document provides the most relevant policy models, including access control (authentication and authorization), channel protection, filtering, traffic diverts, network anonymity, Quality of Service, privacy, data aggregation, operational security policies as well as policies for orchestration.

Finally, the document provides, as examples, two different instantiations of the security policies by applying the previous defined policy models. The instantiations are scoped in IoT building scenario, corresponding to the orchestration of security policies for authentication and authorization.

# 1 INTRODUCTION

## 1.1 AIMS OF THE DOCUMENT

This document is part of ANASTACIA WP2 "Security and Trust by Design Enablers" which aims to analyse and provide formal definition of interoperable security policies for SDNs, NFV and IoT. WP2 also aims to: provide an analysis of attack threats and mitigation measures for SDNs, NFV and IoT-enabled scenarios; perform an analysis and formal definition of privacy risk models; provide a set of secure software development guidelines and procedures.

Concretely, this deliverable is scoped in Task 2.1 of WP2, which aims to analyse and describe the existing proposals for policy modelling and policy orchestration in order to devise innovative declarative interoperable policy models that can be afterwards evaluated and enforced in NFV, IoT and SDN architectures. Policy models defined herein will serve as input for the Policy Interpreter (Security Enforcement Manager) defined in the scope of WP3 for policy analysis and enforcement. Existing models and languages will be adapted and extended to consider context and changes in the environment during interoperation. ANASTACIA will consider and model diverse types of security policies, such as authentication, authorization, filtering, channel protection and forwarding.

Task 2.1 is also providing an authorization model based on capabilities tokens and the policies to manage the access control to device's features and services, including the definition of contextual rights delegation, fine-grained access control rights, and mechanisms to preserve privacy and personal information.

The main goal of this particular deliverable is to provide the security policy models required to deal with the security aspects demanded in the scope of ANASTACIA project.

This document is structured as follows: Section 2 provides the state of the art of related policy models, techniques and technologies. Section 3 highlights the most relevant innovations of ANASTACIA in this subject. Section 4 gives an overview of the ANASTACIA architecture, contextualizing the policy models in the ANASTACIA framework. Section 5 is the core of the deliverable, since it defines the security policy models at both high-level and medium-level of abstractions. Section 6 provides two different instantiations of the security policies by applying the previous defined policy models.

## 1.2 APPLICABLE AND REFERENCE DOCUMENTS

This document refers to the following documents:

- ANASTACIA Grant Agreement N°731558 – Annex I (Part A) – Description of Action.
- ANASTACIA Consortium Agreement v1.0 – December 6th 2016.
- ANASTACIA deliverable D1.1 – Holistic Security Context Analysis.
- ANASTACIA deliverable D1.2 – User-centred Requirement Initial Analysis.
- ANASTACIA deliverable D1.3 – Initial Architecture Design.
- ANASTACIA deliverable D2.1 – Policy based definition and policy for orchestration first report.

## 1.3 REVISION HISTORY

| Version | Date | Author | Description |
|---|---|---|---|
| **0.1** | 05.11.2018 | Alejandro Molina Zarca and Jorge Bernal Bernabé, Antonio Skarmeta (UMU) | Skeleton of expected contents |
| **0.2** | 19.11.2018 | Jorge Bernal Bernabé (UMU) | Section 1 inputs. |
| **0.3** | 20.11.2018 | Alejandro Molina Zarca (UMU) | Section 2 and 3 inputs. |
| **0.4** | 21.11.2018 | Jorge Bernal Bernabé (UMU) | Section 4 inputs. |
| **0.5** | 26.11.2018 | Alejandro Molina Zarca and Jorge Bernal Bernabé (UMU) | Section 5 inputs. |
| **0.6** | 27.11.2018 | Alejandro Molina Zarca (UMU) | Section 5 inputs. |
| **0.7** | 28.11.2018 | Alejandro Molina Zarca (UMU) | Section 5 inputs. |
| **0.8** | 03.12.2018 | Alejandro Molina Zarca (UMU) | Section 6 inputs. |
| **0.9** | 10.12.2018 | Alejandro Molina Zarca, Jorge Bernal Bernabé and Jordi Ortiz Murillo, Antonio Skarmeta (UMU) | Internal review |
| **1.0** | 18.12.2018 | Rubén Trapero Burgos (ATOS) and Alejandro Molina Zarca (UMU) | Final review |

## 1.4 Acronyms and Definitions

| Acronym | Meaning |
|---|---|
| **BMS** | Building Management Systems |
| **CIM** | Common Information Model |
| **CPS** | Cyber Physical System |
| **CRUD** | Create, Read, Update, and Delete |
| **DSPS** | Dynamic Security and Privacy Seal |
| **HSPL** | High-level Security Policy Language |
| **IoT** | Internet of Things |
| **MANO** | Management and Orchestration |
| **MEC** | Mobile (Multi-access) Edge Computing |
| **NFV** | Network Function Virtualization |
| **NSF** | Network Security Functions |
| **PDP** | Policy Decision Point |
| **PEP** | Policy Enforcement Point |
| **PSA** | Personal Security Application |
| **SDL** | System Description Language |
| **SDN** | Software Defined Networking |
| **SEC** | Security Enforcement Manager |
| **SPL** | Security Policy Language (SPL) |

# 2 State Of the Art

This section presents a state of the art for security policy models, the main related technologies as well as the possible relationship among them. Regarding the policy models, Common Information Model (CIM) [1] is the main DMTF standard that provides a common definition of management-related information independent of any specification. The model defines concepts for authorization, authentication, delegation, filtering, and obligation policies. However, for an information model to be useful, it must be mapped into some specification. And for this purpose, CIM models are not suitable by themselves, due to the huge number of classes which is compound. xCIM High-level Security Policy Language (SPL) defined in [2] allows to the administrator the definition of security policies using a friendly language, near to the spoken English. It also has an internal format which is a language for formal modelling and low-level abstraction that is oriented to developers. xCIM System Description Language (SDL) is a sub-model that represents the medium level abstraction representation for system description. Whereas xCIM Security Policy Language (SPL) is a sub-model of CIM that represents the medium/low level abstraction representation for security policies. In the proposed approach there is a translation from the high-level specification to low-level rules specified by a language based CIM-Policy Information Model (i.e. xCIM-SPL or internal format). These policy models have been used in the scope of POSITIF [3] and DESEREC [4] European projects.

Following the previous approach, [5] presents High-level Security Policy Language (HSPL) and the Medium-level Security Policy Language (MSPL). They are two abstractions defined within the European project SECURED [7] to specify the security policy. High-level Security Policy (HSPL) is a policy language suitable for expressing the general protection requirements of typical non-technical end-users, such as “do not permit access to illegal content” or “block access to peer-to-peer networks”. The Medium-level Security Policy Language (MSPL) is the policy abstraction used for expressing specific configurations in a device-independent format. That is, an abstract language with statements related to the typical actions performed by various security controls (e.g. matching patterns against packet headers, keeping track of connection status, identifying the MIME type of a payload), but expressed in a generic syntax. HSPL is refined in MSPL, and the latter in final configurations for PSAs. PSA (Personal Security Application) is the component within the SECURED project architecture which enforces de configuration. It is responsible to enforce a security policy as specified by the user.

In more general terms, The Web Services Policy Framework [8] provides a general-purpose policy model and corresponding syntax to describe the policies of entities in a Web services-based system. It defines a base set of constructs that can be used and extended by other Web services specifications to describe a broad range of service requirements and capabilities. Policy expressions are represented in XML.

Focused on objects, Ponder2 [9] is a policy system suitable for a wide range of environments and applications. Everything in Ponder2 is a Managed Object. Managed Objects include management policies and adaptors to real-world objects such as sensors, alarms, switches etc. Making everything a managed object allows Ponder2 to manipulate them all for management purposes in the same way. The basic Ponder2 objects include Events, Policies and Domains, it is up to the user to create or reuse Managed Objects for other purposes such as adaptors. There are currently two basic policy types in Ponder2, Obligation Policies (Event Condition Action policies) and Authorisation Policies.

Beyond general policy models, it is possible find models or concepts specific purposes, applicable to policy models. E.g. In [10] the requirements of honeynet description are studied and a survey of existing description languages is presented, concluding that a CIM (Common Information Model) match the basic requirements. Thus, a CIM like technology independent honeynet description language (TIHDL) is proposed. The language is defined being independent of the platform where the honeynet will be deployed later, and it can be translated, either using model-driven techniques or other translation mechanisms, into the description languages of honeynet deployment platforms and tools.

Focused on Network Security Functions, in [11][12] is being studied an approach based on a model of capabilities that allows to unambiguously determine what Network Security Functions (NSFs) can do in terms

of security policy enforcement. Furthermore, when an unknown threat (e.g., zero-day exploits, unknown malware, and APTs) is reported by a network security device, new capabilities may be created, and/or existing capabilities may be updated. These new capabilities may be sent to and stored in a centralized repository or stored separately in a local repository. In either case, a standard interface is needed during this automated update process**.** In defining the capabilities of an NSF, the "Event-Condition-Action" (ECA) policy rule set model is used, and it consist of "An Event", defined as the occurrence of an important change in the system being managed, and/or in the environment of the system being managed, "A Condition", which is a set of attributes, features, and/or values that are to be compared with a set of known attributes, features, and/or values in order to make a decision NSFs provide security functions by executing various Actions.

Regarding the use of policies for orchestration [13] is focused on combining SDN and NFV principles aiming to formulate a baseline architecture that will facilitate policy-driven dynamic methods for (i) management of SDN resources, (ii) lifecycle management of VNFs and the associated data, and (iii) orchestration of multiple diverse VNFs to deliver Business Applications as NFV Services (i.e. Service Chains). The salient features of the proposed architecture include: Modular design by enabling hardware elements, VNFs, services and orchestration decoupling, Information Model describing and abstracting network resources and network functions, as well as permitting policy-based management of individual VNFs and orchestration of NFV Service Chains. The instantiation of NFV Services is governed by policy rules and the Policy Service of the NFVO is based on Ponder2.

In [14] authors present a novel SDN-based NFV orchestration framework, called APPLE, to enforce network function policies while providing the above properties. APPLE uses the SDN paradigm, running an Optimization Engine periodically to make adjustment according to the large time-scale network dynamics. It takes the traffic rate, forwarding path, and policy chain of each flow, together with the available hardware resources, as input. In this framework, policies are specified by network operators or applications describing the sequence of NFs that each class of flows need to traverse in order.

[15] introduces a novel model-driven method to orchestrate service policy within system context in design time. By applying top-down SOA (Service-oriented-Architecture) design methodology, firstly the logical service policy model based on WS-Policy is created to describe relation among service policy in logical level, and then it is transformed to physical service policy model with more factors of real system topology considered. Finally, service policy deployment model is generated to describe the relation among policy related artefacts and policy repositories, and guide and automate the deployment of policies in runtime environment as well. This framework is no focused on distributed systems and networks.

Of course, the policy for orchestration approach requires to be conscient of the possible policy conflicts. In that sense, [16] provides a taxonomy of semantic conflicts, it analyses the main features of each of them and provides an OWL/SWRL modelling for certain realistic scenarios related with information systems. It also describes different conflict detection techniques that can be applied to semantic conflicts and their pros and cons. In [17], as part of the solution, authors shows in which way an ontology is used as input for a Trust and Security Decision Support System, in order to assist in the Intercloud security decision making process, quantifying security expectations and trustworthiness about Cloud Service Providers. [18] builds up an efficient rule engine in a practical smart building system. Moreover, a rule engine adaption scheme is proposed to minimize the rule matching overhead.

# 3 Discussion on Progress Beyond the State of the Art

As it can be inferred from the previous section, from the point of view of the system and security management it is interesting to provide different levels of security policies in order to provide different levels of abstraction for different profiles of management. It is also important to highlight the difference between generic models and specific extensible models, as well as remark the policy orchestration features and policy conflict detection. Since ANASTACIA seeks to cover as much as possible use cases, we focus in solutions which provide policy models from the policy languages in order to support the extension and creation of new ones.

| Solution | Policy models |
|---|---|
| xCIM | Authentication, Authorization, Filtering, Channel Protection, Operation |
| SECURED | Authentication, Authorization, Filtering, Channel Protection, Operation (there are more concepts but pending to be extended) |
| WS-Policy Framework | Specific policy models for web services |
| Ponder2 | Obligation (ECA), Authorization |
| **ANASTACIA** | **Authentication, Authorization, Filtering, Channel Protection, Operation, Monitoring, Anonymity, IoT management, Traffic Divert, IoT Honeynet, Privacy, QoS, Data Aggregation, Policy for Orchestration** |

**Table 1: Main identified policy solutions**

*Table 1* shows the available policy models per solution. As it can be seen, xCIM and SECURED are the most suitable as to security policy models, but it is important to highlight that SECURED defines more security concepts pending to be extended. With this in mind, ANASTACIA policy model improves the SECURED current state of the art as well as provide novelty approaches in order to be able increase the security measures and countermeasures in the whole system at different levels. To this aim, ANASTACIA adopts and extend concepts and features from the state of art, in order to provide a unified security policy framework.

*Figure 1* shows the relationship between ANASTACIA and the most suitable projects or frameworks according on the ANASTACIA requirements. As it can be seen, from our point of view, the policy abstraction in two levels is reused and extended by SECURED from the xCIM-SPL/SDL concepts, in order to generate their owns High-level and Medium-level policy languages. Later the I2NSF reuses and extends concepts from SECURED, specially exploding the capability concept, and flow policy rules, but in this case providing an ECA approach. ANASTACIA then extends the HSPL/MSPL policy models since they already provide a good base of specific multi-domain security policies, providing more capabilities and security concepts, as well as providing policy for orchestration properties in the policy itself. Besides, the ECA concept has been adopted and included to HSPL/MSPL models in order to be able dealing not only with conditions and actions, but even events. Finally, the policy for orchestration is supervised by a rule engine based on the new extended models.

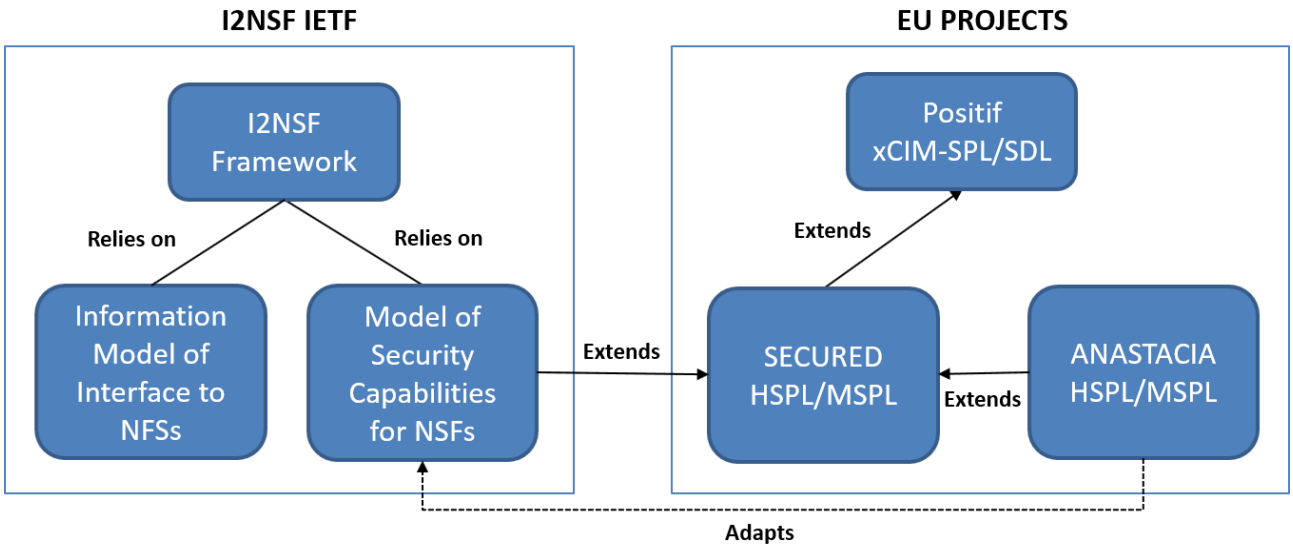


Figure 1: Policy Models Relationship

In this way, **ANASTACIA's policy related main contributions reside in the unification of relevant, new and extended capability-based security policy models (including ECA features), as well as policy orchestration and conflict detection mechanisms. All the former under a unique policy framework.**

The following text highlights the extended and the new features for the ANASTACIA HSPL/MSPL policy models. Some of them has been categorized as new since there were not policy models defined in SECURED for them.

Extended Security Policy Features:

- **HSPL general schema:** Actions and objects have been extended in order to represent the new kind of elements for the new security policies. Purpose and resources model have been extended in order to represent key-value pairs. Base HSPL policy model has been extended in order to provide dependences, priority and bidirectionality.
- **MSPL general schema:** Capabilities has been extended in order to represent the values of the new MSPL security policies.
- **Channel Protection Policy:** The policy has been extended in order to allow the representation of IoT related technologies like the channel protection DTLS protocol.
- **Filtering Policy:** The policy has been extended in order to provide QoS values, as well as IoT application layer conditions (e.g. CoAP).
- **Authorization Policy:** The policy has been extended in order to provide specific authentication actions and conditions.

New Security Policy Features:

- **Authentication configuration**: It has been modelled an authentication policy, providing authentication conditions and actions, including different authentication options for different methods and mechanisms.
- **Monitoring configuration policy:** It has been modelled a monitoring policy which is able to provide multiple monitoring conditions with detection filters and signatures, as well as different actions like alerts or reports.
- **Privacy:** It has been modelled a privacy policy, providing specific privacy conditions and actions, including different privacy methods like attribute based or PKI based.
- **Network Anonymity:** It has been modelled a network anonymity policy capable to specify different anonymity actions and technologies like Onion routing or Traffic Mixing.

- **IoT Control:** It has been modelled IoT control policies in order to perform command and control configurations like power management.
- **IoTHoneynet:** It has been modelled an IoT honeynet policy which provides an IoT network and IoT devices configuration in order to be able to replicate the IoT infrastructure configuration.
- **QoS:** It has been modelled a quality of service policy in order to represent different quality of service profiles as well as its parameters.
- **Data Aggregation:** It has been modelled a data aggregation policy, providing to security administrators specific configurations in order to aggregate specific the data matched in the aggregation conditions.
- **Policy for Orchestration:** It has been modelled the policy for orchestration which allows specifying the deployment of multiple security policies. To this aim the policies included in a policy for orchestration can indicate orchestration parameters like priority or dependences. They have been considered two different kind of dependences, these are, policy dependences and event dependences. A policy dependence indicates that a policy cannot be enforced until other policy is in a specific status. On the other hand, an event dependence (ECA concept) indicates that a policy cannot be enforced until a specific event occurs.
- **Policy conflict detection:** It has identified the main applicable conflicts to the policy for orchestration using directly the proper policy models instead to defining a new ontology. It is also providing a policy conflict detector by specifying generic policy rules in independent way of the rule engine.

# 4 ANASTACIA ARCHITECTURE OVERVIEW

The ANASTACIA system model is structured as a set of layers that provide a broad view of the framework and stand out its integration within IoT infrastructures. ANASTACIA is envisioned as a framework integrated on top of an IoT infrastructure which can be managed by a system administrator by using security policies. The whole information focused on the ANASTACIA architecture can be found on D1.3.

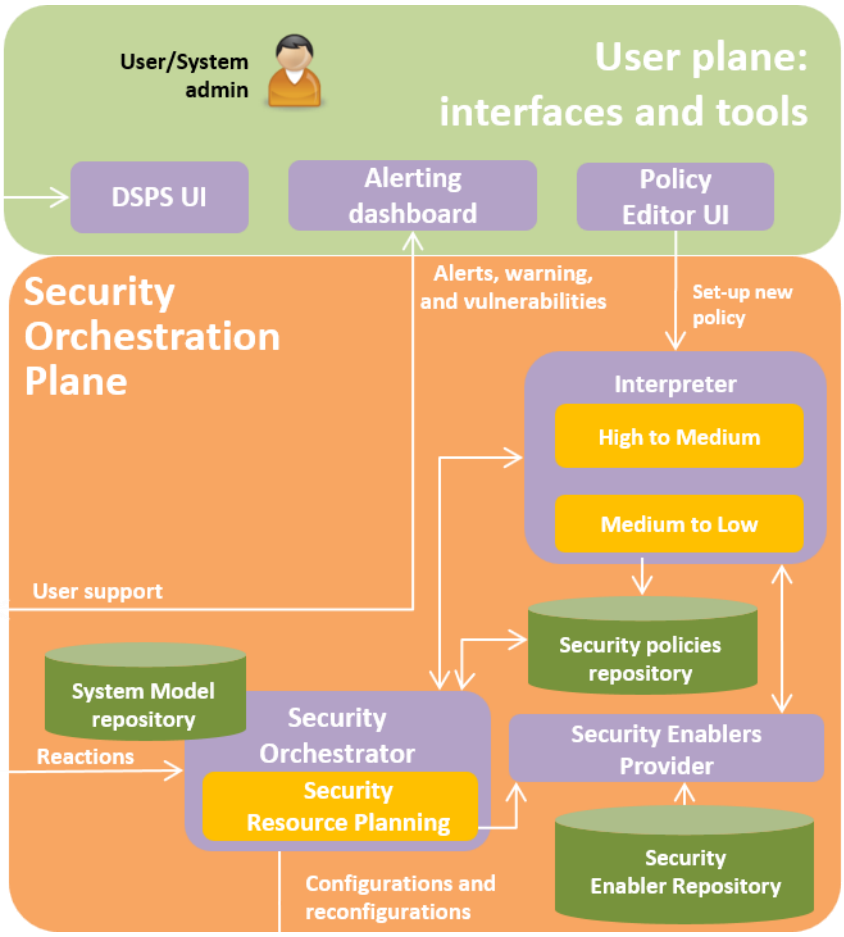


Figure 2: User/Orchestration Planes

*Figure 2* represents a slice of the whole ANASTACIA framework, focused on the most relevant planes involved on policy management, these are the **User Plane** and the **Orchestration Plane**.

- The **User Plane** includes interfaces, applications and tools that help system administrators to manage the IoT platform through the ANASTACIA framework. For example, at this plane system admins are able to edit the security policies that govern the underlying IoT platform.
- The **Security Orchestrator Plane** organizes the resources that support the Enforcement Plane, carrying out activities such as the transformation of security properties to configuration rules and aligning the security policies defined by the security interpreter with the provisioning of relevant security mechanisms. It has the whole vision of the underlying infrastructure and the resources and interfaces available at the Security Enforcement Plane.

By the interaction between both planes, the administrator is able to define security policies and enforce them in the underlaying architecture. In this process the following components of ANASTACIA framework are involved:

- A **Policy Editor Tool** at the User plane that can be used by the User/System admin to set the Security policy to be enforced in the IoT platform.
- An **Interpreter** in the Security Orchestration Plane that will transform the Security policy (closer to a human readable policy) to a machine-readable policy that is able to represent lower configurations parameters.
- A **Security Enabler Provider** in the Security Orchestration Plane, that can identify the security enablers that can provide specific security capabilities to meet the security policies requirements.
- A **Security orchestrator** in the Security Orchestration Plane is responsible for selecting the security enablers to be used in the policy refinement process and configuring the Monitoring and Reaction Plane according to the security policy to enforce.

Once the security policies are translated in configurations for the selected security enablers, the **Security orchestrator** that has the whole vision of the subjacent infrastructure is able to trigger the enforcement of the defined policies using the corresponding configurations or tasks obtained during the policy translation process.

# 5 ANASTACIA SECURITY POLICY MODELS

ANASTACIA framework reuses and extends the security policy models from the European SECURED project [1] which defines two levels of security policies, high and medium level security policies.

The High-level Security Policy Language (HSPL) [2] and the Medium-level Security Policy Language (MSPL) [2] are two policy languages defined within the European SECURED project to specify security policies. HSPL is the policy language suitable for expressing the general protection requirements of typical non-technical end-users, such as “do not permit access to illegal content” or “block access to peer-to-peer networks”. MSPL is an abstract language with statements related to the typical actions performed by various security controls but expressed independent of the final devices, it means, expresses specific configurations by non-technically users in a device-independent format, such as “deny *.sex”, “deny src 192.168”, or “inspect image/* for malware”.

Both policy languages are based on XML and are focused on the capability concept. A capability denotes any kind of security functionality that can be provided by a Security Enabler. A Security Enabler implements some security controls, generally, by a software module, e.g. filtering, logging or authentication. ANASTACIA project extends and adapt HSPL, MSPL models and capabilities by defining new security policies like traffic divert, privacy, QoS or network anonymity in order to comply with the ANASTACIA framework objectives.

## 5.1 HIGH-LEVEL SECURITY POLICY LANGUAGE (HSPL)

The High-level security policy language (HSPL) is a security policy language with a high level of abstraction which allows to model security policies regardless of the underlaying technology, being this a key feature for the framework, allowing multiple implementations and enforcement points for the same high-level policy. This level of abstraction also provides other important features such as allowing a non-technical end-user to specify general protection requirements without a deep knowledge of the lower layers of the system.

Since the main idea of the HSPL is to define high level policies capable to model security requirements in terms of who is allowed or not to apply some action over some element under some conditions, the HSPL structure is defined and extended as:

```
[ sbj ] act obj [ (field_type,value) ...(field_type,value) ] [ dependence ] [ priority ]
```

Where the `sbj` indicates the subject involved in some action over an element, `act` represents the action to be carried out by the subject, the `obj` indicates the element or resource where will be apply the action. A set of `fields` and values indicate some conditions about the action in the security requirement. Finally, `dependence` and `priority` provide relevant information regarding how the policy must be orchestrated.

### 5.1.1 Subject

Considering the nature of the present project, the following subjects have been considered in order to represent the main actors of the ANASTACIA framework:

- **Specific user**: e.g. Alice.
- **User group:** e.g. Administrators.
- **Specific device/VNF:** e.g. Sensor-A.
- **Device/VNF group:** e.g. 1st floor sensors.

### 5.1.2 Action

For the scope of ANASTACIA, it has been considered the following set of HSPL actions, which allows the modelling of security policies capable of enforcing the main security requirements identified in the use cases:

- **Authorized/No Authorized to access:** Indicates if the subject will be able to access or not to some element or resource.
- **Configure authentication:** Indicates that the subject must be configured using a specific authentication mechanism.
- **Configure privacy:** The subject must apply some kind of privacy against a target.
- **Enable:** Action which activates or deactivate some element or resource.
- **Protect confidentiality integrity:** To guarantee communication´s confidentiality and integrity.
- **Configure Monitoring:** It must be performed a monitoring in the specified subject, looking for a specific pattern.
- **Configure QoS:** Indicates that it is required a specific Quality of Service for the subject resources.
- **Data Aggregation:** It is required to perform a data aggregation from different sources.
- **Configure Network Anonymity:** It is necessary to perform a configuration in order to anonymize the subject at network level.

### 5.1.3 Object

As for the element or resource where the action will be performed, the following set of HSPL objects has been considered. The objects correspond to the main use cases of the ANASTACIA project are:

- **All traffic:** Whatever kind of traffic.
- **PANA Traffic:** Traffic for PANA authentication.
- **CoAP-EAP Traffic:** Traffic for specific IoT light way authentication.
- **CoAP Traffic:** Traffic for the Constrained Application protocol.
- **ICMP Traffic:** Traffic for ICMP.
- **Resource:** The element/object where the action is performed.

### 5.1.4 Field condition

The field conditions establish the constraints about the action to perform. HSPL defines several kinds of conditions of which we are using the following ones:

- **Time:** To specify timing conditions, e.g. the security requirement duration.
- **Specific URL:** Concrete URL or website.
- **Type of content:** Specific content affected by the policy.
- **Traffic target:** Specific target affected by the policy.
- **Purpose:** Specific purpose of the condition e.g. specific method or parameter of the request.
- **Resource value:** The specific resource.

### 5.1.5 Dependence, priority and Enabler candidates

Dependence and priority allow establishing dependence relationship between HSPL policies, as well as the magnitude of the security policy respect the other ones. These features are very useful to provide policies for orchestration from the high-level approach. A dependence has been categorized as:

- **Policy dependence:** A high-level security policy depends on the refinement, translation or enforcement of another one.
- **Event dependence:** A high-level security policy depends on some kind of event in the system (e.g. Authentication success).

### 5.1.6 Combining HSPL components

This section exposes how the HSPL components, which are, subject, action, object and condition fields can be combined to build a complete HSPL policy. *Table 2* shows the combination among the actions and the possible objects for the HSPL applied on the ANASTACIA framework.

| Action/Object | All traffic | Internet traffic | Intranet traffic | CoAP traffic | ICMP traffic | PANA traffic | CoAP-EAP traffic | Resource |
|---|---|---|---|---|---|---|---|---|
| **Authorized/No Authorized to access** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Configure authentication** | | | | | | ✓ | ✓ | |
| **Enable** | | | | | | | | ✓ |
| **Protect confidentiality integrity** | ✓ | ✓ | ✓ | ✓ | | | | |
| **Configure Privacy** | | | | | | | | ✓ |
| **Configure Monitoring** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| **Configure QoS** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| **Data Aggregation** | ✓ | | | ✓ | | | | |
| **Configure Network Anonymity** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |

**Table 2: HSPL Action - Object combination**

*Table 3* shows the possible combinations of the conditional fields and objects for the HSPL components in ANASTACIA framework.

| Field/Object | All traffic | Internet traffic | Intranet traffic | CoAP traffic | ICMP traffic | PANA traffic | CoAP-EAP traffic | Resource |
|---|---|---|---|---|---|---|---|---|
| **Traffic Target** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| **Purpose** | | | | | | ✓ | ✓ | ✓ |
| **Resource value** | | | | | | | | ✓ |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Time** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Specific URL** | ✓ | ✓ | ✓ | ✓ | | | | |
| **Type of content** | ✓ | ✓ | ✓ | ✓ | | | | |

Table 3: HSPL Field - Object combination

### 5.1.7 HSPL example

According to the exposed combinations of the components for the HSPL in previous sections, here is shown a simple example in order to model the sentence:

```
IoT-lab is authorized to access PANA traffic
     (time period, {08:00-18:00 GMT+1})
```

That sentence is modelled in XML as the following, specifying the subject and the identification of the HSPL as attribute of the XML element.

```
<hspl subject='IoT-lab' id='HSPL0' >
  <action>authorise_access</action>
  <objectH>PANA_traffic</objectH>
  <fields>
    <traffic_target>
      <target_name>PANA_AGENT</target_name>
    </traffic_target>
    <time_period time-zone='UTC'>
      <interval_time >
        <time_hours start-time='08:00:00' end-time='18:00:00'/>
      </interval_time>
    </time_period>
  </fields>
</hspl>
```

Figure 3: HSPL example

*Figure 3* indicates that the *IoT-lab*, as *subject*, is *authorized to access*, as *action*, *PANA traffic*, as *object*, against the *PANA_AGENT* as target, during a certain period of time as a condition field. As it can be seen, this level only expresses a security requirement without technical or deeper specific implementation information.

## 5.2 Medium-level Security Policy Language (MSPL)

The Medium-level security policy language (MSPL) is a security policy language with a medium level of abstraction. It is still independent on the underlaying technology, but closer to it than an HSPL one. This is, it provides in general way a set of actions suitable by the most common applicable security settings (e.g. ALLOW or DENY IP_ADDRESS sentences), which is also interesting for technical users.

Regarding the structure of MSPL, the main element is the **ITResource**, which is explained following.

### 5.2.1 IT Resource

*ITResource* intends to represent the configuration of a piece of software or hardware (thereinafter Security Enabler) capable to applying the enforcement of the security policy. Each *ITResource* is able to represent ***Configurations***, ***Dependences***, **Priorities** and **EnablerCandidates**.

### 5.2.1.1 Configuration

A configuration is composed by ***Capabilities*** and the required ***Configuration Rules*** in order to configure the future security enabler who will implement the capability.

#### 5.2.1.1.1 Capabilities

The concept of capability exposes a kind of security functionality which can be provided by a Security Enabler. For instance, if the ANASTACIA framework offers a Security Enabler capable of performing filtering actions, it must implement a filtering capability and it must be ready to be configured with the most common filtering fields, e.g. source address, destination address and so on. According to the main identified use cases, *Table 4* shows the main identified ANASTACIA framework capabilities:

| Capability | Description | Enabler example |
|---|---|---|
| AUTHORISE_ACCESS_RESOURCE | Authorizes access to a specific resource. | PDP (XACML) |
| AUTHENTICATION | Allows configure the authentication mechanism. | PANA AGENT |
| PRIVACY | Allows activate some privacy techniques. | CPAB |
| FILTERING_L3 | The Security enabler is able to perform filtering operations up to level 3. | OVS |
| FILTERING_L4 | The Security enabler is able to perform filtering operations up to level 4. | OVS |
| TRAFFIC_DIVERT | The Security Enabler is able to perform traffic divert operations. | OVS |
| TRAFFIC_INSPECTION_L7 | The Security Enabler is able to inspect the network traffic up to level 7. | SNORT |
| NETWORK_TRAFFIC_ANALYSIS | The Security Enabler is able to analyse the network traffic. | SNORT |
| PROTECTION_CONFIDENTIALITY_INTEGRITY | The Security Enabler is able to prepare secure connections using specific configurations. | Apache SSL |
| DTLS_PROTOCOL | The Security Enabler is able to prepare DTLS connections using specific configurations. | DTLS-Proxy |

| | | |
|---|---|---|
| IOT_CONTROL | The Security Enabler is able to perform command and control operations over the IoT domain. | IoT Controller |
| IOT_HONEYNET | The Security Enabler is able to deploy and manage a virtual IoT-Honeynet. | Cooja agent |
| QoS | The Security Enabler is able to provide a specific Quality of Service. | OVS |
| NETWORK_ANONYMITY | The Security Enabler is able to provide anonymity at network level | TOR |
| DATA_AGGREGATION | The Security Enabler is able to aggregate the data received from multiple sources | Kafka Broker |

**Table 4: Capabilities**

#### 5.2.1.1.2 Configuration Rule

The configuration rule element denotes an abstract set of configuration settings which are independent to the final Security Enablers. Specifically, a configuration rule is composed by ***Conditions*** and ***Actions***. In semantic terms, this modelling allows to specify that if the conditions are accomplished, it must be triggered the action. Since different capabilities could require distinct kind of configurations, the schema provides a hierarchical approach with inheritance for actions and conditions.

##### 5.2.1.1.2.1 Conditions

A condition models a set of fields in order to represent the values that the subject must satisfy in order to carry out the action. According on the capabilities, the main conditions can be:

- **Authentication Condition:** Able to represent fields like the authentication subject.
- **Authorization Condition:** It allows to represent fields like the source, destination and the specific resource to access.
- **Filtering Configuration Condition:** Able to represent fields related to filtering e.g. source address, source port and so on.
- **Traffic Divert Configuration Condition:** It allows to represent the network traffic conditions in order to perform the traffic divert action.
- **Data Protection Condition:** Allowing to specify integrity and/or encryption mechanism.
- **Monitoring Configuration Condition:** Able to represent specific monitoring fields like attack signatures.
- **Privacy Condition:** It allows specifying fields like who will be the subject of the privacy.
- **QoS Condition:** It allows providing specific QoS fields like the QoS profile or throughput.
- **Anonymity Configuration Condition:** It allows to represent network data of the subject that must be satisfied in order to perform the anonymity action.

#### 5.2.1.1.2.2 Actions

An action models a set of fields in order to configure the specific action to be carried out once the conditions are satisfied. According on the capabilities, the main actions can be:

- **vIoT-HoneyNet Action**: It allows specifying an IoT-Honeynet action like deploy or remove the IoT-honeynet specified in the model.
- **Traffic Divert Action**: It specifies the action for the traffic divert, e.g. forward.
- **Filtering Action**: The action for the filtering condition, e.g. allow or deny.
- **Power Management Action**: It specifies actions related on power management like turn off or reset the devices.
- **Data Protection Action**: The action specifies how must be configured the channel protection.
- **Privacy Action**: It allows specifying the privacy configuration.
- **Authorization Action**: It specifies the action for the authorization like allow or deny.
- **Authentication Action**: It specifies the authentication mechanism, methods and parameters.
- **Monitoring Action**: It allows represent fields like the monitoring action type, reports and so on.
- **QoS Action**: It specifies the QoS action like specifying a profile, bandwidth…
- **Anonymity Action**: It specifies the anonymity action type, including the anonymity technology parameters depending on the nature of the anonymity solution.

### 5.2.1.2 Dependences, priority and Enabler candidates

In the same way as in HSPL, dependence and priority allow establishing dependence relationship between MSPL policies, as well as magnitude of the security policy respect the other ones in order to provide policies for orchestration from the medium-level approach. A dependence has been categorized as:

- **Policy dependence:** A medium-level security policy depends on the translation or enforcement of another one.
- **Event dependence:** A medium-level security policy depends on some kind of event in the system (e.g. Authentication success).

Regarding the Enabler candidates, this field represents a set of Security Enablers which implements the capabilities required by the security policy. For instance, if the security policy requires traffic filtering, the security enabler candidates could be those related with networking management like SDN controllers, IPTables, specific firewalls…

## 5.2.2 MSPL Components relationship

*Figure 4* shows the relationship among the main MSPL components. The *ITResource* represents the Security Enabler configuration, the priority, dependences and enabler candidates which will be able to enforce a configuration. Regarding the configuration, it contains the capability or capabilities required by the policy, and it is extended as a set of rules. This rule set is composed by configuration rules, and these will provide configuration rule actions and conditions. The configuration rule actions are extended in the action specific action to perform e.g. *FilteringAction*, specifying the action type. On the other hand, the configuration conditions are extended in order to represent the specific conditions that must be accomplished, e.g. The address which will be affected by the action.

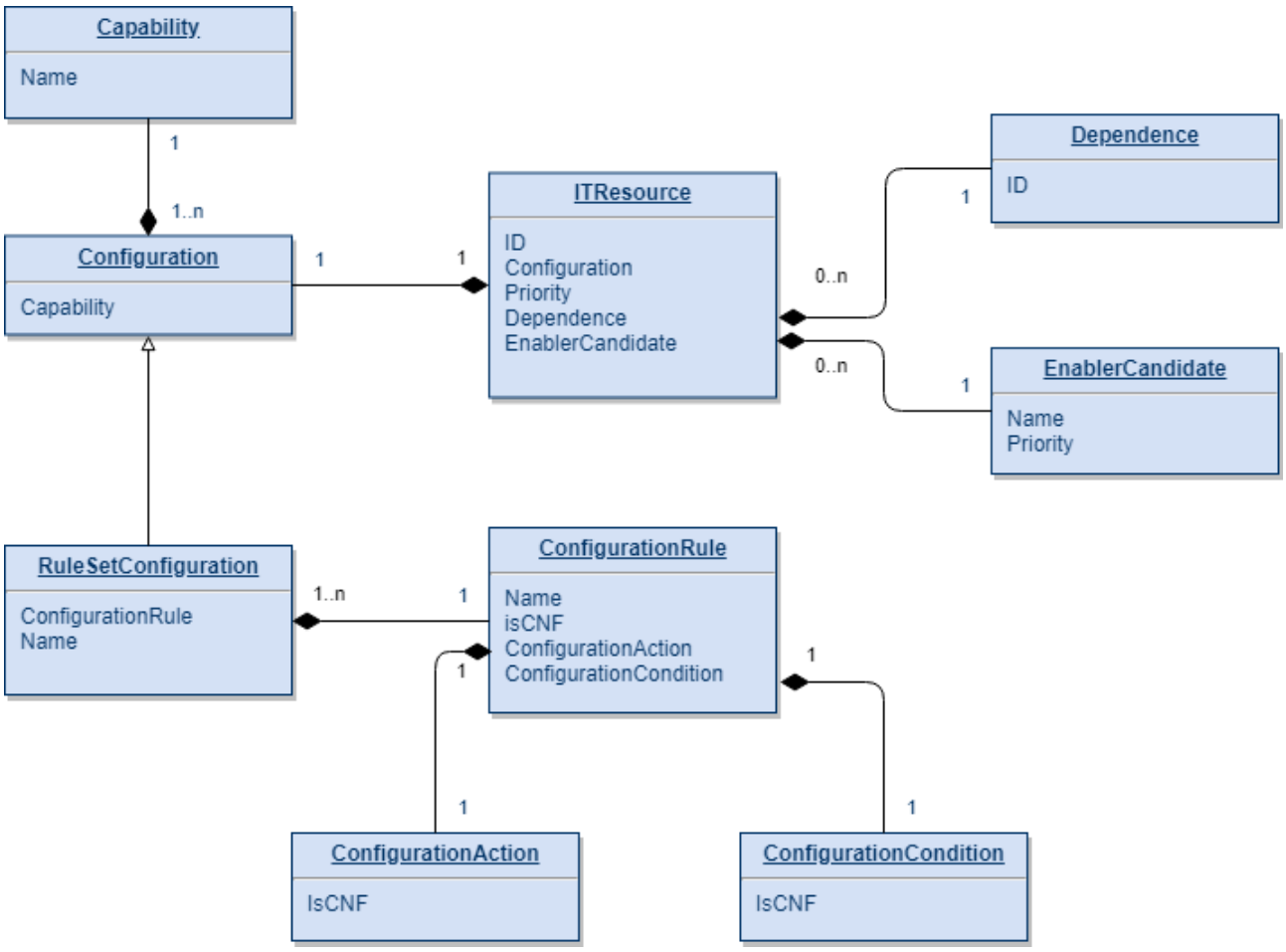


Figure 4: ANASTACIA-MSPL Scheme

Next section provides an example including an XML code where it can be observed in a practical way an instantiation of the main diagram elements.

### 5.2.3 MSPL Example

According to the exposed components for the MSPL, here is shown a simple example corresponding to the following HSPL sentence:

```
IoT-lab is authorized to access PANA traffic
     (time period, {08:00-18:00 GMT+1})
```

The process of how is refined a HSPL into a MSPL policy it is explained in D2.1. At this point we will focus on the MSPL definition for the specified example.

```
<ITResource ID="MSPL_f9b27422-15b3-4bb5-ad21-3e08af5b1a1c">
    <configuration xsi:type="RuleSetConfiguration">
        <capability>
              <Name>Timing</Name>
         </capability>
         <capability>
            <Name>Traffic_Divert</Name>
        </capability>
        <configurationRule>
            <configurationRuleAction xsi:type='TrafficDivertAction' >
```

```
            <TrafficDivertActionType>FORWARD</TrafficDivertActionType>
            <packetDivertAction>
              <isCNF>false</isCNF>
              <packetFilterCondition>
                <Interface>7</Interface>
              </packetFilterCondition>
            </packetDivertAction>
          </configurationRuleAction>
          <configurationCondition
                      xsi:type='TrafficDivertConfigurationCondition'>
            <isCNF>false</isCNF>
              <packetFilterCondition>
                  <SourceAddress> 2001:720:1710:4::0/64</SourceAddress>
                  <SourcePort>5678</SourcePort>
                  <ProtocolType>UDP</ProtocolType>
              </packetFilterCondition>
              <timeCondition>
                  <Weekday></Weekday>
                  <Time>08:00-19:00,</Time>
              </timeCondition>
           </configurationCondition>
          <externalData xsi:type='Priority'>
              <value>60000</value>
          </externalData>
        </configurationRule>
        <Name>Conf0</Name>
    </configuration>
</ITResource>
```

Figure 5: MSPL Example

*Figure 5* shows a MSPL policy example for traffic divert in a specific timing interval, so the ITResource represents a Security Enabler with timing and traffic divert capabilities. Regarding the condition and action, the MSPL models that the UDP datagrams with the specified source port and source Ipv6 address must be forwarded through the specified interface.

## 5.3 Security Policy Models Definition

This section illustrates the policy models for High-level Security Policy Language and Medium-level Security Policy Language adapted and extended to the ANASTACIA framework.

### 5.3.1 HSPL

The HSPL model is compounded by several complex types. Following it is explained the most relevant model components. The main complex type for the HSPL policy model is the **`hspl`** itself, which, as can be observed in *Figure 6*, we are highlighting the action, object, generic fields, dependences, enabler candidates, the identifier of the HSPL in order to provide traceability, and finally, the subject as attribute of the type.

```
<complexType name="hspl">
  <sequence>
    <element name="action" type="tns:action" minOccurs="1" maxOccurs="1"/>
    <element name="objectH" type="tns:objectH" minOccurs="1" maxOccurs="1"/>
    <element name="fields" type="tns:fields" minOccurs="0" maxOccurs="1"/>
    <element name="dependences" type="tns:dependences" minOccurs="0"
      maxOccurs="1"/>
    <element name="candidates" type="tns:candidates" minOccurs="0"
      maxOccurs="1"/>
  </sequence>
  <attribute name="subject" type="string" use="optional"></attribute>
  <attribute name="id" type="ID" use="required"></attribute>
```

```
</complexType>
```

Figure 6: HSPL Complex Type

**Action** is a simple string type constrained to specific values which has been extended for ANASTACIA. *Figure 7* shows the most relevant actions applicable to the ANASTACIA framework according to the main identified policies.

```
<simpleType name="action">
        <restriction base="string">
                <enumeration value="authorise_access"></enumeration>
                <enumeration value="no_authorise_access"></enumeration>
                <enumeration value="enable"></enumeration>
                <enumeration value="prot_conf_integr"></enumeration>
                <enumeration value="config_authentication"></enumeration>
                <enumeration value="config_privacy"></enumeration>
                <enumeration value="config_monitoring"></enumeration>
                <enumeration value="config_qos"></enumeration>
                <enumeration value="data_aggregation"></enumeration>
                <enumeration value="config_network_anonymity"></enumeration>
                …
        </restriction>
</simpleType>
```

Figure 7: Action Simple Type

Regarding the **objectH** type, it is also a simple string type like the previous one. *Figure 8* shows the string restriction for main values related with the ANASTACIA framework which are mainly related with kind of traffic and resources.

```
<simpleType name="objectH">
        <restriction base="string">
                <enumeration value="AllTraffic"></enumeration>
                <enumeration value="Internet_traffic"></enumeration>
                <enumeration value="Intranet_traffic"></enumeration>
                <enumeration value="PANA_traffic"></enumeration>
                <enumeration value="CoAP_traffic"></enumeration>
                <enumeration value="CoAP-EAP_traffic"></enumeration>
                <enumeration value="ICMP_traffic"></enumeration>
                <enumeration value="resource"></enumeration>
                …
        </restriction>
</simpleType>
```

Figure 8: Object Simple Type

**Fields** type is a complex type composed by a sequence of other complex types which provide to the field more accuracy for the high-level security policy specification.

```
<complexType name="fields">
 <sequence>
  <element name="time_period" type="tns:time_period" minOccurs="0"
    maxOccurs="1"/>
  <element name="traffic_target" type="tns:traffic_target" minOccurs="0"
    maxOccurs="1"/>
  <element name="specific_URL" type="tns:specific_URL" minOccurs="0"
    maxOccurs="1"/>
  <element name="type_content" type="tns:type_Content" minOccurs="0"
    maxOccurs="1"/>
  <element name="purpose" type="tns:purpose" minOccurs="0" maxOccurs="1"/>
  <element name="resource_values" type="tns:resource_values" minOccurs="0"
    maxOccurs="1"/>
 </sequence>
```

```
</complexType>
```

Figure 9: Fields Complex Type

*Figure 9* shows the main components for the **`fields`** complex type. Time period is a complex type which allows specifying a time interval, including days and time hours. Traffic target, specific URL, type content, purpose and the resource values are complex types formed by a sequence of simple string types providing some flexibility for the high-level policy specification. Some of them have been extended in order to represent key-value pairs.

Regarding the high-level security policy **`Dependences`**, both aforementioned, policy dependence and event dependence inherit from **`Dependence`** and both contains a `DependenceCondition` which must be solved in order to satisfy the dependence. *Figure 10* shows the model fields in order to specify that an HSPL policy depends on the `status` of another one, where `policyID` codifies a high-level security policy identifier.

```
<complexType name="PolicyDependence">
  <complexContent>
  <extension base="tns:Dependence" >
    <sequence>
      <element name="dependenceCondition" type="tns:PolicyDependenceCondition"/>
    </sequence>
  </extension>
  </complexContent>
</complexType>

<complexType name="PolicyDependenceCondition">
  <complexContent>
  <extension base="tns:DependenceCondition" >
    <sequence>
      <element name="policyID" type="string"/>
      <element name="status" type="string"/>
        </sequence>
  </extension>
  </complexContent>
</complexType>
```

Figure 10: Policy Dependence

On the other hand, *Figure 11* shows a dependence which is solved by a specific event. The kind of event is identified by the `eventID` and it is verified against the `EventDependenceCondition` which is composed by the main fields of a high-level security policy, providing the `subject`, `action`, `object` and `fields`.

```
<complexType name="EventDependence">
  <complexContent>
  <extension base="tns:Dependence" >
    <sequence>
      <element name="eventID" type="string"/>
      <element name="dependenceCondition" type="tns:DependenceCondition"
        minOccurs="0" maxOccurs="1"/>
    </sequence>
  </extension>
  </complexContent>
</complexType>

<complexType name="EventDependenceCondition">
  <complexContent>
  <extension base="tns:DependenceCondition" >
    <sequence>
      <element name="subject" type="string"></element>
      <element name="action" type="tns:action" minOccurs="0"></element>
      <element name="objectH" type="tns:objectH" minOccurs="0"></element>
      <element name="fields" type="tns:fields"></element>
```

```
    </sequence>
  </extension>
  </complexContent>
</complexType>
```

Figure 11: Event Dependence

Regarding the **Candidates**, *Figure 12* shows the candidates field which represents a list of candidates, where a **Candidate** is composed by its name and a priority, allowing to establish an index of suitability between them.

```
<complexType name="Candidates">
    <sequence>
      <element name="candidate" type=" tns:Candidate"
        minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>

<complexType name="Candidate">
    <sequence>
      <element name="name" type="string"/>
      <element name="priority" type="integer"/>
    </sequence>
</complexType>
```

Figure 12: Enabler Candidates

The following section exposes the security policy model definition regarding the main identified policies on ANASTACIA framework.

### 5.3.2 MSPL

In the same way as HSPL, the Medium-level Security Policy Language is also composed by different complex types, but this time, the main component is not an MSPL policy, but an **ITResourceType**.

```
<complexType name="ITResourceType">
  <sequence>
    <element maxOccurs="1" minOccurs="1" name="configuration"
      type="ITResource:Configuration"/>
    <element maxOccurs="1" minOccurs="0" name="priority" type="integer"/>
    <element maxOccurs="1" minOccurs="0" name="dependences"
      type="ITResource:Dependences">
    <element maxOccurs="1" minOccurs="0" name="enablerCandidates"
      type="ITResource:EnablerCandidates"></element>
  </sequence>
  <attribute name="ID" type="string"></attribute>
</complexType>
```

Figure 13: ITResource type

*Figure 13* shows the **ITResourceType** definition, which contains an abstract (non-implementation specific) **configuration** for a Security Enabler. As in the HSPL case, they have been also included the **priority**, **dependences** and **enablerCandidates**, as well as an identifier in order to provide traceability for the MSPL.

```
<complexType name="Configuration">
  <sequence>
    <element maxOccurs="unbounded" minOccurs="1" name="capability"
type="ITResource:Capability"/>
  </sequence>
</complexType>
```

Figure 14: ITResource configuration

*Figure 14* shows the **Configuration** definition for an *ITResource*, which is composed by a sequence of the required **capability** list to enforce the security policy. *Figure 15* shows the **Capability** definition, which is composed by a **CapabilityType**. The latter is a simple string type restricted to the main set of values used on ANASTACIA.

```
<complexType name="Capability">
  <sequence>
    <element maxOccurs="1" minOccurs="1" name="Name"
      type="ITResource:CapabilityType"></element>
  </sequence>
</complexType>

<simpleType name="CapabilityType">
  <restriction base="string">
    <enumeration value="Filtering_L3"/>
    <enumeration value="Filtering_L4"/>
    <enumeration value="Timing"/>
    <enumeration value="TrafficInspection_L7"/>
    <enumeration value="Network_traffic_analysis"/>
    <enumeration value="Protection_confidentiality"/>
    <enumeration value="Protection_integrity"/>
    <enumeration value="AuthoriseAccess_resource"/>
    <enumeration value="Authentication"/>
    <enumeration value="Traffic_Divert"/>
    <enumeration value="IoT_control"/>
    <enumeration value="DTLS_protocol"/>
    <enumeration value="IoT_honeynet"/>
    <enumeration value="Privacy"/>
    <enumeration value="Anonymity"/>
    <enumeration value="QoS"/>
    <enumeration value="Data_aggregation"/>
  </restriction>
</simpleType>
```

Figure 15: Capability Complex Type

To provide the configuration rules, the **Configuration** is used as base class, which is extended in order to model a **RuleSetConfiguration**.

```
<complexType name="RuleSetConfiguration">
  <complexContent>
    <extension base="ITResource:Configuration">
      <sequence>
        <element maxOccurs="unbounded" minOccurs="0" name="configurationRule"
          type="ITResource:ConfigurationRule"/>
        <element maxOccurs="1" minOccurs="1" name="Name" type="string"/>
      </sequence>
    </extension>
</complexContent>
```

Figure 16: RuleSetConfiguration

*Figure 16* shows the components of the rule set configuration, which is basically a set of **ConfigurationRule** and a name or description. A **ConfigurationRule** allows modelling a specific rule. As it can be observed in *Figure 17*, it is achieved by indicating mainly a **configurationRuleAction** and a **configurationCondition**. On the other hand, the field **isCNF** indicates if the rule must be satisfied completely, or it is enough satisfying any field.

```
<complexType name="ConfigurationRule">
  <sequence>
```

```
    <element maxOccurs="1" minOccurs="1" name="configurationRuleAction"
      type="ITResource:ConfigurationAction"/>
    <element maxOccurs="1" minOccurs="1" name="configurationCondition"
      type="ITResource:ConfigurationCondition"/>
    <element maxOccurs="1" minOccurs="1" name="Name" type="string"/>
    <element maxOccurs="1" minOccurs="1" name="isCNF" type="boolean"/>
  </sequence>
</complexType>
```

Figure 17: ConfigurationRule

Since the policy model allows the inheritance, the actions and conditions can be instantiated in different ways depending on the security policy. This is, some actions and conditions have been extended as well as new specific actions and conditions have been defined for new specific security policies in the ANASTACIA framework like, authentication, authorization, privacy, monitoring, traffic divert, QoS, anonymity or the IoT related ones. Following it is shown the security policy model definitions regarding the main identified policies on ANASTACIA framework for the Medium-level Security Policy Language.

### 5.3.2.1 Filtering Policy

MSPL filtering policy models are intended to represent networking related information configuration. *Figure 18* shows how the base configuration condition is extended, allowing to establish specific networking parameters like **`packetFilterCondition`**, `statefulCondition`, `timeCondition`, **`applicationLayerCondition`** and **`qosCondition`** (the latter will be explained in the QoS policy section).

```
<complexType name="FilteringConfigurationCondition">
  <complexContent>
    <extension base="ITResource:ConfigurationCondition">
      <sequence>
        <element maxOccurs="1" minOccurs="0" name="packetFilterCondition"
          type="ITResource:PacketFilterCondition"/>
        <element maxOccurs="1" minOccurs="0" name="statefulCondition"
          type="ITResource:StatefulCondition"/>
        <element maxOccurs="1" minOccurs="0" name="timeCondition"
          type="ITResource:TimeCondition"/>
        <element maxOccurs="1" minOccurs="0" name="applicationLayerCondition"
          type="ITResource:ApplicationLayerCondition"/>
        <element maxOccurs="1" minOccurs="0" name="qosCondition"
          type="ITResource:QoSCondition"/>

      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 18: FilteringConfigurationCondition

*Figure 19* shows the modelling of **`PacketFilterCondition`** related fields, these are, source address, destination address, source and destination ports, the direction of the connection establishment, the interface and the protocol type.

```
<complexType name="PacketFilterCondition">
  <sequence>
    <element maxOccurs="1" minOccurs="0" name="SourceAddress" type="string"/>
    <element maxOccurs="1" minOccurs="0" name="DestinationAddress"
       type="string"/>
    <element maxOccurs="1" minOccurs="0" name="SourcePort" type="string"/>
    <element maxOccurs="1" minOccurs="0" name="DestinationPort" type="string"/>
    <element maxOccurs="1" minOccurs="0" name="direction" type="string"/>
    <element maxOccurs="1" minOccurs="0" name="Interface" type="string"/>
```

```
    <element maxOccurs="1" minOccurs="0" name="ProtocolType" type="string"/>
  </sequence>
</complexType>
```

Figure 19: PacketFilterCondition

The **ApplicationLayerCondition** showed in *Figure 20* exposes several fields related with the application layer, this is, fields related with the level four and above in the OSI stack, as they can be, the URL, the specific HTTP method, the file extension as well as the mime type.

```
<complexType name="ApplicationLayerCondition">
  <sequence>
    <element maxOccurs="1" minOccurs="0" name="URL" type="string"/>
    <element maxOccurs="1" minOccurs="0" name="httpMethod" type="string"/>
    <element maxOccurs="1" minOccurs="0" name="fileExtension" type="string"/>
    <element maxOccurs="1" minOccurs="0" name="mimeType" type="string">
    <element maxOccurs="1" minOccurs="0" name="phrase" type="string"/>
  </sequence>
 </complexType>
```

Figure 20: Application Layer Condition

Regarding the **FilteringAction**, *Figure 21* uses a **FilteringActionType** which is a restricted string in order allow or deny the traffic specified in the filtering condition.

```
<complexType name="FilteringAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element name="FilteringActionType"
          type="ITResource:FilteringActionType"/>
        </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 21: FilteringAction

By using specific filtering configuration conditions fields, the user is able to model filtering policies according to different purposes and protocols. Due to its flexibility to represent the network information, the filtering condition has been reused and extended for other security policies.

### 5.3.2.2 Traffic Divert

MSPL traffic divert policy models are intended to represent different traffic divert operations like traffic forwarding or traffic mirroring. *Figure 22* shows the **TrafficDivertConfigurationCondition** definition. Since the required fields in order to perform a matching against network datagrams is the same that in filtering case, the complex type is just inheriting the `FilteringConfigurationCondition`.

```
<complexType name="TrafficDivertConfigurationCondition">
  <complexContent>
    <extension base="ITResource:FilteringConfigurationCondition" />
  </complexContent>
</complexType>
```

Figure 22: TrafficDivertConfigurationCondition

Regarding the **TrafficDivertAction**, *Figure 23* shows that it is composed by a **TrafficDivertActionType** and a **packetDivertAction**. The first one defines the type of traffic divert operation while the second one allows specifying the new parameters for the network datagrams.

```
<complexType name="TrafficDivertAction">
 <complexContent>
```

```
  <extension base="ITResource:ConfigurationAction">
   <sequence>
    <element name="TrafficDivertActionType"
      type="ITResource:TrafficDivertActionType" />
    <element maxOccurs="1" minOccurs="1" name="packetDivertAction"
      type="ITResource:TrafficDivertConfigurationCondition" />
   </sequence>
  </extension>
 </complexContent>
</complexType>
<simpleType name="TrafficDivertActionType">
  <restriction base="string">
    <enumeration value="FORWARD"></enumeration>
    <enumeration value="MIRRORING"></enumeration>
  </restriction>
</simpleType>
```

Figure 23: TrafficDivertAction

It is important to highlight that the `TrafficDivertAction` is easily extensible to taking into account new action type specific parameters.

### 5.3.2.3 Access control Policy

MSPL access control policy models are intended to provide access control configuration at medium-level. With this purpose they have been modelled two different kind of policies, these are, authentication policies and authorization policies.

#### 5.3.2.3.1 Authentication

The authentication policy model allows specifying the required parameters in order to configure an authentication method for a subject. In order to represent it, they have been defined the specific *Authentication* action and condition.

```
<complexType name="AuthenticationCondition">
  <complexContent>
    <extension base="ITResource:FilteringConfigurationCondition">
      <sequence>
        <element name="AuthenticationSubject" type="string" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 24: Authentication Condition

*Figure 24* shows the new condition defined for authentication, which extends the `FilteringConfigurationCondition` in order to provide network information about the subject, as well as the proper `AuthenticationSubject` that is able to represent some kind of extra identification about the subject. The **`AuthenticationCondition`** then represents the condition must be accomplished by the subject in order to apply the authentication action.

```
<complexType name="AuthenticationAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element name="AuthenticationOption"
          type="ITResource:AuthenticationOption" maxOccurs="unbounded" />
       </sequence>
    </extension>
  </complexContent>
</complexType>
```

```
<complexType name="AuthenticationOption">
  <sequence>
    <element name="AuthenticationTarget"
      type="ITResource:PacketFilterCondition" />
    <element name="AuthenticationMethod" type="string" />
    <element name="AuthenticationMechanism" type="string" />
    <element name="AuthenticationParameters"
      type="ITResource:AuthenticationParameters"/>
  </sequence>
</complexType>
```

Figure 25: Authentication Action

*Figure 25* shows the new **AuthenticationAction** which is composed by a sequence of **AuthenticationOption**. This list of authentication options defines an AuthenticationTarget in order to specify the target of the authentication (it reuses the PacketFilterCondition), an AuthenticationMethod, and AuthenticationMechanism in order to specify the type of authentication must be used as well as AuthenticationParameters (reusing the AuthenticationParameters field for the condition), which is able to represent specific authentication parameters like certificates, ids or key locations.

#### 5.3.2.3.2 Authorization policy

The authorization policy model allows specifying the required parameters in order to authorize or unauthorize to a subject to perform a specific action over a specific resource. In the same way as the previous case, they have been defined a new action and condition as well as the action types.

```
<complexType name="AuthorizationCondition">
 <complexContent>
  <extension base="ITResource:FilteringConfigurationCondition">
   <sequence>
    <element name="AuthorizationSubject" type="string" minOccurs="0" />
    <element name="AuthorizationTarget" type="string" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

Figure 26: AuthorizationCondition

*Figure 26* shows the new condition for the authorization capability, which extends the FilteringConfigurationCondition by adding AuthorizationSubject and AuthorizationTarget in order to provide more information for the subject and target like subject or target identifiers. By the filtering configuration condition extension, it is able to indicate all the information regarding the endpoint who requires authorization.

```
<complexType name="AuthorizationAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element name="AuthorizationActionType"
          type="ITResource:AuthorizationActionType"/>
      </sequence>
    </extension>
</complexContent>
</complexType>
  <simpleType name="AuthorizationActionType">
    <restriction base="string">
        <enumeration value="ALLOW"></enumeration>
        <enumeration value="DENY"></enumeration>
    </restriction>
```

```
</simpleType>
```

Figure 27: Authorization Action

*Figure 27* shows the new ***AuthorizationAction*** which provides an ***AuthorizationActionType*** where it is possible to specify a restricted set of actions. Currently, the model is able to specify if the subject will be able or not to access to the specified resource.

### 5.3.2.4 Channel Protection Policy

MSPL channel protection policy models are intended to specify channel protection configurations in independent way of the underlaying implementation. To model channel protection policies, Figure 28 shows the **`DataProtectionCondition`**, which uses the packet filter condition to define the source and destination involved in the channel protection.

```
<complexType name="DataProtectionCondition">
  <complexContent>
    <extension base="ITResource:ConfigurationCondition">
        <sequence>
         <element maxOccurs="1" minOccurs="0" name="packetFilterCondition"
                type="ITResource:PacketFilterCondition" />
        </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 28: DataProtectionCondition

*Figure 29* shows the main components for the **`DataProtectionAction`**. These are, the `technology` involved, described as string, the `technologyActionParameters` and a sequence of `technologyActionSecurityProperty`.

```
<complexType name="DataProtectionAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element name="technology" type="string"/>
        <element maxOccurs="1" minOccurs="1" name="technologyActionParameters"
          type="ITResource:ActionParameters"/>
        <element maxOccurs="unbounded" minOccurs="0"
          name="technologyActionSecurityProperty"
          type="ITResource:TechnologyActionSecurityProperty"/>
      </sequence>
   </extension>
  </complexContent>
</complexType>
```

Figure 29: Data Protection Action

The `technologyActionParameters` field extends **`ActionParameters`** which is defined in *Figure 30*. The `keyExchange` parameter allows specifying values like the key exchange action, the hash algorithm to use, the symmetric encryption algorithm as well as the authentication type. The `authenticationParameters` allow the user to model important information for the authentication process as the Pre-Shared Key value or its path, the certificate for the Certification Authority, the public key information, this is, all necessary data to carry on an authentication process using certificates or pre-shared keys.

```
<complexType name="ActionParameters">
  <sequence>
    <element maxOccurs="1" minOccurs="0" name="keyExchange"
        type="ITResource:KeyExchangeParameter"/>
    <element maxOccurs="unbounded" minOccurs="1" name="technologyParameter"
        type="ITResource:TechnologySpecificParameters"/>
```

```
    <element maxOccurs="1" minOccurs="0" name="authenticationParameters"
        type="ITResource:AuthenticationParameters"/>
  </sequence>
</complexType>
```

Figure 30: Action Parameters

The `technologyParameter` which is the type `TechnologySpecificParameters` is extended depending on specific technologies in order to provide fields dependant on technology, e.g. `IPsecTechnologyParameter`, `DTLSTechnologyParameter` and so on. On the other hand, the `technologyActionSecurityProperty` is extended depending on the expecting action, e.g. `Integrity`, `Authentication` and `Confidentiality`, providing different fields for each of them.

### 5.3.2.5 Privacy Policy

MSPL privacy policy models are intended to specify privacy configurations, including multiple privacy methods in independent way of the underlaying implementation. *Figure 31* shows the specific configuration condition for the privacy policy which is able to specify the `Subject` and `Target` of the policy, as well as the network configuration for the matching.

```
<complexType name="PrivacyConfigurationCondition">
  <complexContent>
    <extension base="ITResource:FilteringConfigurationCondition">
      <sequence>
        <element name="Subject" type="string" minOccurs="0" />
        <element name="Target" type="string" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 31: PrivacyConfigurationCondition

Regarding the privacy action, *Figure 32* shows the action for the privacy policy, which is composed by a **`PrivacyActionType`** and a **`PrivacyMethod`**. The privacy action type allows specify the kind of privacy (e.g. data privacy).

```
<complexType name="PrivacyAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element name="PrivacyActionType"
          type="ITResource:PrivacyActionType" minOccurs="1" />
         <element name="PrivacyMethod" type="ITResource:PrivacyMethod"
           minOccurs="1" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 32: Privacy Action

On the other hand, the **`PrivacyMethod`** is extended in order to model different kind of privacy methods. Actually, it has been extended in order to provide **`PrivacyIBMethod`**, **`PrivacyABMethod`** and **`PrivacyPKIMethod`** as can be shown in *Figure 33*.

```
<complexType name="PrivacyIBMethod">
  <complexContent>
    <extension base="ITResource:PrivacyMethod">
      <sequence>
        <element name="IB" type="string" />
      </sequence>
```

```
    </extension>
  </complexContent>
</complexType>
<complexType name="PrivacyABMethod">
  <complexContent>
    <extension base="ITResource:PrivacyMethod">
      <sequence>
        <element name="attribute" type="ITResource:KeyValue"
          maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="PrivacyPKIMethod">
  <complexContent>
    <extension base="ITResource:PrivacyMethod">
      <sequence>
        <element name="pkiParameters"
          type="ITResource:AuthenticationParameters"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 33: Privacy methods

**PrivacyIBMethod** is able to model the identity required in an identity-based privacy method. **PrivacyABMethod** allows model a list of key-value attribute pairs for the attribute-based privacy method. Finally, the **PrivacyPKIMethod** specifies a set of `pkiParameters` by extending the `AuthenticationParameters` field.

### 5.3.2.6 Monitoring Policy

MSPL monitoring policy models are intended to specify monitoring configurations allowing to establish the monitoring parameters in independent way of the underlaying technology. This is, providing generic monitoring fields which can be implemented by different monitoring enablers.

```
<complexType name="MonitoringConfigurationConditions">
  <complexContent>
    <extension base="ITResource:ConfigurationCondition">
      <sequence>
        <element maxOccurs="1" minOccurs="0" name="ProbeID"
          type="string"></element>
        <element maxOccurs="unbounded" minOccurs="1"
          name="monitoringConfigurationCondition"
          type="ITResource:MonitoringConfigurationCondition">
        </element>
       </sequence>
     </extension>
  </complexContent>
</complexType>

<complexType name="MonitoringConfigurationCondition">
  <complexContent>
    <extension base="ITResource:FilteringConfigurationCondition">
      <sequence>
        <element maxOccurs="1" minOccurs="0"
           name="detectionFilter" type="string"></element>
        <element maxOccurs="1" minOccurs="0"
          name="signatureList" type="ITResource:SignatureList"></element>
      </sequence>
```

```
    </extension>
  </complexContent>
</complexType>
```

Figure 34: MonitoringCondition

*Figure 34* shows the model for the monitoring condition which in this case is a **MonitoringConfigurationConditions** list in order to specify several monitoring conditions for an action, where a **MonitoringConfigurationCondition** extends the `FilteringConfigurationCondition` in order to be able specifying networking parameters, as well as adding generic monitoring-related parameters like a `detectionFilter` and a `signatureList`.

```
<complexType name="MonitoringAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
          <element maxOccurs="unbounded"
            name="monitoringActionType" type="ITResource:MonitoringActionType" />
          <element name="reportPerFlow" type="boolean" minOccurs="0" />
          <element name="reportPeriodicity" type="integer" minOccurs="0"/>
         <element maxOccurs="1" minOccurs="0" name="count"
           type="integer"></element>
          <element maxOccurs="1" minOccurs="0" name="ruleID"
            type="string"></element>
          <element name="aditionalRuleParameters"
          type="ITResource:KeyValue" minOccurs="0"
          maxOccurs="unbounded"></element>
       </sequence>
    </extension>
  </complexContent>
</complexType>
<simpleType name="MonitoringActionType">
  <restriction base="string">
    <enumeration value="ALERT"></enumeration>
    <enumeration value="ENABLE_SESS_STATS"></enumeration>
    <enumeration value="ENABLE_NO_SESS_STATS"></enumeration>
  </restriction>
</simpleType>
```

Figure 35: MonitoringAction

Regarding the monitoring action, *Figure 35* shows the schema. As it can be seen, the **MonitoringAction** is composed by a field which specifies the action type, this is, **MonitoringActionType**, indicating the nature of the action, depending on the condition. The rest of the fields indicate the configuration of the action, this is, it is possible to specify if it is required sending reports, its periodicity, if it is required count the packages, the rule id and even a list of additional monitoring parameters.

### 5.3.2.7 Network Anonymity

MSPL network anonymity policy models are intended to specify network anonymity configurations allowing to establish different preferences or properties depending on the kind of anonymization desired.

```
<complexType name="AnonymityConfigurationCondition">
  <complexContent>
    <extension base="ITResource:FilteringConfigurationCondition">
    </extension>
  </complexContent>
</complexType>
```

Figure 36: AnonymityConfigurationCondition

The **AnonymityConfigurationCondition** inherits the **FilteringConfigurationCondition** in order to provide network information regarding the subject as can be seen in *Figure 36*. *Figure 37* shows the **AnonymityAction**. It is composed by an **AnonymityActionType**, the anonymityTarget, **anonymityTechnologyParameters** and aditionalAnonymityParameters. The first one represents the kind of anonymity to be configured (sender, receiver or whole communication). The second one allows to include the target network related information. The third one is able to represent specific technology parameters, still independent to the implementation. Finally, the las one allows including additional parameters as key-value pairs.

```
<complexType name="AnonymityAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element maxOccurs="unbounded" name="anonymityActionType"
          type="ITResource:AnonymityActionType" />
        <element maxOccurs="1" minOccurs="0" name="anonymityTarget"
          type="ITResource:AnonymityConfigurationCondition" />
        <element maxOccurs="1" minOccurs="0"
          name="anonymityTechnologyParameters"
          type="ITResource:AnonymityTechnologyParameter" />
       <element name="aditionalAnonymityParameters" type="ITResource:KeyValue"
          minOccurs="0" maxOccurs="unbounded"></element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<simpleType name="AnonymityActionType">
  <restriction base="string">
    <enumeration value="SENDER"></enumeration>
    <enumeration value="RECEIVER"></enumeration>
    <enumeration value="COMMUNICATION"></enumeration>
  </restriction>
</simpleType>
```

**Figure 37: AnonymityAction**

The **anonymityTechnologyParameters** are extended being able to model different kind of anonymity parameters depending on the anonymization technology like Onion routing or traffic mixing.

```
<complexType name="OnionRoutingTechnologyParameter">
  <complexContent>
  <extension base="ITResource:AnonymityTechnologyParameter">
    <sequence>
      <element name="exitRelay" type="boolean" />
      <element name="IPv6Exit" type="boolean" />
      <element name="publicRelay" type="boolean" />
      <element name="hiddenServiceDir" type="string" />
      <element name="hiddenServicePort" type="string" />
      <element name="relayBandwidthRate" type="integer" />
      <element name="relayBandwidthBurst" type="integer" />
    </sequence>
  </extension>
  </complexContent>
</complexType>
```

**Figure 38: OnionRoutingTechnologyParameter**

*Figure 38* shows a modelling example to represent Onion routing parameters. **OnionRoutingTechnologyParameter** allows representing Onion routing related fields like the kind of node in the Onion network, or even data rates.

### 5.3.2.8 QoS

MSPL Quality of Service policy models are intended to specify QoS parameters in order to provide Quality of Service configurations to the underlaying infrastructure.

```
<complexType name="QoSCondition">
  <sequence>
    <element name="profile" type="string" minOccurs="0" maxOccurs="1"/>
    <element name="throughput" type="integer" minOccurs="0" maxOccurs="1"/>
    <element name="transitDelay" type="integer" minOccurs="0" maxOccurs="1"/>
    <element name="priority" type="integer" minOccurs="0" maxOccurs="1"/>
    <element name="errorRate" type="integer" minOccurs="0" maxOccurs="1"/>
    <element name="resilence" type="integer" minOccurs="0" maxOccurs="1"/>
  </sequence>
</complexType>
```

Figure 39: QoSCondition

*Figure 39* shows the fields which compounds a **QoSCondition**. In this way, the security administrator is able to create and identify different QoS profiles by specifying QoS related fields like the `throughput`, `transitDelay`, `errorRate` and so on.

```
<complexType name="QoSAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element maxOccurs="1" minOccurs="0"
          name="qosAction" type="ITResource:QoSCondition" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 40: QoSAction

Regarding the **QoSAction**, as it can be seen in *Figure 40*, basically it is modelled by using the `QoSCondition` previously modelled in order to be able to introduce the QoS related fields.

### 5.3.2.9 Data Aggregation

MSPL data aggregation policy models are intended to specify data aggregation configurations in order to configure security enablers capable to aggregate data based on specific patterns. Unlike other policies, in this case there is not a simple condition but a set of **DataAgreggationConfigurationConditions**. *Figure 41* shows that the filed is basically a list of **DataAggregationConfigurationCondition** which actually, in order to represent network fields, inherits from the `FilteringConfigurationCondition`.

```
<complexType name="DataAgreggationConfigurationConditions">
  <complexContent>
    <extension base="ITResource:ConfigurationCondition">
      <sequence>
        <element maxOccurs="unbounded" minOccurs="1"
          name="dataAggregationConfigurationCondition"
          type="ITResource:DataAggregationConfigurationCondition"></element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="DataAggregationConfigurationCondition">
  <complexContent>
    <extension base="ITResource:FilteringConfigurationCondition">
```

```
    </extension>
  </complexContent>
</complexType>
```

Figure 41: DataAggregationConfigurationCondition

Regarding the **DataAggregationAction**, *Figure 42* show that the model is able to specify the action to be addressed as well as a set of `aditionalRuleParameters` in order to provide additional data aggregation parameters required by the organization. Note that it is easily extensible by adding new values to the **DataAggregationActionType**,

```
<complexType name="DataAggregationAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element maxOccurs="unbounded" name="dataAggregationActionType"
          type="ITResource:DataAggregationActionType" />
        <element name="aditionalRuleParameters" type="ITResource:KeyValue"
          minOccurs="0" maxOccurs="unbounded"></element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<simpleType name="DataAggregationActionType">
  <restriction base="string">
    <enumeration value="SIMPLE_AGGREGATION"></enumeration>
  </restriction>
</simpleType>
```

Figure 42: DataAggregationAction

### 5.3.2.10 Operational policies

Complementing the other security areas policies, the model provides a way to specify operational policies, these are those capable to specify operational actions like concrete deployments or configurations.

```
<complexType name="EnableAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element name="EnableActionType"
          type="ITResource:EnableActionType" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="EnableActionType">
        <attribute name="enable" type="boolean"></attribute>
        <attribute name="objectToEnable" type="string"></attribute>
</complexType>
```

Figure 43: Operational Policy

*Figure 43* shows the ***EnableAction*** definition which extends the *ConfigurationAction* base and it allows indicating an ***EnableActionType*** where it is possible to specify if the action is enabling or disabling the policy through a Boolean attribute, as well as the resource where the action will be performed. Extending this idea, they have been modelled IoT specific security operational policies like IoT control policies and Virtual IoT Honey Net policies.

#### 5.3.2.10.1 IoT control

To be able to model specific actions over the IoT domain devices, the policy model has been extended with a new capability and different IoT actions. This kind of actions can provide different options in order to interact with the IoT devices, e.g. the power management, the IoT security management, or even firmware operations. *Figure 44* shows an example of configuration rule action extension, in particular, the `PowerMgmtAction`, which provides different action types like the ability of turn off or reset the IoT device.

```
<complexType name="PowerMgmtAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
      <sequence>
        <element name="PowerMgmtActionType"
type="ITResource:PowerMgmtActionType" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<simpleType name="PowerMgmtActionType">
        <restriction base="string">
                <enumeration value="OFF"></enumeration>
                <enumeration value="RESET"></enumeration>
        </restriction>
</simpleType>
```

**Figure 44: IoT control capability**

#### 5.3.2.10.2 Virtual IoT Honey Net

The virtual IoT Honey Net allows specifying a policy which will be able to model an IoT Virtual Honey Net deployment by replicating a real physical one. This model relies on the "Technology Independent Honeynet Description language" [3], that has been adapted and extended to cope with IoT scenarios, giving place to the IoT-honeynet model. *Figure 45* shows an example of the model which represents the policy. In this case, the action has been extended as **`VIoTHoneyNetAction`**. That action intends to provide the specific action type **`VIoTHoneyNetActionType`** that will be performed (e.g. *DEPLOY*, *RESET* or *REMOVE*) as well as the **`ioTHoneyNet`** model in order to replicate the IoT physical environment.

```
<complexType name="VIoTHoneyNetAction">
  <complexContent>
    <extension base="ITResource:ConfigurationAction">
    <sequence>
      <element name="VIoTHoneyNetActionType"
        type="ITResource:VIoTHoneyNetActionType" />
      <element name="ioTHoneyNet" sm:type="ioTHoneyNet" />
    </sequence>
    </extension>
  </complexContent>
</complexType>
<simpleType name="VIoTHoneyNetActionType">
        <restriction base="string">
                <enumeration value="DEPLOY"></enumeration>
                <enumeration value="RESET"></enumeration>
                <enumeration value="REMOVE"></enumeration>
        </restriction>
</simpleType>
```

**Figure 45: VIoT Honey Net**

Regarding the iot-honeynet model, *Figure 46* shows the main composition of the complex type, which is compounded by a description of the honeynet and multiple **`net`**, **`router`** and **`gateway`** objects, and finally a sequence of **`ioTHoneyPot`**.

```
<xs:complexType name="ioTHoneyNetType">
    <xs:sequence>
      <xs:element type="xs:string" name="name"/>
      <xs:element name="net" type="tns:netType" maxOccurs="unbounded"
minOccurs="0"/>
      <xs:element name="router" type="tns:ioTRouterType" maxOccurs="unbounded"
minOccurs="0"/>
      <xs:element name="containmentGateway" type="tns:containmentGatewayType"
maxOccurs="unbounded" minOccurs="0"/>
      <xs:element name="ioTHoneyPot" type="tns:ioTHoneyPotType"
maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

Figure 46: IoTHoneyNet Type

*Figure 47* shows the definition for each element of the `ioTHoneyNet`, excluding the `ioTHoneyPotType` which is illustrated later. The **`netType`** is modelled using the network name/description/address and an identifier. For the **`routerType`**, we can specify the name, a sequence of interfaces and routes, even the operating system and an identifier. The **`ifType`** in turn are composed by the name, mac address, IP address, an identifier, and the network to which it belongs. On the other hand, the **`routeType`** allows defining the next hop in order to reach a destination. Finally, we could define a gateway in similar way of the router specification.

```
<xs:complexType name="netType">
  <xs:sequence>
    <xs:element type="xs:string" name="name"/>
  </xs:sequence>
  <xs:attribute type="xs:byte" name="id" use="optional"/>
</xs:complexType>

<xs:complexType name="routerType">
  <xs:sequence>
    <xs:element type="xs:string" name="name"/>
    <xs:element name="if" type="tns:ifType" maxOccurs="unbounded"
      minOccurs="1"/>
    <xs:element name="route" type="tns:routeType" maxOccurs="unbounded"
      minOccurs="0"/>
    <xs:element name="operatingSystem" type="tns:operatingSystemType"/>
  </xs:sequence>
  <xs:attribute type="xs:byte" name="id"/>
</xs:complexType>

<xs:complexType name="ifType">
  <xs:sequence>
    <xs:element type="xs:string" name="name"/>
    <xs:element type="xs:string" name="mac_addr"/>
    <xs:element type="xs:string" name="ip"/>
  </xs:sequence>
    <xs:attribute type="xs:byte" name="id"/>
    <xs:attribute type="xs:string" name="net"/>
  </xs:complexType>

  <xs:complexType name="routeType">
    <xs:sequence>
      <xs:element type="xs:string" name="dst"/>
      <xs:element type="xs:string" name="gw"/>
    </xs:sequence>
    <xs:attribute type="xs:byte" name="id"/>
  </xs:complexType>
  <xs:complexType name="containmentGatewayType">
```

```
    <xs:sequence>
      <xs:element type="xs:string" name="name"/>
      <xs:element name="if" type="tns:ifType"/>
      <xs:element name="operatingSystem" type="tns:operatingSystemType"/>
    </xs:sequence>
  </xs:complexType>
```

Figure 47: ioT HoneyNet Type elements

*Figure 48* shows the **ioTHoneyPotType** which extends the honeyPotType. It allows to specify similar values as in the router case, including the operating system, interface and software specification of the honey pot (e.g. Contiki with CoAP agent). Also, they have been provided relevant specific fields such as the model of the IoT device, its physical location, and the sequence of available resources.

```
<xs:complexType name="honeyPotType">
  <xs:sequence>
    <xs:element type="xs:string" name="name"/>
    <xs:element type="tns:interactionLevel" name="interaction_level"/>
    <xs:element name="if" type="tns:ifType"/>
    <xs:element name="operatingSystem" type="tns:operatingSystemType"/>
    <xs:element name="software" type="tns:softwareType"/>
  </xs:sequence>
  <xs:attribute type="xs:byte" name="id"/>
</xs:complexType>

<xs:complexType name="ioTHoneyPotType">
  <xs:complexContent>
  <xs:extension base="tns:honeyPotType">
    <xs:sequence>
      <xs:element name="model" type="xs:string" minOccurs="0"/>
      <xs:element name="location" type="tns:physicalLocation" minOccurs="0"/>
      <xs:element name="resource" type="tns:ioTResourceType" minOccurs="0"
                maxOccurs="unbounded"/>
     </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 48: HoneyPot type

The model of the IoT device corresponds with the name of the specific hardware model (e.g. “WISMOTE”). The physical location allows specifying the location in 3D coordinates (x,y,z) and finally, the resources can be selected from a specific list of the most common IoT resources (e.g. TEMPERATURE, HUMIDITY...).

### 5.3.2.11 Policy for Orchestration

Policies for orchestration are intended to provide a set of security policies which must be enforced according on the orchestration logic. This is, following a specific order according on the priority, dependences or a combination of these.

```
<complexType name="ITResourcesType">
  <sequence>
    <element maxOccurs="unbounded" minOccurs="1" name="ITResource"
      type="ITResource:ITResourceType"></element>
  </sequence>
</complexType>

<complexType name="ITResourceType">
  <sequence>
    <element maxOccurs="1" minOccurs="1" name="configuration"
      type="ITResource:Configuration"/>
    <element maxOccurs="1" minOccurs="0" name="priority"
```

```
      type="integer"/>
    <element maxOccurs="1" minOccurs="0"
      name="dependences" type="ITResource:Dependences"/>
    <element maxOccurs="1" minOccurs="0"
      name="enablerCandidates" type="ITResource:EnablerCandidates"/>
  </sequence>
  <attribute name="ID" type="string"/>
</complexType>
```

Figure 49: ITResourcesType

*Figure 49* shows the **`ITResourcesType`** which represents a set of security policies which depends on other security policies or events. In this way, the **`ITResourcesType`** are composed by a list of **`ITResource`**, and the latter has been extended with new fields in order to establish a `priority`, `dependence` **and** `enablerCandidates`. The priority is used to decide the importance of the policy for the policy enforcement, while the dependence field is able to represent several dependences which refer to another security policy or some event (e.g. authentication event). Finally, the enabler candidates represent what could be candidate security enablers abler to enforce the security policy.

## 5.4 POLICY EDITOR TOOL

In order to ease the policy modelling, it is provided a Policy Editor tool. By using the Policy Editor Tool, it is possible modelling High-level security policies through a friendly GUI, as well as request the policy translation or even the policy enforcement.

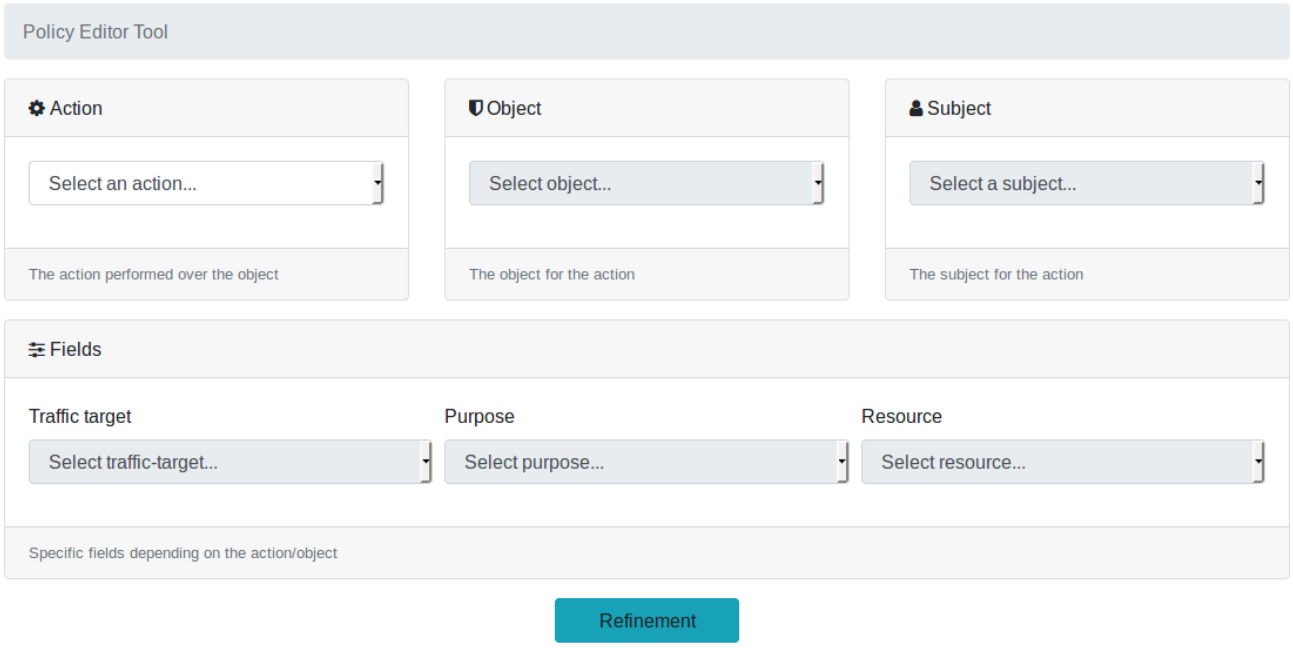


Figure 50: Policy Editor Tool – Refinement

*Figure 50* shows the GUI of the Policy Editor Tool for the policy enforcement. Currently the editor provides two main sections. The first one allows the security administrator to indicate the action, object and subject of the security policy. The second one includes the target, purpose and resource, so, by using this GUI the security administrator is able to specify the required fields to modelling an HSPL policies, taking into account that each selector is filled depending on the rest, according on Table 1 and Table 2. For instance, the available objects are different if the action is related with authorization or with an operational policy. Once the system administrator has defined the High-level security policy, he/she can request the policy refinement by pressing the refinement button.

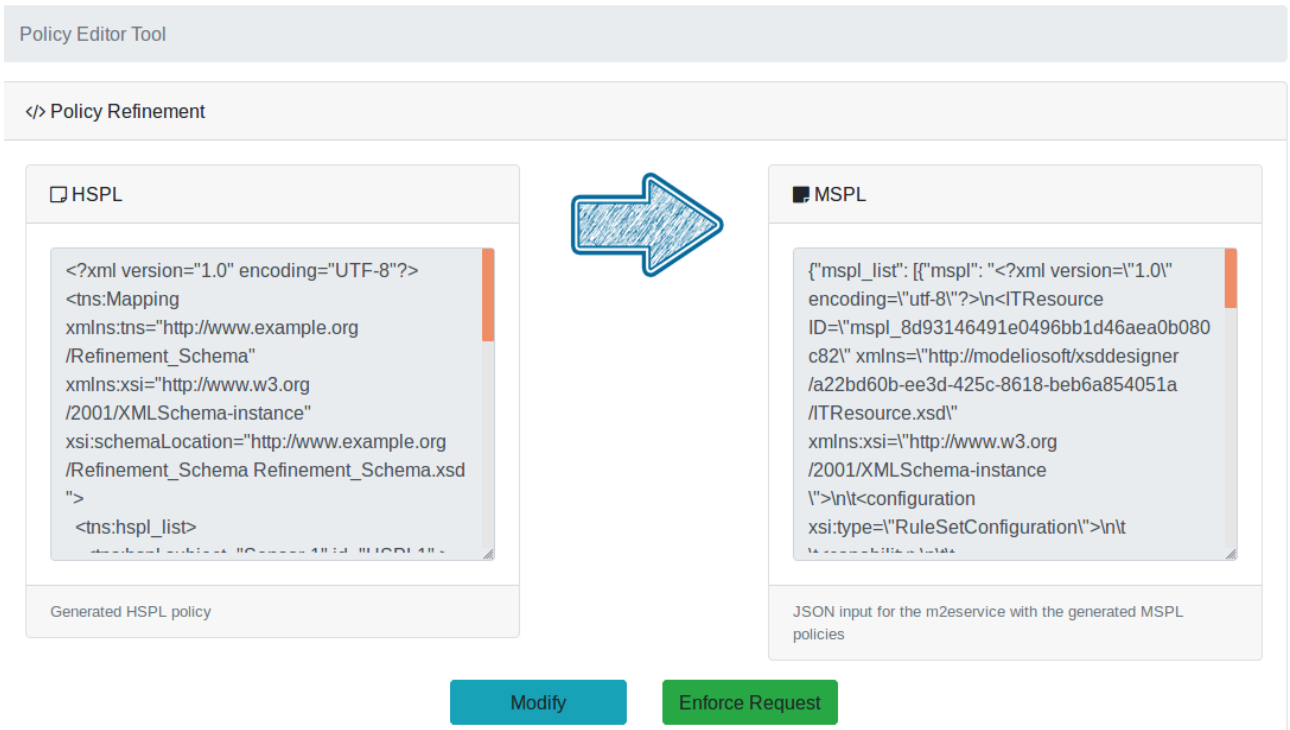


**Figure 51: Policy Editor Tool - Enforce request**

*Figure 51* shows an example of result for the policy refinement. In this case the policy refinement screen provides both, the High-level security policy generated from the specified fields, as well as the Medium-level security policy (or policies) refined from the first one. At this point the security administrator verifies the result and decides if it is necessary to perform some kind of modification over the policy or it can be requested for the enforcement. If the security administrator decides to modify the policy, he/she can click on the *modify* button, getting back to the HSPL definition. On the other hand, if the *enforce request* button is pressed, a Medium-level policy enforcement is requested.

## 5.5 POLICY CONFLICTS DETECTION

The American Heritage Dictionary of the English Language defines a conflict as “A part of discord caused by the actual or perceived opposition of needs, values and interests...”. Applying that definition in the scope of this document, a policy conflict occurs when exists some kind of incompatibility or contradiction in the policy definitions taking into account the capabilities, conditions and actions. In order to detect these conflicts, it is applied a rule-based approach through up the system workflow since this kind of approach allows modelling the behaviour of the system by declaring rules, providing important properties like flexibility and scalability.

*Figure 52* shows a base diagram for the rule engine. A security policy (or a set of them) is sent to the rule engine in order to detect some kind of conflict. Then, the rule engine verifies the security policy/es against the defined rules. To this aim it is possible it requires performing requests to different data sources in order to know the current status of the affected policies, as well as adapting the policy model to the rule engine if it is necessary.

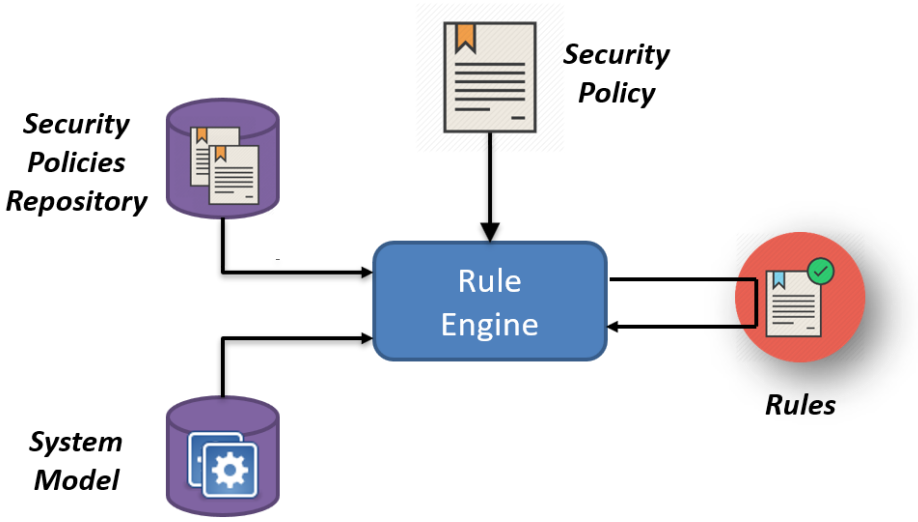


Figure 52: Rule Engine

Regarding the rules, *Figure 53* shows an example of rule declaration in an abstract language. As it can be seen, there are two well-differentiated sections separated by "->", these are, the antecedents and the consequent. The antecedents allow specify different conditions through Boolean expressions.

```
ConflictType: Antecedent1 ^ Antecedent2 | AntecedentN
->
Consequent
```

Figure 53: Rule syntax example

In practice, each `Antecedent` expression can reference different elements like objects, attributes, properties, classes or methods. In this way, if the antecedent's expression is satisfied, it will be executed the consequent. Usually, this consequent applies some kind of action over the system. The kind of antecedents and consequents is determined depending on the conflict we intend to detect. Since the kind of conflict can be different depending on the circumstances, in the scope of ANASTACIA project security policies they have been identified mainly the following ones; **redundancy**, **priority**, **duties**, and **dependence**.

### 5.5.1 Redundancy conflict

This kind of conflict can be identified when the same security policy is tried to be enforced multiple times, or when the security policy behaviour is already defined by a more restrictive one (e.g. we already have filtered all traffic, and we try filter CoAP traffic).

```
Policy: p
RedundancyConflict: ( p.exists() ^ p.enforced() ) |
    ( p.exists() ^ p.status("pending") ) |
    ( p.existByBehavior() )
->
notify(RedundancyConflict,p)
```

Figure 54: Redundancy conflict rule example

*Figure 54* shows an example of the rules which can be verified in order to detect redundancy conflicts. In the example we will detect a redundancy if the same policy already exists (the same object or ID) and it is already enforced, or if the policy exists and is pending to be enforced, or if the policy content already exists. This latter means, if there is another policy which is enforcing the same behaviour. It is important to highlight that, while the example is just notifying the conflict, other kind of consequents could be performed. E.g. in

the latter case we could be interested in register the redundant policy in a queue in order to be executed if the policy with the same behaviour is removed.

### 5.5.2 Conflict of priorities

This conflict is related with a policy for orchestration and it arises when a security policy is trying to be enforced before another one with a higher priority.

```
Policy: p
OrchestrationPolicySet: ops
PriorityConflict: op in ops ^ op.priority > p.priority ^ !op.enforced()
->
notify(PriorityConflict,p)
```

**Figure 55: Priority conflict rule example**

*Figure 55* shows an example where it is being verified if exists a security policy in the policy for orchestration with a higher priority, and it is still unenforced. In this case it is necessary to notify the priority conflict violation.

### 5.5.3 Conflict of duties

This kind of conflicts occurs when a security policy generates a new behaviour in the system which is not compatible with the requirements of another security policy. E.g. A channel protection security policy will generate a conflict with a deep packet inspection security policy.

```
Policy: p
Policies: ps
DutiesConflict: if p.fieldRelated(pd.fields)  ^ p.capability.colision(ps)
->
notify(DutiesConflict,p)
```

**Figure 56: Duties conflict rule example**

*Figure 56* shows a rule example for the conflict of duties. In this case, it is verified if the security policy affects in somehow to the other policies already deployed on the system. This is, it verifies if some field is related with an already enforced security policy (e.g. an IP address which is contained in a subnet which already enforces some kind of policy). If so, it is then verified if the capability collisions with the capabilities of the affected policies. Of course, this requires establishing a capability collision base (e.g. **`DTLS_protocol`** will collide with **`Traffic_inspection_L7`**).

### 5.5.4 Conflict of dependence

This kind of conflict arises when the security policy enforcement depends on other security policy or event and the latter is already not enforced/satisfied.

```
Policy: p
Policies: ps
Events: es
DependenceConflict: p2 in ps ^ p.depends(p2) ^ !p2.enforced() |
   e in es ^ p.depends(e) ^ !e.triggered()
->
notify(DependenceConflict,p)
```

Figure 57: Dependence conflict rule example

*Figure 57* specifies that if there is a security policy which depends on another one which is currently not enforced, or it depends on an event which has not been previously triggered, a dependence conflict will be notified.

### 5.5.5 Conflict of managers

The conflict of managers occurs when a security policy which contradicts a previous one is trying to be enforced.

```
Policy: p
Policies: ps
DutiesConflict: ps.conditionsExists(p.condition) ^ p.action.colision(ps.action) |
      ps.conditionContains(p.condition) ^ p.action.colision(ps.action)
->
notify(ManagersConflict,p)
```

Figure 58: Managers conflict rule example

*Figure 58* shows an example where it is being verifying if the condition already exists as part of another security policy and the actions collide among them, as well as if the condition is not the same but is contained in somehow in other security policy (i.e. An IP address inside a subnet address). In this case it is also required a well-known base of action collisions.

# 6 USE CASE: SECURITY POLICY ENFORCEMENT IN IOT BUILDING SCENARIOS

This section shows specific instantiations of security policies in order to provide authentication and authorization security properties to the IoT infrastructure in a building management scenario where ANASTACIA has been deployed.

## 6.1 AUTHENTICATION

In order to configure the authentication process, the security administrator is able to model an authentication High-level Security Orchestration Policy through the Policy Editor Tool. *Annex 1* shows an authentication orchestration HSPL policy. This policy includes several HSPL policies in order to configure the authentication mechanism for the IoT devices. This is, the IoT-devices will be configured in order to perform the authentication using `PANA` protocol against a `PANA_AGENT`. Actually, it is also specifying the preferred method for the authentication agent. Depending on the policy interpreter implementation, some HSPL policy values can indicate that the HSPL can be refined in one or multiple MSPL policies. For instance, a bidirectional HSPL policy can be refined in two MSPL policies, as well as a subject with identifies multiple devices (e.g. IoT-Net) could be refined in multiple MSPL policies.

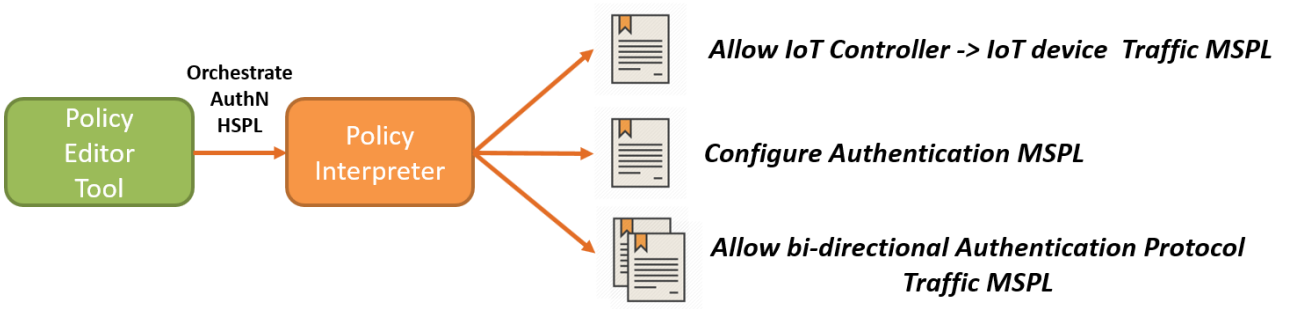


Figure 59: HSPL to MSPL orchestration example

*Figure 59* shows an example where the security administrator defines an authentication orchestration HSPL policy which contains multiple HSPL policies, and it is refined in a MSPL orchestration policy that contains multiple MSPL policies. In this case, the high-level orchestration authentication policy involves enough security policies in order to allow the full authentication process.

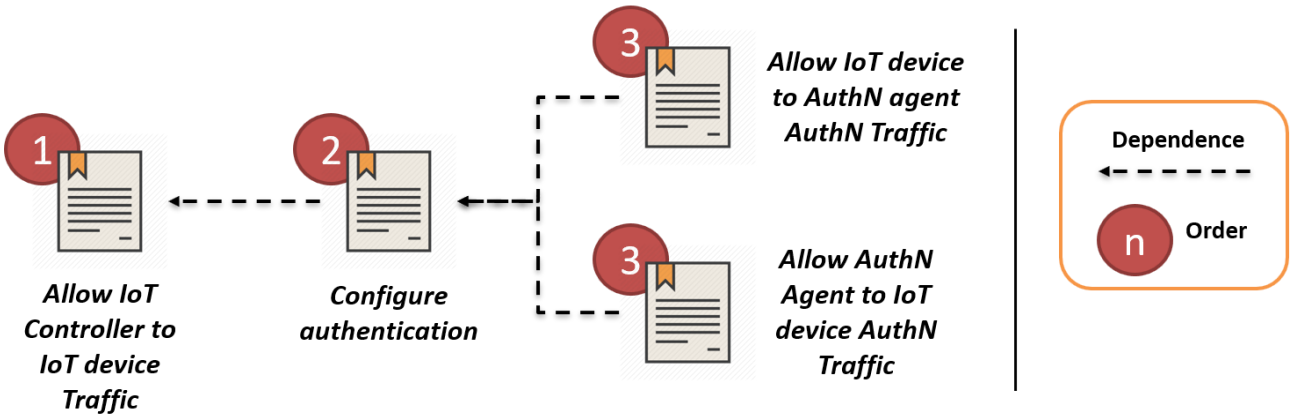


Figure 60: AuthN Orchestration graph

*Figure 60* shows an orchestration MSPL graph. This kind of graph determines the order and dependencies among the security policies. In this case, in order to configure the authentication in the IoT device, the IoT

controller must be able to send the command and control message to the IoT device, so the first MSPL policy is a networking authorization from the IoT Controller to the IoT device. Once the first policy has been enforced correctly, the authentication configuration can be enforced against the IoT-device through the IoT Controller. When the authentication is properly configured, the authentication traffic is allowed bidirectionally regardless of the order. *Annex 2* shows a full example of an authentication orchestration MSPL policy.

## 6.2 AUTHORIZATION

Before the IoT devices are able to access someplace in the architecture in order to put some kind of information, it must be performed an authorization process. *Annex 3* shows an instantiation of a High-level security policy where the security administrator requires that the IoT device (*subject*) is authorised (*action*) to put (*purpose*) temperature (*resource*) measures in the IoT Broker (*target*). In the same way of the previous case, the HSPL policies could be refined in one or multiple MSPL depending on the organization.

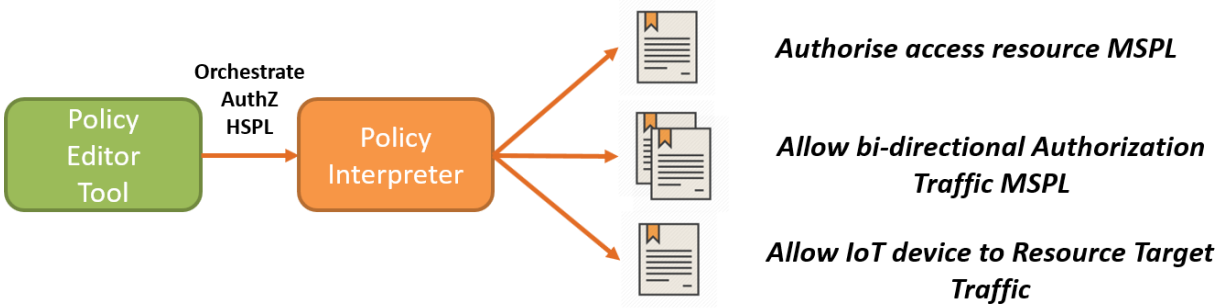


Figure 61: AuthZ HSPL to MSPL Orchestration example

*Figure 61* shows that in this case the HSPL authorization orchestration policy generates four MSPL security policies. The first one defines the resource authorization, the second and third ones are intended to allow the bidirectional communication for the authorization process (they can be generated from one HSPL). Finally, the fourth one allows the unidirectional traffic from the IoT device to the IoT broker.

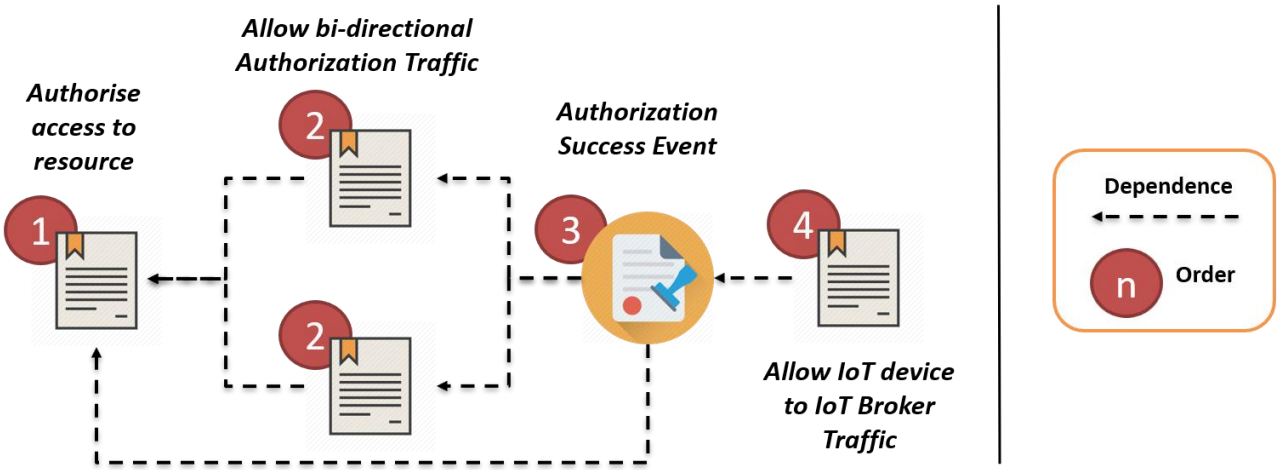


Figure 62: AuthZ Orchestration graph

*Figure 62* shows the orchestration graph generated by the high-level authorization orchestration policy. In this case, the first policy to be applied is the authorization resource itself, which is focused on grant enough permissions to the IoT device in order to write the specified resource in the IoT Broker. Once this policy has been enforced, the authorization traffic must be allowed in order the IoT device is able to perform the authorization process. When the authorization process has been finished successfully the system will receive an authorization success notification. Since the last policy depends on the authorization success event, when

the notification is received, the dependence is satisfied so the network authorization policy is enforced in order to allow the communication from the IoT device against the IoT broker. This is, the traffic from the IoT device to the IoT Broker is not allowed until the authorization has been performed successfully. *Annex 4* shows a full example of an authentication orchestration MSPL policy.

# 7 CONCLUSIONS

This document has detailed the main security policy models to define the policies envisaged in the ANASTACIA framework. To accomplish this aim, the present document exposes how the two-level policy approach and the capability concept have been included and evolved in order to apply them over the ANASTACIA framework. Thus, the models and processes have been extended in order to cope with the main identified security policies for the main scenarios being addressed in ANASTACIA project.

The document has also provided real policy examples for an IoT building management system, focused on the policy orchestration in order to provide a whole authentication and authorization process. In this regard, they have been instantiated different security policies in order to perform resource and network authorization and subject authentication, taking into account the policy dependences.

It should be noticed that this deliverable has provided the foundations for the policy modelling. Nonetheless, this design might be extended and evolved as the project evolves.

# 8 ANNEX

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Mapping xmlns:tns="http://www.example.org/Refinement_Schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/Refinement_Schema hspl.xsd ">
  <tns:hspl_list xsi:type="tns:HSPL_Orchestration"
ID="hsplo_a87cda98d7ca9d8787cdaa">
      <!-- Allow IoT Controller to IoT device Traffic -->
       <tns:hspl id="hspl_a87cda98d7ca9d8787cdca" subject="IoTController">
      <tns:action>authorise_access</tns:action>
      <tns:objectH>CoAP_traffic</tns:objectH>
      <tns:fields>
        <tns:traffic_target>
          <tns:target_name>Sensor1</tns:target_name>
        </tns:traffic_target>
      </tns:fields>
    </tns:hspl>
    <!-- Configure authN in Sensor1 against PANA AGENT -->
     <tns:hspl id="hspl_a87cda98d7ca9d8787cdcb" subject="Sensor1">
      <tns:action>config_authentication</tns:action>
      <tns:objectH>PANA</tns:objectH>
      <tns:fields>
        <tns:traffic_target>
          <tns:target_name>PANA_AGENT</tns:target_name>
        </tns:traffic_target>
      </tns:fields>
      <tns:dependences>
        <tns:dependence xsi:type="tns:PolicyDependence">
            <tns:dependenceCondition xsi:type="tns:PolicyDependenceCondition">
              <tns:policyID>hspl_a87cda98d7ca9d8787cdca</tns:policyID>
            <tns:status>ENFORCED</tns:status>
            </tns:dependenceCondition>
        </tns:dependence>
      </tns:dependences>
    </tns:hspl>
    <!-- Allow authN traffic -->
    <tns:hspl id="hspl_a87cda98d7ca9d8787cdcd" subject="Sensor1"
         bidirectional="true">
        <tns:action>authorise_access</tns:action>
        <tns:objectH>PANA_traffic</tns:objectH>
      <tns:fields>
        <tns:traffic_target>
                  <tns:target_name>PANA_AGENT</tns:target_name>
        </tns:traffic_target>
      </tns:fields>
      <tns:dependences>
        <tns:dependence xsi:type="tns:PolicyDependence">
            <tns:dependenceCondition xsi:type="tns:PolicyDependenceCondition">
              <tns:policyID>hspl_a87cda98d7ca9d8787cdcb</tns:policyID>
            <tns:status>ENFORCED</tns:status>
            </tns:dependenceCondition>
        </tns:dependence>
        <tns:dependence xsi:type="tns:PolicyDependence">
            <tns:dependenceCondition xsi:type="tns:PolicyDependenceCondition">
              <tns:policyID>hspl_a87cda98d7ca9d8787cdcc</tns:policyID>
            <tns:status>ENFORCED</tns:status>
            </tns:dependenceCondition>
        </tns:dependence>
```

```
      </tns:dependences>
    </tns:hspl>
  </tns:hspl_list>
</tns:Mapping>
```

Annex 1: AuthN Orchestration HSPL

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<ITResourceOrchestration ID="mspl_9f1a88b4fc67421b98de270d5a63d35f"
        xmlns="http://modeliosoft/xsddesigner/a22bd60b-ee3d-425c-8618-
beb6a854051a/ITResource.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://modeliosoft/xsddesigner/a22bd60b-ee3d-425c-
8618-beb6a854051a/ITResource.xsd ANASTACIA_MSPL_XML_Schema.xsd">
  <!-- Allow traffic from IoT Controller to Sensor -->
  <ITResource ID="mspl_9f1a88b4fc67421b98de270d5a63d35a">
    <configuration xsi:type='RuleSetConfiguration'>
        <capability>
            <Name>Traffic_Divert</Name>
        </capability>
        <configurationRule>
            <configurationRuleAction xsi:type='TrafficDivertAction' >
              <TrafficDivertActionType>FORWARD</TrafficDivertActionType>
              <packetDivertAction>
                <isCNF>false</isCNF>
                <packetFilterCondition>
                  <DestinationAddress>
                    2001:0720:1710:0004::5001/128
                  </DestinationAddress>
                </packetFilterCondition>
              </packetDivertAction>
            </configurationRuleAction>
            <configurationCondition
               xsi:type='TrafficDivertConfigurationCondition'>
              <isCNF>false</isCNF>
                <packetFilterCondition>
                  <SourceAddress>
                    2001:720:1710:4:5054:ff:feec:e209/128
                  </SourceAddress>
                    <DestinationAddress>
                      2001:0720:1710:0004:0000:0000:0000:5001/128
                    </DestinationAddress>
                </packetFilterCondition>
             </configurationCondition>
            <externalData xsi:type='Priority'>
                <value>60000</value>
            </externalData>
            <Name>Rule0</Name>
            <isCNF>false</isCNF>
        </configurationRule>
        <Name>Conf0</Name>
    </configuration>
  </ITResource>
  <!-- AuthN Configuration -->
  <ITResource ID="mspl_9f1a88b4fc67421b98de270d5a63d35b">
    <configuration xsi:type='RuleSetConfiguration'>
        <capability>
            <Name>Authentication</Name>
        </capability>
        <configurationRule>
```

```
            <configurationRuleAction xsi:type='AuthenticationAction' >
                <AuthenticationOption>
                    <AuthenticationTarget>
                      <DestinationAddress>
                        2001:720:1710:4:5054:caff:fefe:770f/128
                      </DestinationAddress>
                        <DestinationPort>5678</DestinationPort>
                        <ProtocolType>UDP</ProtocolType>
                    </AuthenticationTarget>
                    <AuthenticationMethod>PSK</AuthenticationMethod>
                    <AuthenticationMechanism>PANA</AuthenticationMechanism>
                    <AuthenticationParameters></AuthenticationParameters>
                </AuthenticationOption>
            </configurationRuleAction>
            <configurationCondition xsi:type='AuthenticationCondition' >
                <isCNF>false</isCNF>
                <packetFilterCondition>
                  <SourceAddress>
                    2001:0720:1710:0004::5001/128
                  </SourceAddress>
                </packetFilterCondition>
                <AuthenticationSubject>67a8c95d9f8c</AuthenticationSubject>
            </configurationCondition>
            <externalData xsi:type='Priority'>
                <value>0</value>
            </externalData>
            <Name>Rule0</Name>
            <isCNF>false</isCNF>
        </configurationRule>
        <Name>Conf0</Name>
    </configuration>
    <dependences>
      <dependence xsi:type='PolicyDependence'>
        <configurationCondition>
        <isCNF>true</isCNF>
        <policyID>mspl_9f1a88b4fc67421b98de270d5a63d35a</policyID>
        <status>ENFORCED</status>
        </configurationCondition>
      </dependence>
    </dependences>
</ITResource>
<!-- Allow authN Traffic from IoT -->
  <ITResource ID="mspl_9f1a88b4fc67421b98de270d5a63d35c">
    <configuration xsi:type='RuleSetConfiguration'>
        <capability>
            <Name>Traffic_Divert</Name>
        </capability>
        <configurationRule>
            <configurationRuleAction xsi:type='TrafficDivertAction' >
              <TrafficDivertActionType>FORWARD</TrafficDivertActionType>
              <packetDivertAction>
                <isCNF>false</isCNF>
                <packetFilterCondition>
                  <DestinationAddress>
                    2001:720:1710:4:5054:caff:fefe:770f/128
                  </DestinationAddress>
                </packetFilterCondition>
              </packetDivertAction>
            </configurationRuleAction>
            <configurationCondition
              xsi:type='TrafficDivertConfigurationCondition'>
```

```
            <isCNF>false</isCNF>
              <packetFilterCondition>
                <SourceAddress>
                  2001:0720:1710:0004::5001/128
                </SourceAddress>
                <DestinationAddress>
                  2001:720:1710:4:5054:caff:fefe:770f/128
                </DestinationAddress>
                      <DestinationPort>5678</DestinationPort>
                      <ProtocolType>UDP</ProtocolType>
              </packetFilterCondition>
           </configurationCondition>
          <externalData xsi:type='Priority'>
              <value>60000</value>
          </externalData>
          <Name>Rule0</Name>
          <isCNF>false</isCNF>
      </configurationRule>
      <Name>Conf0</Name>
  </configuration>
      <dependences>
    <dependence xsi:type='PolicyDependence'>
      <configurationCondition>
      <isCNF>true</isCNF>
      <policyID>mspl_9f1a88b4fc67421b98de270d5a63d35b</policyID>
      <status>ENFORCED</status>
      </configurationCondition>
    </dependence>
  </dependences>
</ITResource>
<!-- Allow authN Traffic to IoT -->
  <ITResource ID="mspl_9f1a88b4fc67421b98de270d5a63d35d">
  <configuration xsi:type='RuleSetConfiguration'>
      <capability>
          <Name>Traffic_Divert</Name>
      </capability>
      <configurationRule>
          <configurationRuleAction xsi:type='TrafficDivertAction' >
            <TrafficDivertActionType>FORWARD</TrafficDivertActionType>
            <packetDivertAction>
              <isCNF>false</isCNF>
              <packetFilterCondition>
                <DestinationAddress>
                  2001:0720:1710:0004:0000:0000:0000:5001/128
                </DestinationAddress>
              </packetFilterCondition>
            </packetDivertAction>
          </configurationRuleAction>
          <configurationCondition
            xsi:type='TrafficDivertConfigurationCondition'>
            <isCNF>false</isCNF>
              <packetFilterCondition>
                <SourceAddress>
                  2001:720:1710:4:5054:caff:fefe:770f/128
                </SourceAddress>
                <DestinationAddress>
                  2001:0720:1710:0004::5001/128
                </DestinationAddress>
                <SourcePort>5678</SourcePort>
                <ProtocolType>UDP</ProtocolType>
              </packetFilterCondition>
```

```
            </configurationCondition>
            <externalData xsi:type='Priority'>
              <value>60000</value>
            </externalData>
            <Name>Rule0</Name>
            <isCNF>false</isCNF>
        </configurationRule>
        <Name>Conf0</Name>
    </configuration>
    <dependences>
      <dependence xsi:type='PolicyDependence'>
        <configurationCondition>
        <isCNF>true</isCNF>
        <policyID>mspl_9f1a88b4fc67421b98de270d5a63d35b</policyID>
        <status>ENFORCED</status>
        </configurationCondition>
      </dependence>
    </dependences>
  </ITResource>
</ITResourceOrchestration>
```

Annex 2: AuthZ Orchestration MSPL

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Mapping xmlns:tns="http://www.example.org/Refinement_Schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/Refinement_Schema hspl.xsd ">
  <tns:hspl_list xsi:type="tns:HSPL_Orchestration"
ID="hsplo_a87cda98d7ca9d8787cdaa">
      <!-- Authorize Sensor1 to put resource in IoT Broker -->
       <tns:hspl id="hspl_a87cda98d7ca9d8787cdca" subject="Sensor1">
      <tns:action>authorise_access</tns:action>
      <tns:objectH>resource</tns:objectH>
      <tns:fields>
        <tns:traffic_target>
          <tns:target_name>IoT_Broker</tns:target_name>
        </tns:traffic_target>
        <tns:purpose>
          <tns:purpose_property>
            <tns:key>method</tns:key>
            <tns:value>PUT</tns:value>
          </tns:purpose_property>
        </tns:purpose>
        <tns:resource>
          <tns:resource_property>
            <tns:key>TEMPERATURE</tns:key>
          </tns:resource_property>
        </tns:resource>
      </tns:fields>
    </tns:hspl>
    <!-- Allow authorization traffic (sensor to paa) -->
    <tns:hspl id="hspl_a87cda98d7ca9d8787cdcb" subject="Sensor1"
      bidirectional="true">
      <tns:action>authorise_access</tns:action>
      <tns:objectH>PANA_traffic</tns:objectH>
      <tns:fields>
        <tns:traffic_target>
                  <tns:target_name>PANA_AGENT</tns:target_name>
        </tns:traffic_target>
      </tns:fields>
```

```
      <tns:dependences>
        <tns:dependence xsi:type="tns:PolicyDependence">
            <tns:dependenceCondition xsi:type="tns:PolicyDependenceCondition">
              <tns:policyID>hspl_a87cda98d7ca9d8787cdca</tns:policyID>
            <tns:status>ENFORCED</tns:status>
            </tns:dependenceCondition>
        </tns:dependence>
      </tns:dependences>
    </tns:hspl>
    <!-- Allow IoT device Traffic to Resource target -->
    <tns:hspl id="hspl_a87cda98d7ca9d8787cdce" subject="Sensor1">
      <tns:action>authorise_access</tns:action>
      <tns:objectH>CoAP_traffic</tns:objectH>
      <tns:fields>
        <tns:traffic_target>
          <tns:target_name>IoT_Broker</tns:target_name>
        </tns:traffic_target>
      </tns:fields>
      <tns:dependences>
        <tns:dependence xsi:type="tns:EventDependence">
            <tns:eventID>AUTHZ-SUCCESS</tns:eventID>
            <tns:dependenceCondition xsi:type="tns:EventDependenceCondition">
              <tns:subject>Sensor1</tns:subject>
              <tns:fields>
                <tns:traffic_target>
                  <tns:target_name>IoT_Broker</tns:target_name>
                </tns:traffic_target>
                <tns:purpose>
                  <tns:purpose_property>
                    <tns:key>method</tns:key>
                    <tns:value>PUT</tns:value>
                  </tns:purpose_property>
                </tns:purpose>
                <tns:resource>
                  <tns:resource_property>
                    <tns:key>TEMPERATURE</tns:key>
                  </tns:resource_property>
                </tns:resource>
              </tns:fields>
            </tns:dependenceCondition>
        </tns:dependence>
      </tns:dependences>
    </tns:hspl>
  </tns:hspl_list>
</tns:Mapping>
```

Annex 3: AuthZ Orchestration HSPL

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<ITResourceOrchestration ID="mspl_9f1a88b4fc67421b98de270d5a63d35f"
        xmlns="http://modeliosoft/xsddesigner/a22bd60b-ee3d-425c-8618-
beb6a854051a/ITResource.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://modeliosoft/xsddesigner/a22bd60b-ee3d-425c-
8618-beb6a854051a/ITResource.xsd ANASTACIA_MSPL_XML_Schema.xsd">
  <!-- Resource authorisation -->
  <ITResource ID="mspl_9f1a88b4fc67421b98de270d5a63d35a">
    <configuration xsi:type='RuleSetConfiguration'>
        <capability>
            <Name>AuthoriseAccess_resurce</Name>
```

```
        </capability>
        <configurationRule>
            <configurationRuleAction xsi:type='AuthorizationAction' >
              <AuthorizationActionType>ALLOW</AuthorizationActionType>
            </configurationRuleAction>
            <configurationCondition xsi:type='AuthorizationCondition'>
                <isCNF>false</isCNF>
                <packetFilterCondition>
                  <SourceAddress>
                    2001:0720:1710:0004::5001/128
                  </SourceAddress>
                  <DestinationAddress>IoT Broker IP</DestinationAddress>
                </packetFilterCondition>
                <timeCondition>
                  <Weekday></Weekday>
                  <Time>08:00-19:00,</Time>
                </timeCondition>
                <applicationLayerCondition
                  xsi:type="IoTApplicationLayerCondition">
                  <URL>/60001</URL>
                  <method>PUT</method>
                </applicationLayerCondition>
                <AuthorizationSubject>Wola</AuthorizationSubject>
            </configurationCondition>
            <externalData xsi:type='Priority'>
              <value>0</value>
            </externalData>
            <Name>Rule0</Name>
            <isCNF>false</isCNF>
        </configurationRule>
        <resolutionStrategy xsi:type='FMR'/>
        <Name>MSPL_b22c6384-ed08-487b-a3ca-ce2e557ca434</Name>
    </configuration>
</ITResource>
  <!-- Allow authZ Traffic from IoT -->
  <ITResource ID="mspl_9f1a88b4fc67421b98de270d5a63d35c">
    <configuration xsi:type='RuleSetConfiguration'>
        <capability>
          <Name>Traffic_Divert</Name>
        </capability>
        <configurationRule>
            <configurationRuleAction xsi:type='TrafficDivertAction' >
              <TrafficDivertActionType>FORWARD</TrafficDivertActionType>
              <packetDivertAction>
                <isCNF>false</isCNF>
                <packetFilterCondition>
                  <DestinationAddress>
                    2001:720:1710:4:5054:caff:fefe:770f/128
                  </DestinationAddress>
                </packetFilterCondition>
              </packetDivertAction>
            </configurationRuleAction>
            <configurationCondition
              xsi:type='TrafficDivertConfigurationCondition'>
              <isCNF>false</isCNF>
                <packetFilterCondition>
                 <SourceAddress>
                   2001:0720:1710:0004::5001/128
                 </SourceAddress>
                  <DestinationAddress>
                    2001:720:1710:4:5054:caff:fefe:770f/128
```

```
                </DestinationAddress>
                <DestinationPort>5678</DestinationPort>
                <ProtocolType>UDP</ProtocolType>
              </packetFilterCondition>
            </configurationCondition>
            <externalData xsi:type='Priority'>
              <value>60000</value>
            </externalData>
            <Name>Rule0</Name>
            <isCNF>false</isCNF>
        </configurationRule>
        <Name>Conf0</Name>
    </configuration>
    <dependences>
      <dependence xsi:type='PolicyDependence'>
        <configurationCondition>
        <isCNF>true</isCNF>
        <policyID>mspl_9f1a88b4fc67421b98de270d5a63d35b</policyID>
        <status>ENFORCED</status>
        </configurationCondition>
      </dependence>
    </dependences>
  </ITResource>
  <!-- Allow authZ Traffic to IoT -->
    <ITResource ID="mspl_9f1a88b4fc67421b98de270d5a63d35d">
    <configuration xsi:type='RuleSetConfiguration'>
        <capability>
          <Name>Traffic_Divert</Name>
        </capability>
        <configurationRule>
            <configurationRuleAction xsi:type='TrafficDivertAction' >
              <TrafficDivertActionType>FORWARD</TrafficDivertActionType>
              <packetDivertAction>
                <isCNF>false</isCNF>
                <packetFilterCondition>
                  <DestinationAddress>
                    2001:0720:1710:0004::5001/128
                  </DestinationAddress>
                </packetFilterCondition>
              </packetDivertAction>
            </configurationRuleAction>
            <configurationCondition
              xsi:type='TrafficDivertConfigurationCondition'>
              <isCNF>false</isCNF>
                <packetFilterCondition>
                  <SourceAddress>
                    2001:720:1710:4:5054:caff:fefe:770f/128
                  </SourceAddress>
                  <DestinationAddress>
                    2001:0720:1710:0004::5001/128
                  </DestinationAddress>
                  <SourcePort>5678</SourcePort>
                  <ProtocolType>UDP</ProtocolType>
                </packetFilterCondition>
             </configurationCondition>
            <externalData xsi:type='Priority'>
              <value>60000</value>
            </externalData>
            <Name>Rule0</Name>
          <isCNF>false</isCNF>
        </configurationRule>
```

```
      <Name>Conf0</Name>
  </configuration>
  <dependences>
    <dependence xsi:type='PolicyDependence'>
      <configurationCondition>
      <isCNF>true</isCNF>
      <policyID>mspl_9f1a88b4fc67421b98de270d5a63d35b</policyID>
      <status>ENFORCED</status>
      </configurationCondition>
    </dependence>
  </dependences>
</ITResource>
  <!-- Allow Traffic from IoT Controller to IoT Broker -->
<ITResource ID="mspl_9f1a88b4fc67421b98de270d5a63d35a">
  <configuration xsi:type='RuleSetConfiguration'>
      <capability>
        <Name>Traffic_Divert</Name>
      </capability>
      <configurationRule>
          <configurationRuleAction xsi:type='TrafficDivertAction' >
            <TrafficDivertActionType>FORWARD</TrafficDivertActionType>
            <packetDivertAction>
              <isCNF>false</isCNF>
              <packetFilterCondition>
                <DestinationAddress>
                  2001:0720:1710:0004:0000:0000:0000:5001/128
                </DestinationAddress>
              </packetFilterCondition>
            </packetDivertAction>
          </configurationRuleAction>
          <configurationCondition
            xsi:type='TrafficDivertConfigurationCondition'>
            <isCNF>false</isCNF>
              <packetFilterCondition>
                <SourceAddress>
                  2001:720:1710:4:5054:ff:feec:abcd/128
                </SourceAddress>
                <DestinationAddress>
                  2001:0720:1710:0004::5001/128
                </DestinationAddress>
              </packetFilterCondition>
           </configurationCondition>
          <externalData xsi:type='Priority'>
            <value>60000</value>
          </externalData>
          <Name>Rule0</Name>
          <isCNF>false</isCNF>
      </configurationRule>
      <Name>Conf0</Name>
  </configuration>
  <dependences>
    <dependence xsi:type='EventDependence'>
    <eventID>AUTHZ-SUCCESS</eventID>
    <configurationCondition xsi:type='FilteringConfigurationCondition'>
      <isCNF>false</isCNF>
      <packetFilterCondition>
        <SourceAddress>
          2001:0720:1710:0004:0000:0000:0000:5001/128
        </SourceAddress>
        </packetFilterCondition>
      </configurationCondition>
```

```
      </dependence>
    </dependences>
  </ITResource>
</ITResourceOrchestration>
```

Annex 4: AuthZ Orchestration MSPL

# 9 REFERENCES


[1] Common Information Model (CIM), DMTF Standard.

[2] Jorge Bernal Bernabe, Juan M. Marin Perez, Jose M. Alcaraz Calero, Jesus D. Jimenez Re, Felix J. Garcia Clemente, Gregorio Martinez Perez, Antonio F. Gomez Skarmeta, **"Security Policy Specification"**, Network and Traffic Engineering in Emerging Distributed Computing Applications, IGI Global, pp. 66-93, 2012.

[3] Policy-Based Security Tools and Framework (POSITIF), EU project, FP6, IST-2002-002314

[4] Dependable Security by Enhanced Reconfigurability (DESEREC), IST-2004-026600, EU project, Framework Programme 6.

[5] SECURED EU FP7 project, deliverable D4.1: **Policy specification**.

[6] SECURED EU FP7 project, deliverable D4.2: **Policy transformation and optimization techniques**.

[7] SECURED EU FP7 project, https://www.secured-fp7.eu/

[8] https://www.w3.org/TR/2007/REC-ws-policy-20070904/ws-policy-framework.pdf

[9] Twidle, K., Dulay, N., Lupu, E., & Sloman, M. (2009). Ponder2: A Policy System for Autonomous Pervasive Environments. 2009 Fifth International Conference on Autonomic and Autonomous Systems. doi:10.1109/icas.2009.42

[10] **HoneyNet description language**: W. Fan, D. Fernández and V. A. Villagrá, "Technology independent honeynet description language," 2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Angers, 2015, pp. 303-311.

[11] Diego Lopez et al. **I2NSF Framework: Capabilities, interfaces and framework**. Internet-Draft, IETF. May 2017. I2NSF Working Group. https://tools.ietf.org/html/rfc8329

[12] Basile et al. **Model of Security Capabilities for Network Security Functions**. Internet-Draft , IETF. January 2017. https://tools.ietf.org/html/draft-ietf-i2nsf-capability-04

[13] Giotis, K., Kryftis, Y., & Maglaris, V. (2015). Policy-based orchestration of NFV services in Software-Defined Networks. Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft). doi:10.1109/netsoft.2015.7116145

[14] Li, X., & Qian, C. (2016). An NFV Orchestration Framework for Interference-Free Policy Enforcement. 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). doi:10.1109/icdcs.2016.24

[15] Zhou, Y. C., Liu, X. P., Kahan, E., Wang, X. N., Xue, L., & Zhou, K. X. (2007). Context Aware Service Policy Orchestration. IEEE International Conference on Web Services (ICWS 2007). doi:10.1109/icws.2007.66

[16] Alcaraz Calero, J. M., Marín Pérez, J. M., Bernal Bernabé, J., Garcia Clemente, F. J., Martínez Pérez, G., & Gómez Skarmeta, A. F. (2010). Detection of semantic conflicts in ontology and rule-based information systems. Data & Knowledge Engineering, 69(11), 1117–1137. doi:10.1016/j.datak.2010.07.004

[17] Bernabe, J. B., Perez, G. M., & Skarmeta Gomez, A. F. (2015). Intercloud Trust and Security Decision Support System: an Ontology-based Approach. Journal of Grid Computing, 13(3), 425–456. doi:10.1007/s10723-015-9346-7

[18] Y. Sun, T. Wu, G. Zhao and M. Guizani, "Efficient Rule Engine for Smart Building Systems," in IEEE Transactions on Computers, vol. 64, no. 6, pp. 1658-1669, 1 June 2015. doi: 10.1109/TC.2014.2345385.