D4.2
Policy transformation and optimization techniques

Project number	611458
Project acronym	SECURED
Project title	SECURity at the network EDge
Project duration	36 months (1/10/2013–30/9/2016)
Programme	FP7 (Collaborative Project)

Deliverable type	**R** - Report
Deliverable number	D4.2
Version (date)	v1.3 (9/2/2016)
Work package(s)	WP4
Due date	30/9/2015 – M24

Responsible organisation	POLITO
Editor	Cataldo Basile
Dissemination level	**PU** - Public

Abstract	This documents presents the architecture, workflow, and modes of operation of the H2M Service and M2L Service, the two refinement services offered by the SECURED framework.
Keywords	policy refinement, policy translation, non-enforceability

Editor
Cataldo Basile (POLITO)

Reviewers
René Serral-Gracià (UPC)
Antonio Lioy (POLITO)

Contributors
Christian Pitscheider (POLITO)
Fulvio Valenza (POLITO)
Marco Vallini (POLITO)

Acknowledgement
This work was partially supported by the European Commission (EC) through the FP7-ICT programme under project SECURED(grant agreement no. 611458).

Change Log

Version	Date	Note	Author
v1.0	03.10.2015	Version after internal review	Cataldo Basile
v1.1	05.11.2015	Quality control	Antonio Lioy
v1.2	05.02.2016	Reviewers' comments addressed	Cataldo Basile
v1.3	09.02.2016	Quality control	Antonio Lioy

Executive Summary

This documents presents the policy refinement approach used in the SECURED project. Moreover, the two components that actually implement the SECURED refinement approach are described: the H2M Service and M2L Service.

Three policy abstraction layers are used in SECURED: the High-level Security Policy Language (HSPL), the Medium-level Security Policy Language (MSPL) and the set of the low level configurations needed to configure individual PSAs. Therefore, two refinement processes have been defined to bridge the gap, the refinement of HSPL into MSPL, performed by the H2M Service, and the translation of MSPL policies into low-level configurations, performed by the M2L Service.

The refinement of HSPL policies into MSPL policies is a sophisticated task that can be executed following two different approaches: the policy-driven and application-driven approaches, depending on the required inputs.

The *policy-driven approach* expects as input from the user only the policy to enforce. In this case, the H2M Service first identifies the network security functions needed to enforce the policy, i.e. the *capabilities*. Then it identifies different alternative sets of PSAs, taken from the PSA Repository (PSAR), that own the needed capability and can thus enforce the user policy. Finally, the H2M Service adopts optimization techniques to select the PSAs to use. The optimization criterion is selected by the user from a set of predefined target functions, which optimize on the overall performance, throughput, latency, bandwidth, and, last but not least, on the actual security. As we use multi-objective optimization, target functions can be combined. Once the PSAs to use have been selected, the HSPL policy is mapped into MSPL rules. Moreover, the H2M Service generates the Service Graph, a data structure that describes how packets will be processed by the selected PSAs when instantiating the user Personal Security Controller (PSC).

The *application-driven approach* expects as input from the user the required security policy and the PSA to use to actually enforce the policy. In this case, the H2M Service only maps HSPL policies into MSPL policies and completes, if needed, the information on desired PSAs to generate the Service Graph.

This H2M Service also provides a mechanism to identify cases when the user policy is not enforceable. That is, H2M Service detects if (1) high-level requirements require capabilities or features (like matching conditions, actions) required to enforce the policy is provided by the PSAs selected by the user. In these cases, H2M Service provides remediation hints, i.e. it suggests changes to the policy or to the list of PSAs selected by the user.

The transformation of MSPL policies into PSA configurations performed by the M2L Service mainly involves a change of syntax. Indeed, each PSA may have a different configuration language, but MSPL maps the semantics of every supported PSAs by categorizing it by capability. The translation to each target PSA is performed by a PSA-specific translation module, which takes care of mapping MSPL policies into low-level configurations for one PSA. The M2L Service provides the necessary framework to automatically select the proper translation plugin, obtain the most up to date version and invoke it.

Contents

1 Introduction **1**

2 The SPM architecture **3**
2.1 Online workflows 5
2.2 Data 5
2.2.1 Policies 6
2.2.2 PSAs and service graphs 7
2.2.3 Additional user, network, and mapping information 8

3 HSPL to MSPL transformation **9**
3.1 Basic refinement operations 9
3.2 Policy refinement approaches 9
3.2.1 Policy-driven approach 9
3.2.2 Application-driven approach 12
3.3 Architecture and workflow 16
3.3.1 Requirements identification phase 16
3.3.2 PSA optimization phase 16
3.3.3 Application Graph generation phase 20
3.3.4 MSPL generation phase 21
3.4 Implementation 22

4 MSPL to low-level configuration **24**
4.1 Architecture and workflow 24
4.2 Implementation 27
4.3 Integration in SECURED 28

References **30**


SECURED

1 Introduction

This document presents the refinement models and processes defined in the first two years of the SECURED project and implemented into the SECURED Policy Manager (SPM).

Note that, in the Description of Work, the authorship of this document was assigned to UPC; however, due to the actual distribution of the activities reported in this deliverable, we have preferred to shift leadership to POLITO.

Policy refinement is the process "to determine the resources needed to satisfy policy requirements, to translate high-level policies into operational policies that may be enforced by the system, to verify that the set of lower level policies actually meets the requirements of the high-level policy" [3].

SPM uses three policy abstraction layers, i.e. the High-level Security Policy Language (HSPL), the Medium-level Security Policy Language (MSPL) and the set of the low level configurations needed to configure individual PSAs). The two highest levels have been defined in this project and presented in details in the deliverable D4.1. The HSPL has been designed to express security requirements by means of "subject-verb-object-parameters" sentences. MSPL has been defined to abstract the configuration languages with a vendor and PSA-independent format, which is organized by capability (see Deliverable D4.1 for more details on both MSPL format and capabilities). On the other hand, PSA-specific configuration languages depend on formats and features available at the actual PSA. Furthermore, PSAs are often based on existing security controls, with their own configuration languages, properly wrapped to work with the SECURED architecture. This means that two out of three abstraction layers, HSPL and MSPL, are under our control, that is, we have defined them to meet our purposes, while the third one, the low level configurations are not.

Since there are three abstraction layers, SECURED uses two separate processes for refinement purposes. The first one is the refinement of high-level policies, written in HSPL into medium-level policies represented using MSPL. This is performed by the SECURED component named H2M Service. The second process is the translation of MSPL policies into low-level configurations that are usable by PSAs to actually enforce the policy, performed by the SECURED component named M2L Service.

There is a clear reason for this design, which splits the refinement process and uses MSPL as an intermediate policy language, instead of a monolithic translation from HSPL to low-level configurations. Indeed, we aim at providing enough flexibility to switch from one PSA to another one that has the same capabilities without the need to master different configuration languages or perform manual and error prone translations. By having this additional layer of abstraction, the refinement process is independent from different PSA types. Indeed it is possible to associate the same MSPL policy to another PSA having the same capability (i.e. that can enforce the same type of policy) to change the security control, the M2L Service provides the translation into the new low level configuration settings. Considering the objectives of SECURED, this architecture is flexible enough to be extended (e.g. by adding more capabilities and more PSAs) with limited or no effort.

The refinement of HSPL policies into MSPL policies is a sophisticated task. We propose to perform it with two different approaches: the policy-driven and application-driven approaches. The former expects as input from the user only the policy to enforce. The latter expects from the user the policy and the PSA to use to actually enforce the policy. For this reason, the application-driven approach is not recommended for non technology-savvy users.

For the *policy-driven approach*, it is first necessary to identify the security functions needed to enforce the policy. Since there are several PSAs that own the needed capability, it is possible to choose among several alternatives. Different implementations and combinations of PSAs may have different side-

effects on the overall performance, throughput, latency, bandwidth, and, last but not least, on the actual security. For example a certain PSA implementation requires higher processing power than another, or reduces significantly the bandwidth or latency. The H2M Service evaluates all possible side-effects of different combinations of PSAs and specific configurations and selects the best. To select the PSAs to use, H2M Service uses optimization techniques.

Once the PSAs to use are known (because they have been selected by the optimization process in the policy-driven approach or because they have been manually specified in the application-driven approach), the HSPL policy is mapped into MSPL rules. Moreover, H2M Service generates the Service Graph, a data structure that describes how packets will be processed by the selected PSAs.

The transformation of MSPL policies into PSA configurations mainly involves a change of syntax, as MSPL has been designed to share the same semantics as the PSAs. Each PSA implementation may have a different configuration language and therefore it is needed a PSA-specific translation module, that actually maps MSPL policies into concrete configuration. For example the refinement process requests a firewall PSA and generates the corresponding MSPL, the PSA implementation is not restricted to be compliant with MSPL as there is a translation from MSPL to its configuration language. Therefore, the M2L Service provides the necessary framework to automatically select the proper translation plugin, obtain the most up to date version and invoke it.

It is worth noting that our proposal follows the design principles proposed by Strassner for policy-based network management [3], where the HSPL maps to "Business/System View"-layer, the MSPL maps to the "Administrative View"-layer and the concrete configurations map to the "Device/Instance View"-layer.

In order for the refinement to be feasible, all the concepts at the higher level must have a (maybe equivalent) counterpart at the lower level. Therefore, starting from high-level security requirements, defined by HSPL policies, during the refinement we need a mechanism able to identify the related set of security capabilities needed to enforce the policies by using a set of PSAs (e.g. packet filter PSAs). According to [5], a security policy must be *implementable* through system administration procedures (e.g. publishing of acceptable use guidelines) and *enforceable* with security tools or controls. However in a real scenario, some policies may be less precisely enforceable on some systems than others or in worst case, completely non-enforceable. For example, as suggested by Bishop and Peisert [4], the access control on traditional UNIX systems is much less granular when compared with model authorization policies, so that some authorization policies are not fully supported. In particular, considering the scope of SECURED, two situations can be detected: high-level constraints require a set of capabilities that are not available (non-enforceable) or only a subset of them is available (partially enforceable). Therefore the refinement process is accompanied by an indication of how to handle these situations, e.g. warning the user, suggesting a more relaxed policy, adding a third-party software or install a different PSA to compensate the absent functionality. That is, if user policies are non-enforceable or partially enforceable, the H2M Service provides remediation hints. For example suggest other PSAs not in the user repository or how to modify the policies.

This document is organized as follows. Section 2 introduces the general architecture of the SPM and highlights the components that will be presented in this deliverable. It also presents the policy-related workflows that determine how the SPM components are invoked in the SECURED architecture to perform their services. Finally, it describes the input and output formats managed by the H2M Service and M2L Service. Section 3 presents more in details the H2M Service, its internal architecture, workflow and modes of operation. Finally, Section 4 presents more in details the M2L Service, its internal architecture, workflow and modes of operation.

2 The SPM architecture

Figure 1 presents the last version of the SPM architecture and its relations with the SECURED Repository.

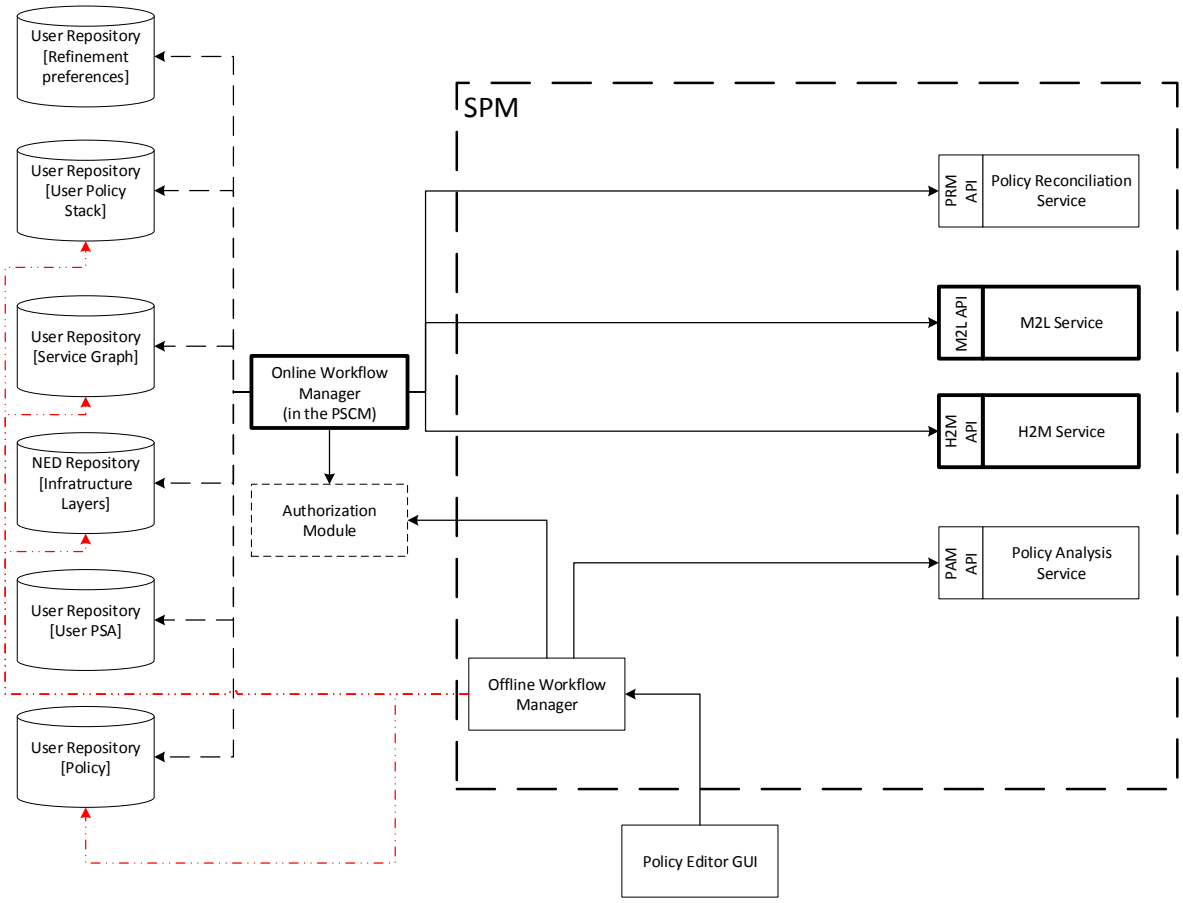

Figure 1: the SPM architecture.

Four main modules are defined:

- Policy Reconciliation Service, which reconciles policies from different actors after the users connects to a NED;
- Policy Analysis Service, which analyses MSPL policies both of the same actor and of different actors taken from the same user policy stack;
- H2M Service, which performs the refinement of HSPL policies into MSPL policies and it also selects an optimal set of PSAs to enforce the HSPL policy, if the PSAs have not been already decided by the user;
- M2L Service, which performs, after the reconciliation, the refinement of MSPL policies into configurations for the user PSAs.

Figure 2 shows the relations between the policy abstractions and the SPM components. It is possible to see how H2M Service accesses HSPL policies and produces MSPL policies that are translated into low-level configuration settings to be enforces by the user PSAs.

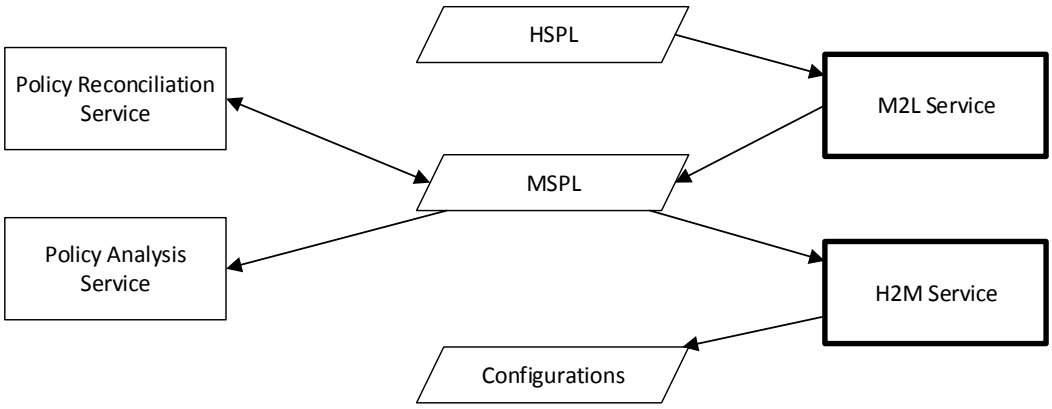

Figure 2: the policy transformation process.

This deliverable covers only the H2M Service and M2L Service, whereas Policy Reconciliation Service and Policy Analysis Service will be described in D4.3. Moreover, there are other components that are not inside the SPM that are needed for the SECURED infrastructure to work correctly:

- Online Workflow Manager, which is a PSCM components that properly calls all the SPM components in order to allow the correct functioning of the SECURED architecture for policy related tasks when a user connects to the infrastructure and during his connection;
- Offline Workflow Manager, which is in charge for implementing the correct workflows to implement offline policy-related operations (like collecting the proper policies in the user stack to perform the analysis and to show and store results);
- Authorization Module, which is the Policy Decision Point that determines whether a SECURED component is allowed to access SECURED Repository objects (to access user or other actor policies).

This deliverable, also covers the Online Workflow Manager, whereas the Offline Workflow Manager will be described in D4.3. Currently, the Authorization Module, foreseen to make the SECURED architecture as general as possible, is not implemented. Indeed, we assumed that:

- users are allowed to access cooperative policies in their policy stack;
- the Online Workflow Manager is inside the PSCM thus it knows the user, as he authenticated to the NED before, and it can act in the user behalf.

It is worth noting that the authorization and delegation issues determined by the current SECURED architecture are not interesting from the research point of view, as they are already addressed by previous work in literature (e.g., by instantiating the XACML architecture and defining the authorization policies with XACML policies). Therefore, the implementation of the Authorization Module would only add complexity the SECURED prototype without giving additional value.

2.1 Online workflows

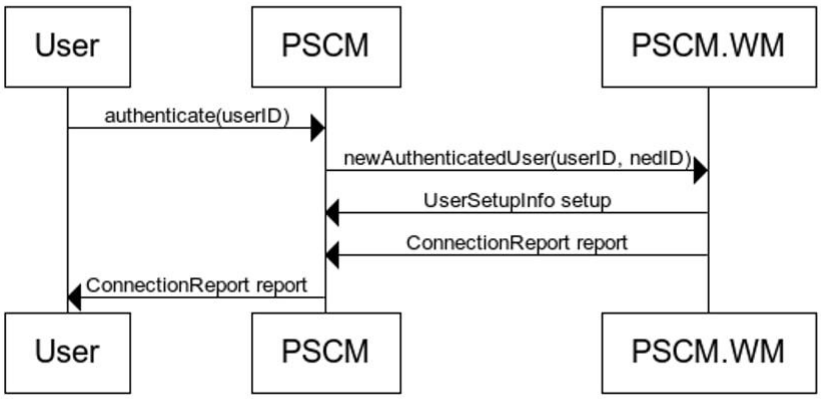

Figure 3: Online workflows.

Three operations (Fig. 3) are performed by the Online Workflow Manager when a new user correctly authenticates, in this order:

1. The Online Workflow Manager collects the user HSPL policies from the SECURED Repository and calls the H2M Service to perform the refinement. Possibly, PSAs to enforce the policies are selected by the H2M Service, if the user has selected the policy-driven approach. Then, MSPL policies and PSAs (if selected) are stored into the SECURED Repository.

2. The Online Workflow Manager reconciles user MSPL policies with the policies from other cooperative actors in the user policy and in the NED infrastructure policy stack. The Online Workflow Manager first contacts the SECURED Repository to collect policy information then it passes everything to the Policy Reconciliation Service. Then the reconciled MSPL policies are stored into the SECURED Repository.

3. The Online Workflow Manager translates the reconciled MSPL policies by calling the M2L Service.

4. The Online Workflow Manager passes the configurations for the user PSA to the PSCM, which will pass them to PSC so that all the PSA are correctly configured (details on this process are available from the WP3 deliverable).

2.2 Data

This section introduces the types of data used by different components of the SECURED policy refinement architecture.

Data used by the H2M Service and M2L Service are all available from the SECURED Repository. In particular, the parts of the SECURED Repository that are accessed by the Online Workflow Manager in order to correctly invoke the two refinement services are (Fig. 1):

- policies, which are the security user requirements expressed in one of the formats (abstraction layers) used in SECURED;

- user Service Graphs (Service Graphs), which are the formal representation of how packets are processed by the SECURED infrastructure;
- user PSAs, which include two separate lists: the PSAs actually chosen (by the user of the refinement process) and the list of PSAs the user would consider when performing the selection during the policy driven refinement.

Moreover, H2M Service and M2L Service produce as output or as intermediate formats the

- low-level configurations, and
- capabilities.

2.2.1 Policies

We summarize here basic information about the policy abstractions defined in SECURED (more detail can be found in the deliverable D4.1).

High-level Security Policy Language

The HSPL policies are the highest policy abstraction we use in SECURED. This language is suitable for expressing the general protection requirements of typical non-technical end-users.

The user HSPL policy is composed of a set of "*subject-verb-object-[attributes]*" or "*verb-object-[attributes]*" statements. The format of these statements is typical of access control languages. Examples of HSPL policies are "block access to Internet every day between 10PM and 8AM" and "Enable basic parental control for Alice".

The complete syntax of HSPL is defined in the deliverable D4.1 together with all the statements that are valid in the SECURED scope.

In the repository, HSPL is stored as a single XML file, whose schema is defined in:

`https://www.secured-fp7.eu/ref/h2m_refinement/refinement-xml-schema/`

Medium-level Security Policy Language

The MSPL policies are the intermediate policy abstraction used in SECURED. MSPL is a language that allows the specification of policies for generic categories of PSAs on a per capability basis. Capabilities determine the kind of policies that a PSA can enforce. Example of capabilities are packet filter, channel protection, and content inspection. MSPL policies serve to express precise configurations by technically-savvy users in a device-independent format. Examples of MSPL rules that can be added to MSPL policies are "`deny access to *.sex`", "`deny access if src_IP=192.168`", or "`inspect image/* for malware`".

MSPL is fully defined in the deliverable D4.1 together with the complete capability description.

In the repository, a user is associated to several MSPL policies, each one stored as a single XML file that represents an MSPL policy for a single capability. That is, there will be one MSPL XML file for each one of the capabilities required to enforce the user policy. The MSPL XMLSchema is defined in:

`https://www.secured-fp7.eu/ref/h2m_refinement/mspl-xml-schema/`

Low-level configurations

Low-level configurations are the lowest policy abstraction used in SECURED. they abstracts the configuration formats used by all the security controls within the PSAs used in SECURED. Since most of the security controls we use in SECURED have not been defined within this project, we cannot control their configuration files. Therefore the low-level configurations is not defined in SECURED and we have to resort to individual security controls formats. For instance, if a PSA that exposes a packet filter capability internally uses iptables, the low-level configuration for this PSA will be a valid iptables configuration (txt) file. Even if in most cases we produce textual files, in some other cases we could produce XML files, depending on the PSA. Moreover, when several files are generated for the same PSA, we produce a single compressed file (like for instance for the PSA based on Dansguardian fitler).

2.2.2 PSAs and service graphs

There are two different data structure used for policy refinement and involve PSAs.

First, some of the refinement tasks require an input list of PSAs. These lists are passed as XML files that follow the XML Schema presented in:

https://www.secured-fp7.eu/ref/h2m_refinement/refinement-xml-schema/

Then, the Service Graph is the formalism used to describe how packets will be processed by the SECURED infrastructure. In the most general case, the Service Graph is a collection of directed graphs in which nodes represent components that process packets (i.e. the PSAs) and arcs represent the flow of the traffic between the two nodes. In its simplest form, Service Graph is a set of chained PSAs active on the user's traffic. Several service graphs can be used to represent how to process different classes of packets (e.g., packets directed to the users can be processed differently from packets produced by the user, as well as HTTP traffic could be processed by PSAs that are not needed for FTP traffic). In alternative, an equivalent representation can be obtained with multi-graphs, if arcs are labelled with the proper traffic filters.

In the current implementation of the SECURED infrastructure, we only use one Service Graph for input, output and all types of traffic. We could support more efficient network infrastructures in Y3, but it is outside the Y2 objectives.

In general, a Service Graph is intended to describe the service, not how the service will be provided. As a consequence, no low-level network information is present, such as physical/virtual ports, applications/VNF placement, topology, and other.

Before presenting the actual data structures to represent Service Graph, we introduce another concept: the *Generic PSA*. We will name Generic PSAs the abstract security controls implementing one specific capability. There are several Generic PSAs, which are identified by the capability they provide, i.e. there is the Generic Packet Filter, the Generic Anti-Virus, etc.

We use for the refinement two types of Service Graphs, depending on the information on the components that process packets stored in nodes:

- Generic Service Graph, or simply Service Graph (SG), is the data structure that describes the potential arrangements of security controls required to enforce user policies, therefore, all the nodes are either known PSAs taken from the PSAR or Generic PSAs.
- Application Graph (AG) is the data structure that describes the actual arrangement of PSAs required to enforce user policies, therefore, all the nodes are known PSAs taked from the PSAR.

An Application Graph can be directly passed to the NED to create the user PSC. In short, an Application Graph is a Service Graph where no abstract security controls are used.

In SECURED, Service Graphs are represented as XML files compliant with the XMLSchema presented in

https://www.secured-fp7.eu/ref/h2m_refinement/refinement-xml-schema/

Furthermore, the associations between a capability owned by a PSA and the MSPL policies they have to enforce are represented by another XML file that follows the XML Schema presented in

https://www.secured-fp7.eu/ref/h2m_refinement/matching-xml-schem/

2.2.3 Additional user, network, and mapping information

There are additional data that are needed to complete the refinement process. Also in this case, they are passed as XML files:

- network level information, which serve to associate the user or the other users specified in HSPL to network addresses;
- URL lists, which are exploded for instance when configuring specific HSPL policies (e.g. to describe which are the phising or blacklisted sites);
- semantic shortcuts, which expand high-level concepts into precise information usable at MSPL level (e.g. to define what are unsafe HTTP methods in case a user would specify this concept in a policy).

3 HSPL to MSPL transformation

This section describes the approaches and the operations performed to refine a HSPL policy into a set of MSPL policies.

3.1 Basic refinement operations

At the very high level, the refinement operations performed by the H2M Service can be summarized here (they are not necessarily performed in this order):

- *Identification of the capabilities needed to enforce a user HSPL policy*. This operation is named *Requirements identification*. This phase processes the HSPL policy statements specified by the user in order to determine all the security controls, thus the capabilities, that serve to enforce the user policy. During this phase it is performed the matching between the capabilities required by the user HSPL policy. This operation is named **capability matching**. Moreover the second task of this phase it is the identification of the PSAs needed to enforce policies that require certain capabilities. This operation is named **PSA identification**.
- *Identification of the non-enforceable policies and notification of potential remediation* This operation is named **non-enforceability analysis**. H2M Service performs two types of non-enforceability analysis: *early non-enforceability analysis* immediately notifies a user that his policy is not enforceable due to lack of capabilities in the PSAs he has selected, and *complete non-enforceability analysis* that notifies a user anomalous situations that may happen when generating his MSPL policy like attributes (conditions, actions) that are not supported by the PSA the MSPL is intended to.
- Translation of an HSPL policy into MSPL policies, one MSPL for each capability. This operation is named **MSPL generation**.
- Generation of the Service Graph that describes how the user packets will be processed by the PSAs. This operation is named **Application Graph generation**.
- Determining a set of PSAs that provide capabilities enough to enforce the user policy and optimize some user selected criteria (like maximize performance, minimize the cost PSAs to buy, etc.). This operation is named **PSA optimization**.

3.2 Policy refinement approaches

In SECURED we foresee two ways to perform policy refinement: *application-driven* and *policy-driven*.

3.2.1 Policy-driven approach

In the policy-driven approach, the user specifies his policy and the enforcement applications are selected by SECURED (from the PSAR or a user selected set of PSAs) based on their capabilities matching those required by the policies. Therefore, the policy-driven approach drastically reduces and simplifies the required effort to the user. Thus, it is recommended for inexperienced users that have not enough technical skills.

There are three ways to use the policy-driven approach, depending on the policy abstraction used to pass the user policy: the HSPL policy-driven refinement, if a HSPL policy is passed, MSPL policy-driven refinement, if an MSPL policy is passed, and hybrid policy-driven refinement, if both HSPL and MSPL are passed as input.

Policy-driven refinement – HSPL inputs

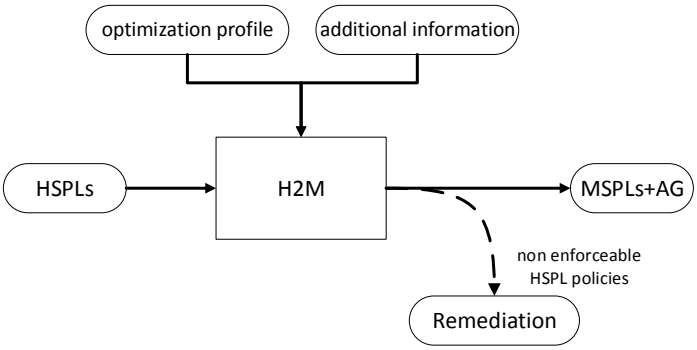

Figure 4: Policy-driven refinement – MSPL inputs

Figure 4 presents the most intuitive way to use the policy-driven approach, the *HSPL policy-driven refinement*. In this case, the user only specifies its HSPL, then selects a PSA optimization criterion, then the H2M Service produces in output the user Application Graph, the MSPL, and all the additional user, network, and mapping information needed to complete the correct enforcement.

Additionally, the user can provide a list of PSA to choose from for the optimization task. This list contains additional information about PSAs, like the fact that the PSAs has been already bought by the user or order of preferences in case of equivalent alternatives.

While it is unlikely that HSPL policies are non-enforceable if PSAs are taken from the PSA Repository, if the user provides a list of PSAs to choose from, some HSPL policy can be non-enforceable. Therefore, H2M Service reports users about non-enforceability cases and provides hints for solving them. Moreover, when generating the MSPL, H2M Service notifies also about complete non-enforceability issues.

In this case, the following basic operations are performed:

- **capability matching**: identification of the capabilities needed to enforce the input HSPL policy;
- **PSA identification**: identification of the different sets of PSAs that own the capabilities needed to enforce the input HSPL policies;
- **non-enforceability analysis**: identification of non-enforceable HSPLs due to lack of capabilities in the PSAs available in the UPR;
- **PSA optimization**: selection of the optimal set of PSAs according to the user selected criterion;
- **MSPL generation**: translation of the input HSPL policy into MSPL policies;
- **Application Graph generation**: generation of the user Application Graph.

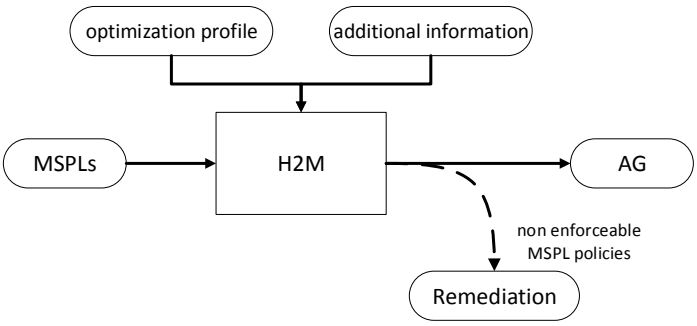

Figure 5: Policy-driven refinement – MSPL inputs

Policy-driven refinement – MSPL inputs

Figure 5 presents another way to use H2M Service, the *MSPL policy-driven refinement*. In this case, the users specifies the MSPL policies for the capabilities he wants to configure but leaves SECURED the decision of the PSAs to use according to an optimization criterion he selects. Also in this case, the user can provide a list of PSA to choose from. H2M Service produces in output the user Application Graph, and all the additional user, network, and mapping information needed to complete the correct enforcement of previously specified MSPL.

Moreover, H2M Service reports users about non-enforceability cases and provides hints for solving them.

In this case, the following basic operations are performed:

- **PSA identification**: identification of sets of PSAs that can be used to enforce the capabilities of MSPLs passed as input;
- **non-enforceability analysis**: identification of non-enforceable MSPLs due to lack of capabilities in the sets of PSA passed as input;
- **PSA optimization**: selection of the optimal set of PSAs according to the user selected criterion;
- **Application Graph generation**: generation of the user Application Graph.

Policy-driven refinement – HSPL and MSPL inputs

Figure 6 presents the last way we have foreseen to use H2M Service, the *hybrid policy-driven refinement*. In this case, the user can specify its policy by means of both HSPL and MSPL policies. For instance, this way of using H2M Service can be helpful in case of an expert user, trusting the SECURED infrastructure, that uses HSPL to define the basic security requirements then adds fine tuned rules directly in MSPL.

In this case, the user specifies its HSPL and MSPL, then he selects a PSA optimization criterion. The H2M Service produces in output the user Application Graph, the MSPL, and all the additional user, network, and mapping information needed to complete the correct enforcement.

Optionally, the user can provide a list of PSA to choose from.

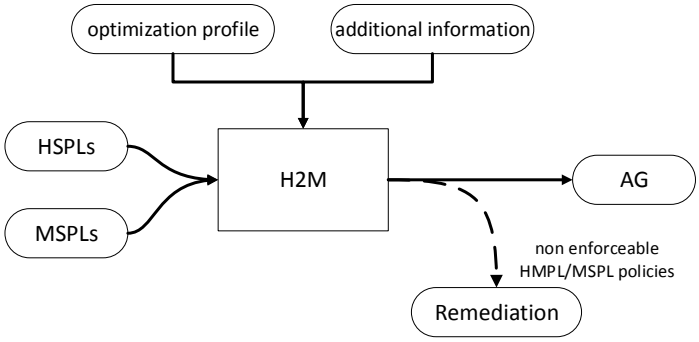

Figure 6: Policy-driven refinement – HSPL and MSPL inputs.

Also in this case, H2M Service reports users about non-enforceability cases and provides hints for solving them. Moreover, when generating the MSPL, H2M Service notifies also about complete non-enforceability issues.

In this case, the following basic operations are performed:

- **capability matching**: identification of the capabilities needed to enforce the input HSPL policy;
- **PSA identification**: identification of the different sets of PSAs that own the capabilities needed to enforce the input HSPL and MSPL policies;
- **non-enforceability analysis**: identification of non-enforceable HSPLs and MSPLs due to lack of capabilities in the sets of PSA passed as input;
- **PSA optimization**: selection of the optimal set of PSAs according to the user selected criterion;
- **MSPL generation**: translation of the input HSPL policy into MSPL policies;
- **Application Graph generation**: generation of the user Application Graph.

An additional operation is required in this use case: the reconciliation. Since some MSPL policies generated from the input HSPL by refinement may conflict with the manually written MSPL policies, a reconciliation step is necessary. By default, we assumed that manually written MSPL override the ones generated from the HSPL. Moreover, it is mandatory to inform the user about MSPL policies generated from the HSPL that conflict with the manually written MSPL.

At M24 only the first two methods are available. It has not been decided yet if the third approach will be implemented during the project lifetime. Indeed, it is a mere re-engineering of basic functionalities provided by SECURED, but the additional reconciliation step makes it less usable for mobility scenarios, also because reconciliation may require explicit user approval in case the user policy is overwritten by policies from other actors higher in the policy stack. Moreover, it does not add to the potential of the research presented here.

3.2.2 Application-driven approach

In the application-driven approach, the user selects the set of PSAs he wants to use to enforce his policy then he defines the related policies. We expect technology-savvy users and security experts to be more

interested in the application-driven than non-experts, as non-experts should find simpler to define their policy and then enforce it through the policy-driven approach.

There are three ways to use the application driven approach, depending on the policy abstraction used to pass the user policy: the HSPL application-driven refinement, if a HSPL policy is passed, MSPL application-driven refinement, if an MSPL policy is passed, and hybrid application-driven refinement, if both HSPL and MSPL are passed as input.

Application-driven refinement – HSPL inputs

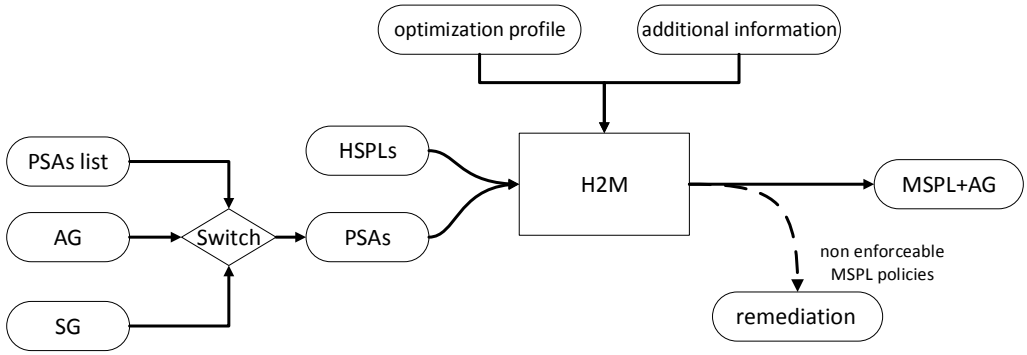

Figure 7: Application-driven refinement – HSPL inputs.

Figure 7 presents the first case, the *HSPL application-driven refinement*. In this case the user specifies his HSPL policy and the PSAs he wants to use to enforce them. PSAs to use can be passed in three alternative forms, either as a PSA list, or an Service Graph, or a Application Graph. Regardless of the inputs, at the end of the process H2M Service makes available the user Application Graph[1], the MSPL, and all the additional user, network, and mapping information needed to complete the correct enforcement.

Since the user provides the PSAs to use, some HSPL policy can be non-enforceable. Therefore, the H2M Service reports users about non-enforceability cases and provides hints for solving them. Moreover, when generating the MSPL, the H2M Service notifies also about complete non-enforceability issues.

- **capability matching**: identification of the capabilities needed to enforce the input HSPL policy;
- **PSA identification** (optional, if a list of PSAs is not passed as input): identification of the different sets of PSAs that own the capabilities needed to enforce the input HSPL policies;
- **non-enforceability analysis**: identification of non-enforceable HSPL policies due to lack of capabilities in the set of PSA passed as input;
- **MSPL generation**: translation of the input HSPL policy into MSPL policies;
- **Application Graph generation** (optional, if the Application Graph is not passed as input): generation of the user Application Graph.

[1] If the Application Graph is passed as input, the H2M Service only copies it.

Application-driven refinement – MSPL inputs

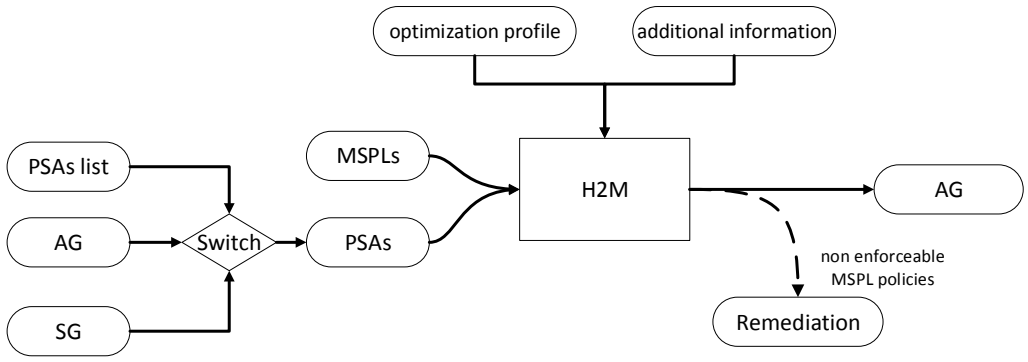

Figure 8: Application-driven refinement – MSPL inputs.

Figure 8 presents the second use case of the application-driven approach, the *MSPL application-driven refinement*. In this case the user specifies his MSPL policy and the PSAs he wants to use to enforce them. PSAs to use can be passed in three alternative forms, either as a PSA list, or a Service Graph, or an Application Graph. Regardless of the inputs, at the end of the process H2M Service makes available the user Application Graph[2], the H2M Service outputs the user Application Graph, the MSPL, and all the additional user, network, and mapping information needed to complete the correct enforcement.

Also in this case, when associating MSPL capabilities to PSAs there may be non-enforceability issues that are reported to the user. Since MSPL is passed as input, the H2M Service does not notify about complete non-enforceability issues.

Application-driven refinement – HSPL and MSPL inputs

In this case, the following basic operations are performed:

- **PSA identification** (optional, if a list of PSAs is not passed as input): identification of the different sets of PSAs that own the capabilities needed to enforce the input MSPL policies;
- **non-enforceability analysis**: identification of non-enforceable MSPLs due to lack of capabilities in the set of PSA passed as input;
- **Application Graph generation** (optional, if the Application Graph is not passed as input): generation of the user Application Graph.

Figure 9 presents the third use case of the application-driven approach, the *hybrid application-driven refinement*. In this case the user specifies both his HSPL and MSPL policies and the PSAs he wants to use to enforce them. The PSAs to use can be passed in three alternative forms, either as a PSA list, or a Service Graph, or an Application Graph. Regardless of the inputs, at the end of the process the H2M Service makes available the user Application Graph[3], the H2M Service outputs the user Application

[2] If the Application Graph is passed as input, the H2M Service only copies it.
[3] If the Application Graph is passed as input, the H2M Service only copies it.

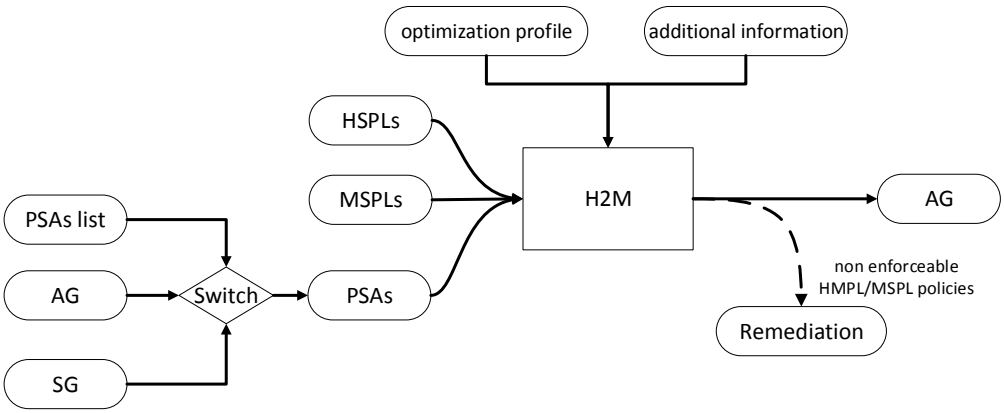

Figure 9: Application-driven refinement – HSPL and MSPL inputs.

Graph, the MSPL, and all the additional user, network, and mapping information needed to complete the correct enforcement.

Since the user provides the PSAs to choose from, some HSPL policy can be non-enforceable. Therefore, H2M Service reports users about non-enforceability cases and provides hints for solving them. Moreover, when generating the MSPL policies, the H2M Service notifies also about complete non-enforceability issues.

In this case, the following basic operations are performed:

- **capability matching**: identification of the different sets of PSAs that own the capabilities needed to enforce the input HSPL and MSPL policies;
- **PSA identification** (optional, if a list of PSAs is not passed as input): identification of the different sets of PSAs that own the capabilities needed to enforce the input MSPL policies;
- **non-enforceability analysis**: identification of non-enforceable HSPL and MSPL policies due to lack of capabilities in the PSAs passed as input;
- **MSPL generation**: translation of the input HSPL policy into MSPL policies;
- **Application Graph generation** (optional, if the Application Graph is not passed as input): generation of the user Application Graph.

As in the policy-driven case that involves both HSPL and MSPL policies, an additional operation is required for this case: the reconciliation. The same considerations apply also in this case.

It is worth noting that, although the user's choice is based on desired capabilities, the capabilities and PSAs that can be selected with the application driven approach also depend on the support by the network provider and/or contractual availability. Let's consider an Internet Service Provider (ISP) that owns a NED. If this NED only supports a predefined set of PSAs, the user's choice is restricted to the set of available PSAs for a specific NED.

3.3 Architecture and workflow

The refinement performed by the H2M Service is divided in four main phases (Figure 10): *Requirements identification*, *PSA Optimization*, *Application Graph generation* and *MSPL generation*. Furthermore, we support two types of non-enforceability analysis: *early non-enforceability* and *complete non-enforceability*.

3.3.1 Requirements identification phase

The Requirements identification phase is composed of two parts: *capability identification* and *PSA identification*. The capability identification process analyses the policies (HSPL and/or MSPL depending on the case) and identifies the required capabilities. On the other hand, the PSA identification process finds the PSAs that support the capabilities needed to enforce the policies.

In order to reduce the solution space, we have defined a set of pruning criterion that reduce the set of PSAs than be used to enforce each capability/policy. For instance, we implemented the pruning criterion that selects the *n* cheapest PSAs with best rating if cost minimization will be used later during the PSA Optimization phase. We also implemented other pruning criteria that consider the business model of PSAs. For example, a PSA can be bought once and used forever or can follow the pay per use paradigm.

In the policy-driven refinement approach, the PSAs used for the optimization are selected either from the entire set of PSAs available in the PSAR or from a subset of them, already prepared by the user, stored into the user repository (they are named the *user PSAs*). In the application-driven refinement approach, the PSAs are selected from the set of selected ones (they are named the *selected PSAs*). In addition in application driven refinement, if the Service Graph is provided in input, the PSA identification phase is performed for each of the Generic PSAs in the Service Graph.

During the PSA identification phase, to produce the combination of PSAs that satisfy the HSPL policies, the process performs the *early enforceability* analysis to check if security capabilities, required by the policies match the ones offered by the available PSAs (user or selected PSAs). This is the capability matching process.

For example (Figure 11), if the user specifies the HSPL "*Alice is not authorized to access Internet age-inappropriate content*", the enforcement of this policy requires to have either one parental control PSA, which owns all the needed capabilities, or three PSAs, one URL filter, one content filter, and one for packet filter. Similarly, when a user specifies an MSPL for a generic layer 4 filter, the set of PSAs to be selected must contain at least one PSA which owns the packet filter capability. If this operation fails, the policies are not enforceable with selected PSAs and the user is warned.

To cope with this issue the refinement process should warn the user providing details on lack of capabilities and help him by suggesting a remediation strategy. For example, *early enforceability* provides suggestions on missing capabilities or other available PSAs.

3.3.2 PSA optimization phase

Considering future trends, the SECURED marketplace could contain a large set of PSAs. For example, different parental control applications with different performance and cost can be provided. Therefore, the refinement process requires an approach to choose among different PSAs. The objective of the optimization phase is to identify which PSAs should be used to enforce a set of policies.

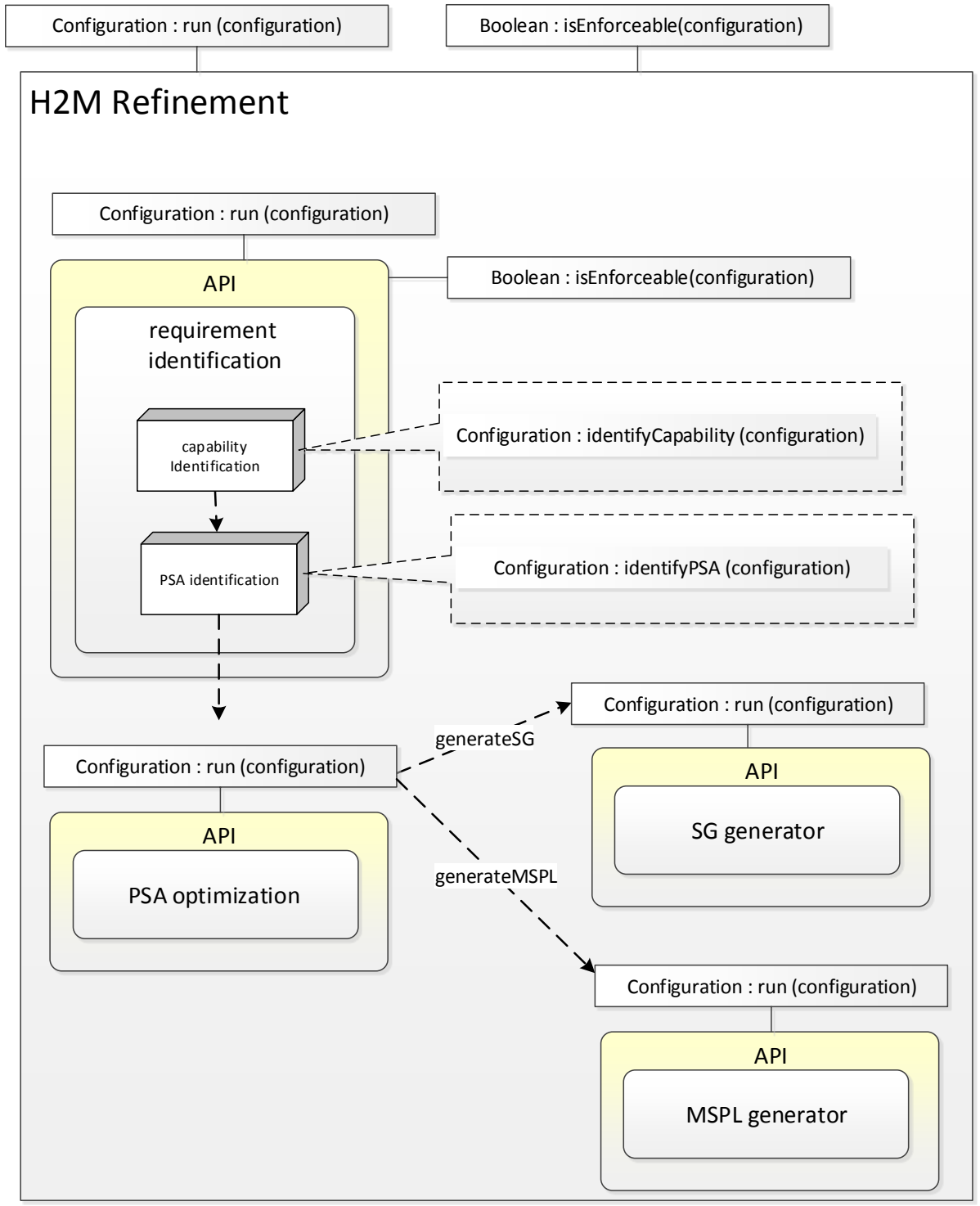

Figure 10: H2M refinement architecture.

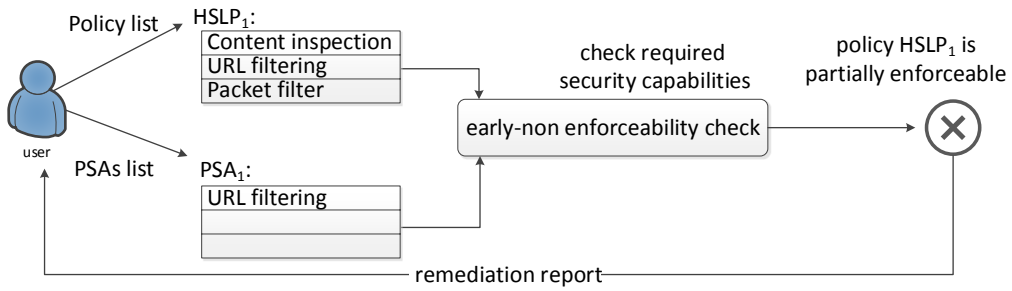

Figure 11: Early enforceability analysis.

First of all, a set of criteria needs to be defined to make a choice. Examples of criteria are the minimization of economic costs, maximization of performance[4]. Moreover, other constraints can be added, like the type of license or the vendor.

In order to simplify the usage of criteria for unskilled users, the user may choose a set of standard profiles that specify a predefined set of criteria (e.g. maximize performance vs. minimize costs). In addition, the PSA aggregation concept ("PSA bundle"), i.e. the grouping of different functionalities in the same application, will also be specified to create complex security services and optimize resource usage at the NED.

Once a profile or a criterion is selected by the user, the refinement process optimizes the chosen PSAs. Before generating the related MSPL policies, other optimizations are possible. For example, the performance of software-based packet filters greatly benefits from rule grouping and reordering (possibly based on statistical traffic analysis).

Model

The objective of the optimization model is to identify which security applications should be used to enforce the high-level policies. As defined before, the refinement process transforms the HSPL into more specific requirements (i.e. security capability) and each PSA enforces one or more capability. The optimization model proposed in this document is a modified version of the set-covering problem [7].

The capabilities set is defined as $\mathbb{C} = \{c_1, \ldots, c_l\}$, where the cardinality is l, i.e. $|\mathbb{C}| = l$. In the set covering notation, the capability set is denoted as the *universe*.

The set of security applications (PSA set) is defined as $\mathbb{P} = \{p_1, \ldots, p_n\}$, where the cardinality is n, i.e. $|\mathbb{P}| = n$.

The function $\psi(p_i)$ is introduced to determine the capabilities supported by a PSA. This function is defined as:

$$\begin{array}{rccc} \psi: & \mathbb{P} & \rightarrow & 2^{\mathbb{C}} \\ & p & \mapsto & C_p \subseteq \mathbb{C} \end{array}$$

Hence, ψ identifies the capabilities of each PSA.

[4]The optimization of the performance may consider data from different like CPU usage, RAM, network throughput, network delay

To optimize the choice of the security applications a set of metrics is proposed. In particular, we assume that there are m metrics $\mu_1, \ldots, \mu_m$. Formally, a metric μ_i is a function that associates for each security application p a real value r and it is defined as:

$$\begin{array}{rccc} \mu_i: & \mathbb{P} & \rightarrow & \mathbb{R} \\ & p & \mapsto & r \end{array}$$

Several metrics can be considered to evaluate a PSA. We will use a generic function c (named *cost*) that identifies the aggregated cost (a real number) obtained by linear combination of the values of the metrics μ_i computer over a security application p. More formally:

$$\begin{array}{rccc} c: & \mathbb{P} & \rightarrow & \mathbb{R} \\ & p & \mapsto & \sum_{i=1}^{m} a_i \mu_i(p) \end{array}$$

where a_i represents the weight of the metric μ_i.

In general, we need to use more than one PSA to implement an HSPL policy, that is, the optimization process will select a subsets S of $\mathbb{P}$ ($S \subseteq \mathbb{P}$). The function Ψ is therefore introduced to map set of PSAs to the capabilities they provide. Ψ is defined as:

$$\begin{array}{rccc} \Psi: & 2^{\mathbb{P}} & \rightarrow & \mathbb{C} \\ & S = \{p_m, p_n, \} & \mapsto & \bigcup_{p \in S} \psi(p) \subseteq \mathbb{C} \end{array}$$

Therefore, the general optimization problem we have to solve to find the optimal set of PSAs is defined as:

$\min \sum_{i=1}^{n} c(p_i) x_i$	target function i.e. minimize the cost associated to the selected PSAs
constrained to	
$\Psi(S) = C^H \subseteq \mathbb{C}$	ensures that all the capabilities needed by the HSPL policy are available
$x_i \in \{0,1\}$	every PSA is either in the cover or not

This type of problem is often addressed by using Integer Linear Program (ILP) solvers. However, in order to evaluate two or more target functions at the same time, the MOEA multi-objective framework is adopted [6].

Target functions

We have defined the following (atomic) target functions: `minBuyCost`, `minLatency`, `minTransferCost` and `min[Cost/Rating]`.

The objective of `minBuyCost` minimizes the economic cost associated to a set of PSAs. The cost function c sums up the value of the metric that expresses the cost of each PSA (p_i) as a real value.

`minLatency` minimizes the delay introduced by traffic processing. Currently, a metric has been defined to express the delay to internally process one Mega bit of traffic of each PSA (p_i). As a future work, other metrics can be added for example to consider network delays.

The goal of `minTransferCost` is to minimize the economic cost of transferring a particular volume of traffic. The cost function c sums up the values of a metric that expresses the cost of each PSA (p_i) for processing one Mega bit of traffic. Again, as a future work, other metrics can be added for example to consider network transferring cost for different types of service.

Finally, `min[Cost/Rating]` minimizes the ratio between the cost of a PSA and its rating. This rating, associated to a PSA, collected into SECURED catalogue/store, can be expressed by other users and experts. This rating is expressed by following a set of metrics, for example: number of downloads, number of satisfied users, etc. Therefore, the target function c is composed by a metric that compares the economic cost and the estimation of each PSA (p_i).

Profiles

As introduced before, an optimization profile groups together a set of atomic criteria. A typical optimization problem is defined as a trade-off among competing objectives, also known as multi-objective optimization. Therefore a profile defines the set of competing objectives to identify the optimal trade-off.

In the scope of SECURED, a profile defines how to select the set of PSAs to enforce the set of HSPL policies. To this purpose, the following profiles are proposed:

- trade off the cost of PSAs and their network latency (`minBuyCostminLatency`)
- trade off the cost of PSAs and their rating (`min[Cost/Rating]`)
- the traffic cost of a PSAs and their network latency (`minTransferCostminLatency`)

The first profile (`minBuyCostminLatency`) is useful when a user wants buy the PSAs to enforce his policies. It is composed by two objectives: 1) minimize the buying cost of PSAs; 2) minimize the network latency. Typically maximizing performance requires expensive PSAs, therefore this profile optimizes the trade-off among cost and network performance.

The second profile (`min[Cost/Rating]`) is useful to achieve a trade-off between the cost of a PSA and its quality, expressed by means of a rating.

The last profile (`minTransferCostminLatency`) is suitable for a pay per use cost model, where the cost is calculated by considering the network traffic transferred among PSAs. Therefore, the first objective minimizes the transfer cost, i.e. it selects the PSAs with lower cost, while the second minimizes the network latency, i.e. maximizing the performance.

It is worth noting that each atomic criterion can be considered as a basic block. Therefore the creation of a new profile is simple and straightforward as a new profile can be created grouping together existing criteria, or grouping a new set of criteria.

3.3.3 Application Graph generation phase

If the policy-driven approach is used or if the application-driven approach is invoked passing a list of PSA, the Application Graph generation phase creates the Application Graph. If the application-driven

approach is invoked passing an Service Graph in input, the Application Graph generation substitutes all generic PSAs with the actual PSAs selected with the PSA Optimization phase.

At M24, we implemented a simplified version of the Application Graph generation phase.

The Application Graph generation phase takes as input the PSAs selected by the PSA Optimization phase, and if available the Service Graph. If the Service Graph is passed as input, this phase simply substitutes the generic PSAs with the selected ones.

If the Service Graph is not available in input, the PSAs are passed as a list and we determine the optimal order of processing based on two distinct information: mandatory dependencies and best practice dependencies. These dependencies are built on capabilities / generic devices, not on actual PSAs.

Mandatory dependencies are a set of constraints that describe ordered pairs of Generic PSAs that are forbidden. These arrangements are forbidden because if the first PSA in the pair would process the traffic before the second PSA in pair, the second PSA in the pair would not be able to enforce its policy. For instance, a PSA that performs channel protection and ciphers the content renders useless the operations of a downstream PSA performing content inspection.

Best practice dependencies suggest the best order to place PSAs whose capabilities are that are not forbidden by mandatory dependencies. Even in this case, best practice dependencies are represented as pairs of Generic PSAs. The best order is determined by statistical information on the time required to process packets or known best practice. For instance, it is known that packet filters have better performance than application layer filters that have to decapsulate more layers and perform more complex operations to determine the traffic to deny (like using regex to determine URLs to filter). It is therefore suggested to place packet filters before application layer filters, as packet filters can already reduce the traffic arriving at the application layer ones.

We will define the most complex and general case of Application Graph generation during the third year of the project. In the most complex and general case, PSA information is complemented with the MSPL policies associated to all the capabilities provided by the selected PSAs. MSPL can be used to determine all the classes of traffic that are processed by the PSAS in the most precise way. Indeed, by analysing all the conditions in the MSPL policy, it is possible to know what are the packets that will be processed by a PSA. It is therefore possible determining the minimum number of PSAs that have to be traversed by each packet, that is, the per-packet PSA path. The per-packet PSA paths are later generated based on mandatory and best practice dependencies. It is an exploitation task to improve this Application Graph generation phase.

3.3.4 MSPL generation phase

During the MSPL generation phase, the HSPL concepts are expanded into medium-level ones (e.g. the black list in the actual set of URLs or IP addresses; the service into ports). HSPL conditions are transformed into MSPL conditions while the HSPL action fields determine the action to enforce.

There are several high-level concepts into HSPL policies, like “phishing sites”, “blacklisted sites”, that need to be expanded to be used at MSPL level, where, for example, application layer filters process URLs. Therefore, this process needs some auxiliary files that contain information on how to map high-level concepts to low level configuration parameters like IP address, Port, URLs, cipher algorithm, key or certificate path on a per user basis. These files are organised based on the high-level concept they expand. For the M24 prototype, these files have been defined by the SECURED consortium. Every user is theoretically authorized to change these files and determine in a more precise way the meaning. However, we expect that only the expert users could be really interested in customising them.

```
rule "Rule ID"
when
  //conditions (query language)
then
  //actions (java)
end
```

Figure 12: Structure of the Drools language.

```
rule "Early non-enforceability"
when
c : Capability()
h : Hspl( h.getCapabilities().contains(c))
not ( p : SuitablePSA(p.getCapabilities().contains(c) )

then
h.setEnforceability(false);

end
```

Figure 13: Rule for early non-enforceability.

During the MSPL generation, other cases of non-enforceability may appear (*complete non-enforceability*). For example, when a policy requires to inspect the content of a HTTP protocol field, but the selected PSA does not support this feature, the policy is not enforceable. Similarly, when it does not support a particular option, the policy is partially enforceable. Let us consider a parental control scenario to protect children access to Internet, where applications with different features are available. Two distinct PSAs, PSA_1 and PSA_2 are available, both capable of enforcing a parental control policy but with different functions. PSA_1 includes a "application content inspection" function and a "URL filtering" function. PSA_2 supports the functions of PSA_1 and a feature to specify time-based policies for "URL filtering". Hence, if a user wants to specify that access to Facebook web site is permitted only after dinner from 20 pm to 22 pm PSA_1, he is not allowed to enforce that policy. Therefore, the *MSPL generation phase* produces a *complete non-enforceability report* where all types of enforceability errors/issues are shown to the user.

3.4 Implementation

The H2M process has been implemented in Java 1.7. It uses two open source framework Drools [1], and MOEA. Drools is a Rule Based Systems (RBS) based around the Rete algorith [2]. Figure 12 shows the simple structure of the Drools language. The rule engine has been used in the Requirement identification phase by the capability matching and in the PSA identification operations. Figure 13 contains an example of rules used in the Requirement identification phase. In detail, it checks if there exists a PSA that support the capability required by the HSPL policy as required during the early non-enforceability analysis.

The PSA Optimization phase, as introduced before, uses MOEA to perform multi-objective optimization. With MOEA, it is simple to model new types of optimization problems by extending the class

`Abstract-Problem`. In particular, a new problem class must contains the definition of two methods:

- `public void evaluate(Solution solution)` that contains the definition of the model (i.e. optimization variables (e.g. types, constraints) and the target function(s);
- `public Solution newSolution()` that creates the `Solution` object populated by reading the variables values.

Each optimization profile is implemented by introducing a new problem, i.e. by extending the `Abstract-Problem` class. However, there are several parts of the models that can be reused. First of all, all profiles share the model part. Moreover, some target functions may shared by several profiles, like the `minLatency` and `minBuyCost` target functions, which are shared by the (`minBuyCostminLatency` and `minTransferCostminLatency`) profiles.

4 MSPL to low-level configuration

In SECURED, an abstract security configuration (MSPL) must be translated into a specific security configuration for a specific PSA. Since a concrete configuration depends on the settings used by each PSA (e.g. two packet filters offer the same security capability but may be configured via two completely different languages) a translation operation is required.

This is performed by invoking the M2L module for each PSA. More in details, the coordinator transforms the statements expressed in MSPL to configuration settings required by the PSA.

This process is simpler than the previous one (transformation from HSPL to MSPL) because it uses a "syntax mapping" approach to translate a MSPL configuration into a specific language without adding any information.

4.1 Architecture and workflow

As introduced before, once the MSPL is generated by the reconciliation process, it must be translated into a low-level settings for a specific PSA. To support a wide set of low-level security controls, the translation must be designed to support different languages (e.g. netfilter/iptables or PF for a stateful firewall). The proposed approach develops the M2L translation module (with related API) that takes as input MSPL statements, identifies the set of security controls of the PSA and invokes the related *M2L plugin(s)* to generate the specific configuration. Each *M2L plugin* must contain the logic to perform the translation for a specific security control implementation.

Figure 14 sketches the architecture of the M2L translation module that contains the core element, i.e. *M2L service coordinator*. The goal of this service is to: i) identify the set of security controls to configure (analysing the MSPL) and ii) generate the configuration settings for a PSA (specified by passing the related identifier). The coordinator is invoked by the Online Workflow Manager by using the related API. To guarantee flexibility and scalability, the translation logic cannot be defined into this service. Since each PSA may have more than one security control, the *M2L service coordinator*, for each security control, contacts the *M2L service* to perform the translation by selecting and invoking specific *M2L plugin(s)*. Each plugin is stored and retrieved from the *M2L plugin repository*. For example, when MSPL statements must be translated into netfilter/iptables format, the related plugin is downloaded from the repository and invoked to generate the real configuration. This configuration is managed by the *M2L service coordinator* (e.g. bundled with other security control configuration of the PSA) that returns the configuration settings for the PSA.

The external API, offered by *M2L service coordinator*, exposes the following operations:

- **translate** that takes as input: the MSPL (given by using XML document) the PSA id (an identifier defined by a string), i.e. the reference of a PSA. Then the service analyses the MSPL, identifies the related security controls of the PSA and contacts the *M2L service* (for each security control) to perform the translation. Then it merges the configuration of every security controls and returns the *PSAConfiguration*;
- **isConfigurationSetChanged** that takes as input the information about the PSA configuration and returns a Boolean value. It is useful to check when a PSA configuration is changed.

The external API, offered by *M2L service*, exposes the following operation:

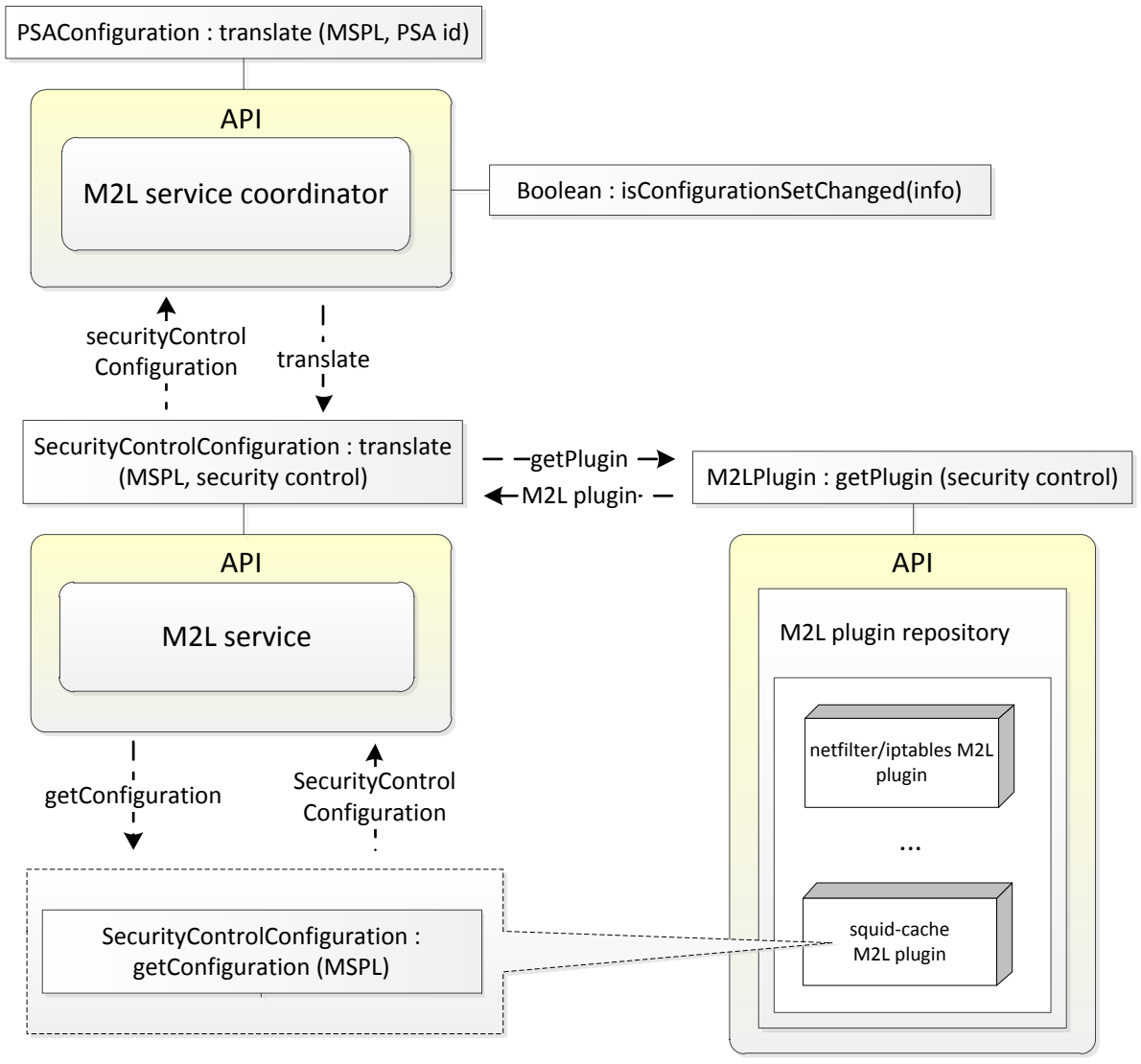

Figure 14: M2L architecture.

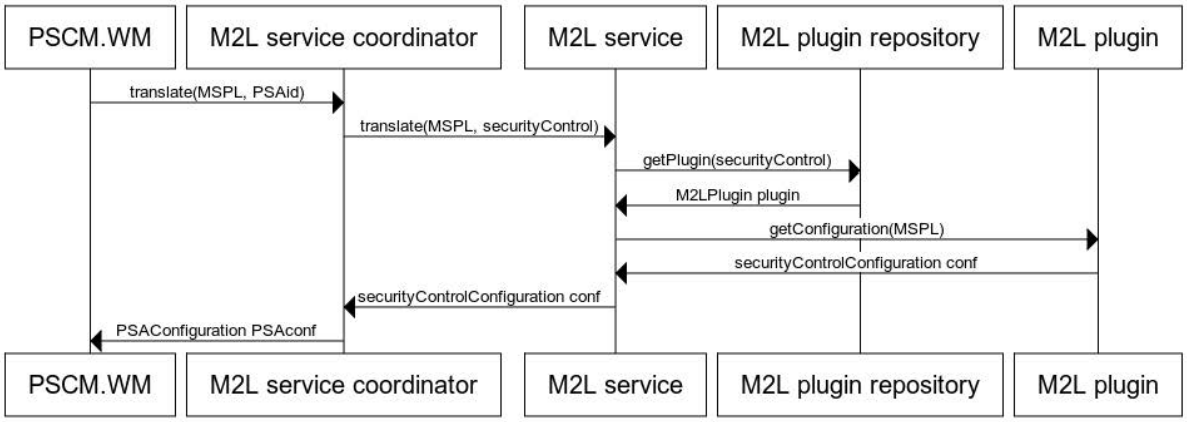

Figure 15: M2L translation workflow.

- **translate** that takes as input: the MSPL (given by using XML document) and the target security control (a string). Then the service contacts the *plugin repository* and downloads the *M2LPlugin* to perform the translation from MSPL to configuration settings for the security control. Then, M2L service returns the *SecurityControlConfiguration*.

Therefore, the API of M2L plugin repository offers an interface with the following operation:

- **getPlugin** that takes as input the type of security control and returns the *M2LPlugin* to perform the translation. The available plugins are currently stored on file system and indexed by using a XML file that contains the *M2L plugin* identifier, its name and the path of the plugin.

A *M2L plugin* is designed to be compact and efficient, hence it offers a single interface with the following operation:

- **getConfiguration** that takes as input the MSPL (given by file) and returns the generated configuration as a file (*SecurityControlConfiguration*).

The workflow, i.e. how M2L translation module works, is depicted in Figure 15. More in detail, the M2L workflow follows these steps:

- the Online Workflow Manager performs a translation request by invoking the **translate** method of *M2L service coordinator* and passing the XML document (MSPL) and the PSA id (PSAid);
- the *M2L service coordinator* performs a translation request by invoking the **translate** method and passing the XML document (MSPL) and the type of security control;
- the *M2L service* stores the XML document into a file, then contacts the *plugin repository* by using **getPlugin** method to select the corresponding M2L plugin and to download it. Then the *plugin* is stored locally;
- the *M2L service* invokes the plugin by using the **getConfiguration** method and passing the XML document (MSPL);
- the plugin returns the configuration (*securityControlConfiguration*) to *M2L service*;

- the *M2L service* returns the configuration (*conf*) to *M2L service coordinator*;
- finally, the *M2L service coordinator* returns the PSA configuration (*PSAconf*) settings to Online Workflow Manager.

4.2 Implementation

The translation service is organized in four modules: **M2LServiceCoordinator**, **M2LPlugin**, **M2LPluginService**, and **M2LService**, where each module is implemented as a Java project.
In particular, the M2LServiceCoordinator interacts with the M2LService (to get the configuration) by using a REST service. The M2LServiceCoordinator exposes an API based on the following methods:

`POST /translate/{PSAid}`
where:

- the inputs are a XML Document and a PSA id;
- the output is a PSA configuration file after translation.

`POST /isConfigurationSetChanged/{info}`
where:

- the input is the information of a configuration;
- the output is a Boolean value.

The M2LService exposes an API based on the following method:

`POST /translate/{securityControl}`
where:

- the inputs are a XML Document and a string to express the type of security control;
- the output is the configuration file after translation.

Then, the M2LService contacts the M2LPluginService by using a REST service to get the corresponding plugin. The method exposed by M2LPluginService is:

`GET /getplugin/{securityControl}`
where:

- the input is a string to express the type of security control;
- the output is the plugin JAR file.

Finally, each plugin is developed by customizing the general project **M2LPlugin**. In particular, each plugin must contain the default method:

```
int getConfiguration( String MSPLFileName, String securityControlFileName )
```

where:

- `MSPLFileName` is the file that contains MSPL (a serialized XML document), i.e. the input;
- `securityControlFileName` is the file that contains the low-level configuration for the control, i.e. the output.

This method is automatically invoked locally by the M2LService during the translation. Then the customized plugin must be exported by using the JAR format and imported into the M2LPluginService.

4.3 Integration in SECURED

The integration of the M2L translation process in the SECURED workflow is depicted in Figure 16. In particular:

- the *Online Workflow Manager* is the software that invokes the translation service;
- the *M2L service coordinator* and the *M2L service* are integrated into SPM component;
- the *plugin service repository* is deployed as external service.

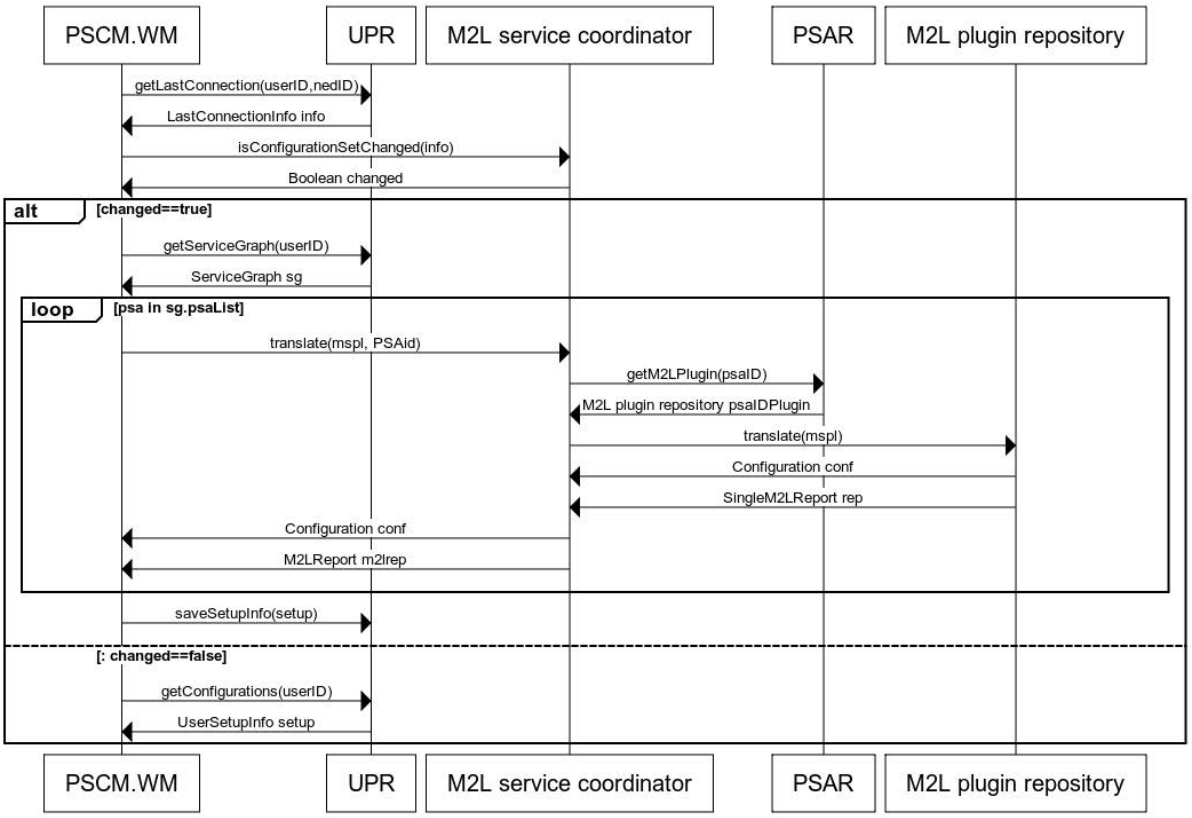

Figure 16: Integration of M2L in the SECURED workflow.

List of abbreviations

AG	Application Graph
ACL	Access Control List
API	Application Programming Interface
GUI	Graphical User Inferface
H2M	High to Medium level refinement
HSPL	High-level Security Policy Language
HTTP	Hyperte xt Transfer Protocol
ISP	Internet Service Provider
IT	Information Technology
M2L	Medium to Low level translation
MSPL	Medium-level Security Policy Language
NED	Network Edge Device
PCIM	Policy Core Information Model
PSC	Personal Security Controller
PSCM	Personal Security Controller Manger
PSCM.WM	Personal Security Controller Manger, Workflow Manager component
PSA	Personal Security Application
PSAR	Personal Security Application Repository
SECURED	SECURity at the network EDge
SG	Service Graph
SPM	Security Policy Manger
URL	Uniform Resource Locator
UPR	User Policy Repository
WFM	Work-flow Manger
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

References

[1] M.Proctor, "Drools: A Rule Engine for Complex Event Processing", *AGTIVE2011: 4th International Conference on Applications of Graph Transformations with Industrial Relevance*, Budapest (Hungary), October 4–7, 2011, p.2, doi:10.1007/978-3-642-34176-2_2

[2] C.L.Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem", *Artificial Intelligence*, Vol.19, No.1, 1982, pp.17-37, doi:10.1016/0004-3702(82)90020-0

[3] J.Strassner, "DEN-ng: achieving business-driven network management", NOMS2002: Network Operations and Management Symposium, Firenze (Italy), April 15-19, 2002, pp.753-761, doi:10.1109/NOMS.2002.1015622

[4] M.Bishop, S.Peisert, "Your security policy is *What*??", UC Davis: College of Engineering, 2006, http://web.cs.ucdavis.edu/~peisert/research/security-policy-report.pdf

[5] J.Weise, C.R.Martin, "Developing a security policy", *SUN blueprints*, December 2001, 14 pages, http://www.smps.reading.ac.uk/~swsellis/tech/solaris/performance/doc/blueprints/1201/secpolicy.pdf

[6] The MOEA framework (version 2.5), http://www.moeaframework.org/

[7] T. H. Cormen, "Introduction to Algorithms, Second Edition", *MIT Press*, 2001, ISBN 0-262-03293-7