D5.2

Specification of PSA management service and repository

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	D5.2
Version (date)	v1.2 (24/10/2014)
Work package(s)	WP5
Due date	30/09/2014 – M12

Responsible organisation	TID
Editor	Antonio Pastor
Dissemination level	**PU** - Public

Abstract	This document describes the PSA Manager (PSAM) and the PSA Repository (PSAR) systems. It includes a full description of the architecture components, interactions and PSA management and repository processes with the rest elements of the SECURED. Finally there is a a detailed specification of the PSAM and PSAR APIs and a guideline for the PSAM and PASR development process.
Keywords	PSAM, PSAR, specification, interface, API

Editor
Antonio Pastor (TID)

Reviewers
Cataldo Basile (POLITO)
Francesco Ciaccia (BSC)
Antonio Lioy (POLITO) – quality control

Contributors
Antonio Pastor (TID)
Diego Lopez (TID)
Adrian L. Shaw (HPLB)
Ludovic Jacquin (HPLB)

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	7.10.2014	First version	Antonio Pastor
v1.1	8.10.2014	Revised version	Cataldo Basile
v1.2	24.10.2014	Second revision and quality control	Antonio Lioy

Executive Summary

This document defines the general architecture of the PSA Repository and PSA Manager services (PSAR and PSAM, respectively), which provide storage for the SECURED Personal Security Applications (PSAs) and offer access to the repositories to users, managers and developers through the appropriate interfaces. This document identifies several deployment choices, named “ecosystems”, to support the stakeholders in selecting the best option for a PSAM/PSAR deployment, and also shows OpenStack as a feasible reference implementation to deploy PSAM and PSAR services.

The PSAM is composed by a User Portal and a Onboarding Process. The User Portal offers a Web Portal plus an API front-end to interact with the users (and other management components). It allows end-users and management system to subscribe PSAs. The Onboarding process offers an additional API to allow incorporating new PSAs in SECURED, that can be added by PSA developers or management systems.

The PSAR supports PSAM as a storage and content server where PSAs and all the related information is stored. PSAR includes several managed repositories and a back-end API to manage a programmatic request/response protocol. The PSAR can be deployed in a distributed environment to improve the delivery of large PSAs and support high availability or mobility scenarios.

End-users, PSA developers and Management systems are the main actors interacting with PSAM to upload or subscribe PSAs. On the contrary, other SECURED entities like the NED, User Profile Repository (UPR) or the Security Policy Manager (SPM) can contact directly the PSAR as an auxiliary server to obtain data related to a PSA.

The API included in this specification is RESTful and permits several actions, the most relevant ones being subscribe or remove PSAs from user preferences, add or delete PSAs from the SECURED system, activate or block PSA availability, or download PSA, PSA manifest, and M2L plugin to external entities.

Contents

1 Introduction

The SECURED project aims to protect end-users from network threats by moving security controls from the end devices to a programmable network device called NED (Network Edge Device). In this way end users could potentially do without any type of security applications on their devices while being protected against possible attacks. This shift of the security functions from the end devices to the edge of the network is achieved through the creation of Personal Security Applications (PSAs) that run on the NED.

To manage the PSAs, SECURED defines two services, covered in this deliverable: the PSA Repository (PSAR) and PSA Management (PSAM) services. The PSAR is a service used for storing in a structured way the PSAs available in a certain domain. On the other hand, PSAM is a service used for managing the PSAR and providing also user and programmatic interfaces to handle the PSAs.

The PSA specification is provided in D5.1 [3] but the project supports developers also by providing specifications for development environments as well as several support services needed for deploying the applications: here is where the PSAR and PSAM services come into play.

The rest of this document is organized as follows: Section 2 presents the high-level architecture of PSAM/PSAR, Section 3 describes the processes and workflows that involve these components, Section 4 defines the REST APIs to interface with PSAM/PSAR, Section 5 provides the requirements for implementing the user portal, and finally Section 6 offers guidelines about PSAM/PSAR implementation, including a possible mapping onto the widely used open-source OpenStack framework.

2 General architecture

Figure 1 shows the subset of the general SECURED architecture – presented in D2.3.1 [1] – that focuses on the services offered by PSAM and PSAR.

As shown in the figure, PSAM and PSAR interact with other modules that are part of SECURED or external to it. These entities are not part of this specification, but described in D2.3.1:

- the Authentication module is in charge of authenticating the access to all the modules with the user credentials and interacts with both PSAM and PSAR;
- the User Profile Repository (UPR) stores the User Profile and Service Graph and PSAM and PSAR will exchange information with the UPR about the PSAs and end user credentials;
- the BSS/OSS/Orchestrator represents the set of existing entities in ISP, network operators or big corporation networks (Business Support System, Operative Support System, or Network Orchestrator) that organize and manage the network nodes. These entities use a business-to-business (B2B) communication model with PSAM for their activities.
- the NED (network Edge Device) is the entity where the PSAs are deployed and executed. The NED will contact directly the PSAR to obtain the PSAs. Details of the NED internal structure and operations are available in D3.1.1 [2].
- the SPM (Security Policy Manager) is the service where the PSA selection may be executed based on policy decision, instead of direct user selection. The SPM may retrieve PSA information from the PSAR to satisfy specific security policies.

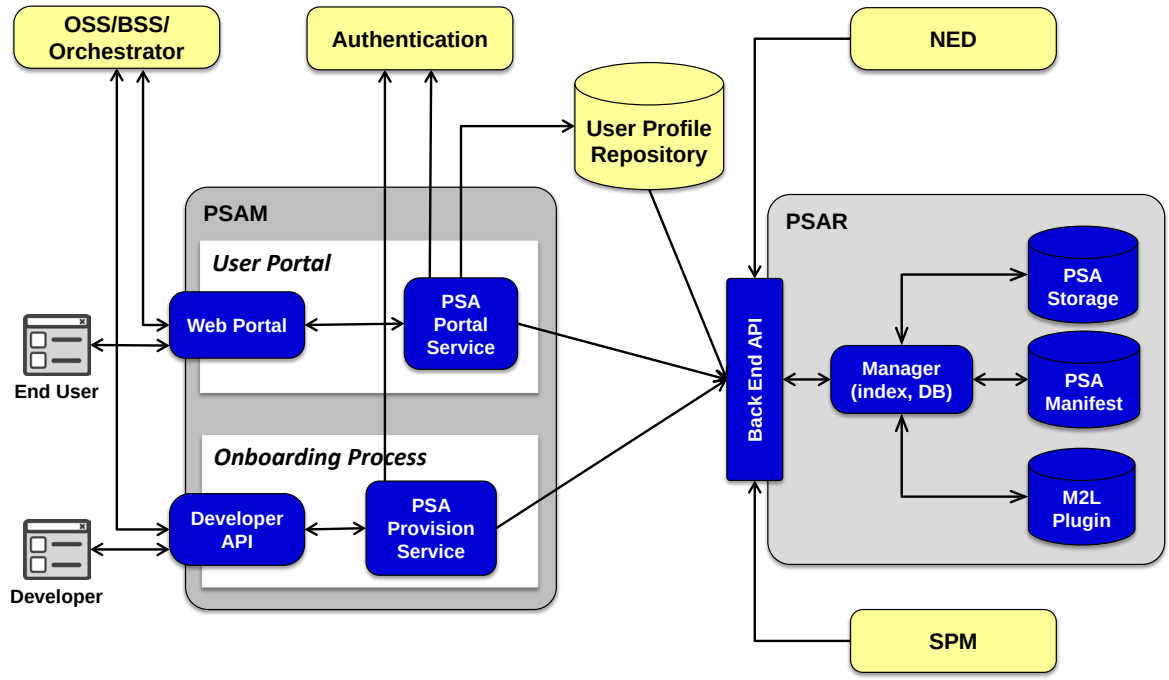

Figure 1: PSAM and PSAR architecture and interactions.

Apart from these entities, also some actors appear in Figure 1:

- End Users – they access the PSAM Web Portal to manage their PSAs;
- PSA developers – they access directly, or through some development framework, the system to manage their developed PSAs.

The network environment and the administration domain where all this entities resides and communicate among themselves can be referred as the implementation ecosystem. Section 6.1 includes detailed information about possible alternatives.

2.1 The PSA Manager (PSAM)

The PSAM is the single entry point for the actors. Figure 1 show the two sub-entities in PSAM that offer external access:

- User Portal for user-related activities;
- Onboarding Process for PSA management.

2.1.1 User Portal

The User Portal is a user-oriented sub-entity that allows an End User to manage the SECURED application-driven service, including the main tasks: subscribing and configuring PSAs. More details about the application-driven concept can be found in D2.3.1.

This module involves two components:

- the Web Portal provides a friendly user interface for interaction with the end-user. It is known as the presentation tier of a three-tier architecture and the usual design includes also an API service for automatic and massive provision of PSAs.
- the PSA Portal Service is the component responsible for executing the internal logic of the Web Portal. It is the business or logic tier in a three-tier architecture. As part of this logic there are also auxiliary processes, like read or write a selected PSA in the User Profile Repository, or the authentication procedure.

The last tier in this three-tier architecture, data tier, is not part of the User Portal, instead is moved to PSAR and explained in Section 2.2.

Specification of the User Portal can be found in Section 5.

2.1.2 Onboarding Process

Onboarding Process is a developer-oriented sub-entity where SECURED managers and developers can upload PSAs. This service consists of two modules:

- the Developer API is a front-end for developers and managers. It is a sort of presentation tier of a three-tier architecture, offering only a REST API that provides an interface to upload and validate PSAs. This API could be integrated in a development framework or be invoked directly.
- the PSA Provision Service is the back-end of the Onboarding process, executing the logic related to the PSA validation, testing and publishing.

Again the data tier is not part of the Onboarding process, but is implemented by the PSAR.

2.2 The PSA Repository (PSAR)

The PSAR is THE software component used for storing all PSA-related information in a structured way. PSAR can be deployed in a distributed or centralized environment.

The PSAR consists of three storage modules (respectively for the PSA manifests, PSA binaries, and M2L Plugins), a manager module and the Back End API, which receives requests from the PSAM, forwards them to the manager and send responses to the PSAM.

The analogy with a three-tier architecture is straightforward from Figure 1:

- Presentation tier = Back End API;
- Logic tier = Manager;
- Data tier = PSA Storage, PSA Manifest storage and M2L Plugin storage.

2.2.1 Backend API

The Back End API is a webservice API to request the execution of specific tasks by the PSAR. The request is translated into the internal logic of the Manager. Figure 1 shows the main interactions of the Back End API:

- interactions with PSAM to support PSAM as data tier;
- interactions with NED to support the delivery of PSAs;
- interactions with SPM to support the policy-driven model[1] (defined in D2.3.1);
- interactions with UPR to support user profile maintenance.

Detailed explanation of these interactions is provided in Section 3.

2.2.2 Manager

The Manager executes the internal logic of the PSAR. The Manager indexes, locates, and updates each of the data repositories modules: the images (PSA Storage), manifests (PSA Manifest Storage) and M2L plugins.

In order to deliver correct and timely answers to the rest of the SECURED elements, the Manager needs to keep a clear reference of where is any piece of information. Technical implementations on how this is done can vary from simple files to complex object-oriented or relational databases. An agnostic entity-relationship model is specified in Figure 2 to allow a PSAR developer to create a valid Manager reference model, based on his preferred implementation model.

In general, this model orbits around the PSA entity and relations with entities associated with the M2L Plugin and PSA storage modules. Also it includes the PSA status to know when a PSA is ready and open to be used by users or when PSA is in maintenance, upgrade or any non-operational status. Finally this model allows finding the location of the storage modules. One implementation would be using the location based on a URI (Universal Resource Identifier):

- HTTP web resource = `http://[host]/path/file`
- FTP resource: =`ftp://[host]/path/file`
- local file = `file:/absolute/path/to/file`

The PSAR can be implemented in a distributed fashion as depicted in Figure 3 (where PoP stands for Point-of-Presence, a SECURED-enabled network attachment point).

The PSA Local Repository (PSALR) is similar to a standard PSAR but includes additional capabilities in its Back End API module that allow it to synchronize its content with the Central PSAR.

The Manager module of the PSALR regularly checks for changes in PSA contents. If a new PSA image, manifest, or M2L plugin is detected, then it is downloaded and stored in the appropriate local storage module.

This way of deploying the PSAR aims to promote three important aspects: mobility, high capacity and high availability. As we can see in the figure, we have a Central PSAR as the parent of multiple PSALRs. In this way we can avoid network overload and large delays when downloading a large file to the NED from the central repository if it is available locally in the PSALR. Also this configuration offers a higher availability by having several distributed copies of the PSA. This is specially useful for the mobility use case when we need to offer PSAs to two different NEDs (the outgoing and the incoming ones).

[1] The policy–driven approach is when the user defines the high–level policies (HSPL) and lets the SECURED infrastructure is in charge of translating these policies to a set of PSAs and their corresponding configuration

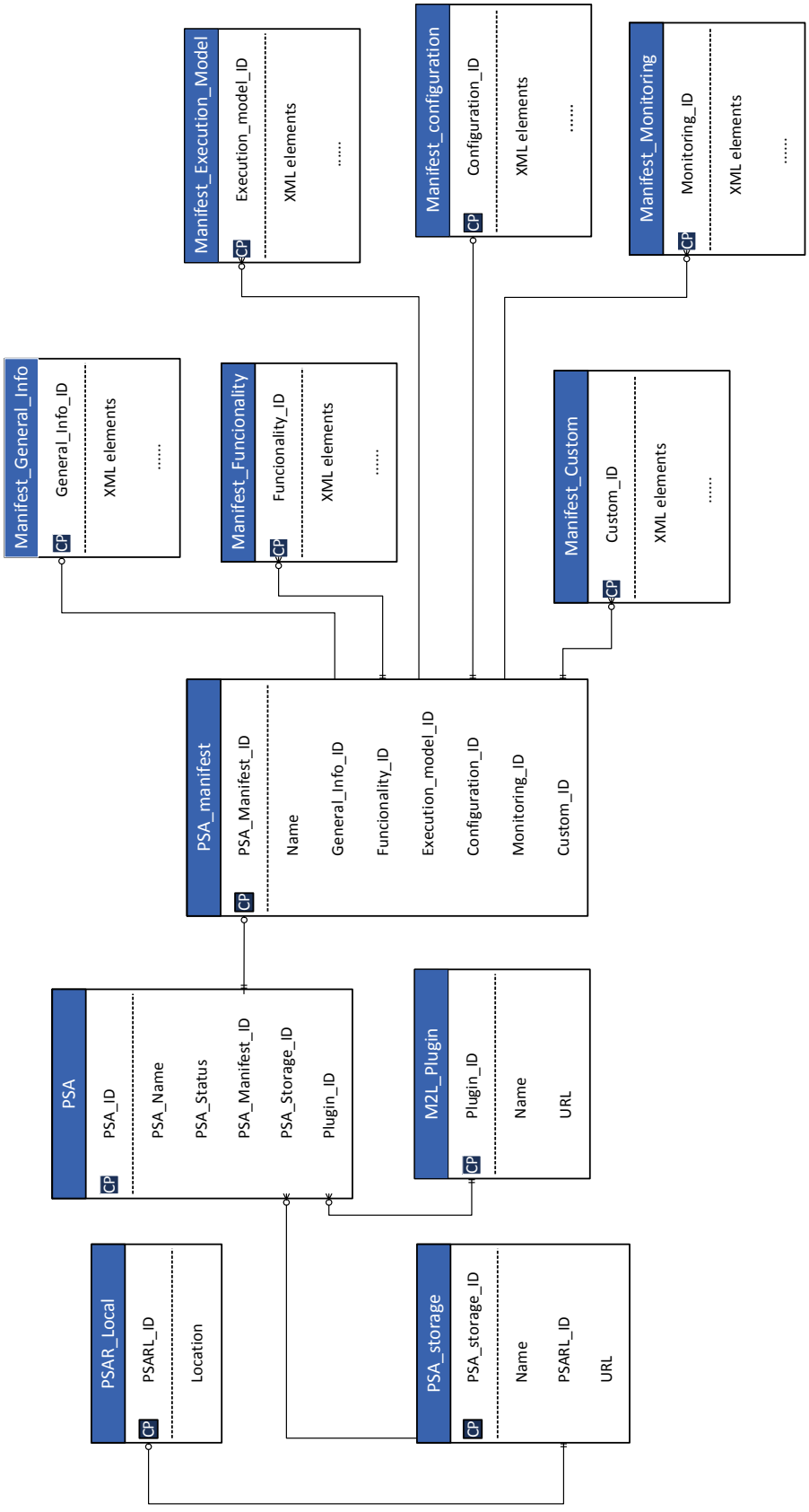

Figure 2: Entity relationship model for the PSAR Manager.

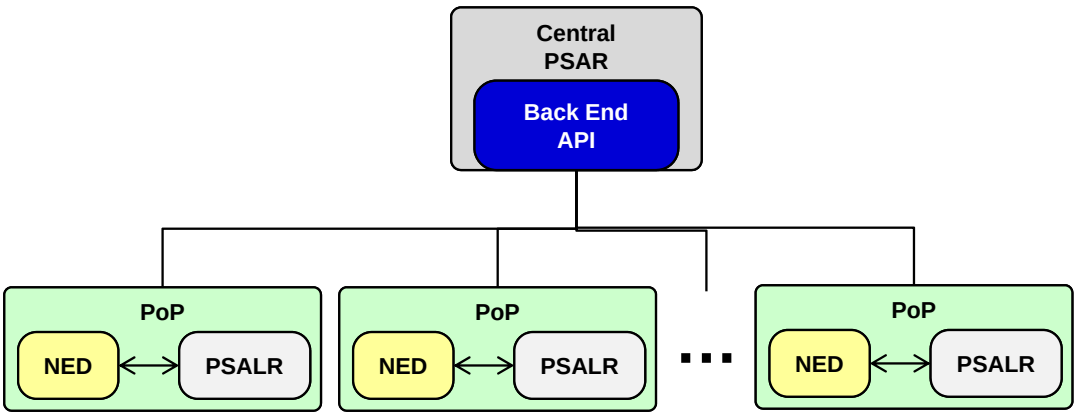

Figure 3: PSAR distributed model

2.2.3 PSA Storage

PSA Storage is a centralized or distributed repository of PSA images that stores the binaries needed for execution of the PSA itself. Depending on the model, PSA image can vary from a full virtual machine image to a simple application binary. Details of possible implementations of PSA are given in D5.1. One example of a PSA Storage could be a file server with high capacity to store the images.

2.2.4 PSA Manifest Storage

PSA Manifest Storage is in charge of storing the PSA Manifest file. The nature of the data to store shape the design of the PSA Manifest Storage to be a structured object database, but not exclusively. An example is an XML database. Details of the PSA Manifest structured format and content are available in D5.1.

2.2.5 M2L Plugin Storage

Conceptually similar to PSA Storage, M2L Plugin Storage is a centralized or distributed repository of executables or scripts. One example could be a file server with high capacity to store the modules that perform the M2L translation to specific configurations.

3 Processes and workflows

Each step and interaction between actors and entities described in Section 2 is clustered in a business process. This section specifies existing business processes where each one encompasses several workflows between various entities and their internal logic. Developers must understand each process and develop the internal logic for each entity to execute the workflows. The processes are arranged in use cases to simplify the model:

End user provision. This use case includes the actions done by an end user who wants to access a Web Portal to search for and subscribe to PSAs.

use case	*group*	*business processes*	*access interface*
end user provision	user-related interactions	User portal (app-driven approach): - retrieve PSA information - subscribe PSA - configure PSA - cancel PSA	browser
BSS/OSS provision	user-related interactions	B2B provision of PSA: - retrieve PSA information - subscribe PSA - configure PSA - cancel PSA	API
open marketplace publishing	onboarding PSAs	Developer service: - publish PSA - update PSA - remove PSA	API
closed-network publishing	onboarding PSAs	PSA provision: - publish PSA - update PSA - remove PSA - health monitor - PSA maintenance	API
PSA deployment	service deployment in NED	- select PSA (policy-driven approach) - deploy PSA	API

Table 1: PSAM and PSAR business processes per use case.

BSS/OSS provision. An alternative representation of previous the case, oriented to massive provision of users from an enterprise IT system or an ISP OSS that do not need human interaction but a B2B communication through an API.

Publishing in an open marketplace. Developer-oriented logical process for the case when, after completing PSA development, a developer wants to offer it in an open SECURED environment.

Publishing in a closed network. This case supports the manager of a closed network infrastructure that wants to publish a set of PSAs for exclusive use in his network.

PSA deployment. This case is related to providing the required PSAs when a user accesses a NED for the first time.

Table 1 summarizes the relation between use cases and the set of business processes described in the next sections, and highlights as well the access interface to PSAM/PSAR to be used by the actors.

3.1 User-related interactions

This section covers the specification of all the workflows associated with the end-user, specifically those related to Table 1 “end user provision” and “BSS/OSS provision” use cases. It includes sign up process, search and selection, subscriptions or acquisition and management of the user’s PSAs.

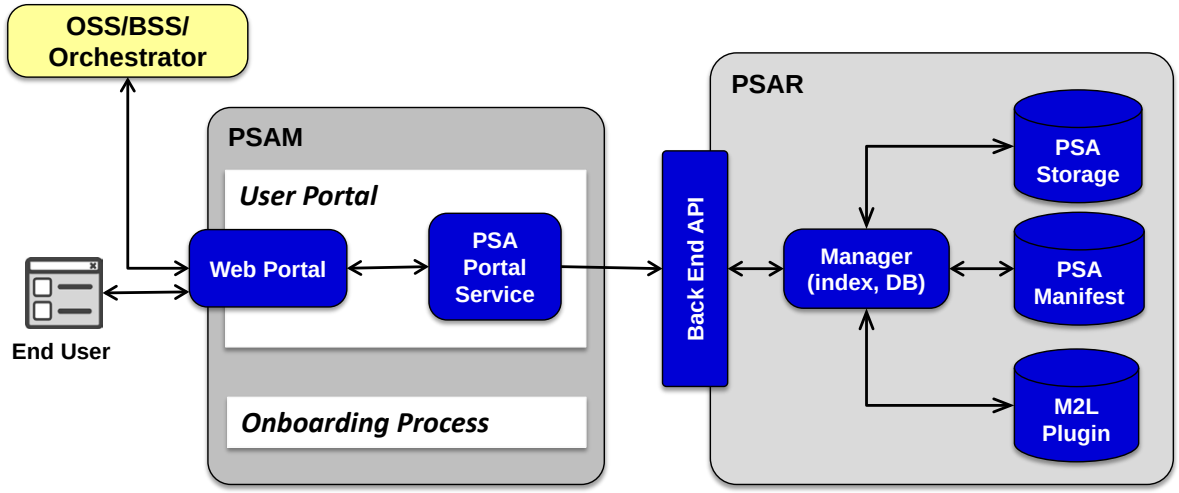

Figure 4: PSA Provision process.

These processes suport the application-driven model defined in the SECURED general architecture (D2.3.1).

Figure 4 extracts from the general architecture the entities, modules and components involved in these processes. Connections between Actors and Web Portal (PSAM), and between PSA Portal Service (PSAM) and BackEnd API (PSAR) happen through a REST API defined in Section 4.

Interactions between the Actors and the Web Portal may happen through a GUI or a programmatic interface.

The User Portal process is a human interaction interface. It offers a window to the End User for working with the PSAs. Open ecosystem (Section 6.1) is the common model for this portal, where end-user interacts using a web interface in public repositories. These functionalities and the detailed workflows of the User Portal are detailed in Section 5.

Automated enrolment is a particularly useful feature for the enterprise and network provider use cases. A new employee or a new subscriber joins the domain and must be automatically granted a user profile and possibly automatic access to a set of PSAs. The OSS/BSS system or Security Policy Manager (SPM) in the single-domain scenarios should be able to insert policies directly into the user profile repository. API definitions are provided for the application-driven approach in a simple scenario where the UPR and the PSAM are within the same administrative infrastructure domain, a closed ecosystems (Section 6.1). This process is composed of a set of common workflows:

- retrieve PSA information to feed BSS/OSS;
- user subscription to a PSA from BSS/OSS;
- user configuration of a PSA from BSS/OSS;
- cancellation of the subscription to a PSA.

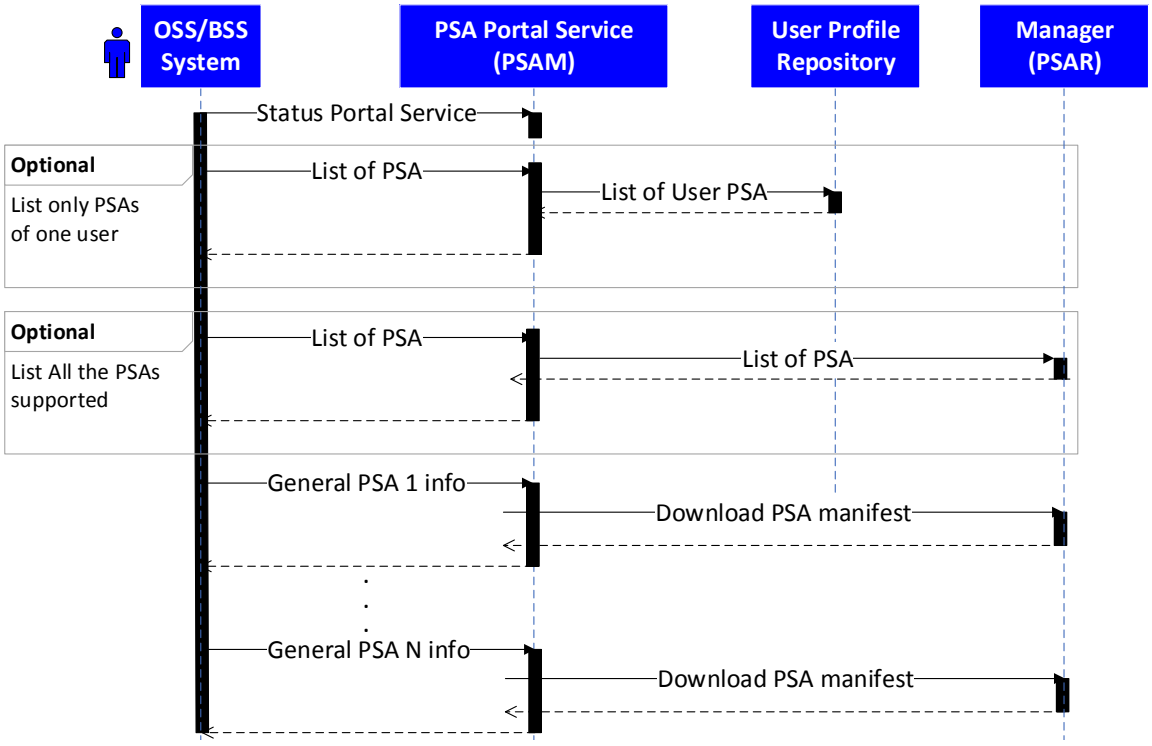

Figure 5: PSA retrieve process.

3.1.1 Retrieve PSA information

The objective of this workflow is to obtain a list of PSAs and the corresponding details for each of them. Two step are considered and shown in Figure 5:

Retrieve the list of PSAs. If there is no user specified by a parameter, only the general list of available PSAs is returned. On the contrary if the request is related to a user, only PSAs subscribed or acquired by a user is returned. The internal logic in Figure 6a has to be implemented in the PSA Portal Service. In the first case, the PSA Portal Service contacts the PSAR Backend API to obtain the list of PSAs available in the PSAR. In the second case, the PSA Portal Service contacts the UPR to learn the list of PSAs of the user.

Retrieve the details of each PSA. This workflow is iterative for each PSA. The developer must implement the internal logic detailed in Figure 6b to accomplish this process in the PSA Portal Service. In this case the PSA Portal service will request the information directly to the PSAR.

3.1.2 Subscription to a PSA

This workflow allows the administrator or management software (BSS/OSS system in network operators) to assign a PSA to an end user or, in other words, add a PSA to a User Profile. Figure 7 shows how the PSA Portal Service receives from OSS/BSS, through the Web Portal, the subscription solicitation of a PSA for a user and how it re-sends the query to the UPR about the new PSA addition and Figure 10a describes the internal logical of the PSA Portal Service.

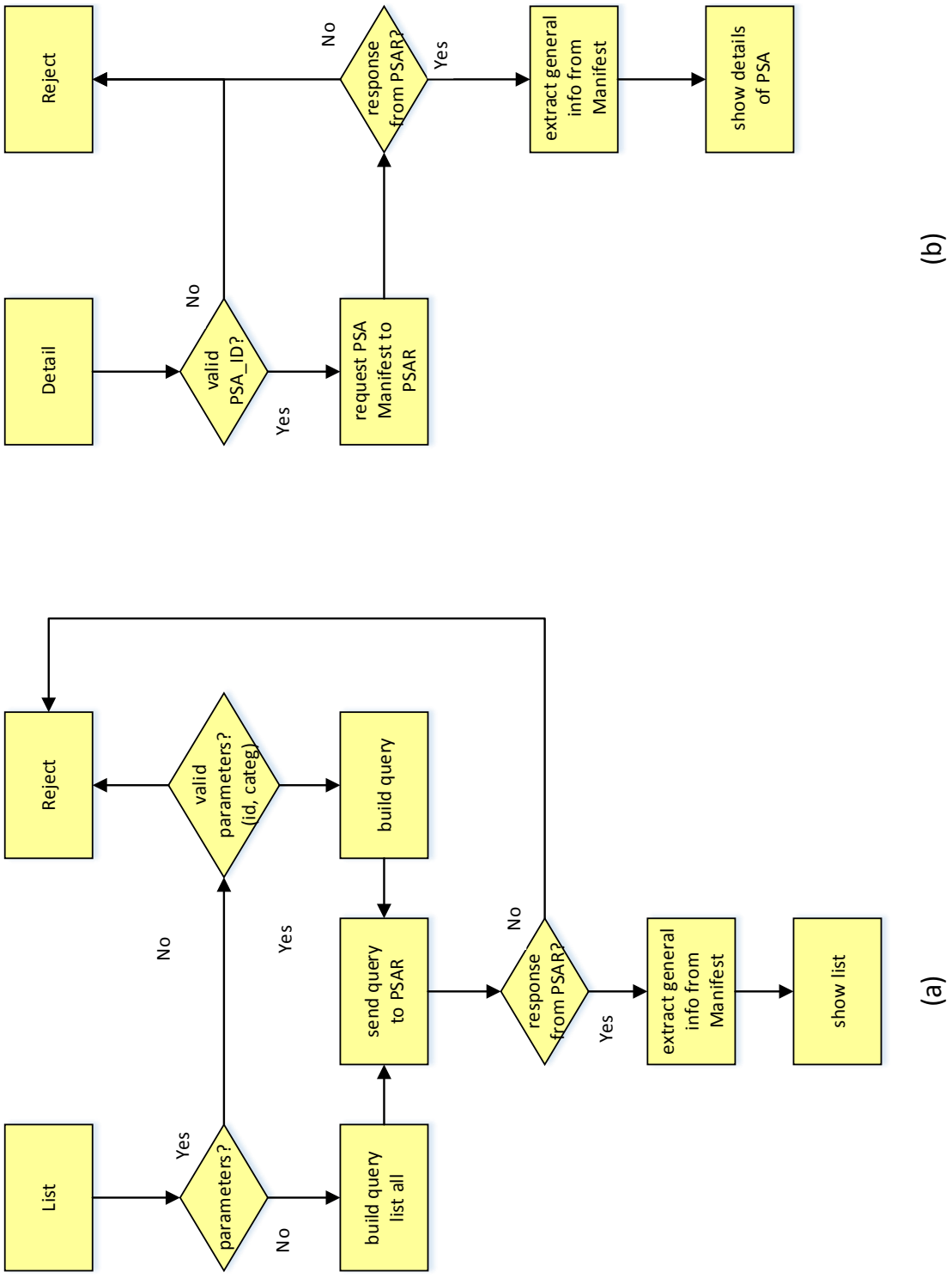

Figure 6: PSA Portal Service logic: (a) retrieve PSA List, (b) retrieve details of a PSA.

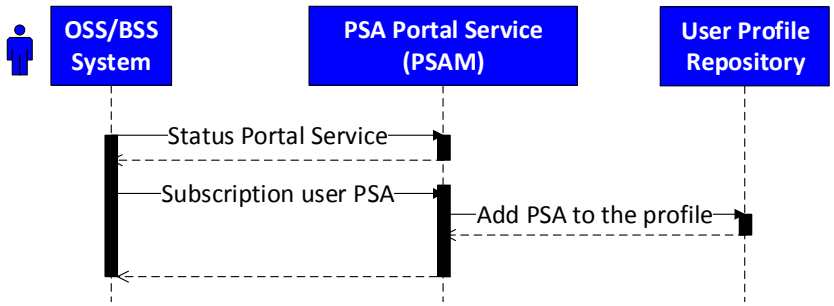

Figure 7: PSA subscription process.

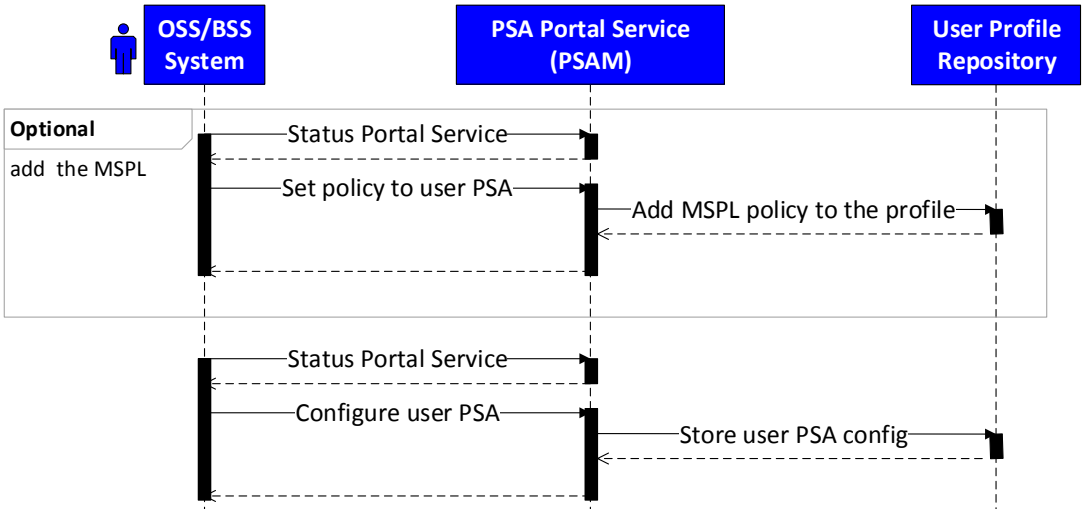

Figure 8: PSA policy and configuration process.

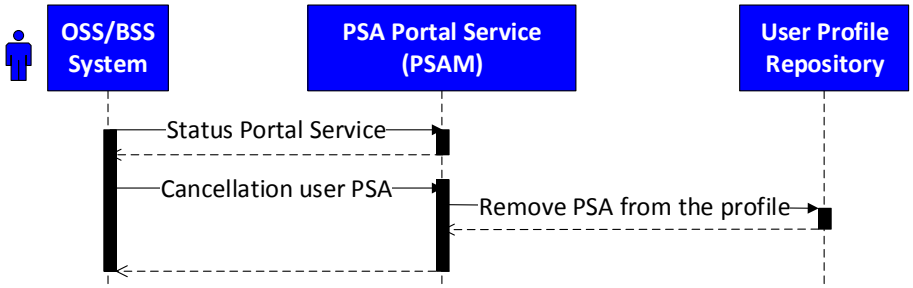

Figure 9: PSA Cancellation process.

3.1.3 Configure a PSA

The administrator or management software may use the following API if an application-driven approach is being used instead of a policy-driven one. A MSPL policy (optional) and the associated configuration for a particular user PSA is delivered. Figure 8 shows how the PSA Portal Service receives through Web Portal the MSPL policy and the PSA's configuration, and send it to the User profile Repository for a specific user. The Internal logic of the PSA Portal Service is shown in Figure 10b.

3.1.4 Cancellation of the subscription to a PSA

When a user subscription ends, or if a new type of PSA is needed, a cancellation process is activated. The process includes removing the PSA from the PSA list of the user profile and removing any corresponding policies or configurations.

The workflow between the PSA Portal Service in PSAM and UPR is described in Figure 9. The internal logic of the PSA Portal Service is shown in Figure 10c. This logic allows to cancel one or all user's PSA in an atomic call.

3.2 Onboarding PSAs

This section covers the specification of an additional task that the PSAM/PSAR must support: loading and publishing the PSA, in short "onboarding". All publishing use cases listed in Table 1 are detailed in this section. This process is transparent to the end-user and prepares the PSAs to be selected and used in the SECURED infrastructure.

The processes involved (Figure 11) are again composed of interactions between various elements from the general architecture. These interactions happen through a client-server REST API described in Section 4.

Depending on the type of actor, this specification foresees two potential cases for the onboarding process:

Developer Service. A PSA developer wants to publish his product in a public applications marketplace, where the actor is the developer that uploads his PSA. Thus the developer service needs to execute functional and requirement validation before opening the PSA for public deployment. This specification does not include contextual validations (like illegal applications) that are responsibility of the repository manager.

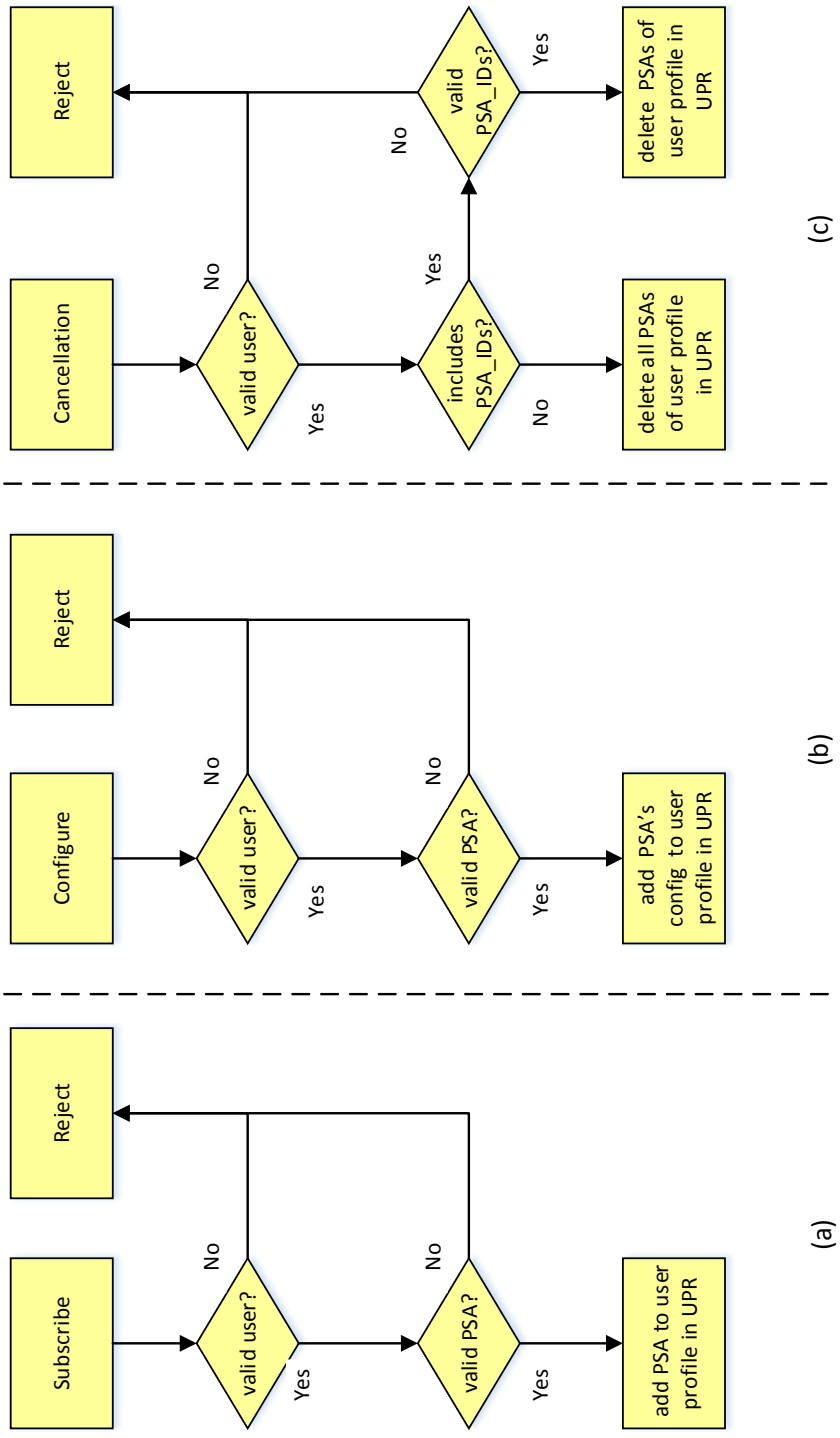

Figure 10: PSA Portal Service logic to (a) subscribe, (b) configure, or (c) cancel a PSA.

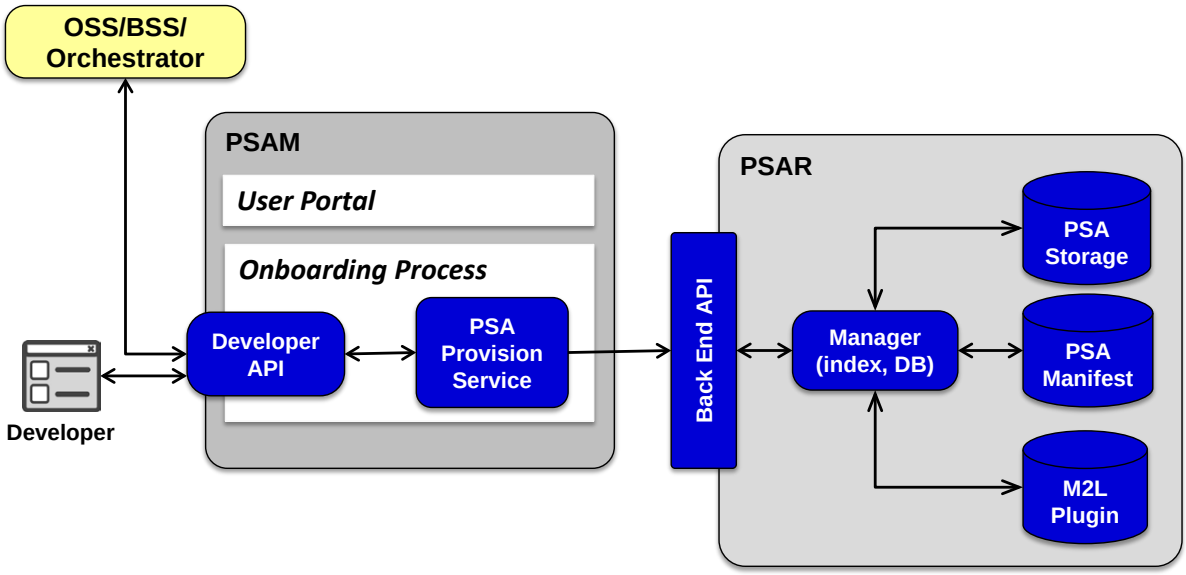

Figure 11: PSA onboarding

PSA Provision. Directly associated with closed ecosystems, there is an OSS/BSS system that commands the storage of PSA in SECURED, after acceptance by the owner of the ecosystem. In this environment usually the PSA is subject to an extensive former process of testing and certification. This process is not done by PSAM, which only executes the internal basic validations. This case has strong relations with the advanced management processes defined in Section 3.2.4.

Both these cases use some or all of the workflows detailed in the next subsections.

3.2.1 PSA Publishing process

The process in Figure 13 is as follows:

1. the PSAM is contacted by a developer (or by an OSS/BSS system) through the "Developer API" using a RESTful interface;
2. the "PSA Provision Service" module executes its internal logic (Figure 12a) including the validation process;
3. the "PSA Provision Service" module interacts with the Manager module in the PSAR through the "Backend API" to provide all the needed data;
4. finally the PSAR Manager opens to a public state the PSA, to make it available to the end-users.

3.2.2 PSA Updating process

This workflow is similar to the publishing one. The only differences are that the user includes the PSA id that identifies an existing PSA and there is no need to update all elements related with this PSA (manifest, binary, or M2L Plugin) but only the new ones, thanks to the internal logic of Figure 12b. The details are given in Figure 14.

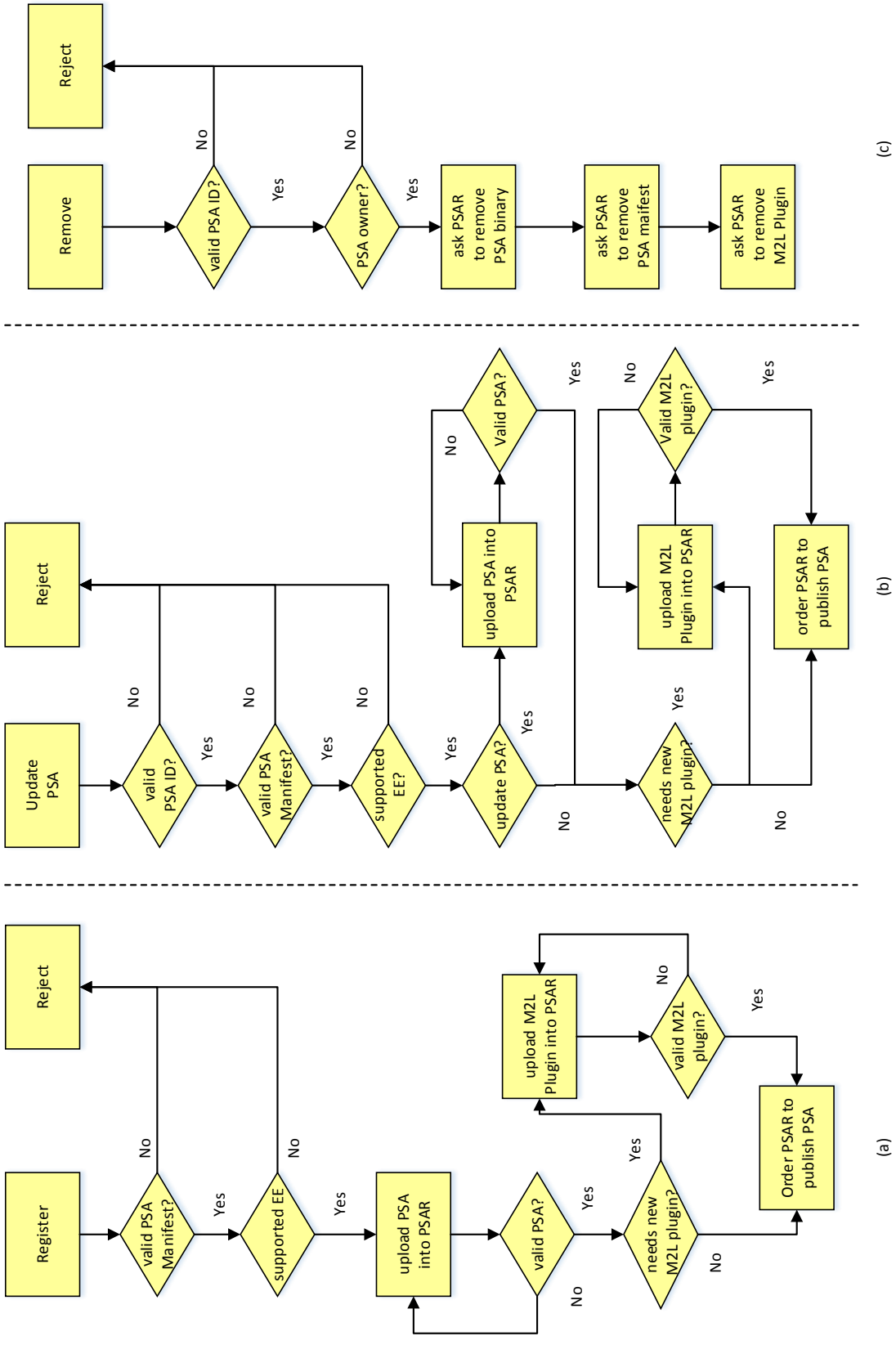

Figure 12: Internal logic of PSA Provision Service to (a) publish, (b) update, and (c) remove a PSA.

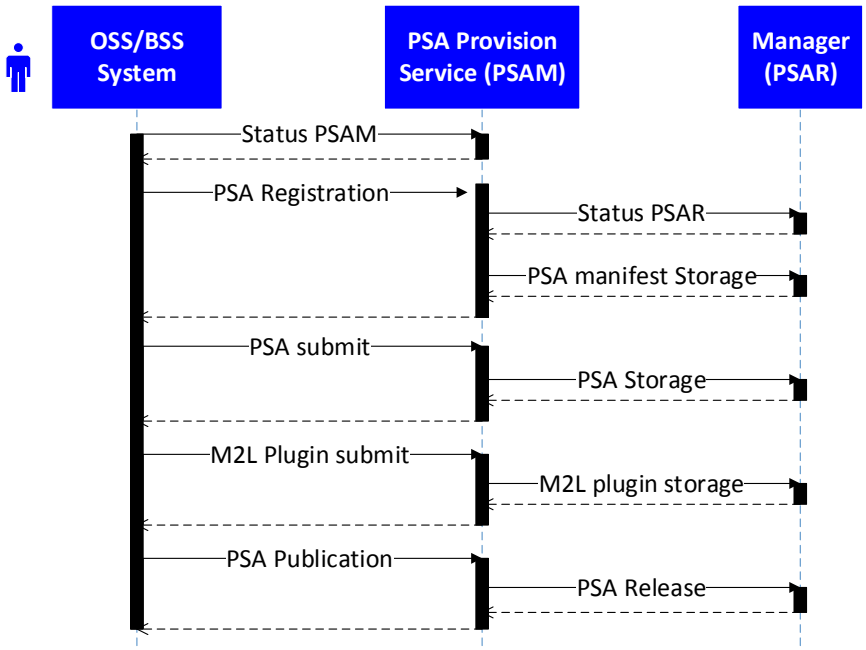

Figure 13: PSA publishing process.

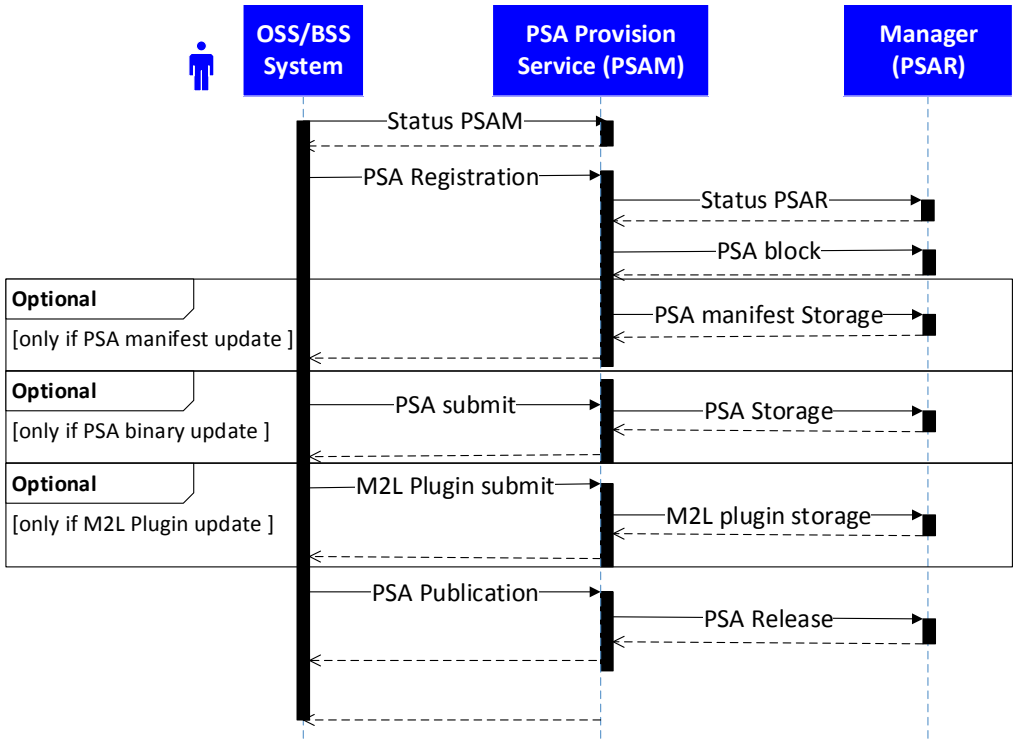

Figure 14: PSA updating process.

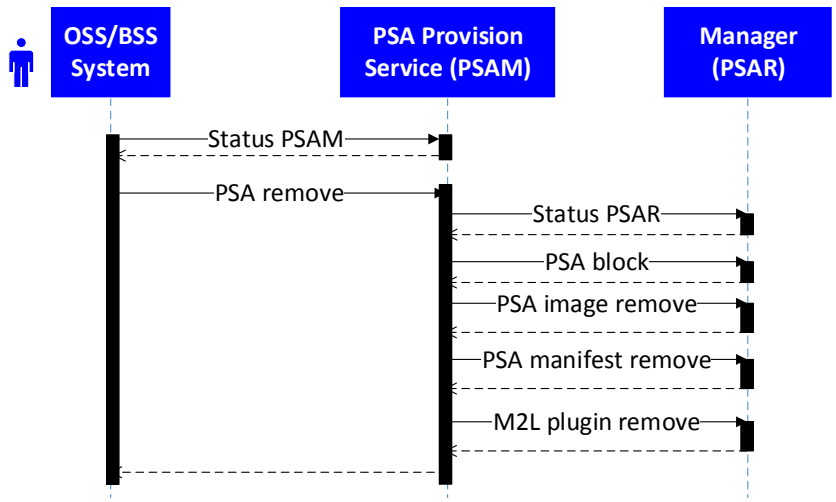

Figure 15: PSA removing process.

3.2.3 PSA removing process

Removing a PSA that is outdated or no longer required involves coordination between PSAM, PSAR and the internal logic of PSAM (shown in Figure 12c). The sequence of actions is detailed in Figure 15 and happens in reverse order with respect to the publishing process.

3.2.4 Advanced management process

The processes described in the previous subsections are those executed more frequently, but in some situations special needs can arise. For example management optimizations can require direct access to the PSAR for health monitoring. These needs can be satisfied with direct calls to specifics methods in the API. Here is a brief description of these processes (illustrated in Figure 16):

Health monitoring. Status reports can be directly invoked from OSS in order to obtain information of the status of the management platforms (PSAM and PSAR). For this process direct API calls to Status PSAM and Status PSAR can be used.

PSA content maintenance. In some situations an external system could require to stop the use of a PSA for specific changes in the data, i.e. block a recently vulnerable PSA until the problem is corrected. A potential optimization in these situations could allow direct calls to PSAR "Backend API" from external system: PSA block, PSA release, PSA manifest remove, etc. All of them are defined in section 4.2.

3.3 Service deployment in NED

This section details the PSAR back-end API interaction for the user's service enforcement: the PSA choice when using the policy–driven configuration and the PSAs deployment inside a NED, illustrated in Figure 17.

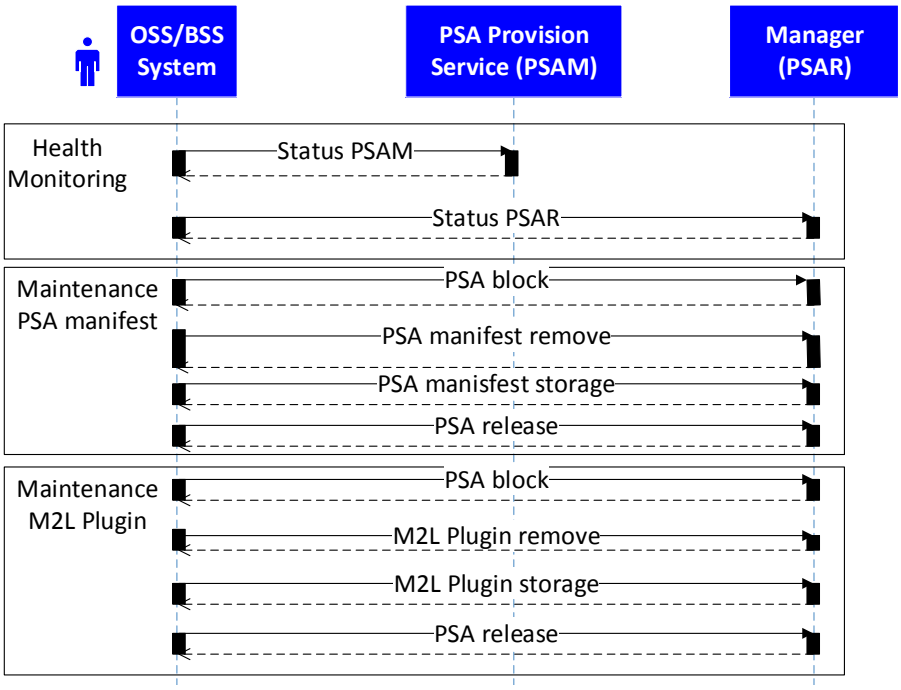

Figure 16: Advanced processes.

3.3.1 PSA selection in a policy-driven configuration

When a user configures his security controls with the policy-driven approach, at some point the SPM will need to query the PSAR for the list of PSAs that match the requirements dictated by the high-level policies of the user. Then, the SPM creates the user's service graph that contains the PSAs and their policy, which will be deployed in the NED.

3.3.2 PSA deployment in a NED

When a user connects to a NED, once the NED attestation and user authentication are done, the NED retrieves the user profile – which contains the service graph – and then contacts the PSAR to get the user's PSAs. The NED requests the PSA manifest to later fetch the PSA version that matches its available TVD (e.g. OS, CPU/memory available per PSA) or to generate a TVD matching the requirements in teh PSA manifest.

For the case of a PSAR in a distributed environment, some additional considerations are needed in the download of the PSA. The Manager module checks for the location of the originating NED and, if a PSA image is available in a nearby Local PSAR, the Manager can inform the NED of a different location of the PSA to be downloaded.

4 Programmatic interfaces

This section describes the API that has been designed to support the workflows in Section 3. Hence two different APIs are defined, one each for activities involving respectively the PSAM and the PSAR.

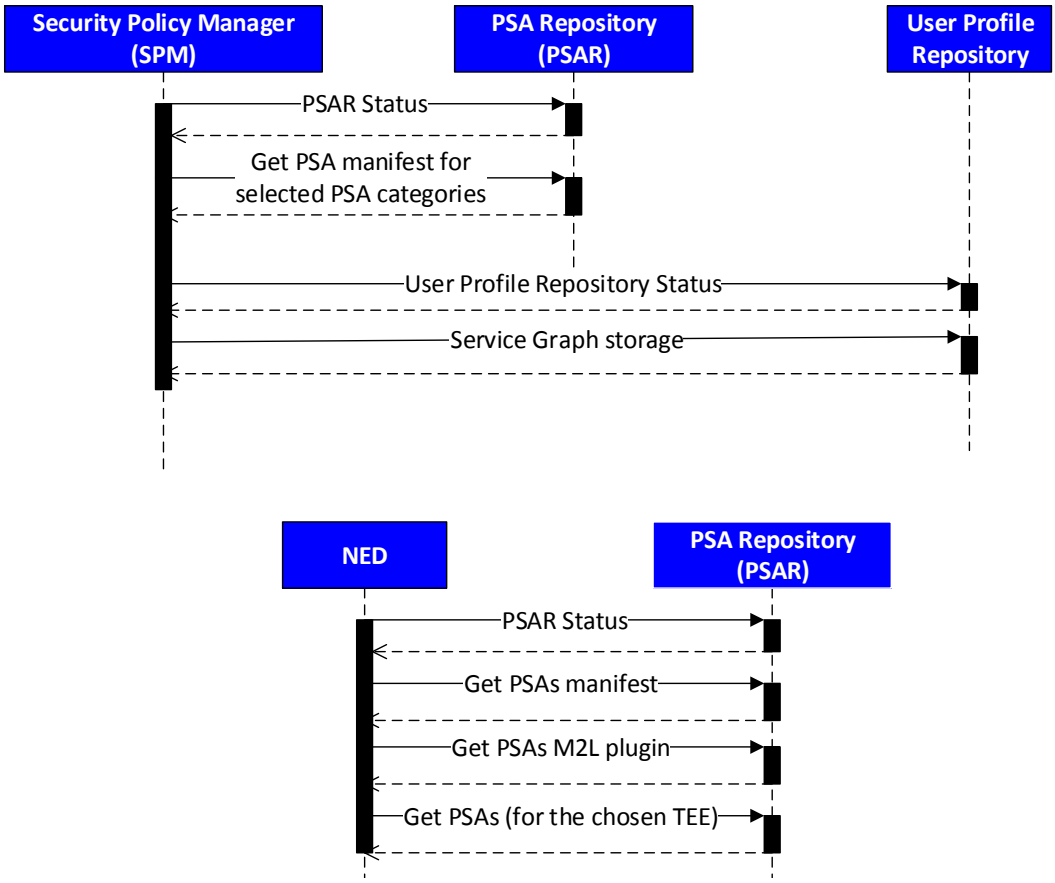

Figure 17: Service deployment.

As this specification will likely evolve over time, the various versions are maintained online at the following pages:

- https://www.secured-fp7.eu/ref/PSAM/
- https://www.secured-fp7.eu/ref/PSAR/

4.1 PSAM API

The PSAM API is composed by two different interface families: one API for the End User to interact with the Web Portal, and one API for PSA Developers to interact with the Developer API.

The PSAM API uses HTTP syntax as it is a REST interface. Each call returns a HTTP status code: `200` if the operation completed successfully, `4xx` or `5xx` if the operation failed due to a temporary or permanent error. The return body (if present) is encoded in JSON notation, while the request body (if any) is formatted according to the type of data being transmitted (e.g. `text/xml` if a XML manifest is uploaded to the repository, or `application/octet-stream` if a binary image is transferred).

4.1.1 Web Portal API

This section addresses the management needs of the PSA subscription process.

In order to help PSAM developers and software architects the next paragraphs include a conceptual description and the list of calls to contact with PSA Web Portal service. There are two ways of interacting with this service, either directly through the user-oriented GUI User Portal or via direct API calls (to facilitate external integration):

User Portal. End users are accustomed to work with graphical interfaces for configuring and accessing their services so also SECURED offers to users a web-based friendly interface for managing the PSAs in the application-driven model. The detailed specification of the web-based User Portal is given in Section 5.

BSS/OSS/Orchestrator. Complementary API calls are offered in case the PSAs are managed not by end-users but by an eneterprise, a service provider, or a network operator. A specific API component is provided to integrate commercial channels and provisioning system of these stakeholders in SECURED.

The defined API calls are briefly described here. Full reference is available at the on-line version. Items in square brackets are optional. Items in curly braces are placeholders (to be replaced with actual values at call time or * to refer to all possible values) or enclose a choice among different values. Some calls will behave differently depending on the identity of the caller: a normal user will be able to access only information pertaining to himself or public, while an administrator will have access to information about all users.

Status of the Web Portal service. Get the status of this service to verify if it is operational.

```
GET /v1/PortalService/status
```

List PSAs. Search and show the list of PSAs available at the PSAR. Can include optional search parameters (e.g. category). The result returned contains the PSA ID, name and category.

```
GET /v1/PSA/images?id={psa_id}[&category={filter}]
```

General PSA info. Retrieve information about a specific PSA, extracted from its manifest. The level of detail is controlled by a specific argument.

```
GET /v1/PSA/images/{psa_id}[?details={yes|no}]
```

List user PSAs. Provides the list of PSAs subscribed by a specific user or by all users (when using * as `user_id`).

```
GET /v1/users/{user_id}/PSA
```

Subscribe user PSA. Provision of a PSA to a specific user.

```
PUT /v1/users/{user_id}/PSA/{psa_id}
```

Configure user PSA. Directly store a specific configuration (passed in the request body) for the user and PSA specified as parameters. This assumes the will to upload a specific low-level configuration without using MSPL. Can be used to replace an existing configuration if the optional configuration ID is provided, else a new configuration will be created The call always return the ID of the configuration created or overwritten.

```
PUT /v1/users/{user_id}/PSA/{psa_id}/configuration/[{config_id}]
```

Set Policy to user PSA. Store an MSPL policy (passed in the request body) for the user and PSA specified as parameters. Can be used to replace an existing policy if the optional policy ID is provided, else a new policy will be created The call always return the ID of the policy created or overwritten.

```
PUT /v1/users/{user_id}/PSA/{psa_id}/mspl/[{mspl_id}]
```

Unsubscribe user PSA. Unsubscribe one or all PSAs of a specific user.

```
DELETE /v1/users/{user_id}/PSA/{psa_id}
```

4.1.2 Developer API

D5.1 describes a life-cycle for the PSA development that includes publishing the PSA. Details of the workflow are given in Section 3.2. This section contains the API calls needed for integration with a developer publishing framework, which may be controlled by a single specific developer (typical in open ecosystems) or by Enterprise/OSS managers in closed environments.

Status of Developer Service. Get the status of this service, to verify if it is operational.

```
GET /v1/DeveloperService/status
```

Register PSA. Register a new PSA by uploading its manifest (provided in the request body). The server validates the XML manifest format and, if it is valid, a new unique identifier is assigned to the PSA and returned in teh respons ebody. Can also be used to upload a new version of the manifest for an existing PSA, by specifying its ID.

```
PUT /v1/PSA/manifests/[{psa_id}]
```

Submit PSA image. Upload the software package (provided in the request body) that represents the executable part of the PSA. The package must be compliant with the PSA manifest provided at registration time.

```
PUT /v1/PSA/images/{psa_id}
```

Submit M2L plugin. Upload the software package (provided in the request body) that represents the M2L plugin, if this it is not already part of the PSA itself.

```
PUT /v1/PSA/M2Lplugins/{psa_id}
```

Publish PSA. The developer requires his PSA to be made available to the end users. It assumes that everything related to the PSA has already been uploaded to the PSAR (image, manifest. M2L plugin) else the call will fail with error. This process can be supervised in closed ecosystems. Periodic calls will offer a result of in progress or published status and could be used as a health monitor.

```
PUT /v1/PSA/publications/{psa_id}
```

Remove PSA. This call triggers the deletion of a PSA from the PSAR, with all its components.

```
DELETE /v1/PSA/images/{psa_id}
```

4.2 PSAR API

This section lists the calls offered by the PSAR Backend API, which has been designed to support the workflows defined in Section 3.

Status PSAR. Verify if the PSAR is online and available to receive queries.

```
GET /v1/status
```

List PSAs. List the PSA stored in the PSAR. This operation accepts several types of query parameters to filter the results of the returned collection.

```
GET /v1/PSA/images
```

Store PSA manifest. Upload and store the PSA manifest in the PSAR, specifically in the PSA Manifest Storage component. The Manifest file must be provided in the request body. This method can be use for new PSAs (in which case a new PSA ID is created and returned) or to update the manifest of an existing PSA (whose ID is passed as argument). Validation of the manifest has been done previously by the PSAM so it is not needed at this stage.

```
PUT /v1/PSA/manifests/[{psa_id}]
```

Store PSA image. Request to store the software package (provided in the request body) that represents the PSA in the PSA Storage component. Can be be used to store a new pacakge or to update an existing one.

```
PUT /v1/PSA/images/{psa_id}
```

Store M2L plugin. Request to store the software package (provided in the request body) that represents the M2L plugin in the M2L Plugin storage. Can be be used to store a new pacakge or to update an existing one.

```
PUT /v1/PSA/M2Lplugins/{psa_id}
```

Change PSA status. Used to move the PSA status from ready to open (i.e. available for deployment) or from open to freeze (i.e. unavailable for deployment, typically for maintenance reasons). The request body specifies the new status.

```
PATCH /v1/PSA/images/{psa_id}
```

Remove PSA image. Used to delete the PSA image.

```
DELETE /v1/PSA/images/{psa_id}
```

Remove PSA manifest Used to delete the PSA manifest.

```
DELETE /v1/PSA/manifests/{psa_id}
```

Remove M2L plugin. Used to delete the PSA M2L Plugin.

```
DELETE /v1/PSA/M2Lplugins/{psa_id}
```

Get PSA manifest. Gets the PSA manifest of a specific PSA.

```
GET /v1/PSA/manifests/{psa_id}
```

Get PSA image. Gets the software package that represents the PSA itself. The PSA image can be returned in the response body or the response can signal a HTTP Redirect (code `3xx`) to point the client to the URI where the image is available for download.

```
GET /v1/PSA/images/{psa_id}
```

Get M2L plugin. Gets the software package that represents the M2L plugin of a PSA. The plugin can be returned in the response body or the response can signal a HTTP Redirect (code `3xx`) to point the client to the URI where the plugin is available for download.

```
GET /v1/PSA/M2Lplugins/{psa_id}
```

Register PSARL. Used in distributed environments to inform the Central PSAR that a new Local PSAR is available and ready. The request body contains information to identify and locate the PSARL. The server validates the provided information (e.g. by contacting the PSARL) and, if it is valid, a new unique identifier is assigned to it and returned in the response body. Can also be used to change information for an already registered PSARL, by specifying its ID.

```
PUT /v1/PSARLs/[{psarl_id}]
```

Delete PSARL. Used in distributed environments to inform the Central PSAR that a Local PSAR is no more available. The request body contains information to identify and locate the PSARL.

```
DELETE /v1/PSARLs/{psarl_id}
```

Note that to synchronize a PSARL with the Central PSAR no new call is needed: the PSARL can use the calls to get the PSA manifest, image, or plugin with conditional HTTP headers (e.g. `If-None-Match` or `If-Modified-Since`) so that a response is returned only if newer versions are available.

5 Design of the User Portal

This section provides the functional specification for the User Portal, describing its minimum expected capabilities. Design, navigation, site layout, and accessibility are not defined.

5.1 Web Portal design

Each PSA has a dedicated web page, which renders the following information:

- the PSA title and functional description;
- TEE platform compatibility (i.e. which execution environment is needed to run the PSA);
- pricing or eligibility (e.g. if tied to a subscription);
- capabilities and system requirements;
- user reviews describing the quality of the PSA;
- legal disclaimer from the PSA provider;
- summary of developer identity and link to contact. developer

An example page is shown in Figure 18.
All attributes in the PSA manifest must be searchable via the user portal, either by browsing by category or through a direct query. The PSA manager will access information about the latest PSAs assigned to each user by looking up the corresponding information stored in the User Profile Repository (UPR).

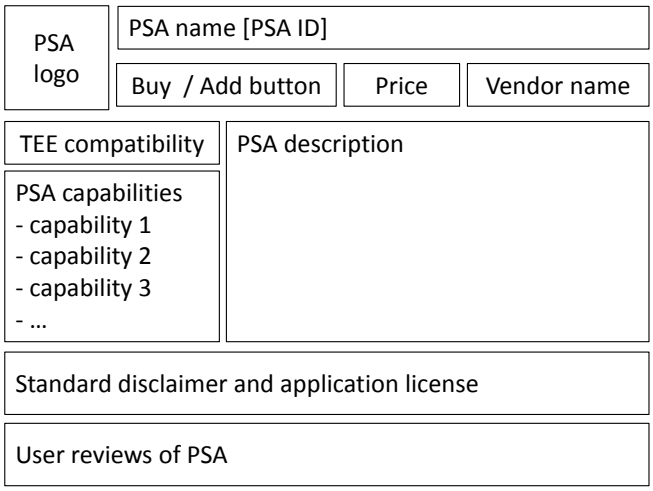

Figure 18: Example PSA webpage in the User Portal.

5.1.1 Integration with SPM

To provide a unified user experience, the policy editor of the SPM may be accessed through the same portal used for the PSA selection process. This is scenario-dependent and can be implementation defined.

5.2 User authentication

The user should log in with their UPR credentials to allow access to his profile. This should be handled by the particular deployment scenario, depending on the number of managed domains and if a two-tier authentication system is used. Since the user authentication model varies with the use case scenarios, the structure of the user authentication process is implementation defined, and not within the scope of the general specification. However, examples of the more common options are presented here.

Single SECURED domain. In this case a single solution is used to handle the credentials for the UPR, PSAM and PSAR. Examples range from passwords stored in the UPR to more complex system such as LDAP or Kerberos authentication and user management systems. In a mobile provider environment this may be handled by existing authentication, authorization and accounting systems (AAA) functionality in the OSS/BSS, such as those based on the RADIUS protocol.

Open identity/authentication. The adoption of systems such as OpenID Connect or OAuth-2.0 permits the use of a single credentials for different managed domains (particularly useful for the open ecosystems). When using this schema, the web portal must embed the front-end to the authentication procedure with the user-selected identity provider (e.g. Google or Facebook) and – upon successful authentication with the third-party identity provider – forward the identity certificate or token to the UPR. Note that, although the authentication is managed by a third-party, the UPR still needs to have a profile for the user, tied to his external identity.

Two-tier authentication. In the case of mixed ecosystem (where the SECURED provider allows users to store PSAs at an external PSAR) the user profile may contain the credentials for the foreign PSAR, or the web portal can embed an interface to forward the credentials to the foreign PSAR.

5.3 Payment gateway

A payment gateway may be required if the SECURED provider wishes to sell PSAs to end users. When a user requests to purchase a PSA, he should be redirected to a third-party payment gateway (e.g. PayPal, Google Wallet, BitCoin). To avoid significant transaction costs, a PSA provider may set up his own payment gateway system. However, this is implementation-defined and is not within the scope of this specification.

5.4 Service graph composer

In the application-driven configuration, the user has to define himself the service graph. Therefore, the user portal must provide an interface (ideally a graphical one) to create the service graph. A default service graph consisting of a static linear chain of the selected PSAs will be proposed to the user, and the user can then choose to reject the default graph and connect the PSAs manually. In a nutshell, the service graph composer workflow (for each traffic direction) is:

1. select a PSA from those available in the user profile;
2. connect the ingress port of the PSA to the egress port of the previous PSA (or to the IN port of the service graph), optionally specifying traffic matching rules (e.g. `TCP port = 80` for the link to a HTTP proxy);
3. repeat the previous steps as much as needed and then finish by connecting the egrees port of the last PSA in the chain to the OUT port of the service graph.

5.5 Web-based User Portal

Procedure to add a PSA to a User Profile using the User Portal web interface:

1. user visits the PSAM User Portal;
2. user searches for a particular PSA or browses for PSAs by category;
3. user selects a PSA page to view more information;
4. user reads from the PSA and all information related to the PSA;
5. user selects to add the PSA to his profile using a“Buy” or “Add” button or equivalent;
6. if the user is not already logged in then a login page is presented to the user for authentication according to the selected method (which may include transparent login through the use of TLS client authentication with a client certificate);
7. upon successful authentication, the user is redirected to a page with a list of all PSAs he has bought or added (the most recently added PSAs appear at the top of the list);
8. when finished the list of user PSAs is updated in the UPR.

5.6 Performance requirements

The availability of public user portals should comply with the SLA of the SECURED service provider. However, should the User Portal or PSA Manager service be unavailable, this should have no effect on the availability of the PSAR.

6 Implementation guidelines

This section offers some guidelines for developers about the implementation of PSAM and PSAR.

A PSAM and/or PSAR developer should follow these steps:

1. Identify the type of ecosystem to use (from Section 6.1) to develop and decide the list of functionalities associated with the model chosen: security controls, authentications system, channel encryption, performance, capacity and availability requirements, or legal impacts. This list allows create a high level design.
2. Decide the components to develop (from the general architecture in Section 2). A developer can decide to create only a web portal module, a PSAR or the whole management and repository system.
3. Decide the processes associated to the selected components and use cases from those described in Section 3.
4. Develop the internal logic of each module and if the design includes several modules, include also the intercommunication channels for the interactions workflows. These internal logic processes and workflows are specified in Section 3.
5. Implement the required API calls (as client or server, depending on the module) according to the specifications in Sections 4 and 5.

A simple example of a design to implement a PSAM and PSAR for integration in the environment of a Network Operator (NO) is given here:

1. Closed ecosystem is selected. Integration in the Identity manager system of the NO (i.e. LDAP) is added to the specification. Also, as all the elements are internal to the NO domain, it is decided that encryption for the communications is not required because no external traffic is expected between the PSAM and PSAR components. Finally thelist of available PSAs will be statically pre-configured and no option for adding new PSAs by the user are planned.
2. No PSAM User portal is needed because users will interact throughout the appropriate marketing channel of the NO (i.e. the Commercial Web Portal). The PSAM developer will need to include the rest of the modules in PSAM.
3. Selected processes are those specified in “B2B Provision of PSA” use case (Section 3.1).
4. Developer will create all the internal logic and PSAM / PSAR API (Section 4) using the programming language and platform of his interest (i.e. LAMP architecture with python API framework).

Additional hints are included in the next subsections. Specifically:

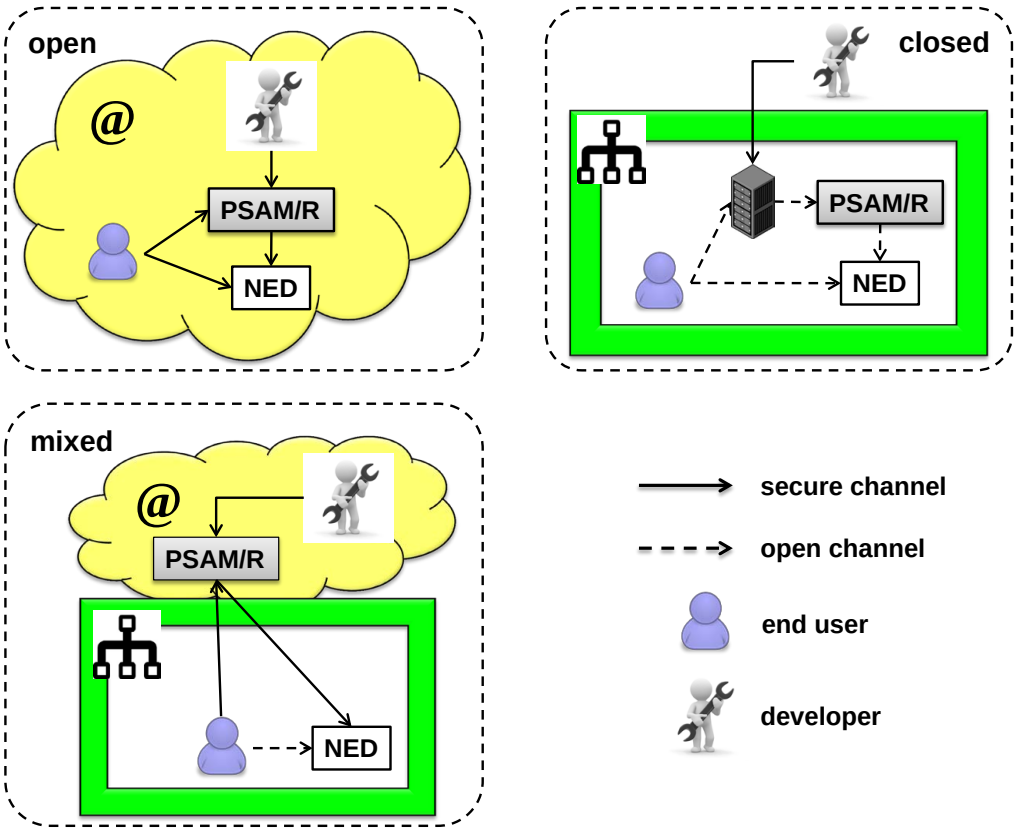

Figure 19: PSAM/PSAR ecosystem models.

- description of different ecosystems or environments where the PSAM and PSAR can be deployed and some considerations needed for each of them;
- analysis of OpenStack as a possible implementation platform because of its nice alignment with the PSAM and PSAR concepts.

6.1 Ecosystems

PSAM/PSAR implementations and deployments can be very different depending on the environment where they must operate. We could have Network Operators that have specific requirements and characteristics, totally different from an academic lab or a open cloud infrastructure. Some of these requirements can be availability levels, IT system integration or trusted channels, among others.

In order to help a developer to decide the best way to create a PSAM/PSAR environment, a set of possible ecosystems has been defined (Figure 19). These are not rigid and separate models, indeed a mixed ecosystem is included to make clear this option too.

6.1.1 Open ecosystem

This type of environment is oriented to business models based on Internet and an open cloud infrastructure, and usually exploit open source code. The main properties of this environment are:

- Internet reachability through public addresses to the infrastructure, including the management network between the main components of SECURED (NED, PSAM, PSAR, SPM);
- open management repositories where to upload and store the PSAs developed, accessible by any person or stakeholder;
- security achieved by establishing trusted communication channels among the infrastructure elements and various open authentication and access control frameworks (e.g. OAuth, OpenID connect, token models).

An open ecosystem is ideal for end-user home network NED, academic institutions, and research labs, that promote open-source business models and projects. The common goal is that the end-user will have direct control of his policies and PSAs.

6.1.2 Closed ecosystem

Closed environments are focused on stakeholders with interest in having a strong control of the applications, deployment infrastructure, controlled access and use permissions. The main properties of this environment are:

- closed management network between the main components of SECURED (NED, PSAM, PSAR, SPM) – if based on Internet (which is not the preferred medium) then it would require secure private channels to reduce risk and exposure;
- restricted management repositories where strict validation tests are expected to happen before allowing to upload and store the PSAs developed and the administrator has the control to decide which stakeholder and when can have access;
- stakeholder owners establish the security management policies for authentication and access (in this ecosystem is common to work with internal identity management systems, like corporate or operators LDAP directories, where the registration and access process is not public);
- support for carrier-grade models in terms of availability and performance.

Closed ecosystem are mostly designed for Corporations, Service Providers and Network Operators. The common goal is that the end-users delegate to managers and OSS/BSS the control of their policies and PSAs.

6.1.3 Mixed ecosystem

A mixed environment can use characteristics from both the open and closed ecosystem. One potential use case is SMEs that do not want (or cannot afford) a fully dedicated PSAM and PSAR, but it can install a NED on their own. In this case some management communication will happen via Internet (e.g. PSA download to the NED, or User Portal access) and others in a closed environment (e.g. user access and use of his PSAs). This can be seen as cloud-based services for PSAM/PSAR. The general rule is that communication between different trust domains must be adequately protected (i.e. authenticated and encrypted). An external security gateway could be used to enforce these requirements.

6.2 A possible reference implementation

To facilitate acceptance and standardization of the SECURED architecture, we would like to find an open and widely accepted technology to serve as a reference providing an environment with all the capabilities needed by the PSAM/PSAR services, as an alternative to creating them from scratch. This section analyses OpenStack as a support for an open reference implementation but this does not restrict in any way the design choice of implementers.

6.2.1 Introduction to OpenStack

OpenStack is a set of open-source software tools and projects that enables building and managing cloud computing platforms for public and private networks. OpenStack is probably a good choice in case of multi-server deployment and can provide important advantages for the distributed architecture discussed previously. But OpenStack is also suitable for standalone installations, providing an easy growth path to multiple boxes seamlessly if needed.

In general the SECURED architecture and OpenStack reference model are aligned in terms of Virtualization of applications, or Virtual Machines. If we focus on the PSAM/PSAR service we find that it is possible to map modules of OpenStack onto our architecture. The modules of OpenStack offer nine services, namely, Heat (Orchestration service), Horizon (Dashboard), Neutron (Networking service), Nova (Compute service), Glance (Image service), Swift (Object storage service), Cinder (Block storage service), Celiometer (Telemetry service) and Keystone (Identity service). Figure 20 shows a simple scheme of an OpenStack standard configuration. All interactions between the different services and modules are carried out via RESTFul APIs, which is aligned with the PSAM/PSAR architecture.

6.2.2 OpenStack reference

As said before, it is possible to map modules of OpenStack architecture with modules of SECURED. In the case that concerns this deliverable (PSAM/PSAR service) we will talk mainly about the following OpenStack services: Horizon, Keystone, Glance and Swift. If we consider the scheme of the general architecture(Section 2, Figure 1), the mapping with these OpenStack modules could be as shown in Figure 21 and discussed below.

Horizon (Dashboard service) is a web interface that enables cloud administrators and users to manage various OpenStack resources and services, such as selecting, launching and managing images, VMs/images deployment or complex per-tenant networking. These functionalities could be useful for SECURED to develop parts of the PSAM/PSAR infrastructure by exploring the Horizon APIs (Object Storage, Image Service, Identity, Compute, and Networking APIs).

Glance (Image service) stores, processes, and retrieves metadata about images This service supports a variety of repositories including normal file systems and Object Storage (Swift). This service consists of a Glance API (to receive calls from other services) and a Glance registry which stores, processes, and retrieves metadata about images (manager role). Glance uses the following OpenStack APIs: Object Storage, Image Service, and Identity APIs.

Keystone (Identity service) provides Identity, Token, Catalog, and Policy services. This service is organized as a group of internal services that can be configured to use a back-end to fit a variety of environments and needs. It is common to find that Keystone has its users stored in an existing centralized authentication service, typically a LDAP server. Keystone has an LDAP driver for the identity back-end to allow it to use LDAP for authentication and storage users, plus some details that

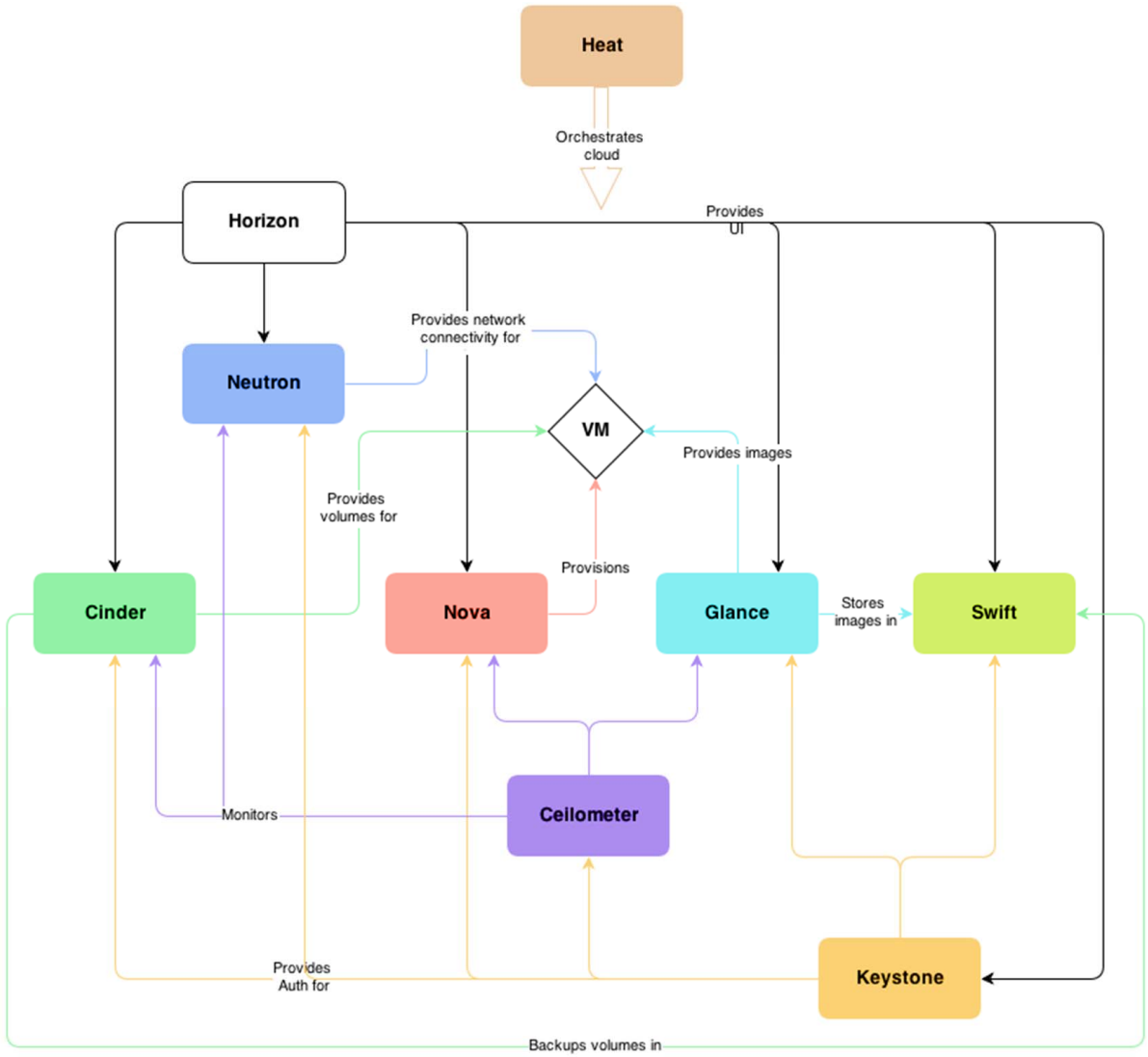

Figure 20: OpenStack architecture.

are not involved on the authentication process. To perform this service, Keystone uses the OpenStack Identity API.

Swift (Object Storage service) is a highly scalable and durable multi-tenant object storage system for large amounts of unstructured data at low cost through a RESTful HTTP API. Each Object Storage service consists of a proxy server that accepts calls from Glance to upload files, modify metadata and create containers, an object server which is a simple storage server that can store, retrieve and delete objects stored on local devices (objects are stored as binary files on the filesystem with metadata stored in the file extended attributes), a container server whose job is to handle listings of objects and an account server responsible for listings of containers. Swift uses the following OpenStack APIs: Object Storage and Identity APIs. Swift can also be used to store snapshot backups. OpenStack manages these backups thanks to the Cinder module (the Block Storage service). This module is provides extra block level storage to OpenStack instances, but also is the module that manages the snapshot backups which

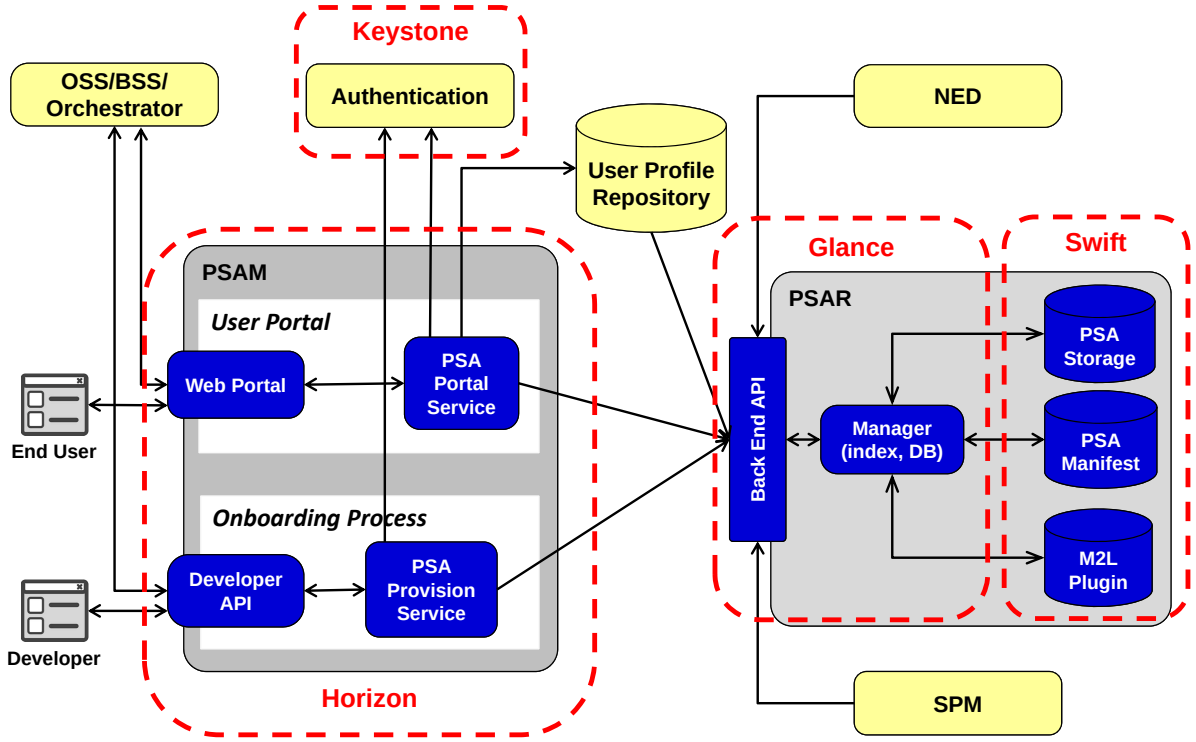

Figure 21: PSAM/PSAR mapping in the OpenStack architecture.

can be a very useful reference for the PSAM/PSAR service.

A brief description of each of the OpenStack APIs mentioned before follows:

Object Storage API. Manage the accounts, containers, and objects in the Object Storage system;

Image Service API. Load images for use by the Compute API and assigns metadata to images too;

Identity API. Get an authentication token that permits access to the OpenStack services RESTful API;

Compute API. Launch virtual machines from images or images stored on persistent volumes;

Networking API. Use virtual networking services among devices managed by the Compute service.

7 Conclusions

This document provides the specification of the PSAM and PSAR interfaces as well as the main processes and workflows where they are involved in the general SECURED architecture.

The specification clearly differentiates between two different management process, one related to the User's PSA, including subscriptions, configuration and remove, and another related to the PSA provisioning PSA. The repository functionality is not only a storage resource but offers full support to other key components of SECURED, as the NED (which needs to obtain the PSAs) or the SPM (which needs to manipulate the policies).

As a result of the specification, a general vision of the PSAM/PSAR and how they fit into the general architecture is achieved. Also detailed information about the different modules and relations is depicted.

In particular the model described support a modular design, offering a highly scalable and distributed solution. Developers with interest in creating their PSAM or PSAR will find a flexible architecture that allows them to work on the processes of their interest.

Guidance for PSAM and PSAR development is provided using some key concepts. The inclusion of ecosystems in the equation has the objective to provide clues about how to move from high-level to low-level design and develop a solution that fits the target network environment.

A possible implementation based on OpenStack is discussed to offer a mature model for the implementation of PSAM/PSAR as a virtualized environment.

A special mention is necessary for the user portal. End users will have this portal as the first point of contact with the platform, when they work in the application-driven model. This specification lists the minimum requirements expected and the general logic to implement, but leaves the door open for any particular implementation preferred by developers.

References

[1] SECURED project, “D2.3.1 – Specification of the SECURED architecture”, September 2014

[2] SECURED project, “D3.1.1 – NED specifications”, June 2014

[3] SECURED project, “D5.1 – Specification of PSA and associated data and interfaces”, September 2014