VPP4ISLANDS

Virtual Power Plant for Interoperable and Smart isLANDS

VPP4Islands

LC-SC3-ES-4-2020

GA 957852

Deliverable Report

Deliverable ID	D5.4	**Version**	1.2
Deliverable name	AAI Infrastructure API Specification and Prototype		
Lead beneficiary	Dominic Heutelbeck (FTK)		
Contributors	Dominic Heutelbeck		
Reviewer	Habib NASSER (RDUIP), E.Metai (BC2050)		
Due date	31/03/2022		
Date of final version	03/05/2022		
Dissemination level	Public		
Document approval	Seifeddine Ben Elghali	**Date**	03/05/2022

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

Acknowledgement: VPP4ISLANDS is a Horizon 2020 project funded by the European Commission under Grant Agreement no. 957852.

REVISION AND HISTORY CHART

Version	Date	Main Author(s)	Summary of changes
1.0	03.12.2021	Dominic Heutelbeck	First Version
1.1	07.12.2021	Dominic Heutelbeck	Incorporate review comments of RDUIP
1.2	03.05.2022	Dominic Heutelbeck	Incorporate review comments from BC2050

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

List of Tables

List of Figures

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

Table 1: List of abbreviations and Acronyms

Abbreviation	Meaning
ABAC	Attribute-based Access Control
ASBAC	Attribute stream-based Access Control
BC2050	Blockchain2050
DLT	Digital Ledger Technologies
DSL	Domain Specific Language
EBAC	Entity-based access control
FTK	FTK Forschungsinstitut für Telekommunikation und Kooperation e.V.
JSON	JavaScript Object Notation
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RAP	Resource Access Point
RBAC	Role-based Access Control
SAPL	Streaming Attribute Policy Language
SSE	Server-Sent Events

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

EXECUTIVE SUMMARY

This Deliverable contains the specifications of the SAPL Policy language and documentation of interfaces for integrating applications using a Server-Sent-Events (SSE)-based web API, and Java applications with a specific focus on the Spring Ecosystem.
The SAPL policy language is a feature-rich extensible domain specific language for expressing access control policies based on the Attribute-Based Access Control (ABAC) or more specific, Attribute Stream-Based Access control (ASBAC). This document provides a thorough introduction into the language, usable as a reference guide during the development of access rights policies.
SAPL is implemented in an Open-Source Authorization Engine which offers several interfaces for integrating external data sources, for realizing embedded or cloud-based policy decision points (PDPs), and for implementing policy enforcement points (PEPs) within applications. The fundamental SAPL engine was brought into the project by FTK as background IP. With Virtual Power Plant as a part of the critical infrastructure of the islands, new requirements arose with regards to validation of correctness of access rights policies. Therefore, a complete testing framework for policies has been developed allowing developers to treat policies the same way they treat application code with regards to testing. This resulted in opportunities for seamless integration of policy authoring and administration in a GitOps style DevOps, Continious Delivery and Integration environment.
For the hybrid authorization approach integrating with the distributed ledger technologies and smart contracts driving the VPP4ISLANDS scenarios together with the other partners, BC2050 has identified Ethereum and LTO in tandem as a platform for the DLT-based services.
This document describes SAPL, key APIs, and the prototypical integration with the Ethereum Blockchain required for adopting SAPL within the VPP4ISLANDS development processes.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

1. INTRODUCTION

In VPP4ISLANDS, the hybrid Authentication and Authorization Infrastructure is a central component of the VPPI-Node, responsible for the authorization of access to the APIs exposed by the node. The AAI implements Attribute Stream-based Access Control (ASBAC) (Heutelbeck, Attribute stream-based access control (ASBAC)-functional architecture and patterns, 2019) to accommodate dynamic requirements of the APIs. By exposing state of the distributed ledger technologies (DLT) all APIs can use the state of smart contracts on the blockchain for making access control decisions without the need for additional DLT intrgrations, if the respective services to not directly interact with the DLT otherwise. This way of allowing for local authorization policies using ASBAC and trusted smart contracts together is what makes the VPP4I AAI hybrid.

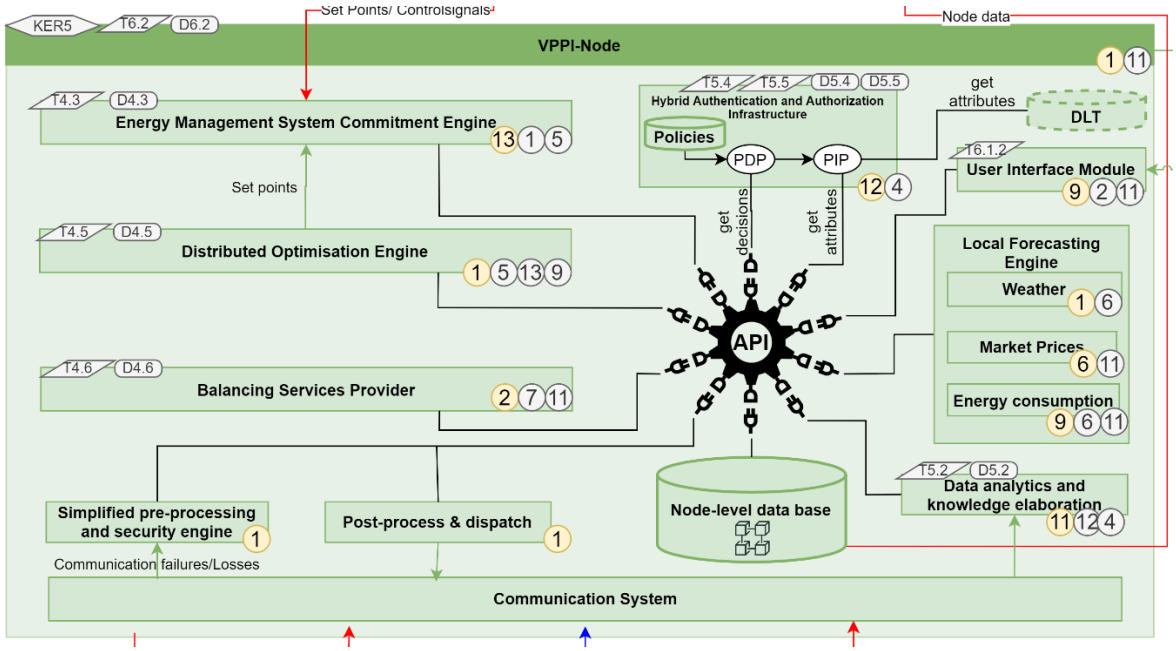

Figure 1: The AAI in the Context of the VPPI-Node (Garner, Jansen, & Dehouche, 2021)

This deliverable is structured as follows, first ASBAC and the SAPL (Streaming Attribute Policy Language) are introduced together with the key system integration APIs for authorization. Then Finally, a general prototype implementation of the DLT/ASBAC integration is discussed.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

2. SAPL

SAPL (Streaming Attribute Policy Language) (Heutelbeck, The Structure and Agency Policy Language (SAPL) for Attribute Stream-Based Access Control (ASBAC), 2019) describes a **domain-specific language (DSL)** for expressing access control policies and a **publish/subscribe protocol** based on JSON. Policies expressed in SAPL describe conditions for access control in applications and distributed systems. The underlying policy engine implements a variant of Attribute-based Access control (ABAC) which enables processing of data streams and follows reactive programming patterns. Namely, the SAPL policy engine implements Attribute Stream-based Access Control (ASBAC).
A typical scenario for the application of SAPL would be a subject (e.g., a user or system) attempting to take action (e.g., read or cancel an order) on a protected resource (e.g., a domain object of an application or a file). The subject makes a subscription request to the system (e.g., an application) to execute the action with the resource. The system implements a **policy enforcement point (PEP)** protecting its resources. The PEP collects information about the subject, action, resource, and potential other relevant data in an authorization subscription request and sends it to a policy decision point (PDP) that checks SAPL policies to decide if it grants access to the resource. This decision is packed in an authorization decision object and sent back to the PEP, which either grants access or denies access to the resource depending on the decision. The PDP subscribes to all data sources for the decision, and new decisions are sent to the PEP whenever indicated by the policies and data sources.

There exist several proprietary platforms dependent or standardized languages, such as XACML, for expressing policies. SAPL brings several advantages over these solutions:

- **Universality**. SAPL offers a standard, generic, platform-independent language for expressing policies.
- **Separation of Concerns**. Applying SAPL to a domain model is relieved from modeling many aspects of access control. SAPL favors configuration at runtime over implementation and re-deployment of applications.
- **Modularity and Distribution**. SAPL allows managing policies in a modular fashion allowing the distribution of authoring responsibilities across teams.
- **Expressiveness**. SAPL provides access control schemata beyond the capabilities of most other practical languages. It allows for attribute-based access control (ABAC), role-based access control (RBAC), forms of entity-based access control (EBAC), and parameterized attribute access and attribute streaming.
- **Human Readability**. The SAPL syntax is designed from the ground up to be easily readable by humans. Basic SAPL is easy to pick up for getting started but offers enough expressiveness to address complex access control scenarios.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

- **Transformation and Filtering**. SAPL allows transforming resources and filtering data from resources (e.g., blacken the first digits of a credit card number or hiding birth dates by assigning individuals into age groups).

SAPL supports **session and data stream-based applications** and offers low-latency authorization for interactive applications and data streams.
SAPL supports **JSON-driven APIs** and integrates easily with modern JSON-based APIs. The core data model of SAPL is JSON offering straightforward reasoning over such data and simple access to external attributes from RESTful JSON APIs.
SAPL supports **Multi-Subscriptions**. SAPL allows bundling multiple authorization subscriptions into one multi-subscription, thus further reducing connection time and latency. The following sections will explain the basic concepts of SAPL policies and show how to integrate SAPL into a Java application easily. Afterward, this document explains the different parts of SAPL in more detail.

2.1 Authorization Subscriptions

A SAPL authorization subscription is a JSON object, i.e., a set of name/value pairs or *attributes*. It contains attributes with the names `subject`, `action`, `resource`, and `environment`. The values of these attributes may be any arbitrary JSON value, e.g.:

```
{
  "subject"     : {
                    "username"    : "alice",
                    "tracking_id" : 1234321,
                    "nda_signed"  : true
                  },
  "action"      : "HTTP:GET",
  "resource"    : "https://vpp4islands.eu/api/meters/123",
  "environment" : null
}
```

Figure 2: Sample Authorization Subscription

This authorization subscription expresses the intent of the user `alice`, with the given attributes, to `HTTP:GET` the resource at `https://vpp4islands.eu/api/meters/123`. This SAPL authorization subscription can be used in a RESTful API, implementing a PEP protecting the API's request handlers.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

2.2 Structure of a SAPL Policy

A SAPL policy document generally consists of:

- the keyword policy, declaring that the document contains a policy (opposed to a policy set; more on policy sets see below)
- a unique (for the PDP) policy name
- the entitlement, which is the decision result to be returned upon successful evaluation of the policy, i.e., permit or deny
- an optional target expression for indexing and policy selection
- an optional where clause containing the conditions under which the entitlement (permit or deny as defined above) applies
- optional advice and obligation clauses to inform the PEP about optional and mandatory requirements for granting access to the resource
- an optional transformation clause for defining a transformed resource to be used instead of the original resource

A simple SAPL policy that allows `alice` to `HTTP:GET` the resource `https://vpp4islands.eu/api/meters/123` , describing a smart meter, could look as follows (in a real-world scenario, this policy is too specific):

```
policy "permit_alice_get_meter123" (1)
permit resource =~ "^https://vpp4islands\.eu/api/meters.*" (2)
where (3)
  subject.username == "alice"; (4)
  action == "HTTP:GET";
  resource == "https://vpp4islands.eu/api/meters/123";
```

Figure 3: Sample Policy 1

1. This statement declares the policy with the name `permit_alice_get_meter123`. The JSON values of the authorization subscription object are bound to the variables subject, action, resource, and environment that are directly accessible in the policy. The syntax `.name` accesses attributes of a nested JSON object.
2. This statement declares that if the resource is a string starting with `https://vpp4i.net/api/meters` (using the regular expression operator =~) and the conditions of the where clause applies, the subject will be granted access to the resource. Note that the `where` clause is only evaluated if the condition of the target expression evaluates to true.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

3. This statement starts the `where` clause (policy body) consisting of a list of statements. The policy body evaluates to true if all statements evaluate to true.

2.3 Authorization Decisions

The SAPL authorization decision to the authorization subscription is a JSON object as well. It contains the attribute decision as well as the optional attributes resource, obligation, and advice. For the introductory sample authorization subscription with the preceding policy, a SAPL authorization decision would look as follows:

```
{
  "decision"    : "PERMIT"
}
```

Figure 4: Sample Authorization Decision

The PEP evaluates this authorization decision and grants or denies access accordingly.

2.4 Accessing Attributes

In many use cases, the authorization subscription contains all the required information for making a decision. However, the PEP is usually not aware of the specifics of the access policies and may not have access to all information required for making the decision. In this case, the PDP can access external attributes. The following example shows how SAPL expresses access to attributes.

Extending the example above, in a real-world application, there will be multiple patients and multiple users. Thus, policies need to be worded more abstractly. In a natural language, a suitable policy could be *Permit operators to HTTP:GET data about any meter*. The policy addresses the profile attribute of the subject, stored externally. SAPL allows to express this policy as follows:

```
policy "operators_get_meter"
permit
  action == "HTTP:GET" &
  resource =~ "^https://vpp4islands\.eu/api/meters/\d*$"
where
  subject.username.<user.profile>.function == "operator";
```

Figure 5: Sample Policy 2

In *line 4* a regular expression is used for identifying a request to any meter's data (operator =~). The authorization subscription resource must match this pattern for the policy to apply.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

The policy assumes that the user's function is not provided in the authorization subscription but stored in the user's profile. Accordingly, *line 6* accesses the attribute user.profile (using an attribute finder step .<finder.name>) to retrieve the profile of the user with the username provided in subject.username. The fetched profile is a JSON object with a property named function. The expression compares it to "operator".

Line 6 is placed in the policy body (starting with where) instead of the target expression. The reason for this location is that the target expression block is also used for indexing policies efficiently and therefore needs to be evaluated quickly. Hence it is not allowed to include conditions that may need to call an external service.

2.5 Reference Architecture

The architecture of the SAPL policy engine is follows the terminology defined by RFC2904 "AAA Authorization Framework".

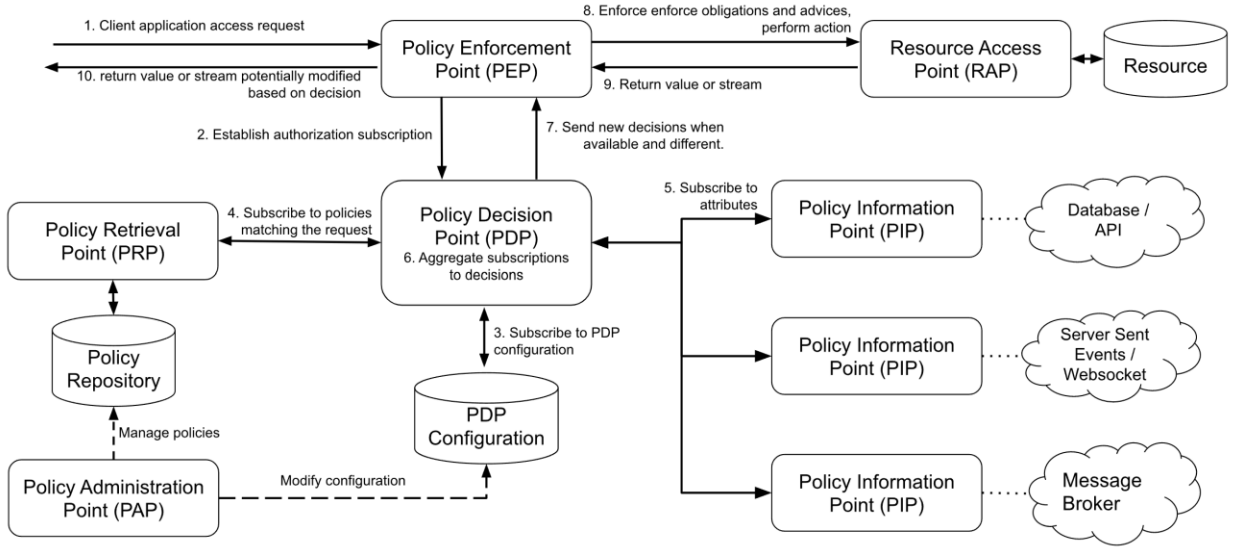

Figure 6: Reference Architecture

2.5.1 Policy Enforcement Point (PEP)

The PEP is a software entity that intercepts actions taken by users within an application. Its task is to obtain a decision on whether the requested action should be allowed and accordingly either let the application process the action or deny access. For this purpose, the PEP includes data describing the subscription context (like the subject, the resource, the action, and other environment information) in an authorization subscription object which the PEP hands over to a

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

PDP. The PEP subsequently receives an authorization decision object containing a decision and optionally a resource, obligations, and advice.

The PEP must let the application process the action if the decision is PERMIT. If the authorization decision object also contains an obligation, the PEP must fulfill this obligation. Proper fulfillment is an additional requirement for granting access. If the decision is not PERMIT or the obligation cannot be fulfilled, the PEP must deny access. Policies may contain instructions to alter the resource (like blackening certain information, e.g., credit card numbers). If present, the PEP should ensure that the application only reveals the resource contained in the authorization decision object.

> A PEP strongly depends on the application domain. SAPL comes with a default PEP implementation using a passed in constraint handler service to handle obligations and advice contained in an authorization decision. Developers should integrate PEPs with the platforms and frameworks they are using. SAPL ships with a set of modules for deep integration with Spring Security and Spring Boot.

2.5.2 Policy Decision Point (PDP)

The PDP has to make an authorization decision based on an authorization subscription object and the access policies it receives from a **Policy Retrieval Point (PRP)** connected to a policy store. Beginning with the authorization subscription object, the PDP fetches policy sets and policies matching the authorization subscription, evaluates them, and combines the results to create and return an authorization decision object. There may be multiple matching policies that might evaluate to different results. To resolve these conflicts, the administrator or developer using a PDP must select a **combining algorithm** (e.g., permit-overrides stating that the decision will be permit if any applicable policy evaluates to permit).

A policy may refer to attributes not included in the authorization subscription object. It will have to obtain them from an external **Policy Information Point (PIP)**. The PDP fetches those attributes while evaluating the policy. To be able to access external PIPs, developers can extend the PDP by adding custom attribute finders. Policies might also contain functions not included in the default SAPL implementation. Developers may add custom functions by implementing **Function Libraries**.

SAPL provides two simple PDP implementations: An **embedded PDP** with an embedded PRP which can be integrated easily into a Java application, and a **remote PDP client** that obtains decisions through a RESTful interface.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

2.5.3 Policy Administration Point (PAP)

The PAP is an entity that allows managing policies contained in the policy store. In the embedded PDP with the Resources PRP, the policy store can be a simple folder within the local file system containing .sapl files. Therefore, any access to files in this folder (e.g., FTP or SSH) can be seen as a straightforward PAP. The PAP may be a separate application or can be included in an existing administration panel.

2.6 Publish / Subscribe Protocol

The PDP receives an authorization subscription from a PEP and sends an authorization decision. Both subscription and decision are JSON objects consisting of name/value pairs (also called attributes) with predefined names. A PEP must be able to create an authorization subscription and process an authorization decision object.

2.6.1 SAPL Authorization Subscription

A SAPL authorization subscription contains attributes with the names subject, resource, action, and environment. Each attribute value can be any JSON value (i.e., an object, an array, a number, a string, true, false, or null).

2.6.2 SAPL Authorization Decision

The SAPL authorization decision contains the attributes decision, resource, obligation, and advice.

Decision

The decision tells the PEP whether to grant or deny access. Access should be granted only if the decision is "PERMIT". The decision attribute can be one of the following string values with the described meanings:

- "PERMIT": Access must be granted.
- "DENY": Access must be denied.
- "NOT_APPLICABLE": A decision could not be made because no policy is applicable to the authorization subscription. The PEP should deny access in this case.
- "INDETERMINATE": A decision could not be made because an error occurred. The PEP should deny access in this case.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

Resource

The PEP knows for which resource it requested access. Thus, there usually is no need to return this resource in the authorization decision object. However, SAPL policies may contain a `transform` statement describing how the resource needs to be altered before it is returned to the subject seeking permission. This can be used to remove or blacken certain parts of the resource document (e.g., a policy could allow doctors to view patient data but remove any bank account details as they can only be accessed by the accounting department). If a policy that evaluates to `PERMIT` contains a `transform` statement, the authorization decision attribute `resource` contains the transformed resource. Otherwise, there will not be a `resource` attribute in the authorization decision object.

Obligation

The value of `obligation` contains assignments that the PEP must fulfill before granting or denying access. As there can be multiple policies applicable to the authorization subscription with different obligations, the `obligation` value in the authorization decision object is an array containing a list of tasks. If the PEP is not able to fulfill these tasks, access must not be granted. The array items can be any JSON value (e.g., a string or an object). Consequently, the PEP must know how to identify and process the obligations contained in the policies. An `obligation` attribute is only included in the authorization decision object if there is at least one obligation.

An authorization decision could, for example, contain the obligation to create a log entry.

In case the obligation is contained in a `DENY` decision, the access must still be denied. An obligation in a `DENY` decision acts like `advice` because the unsuccessful handling of the obligation cannot change the overall decision outcome.

Advice

The value of `advice` is an array with assignments for the PEP as well and works similar to obligations with one difference: The fulfillment of the tasks is no requirement for granting access. I.e., in case the `decision` is `PERMIT`, the PEP should also grant access if it can not fulfill the tasks contained in `advice`. An `advice` attribute is only included in the authorization decision object if there is at least one element within the `advice` array.
In addition to the obligation to create a log entry, a policy could specify the advice to inform the system administrator via email about the access.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

2.6.3 Policy Evaluation

To come to the final decision included in the authorization decision object, the PDP evaluates all existing policy sets and top-level policies (i.e., policies which are not part of a policy set) against the authorization subscription and combines the results. Each policy set and policy evaluates to PERMIT, DENY, NOT_APPLICABLE, or INDETERMINATE (see below). The PDP can be configured with a **combining algorithm** which determines how to deal with multiple results. E.g., if access should only be granted if at least one policy evaluates to PERMIT and should be denied. Otherwise, the algorithm deny-unless-permit could be used.

Available combining algorithms for the PDP are:

- deny-unless-permit
- permit-unless-deny
- only-one-applicable
- deny-overrides
- permit-overrides

The algorithm first-applicable is not available for the PDP since the PDP's collection of policy sets and policies is an unordered set.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

2.6.4 Multi-Subscriptions

SAPL allows for bundling multiple authorization subscriptions into one multi-subscription. A multi-subscription is a JSON object with the following structure:

```
{
  "subjects"                   : ["bs@vpp4islands.eu", "ms@vpp4islands.eu"],
  "actions"                    : ["read"],
  "resources"                  : ["file://vpp4islands/eu/record/islands/Grado",
                                  "file://vpp4islands/eu/record/islands/Bozcaada"],
  "environments"               : [],

  "authorizationSubscriptions" : {
                                   "id-1" : { "subjectId": 0, "actionId": 0, "resourceId": 0 },
                                   "id-2" : { "subjectId": 1, "actionId": 0, "resourceId": 1 }
                                 }
}
```

Figure 7:Multi-Subscriptions - JSON Structure

It contains distinct lists of all subjects, actions, resources, and environments referenced by the single authorization subscriptions being part of the multi-subscription. The authorization subscriptions themselves are stored in a map of subscription IDs pointing to an object defining an authorization subscription by providing indexes into the four lists mentioned before.

The multi-subscription shown in the example above contains two authorization subscriptions. The user `bs@vpp4islands.eu` wants to read the file `file://vpp4islands/eu/record/islands/Grado` , and the user `ms@vpp4islands.eu` wants to read the file `file://vpp4islands/eu/record/islands/Bozcaada`.

The SAPL PDP processes all individual authorization subscriptions contained in the multi-subscription in parallel and returns the related authorization decisions as soon as they are available, or it collects all the authorization decisions of the individual authorization subscriptions and returns them as a multi-decision. In both cases, the authorization decisions are associated with the subscription IDs of the related authorization subscription. The following listings show the JSON structures of the two authorization decision types:

```
{
  "authorizationSubscriptionId" : "id-1",
  "authorizationDecision"       : {
                                    "decision" : "PERMIT",
                                    "resource" : { ... }
                                  }
}
```

Figure 8:Single Authorization Decision with Associated Subscription ID - JSON Structure

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

```
{
  "authorizationDecisions" : {
                               "id-1" : {
                                          "decision" : "PERMIT",
                                          "resource" : { ... }
                                        },
                               "id-2" : {
                                          "decision" : "DENY"
                                        }
                             }
}
```

Figure 9: Multi-Decision - JSON Structure

2.7 PDP APIs

A SAPL PDP must expose a publish-subscribe API for subscribing via the subscription objects laid out above. SAPL defines two specific APIs for that. One is an HTTP Server-Sent Events (SSE) API for deploying a dedicated PDP Server, the other for using a PDP in reactive Java applications. The Java API may be implemented by an embedded PDP or by using the SSE API of a remote server.

2.7.1 HTTP Server-Sent Events API

A PDP to be used as a network service has to implement some HTTP endpoints. All of them accept POST requests and application/json. They produce application/x-ndjson as Server-Sent Events (SSE). A PDP server must be accessed over encrypted TLS connections. All connections should be authenticated. The means of authentications are left open for the organization deploying the PDP to decide or to be defined by a specific server implementation. All endpoints should be located under a shared base URL, e.g., https://pdp.sapl.io/api/pdp/.

A PEP which is a client to the SSE PDP API encountering connectivity issues or errors, must interpret this as an INDETERMINATE decision and thus deny access during this time of uncertainty and take appropriate steps to reconnect with the PDP, using a matching back-off strategy to not overload the PDP.

A PEP must determine if it can enforce obligations before granting access. It must enforce obligation upon granting access at the point in time (e.g., before or after granting access) implied by the semantics of the obligation, and it should enforce any advice at their appropriate point in time when possible.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

Upon subscription, the PDP server will respond with an unbound stream of decisions. The client must close the connection to stop receiving decision events. A connection termination by the server is an error state and must be handled as discussed.

Decide

- URL: {baseURL}/decide
- Method: POST
- Body: A valid JSON authorization subscription
- Produces: A SSE stream of authorization decisions

Multi Decide

- URL: {baseURL}/multi-decide
- Method: POST
- Body: A valid JSON multi subscription
- Produces: A SSE stream of Single Authorization Decisions with Associated Subscription ID JSON Objects

Multi Decide All

- URL: {baseURL}/multi-decide-all
- Method: POST
- Body: A valid JSON multi subscription
- Produces: A SSE stream of Multi Decision JSON Objects

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

Implementations

The SAPL Policy engine comes with two implementations ready for deployment in an organization:

- SAPL Server LT: This light (LT) PDP server implementation uses a configuration and policies stored on a file system.
- SAPL Server CE: This community edition (CE) PDP server implementation uses a relational database (MariaDB) for persistence configuration and offers a convenient graphical Web interface to manage policies, configuration, and clients.

In addition, libraries for embedding the PDP in applications are provided. These components will be described in more detail in D5.5.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

3. The SAPL Policy Language

SAPL defines a feature-rich domain-specific language (DSL) for creating access policies. Those access policies describe when access requests will be granted and when access will be denied. The underlying concept to describe these permissions is an attribute-based access control model (ABAC): A SAPL authorization subscription is a JSON object with the attributes subject, action, resource and environment each with an assigned JSON value. Each of these values may be a JSON object itself containing multiple attributes. Policies can use of Boolean conditions referring to those attributes (e.g., `subject.username == "admin"`).
However, a role-based access control (RBAC) system in which permissions are assigned to a certain role and roles can be assigned to users can be created with SAPL as well.
A full documentation/specification of the language can be found on: https://sapl.io/docs/2.0.0-SNAPSHOT/sapl-reference.html.

4. Testing SAPL policies

The power grid infrastructure controlled by VPP4ISLANDS components and services

The SAPL policy engine provides a framework to test SAPL policies. This framework supports unit tests of a single SAPL document or policy integration test of all SAPL policies of an application via the PDP interface.

4.1 Usage scenarios

With the SAPL test framework, developers can test SAPL policies whether they use SAPL via an embedded PDP in an application or via a central SAPL server.

4.1.1 Embedded PDP

If an application uses an embedded PDP, SAPL policy tests are treated like traditional unit and integration tests. Developers can deploy policy tests alongside these tests and execute them identically via the Maven lifecycle on a local workstation or in a CI pipeline.

4.1.2 SAPL-Server

The following repository GitOps Demo showcases a deployment pipeline with SAPL policy tests in a GitOps-Style for the headless SAPL-Server-LT. Here every change to the policies is introduced via a pull request on the main branch. The CI pipeline executes the policy tests for

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

every pull request and breaks the pipeline run if policy tests are failing. Merging a pull request on the main branch triggers automatic synchronization of the policies to a SAPL-Server-LT instance.

SAPL tests use Java. Therefore, it is impossible to use the SAPL test framework when deploying SAPL-Server-Implementations with GUI-based PAP (i.e., SAPL-Server-CE or SAPL-Server-EE).

4.2 Unit-Tests

SAPL tests use JUnit for executing SAPL unit test cases. Each test is prepared by creating SaplUnitTestFixture. This can be done in the @BeforeEachStep of a JUnit test case.

The SaplUnitTestFixture defines the name of the SAPL document under test or the path to its file. In addition, the fixture sets up PIPs and FunctionLibrarys to be used during test execution.

```
private SaplTestFixture fixture;

  @BeforeEach
  void setUp() throws InitializationException {
      fixture = new SaplUnitTestFixture("policyStreaming")
              //.registerPIP(...)
              .registerFunctionLibrary(new TemporalFunctionLibrary());
  }
```

Figure 10: Test fixture setup

4.3 Policy-Integration-Tests

Instead of testing a single SAPL document, all policies can be tested together using the PDP interface, just like when an application uses an embedded PDP or a SAPL server.

The SaplIntegrationTestFixture manages these kinds of integrations tests.

```
private SaplTestFixture fixture;

  @BeforeEach
  void setUp() {
      fixture = new SaplIntegrationTestFixture("policiesIT")
          .withPDPPolicyCombiningAlgorithm(
              PolicyDocumentCombiningAlgorithm.PERMIT_UNLESS_DENY
          );
  }
```

Figure 11: Test fixture setup for integration tests

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

4.4 Writing test cases

The Step-Builder-Pattern is used for defining the concrete test case. It consists of the following four steps:

1. Given-Step: Define mocks for attributes and functions
2. When-Step: Specify the AuthorizationSubscription
3. Expect-Step: Define expectations for generated AuthorizationDecision
4. Verify-Step: Verify the generated AuthorizationDecision

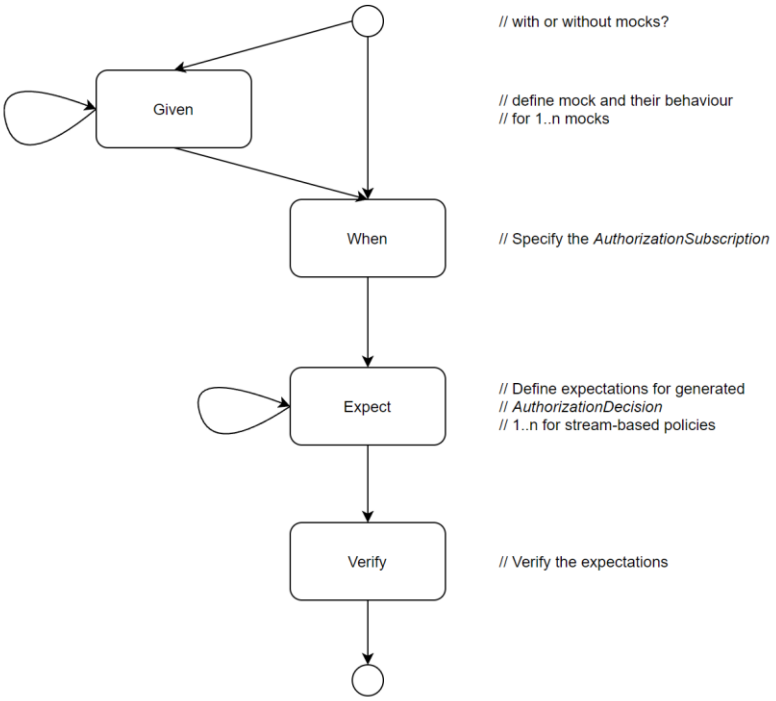

Figure 12: Test structure

Starting with `constructTestCaseWithMocks()` or `constructTestCase()` called on the fixture, the test case definition process is started at the Given-Step or the When-Step.
A lot of examples showcasing the various features of this SAPL test framework can be found in the demo project here.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

4.5 Code-Coverage Reports via the SAPL-Maven-Plugin

For measuring the policy code coverage of SAPL policies, developers can use the sapl-maven-plugin to analyze the coverage and generate reports in various formats.
Currently, three coverage criteria are supported:

1. **PolicySet Hit Coverage**: Measures the percentage of PolicySets that were at least once applicable to an AuthorizationSubscription in the tests.
2. **Policy Hit Coverage**: Measures the percentage of Policies that were at least once applicable to an AuthorizationSubscription in the tests.
3. **Condition Hit Coverage**: Measures the percentage of conditions evaluated to true or false during the tests. The number of conditions times two is compared with the number of positively and negatively evaluated conditions.

5. Hybrid Authorization Prototype

In preparation of the integration of the Authorization processes a prototype has been developed to demonstrate the bridging between SAPL authorization and smart contracts using Ethereum.

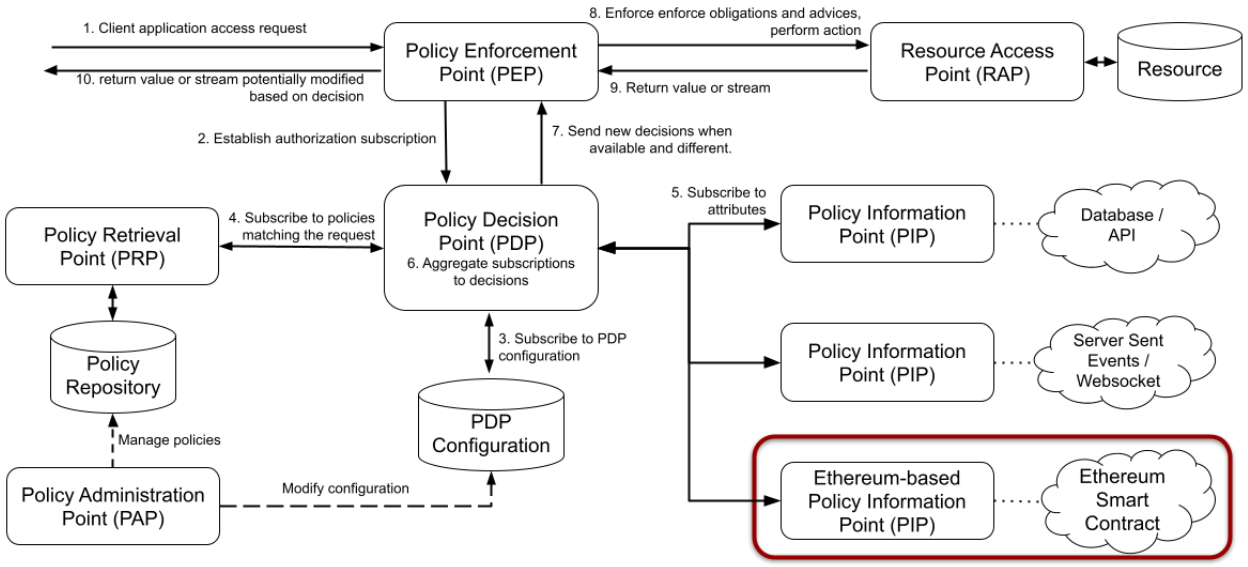

Figure 13: Connecting Ethereum as a Policy Information Point

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

The goal of the integration is to enable the formulation of access control polices referencing the state of the processes implemented in the smart contracts. To achieve this goal, properties of the process related to subjects, actions, resources, and the environment must be accessible within policies as attributes.

There are two pathways to making attributes accessible. First, the developer of the application with a PEP can provide the attributes as part of the authorization subscription. This approach requires the application to have full knowledge of the smart contracts state at time of access control. In this case the attributes cannot be dynamic and will remain constant throughout the lifetime of the subscription. Second, the PDP can be extended with dedicated policy information points to access the smart contract's state on demand. This has the advantage that attributes from smart contracts can be dynamic and application developers are not required to know which data from smart contracts is required for policy enforcement.

The first approach is supported by SAPL out-of-the-box and does not require any additional development. For the second approach with attributes dynamically sourced from smart contracts a generic Ethereum PIP has been developed, exposing Ethereum APIs via SAPL attributes. This generic PIP is available on GitHub: https://github.com/heutelbeck/sapl-extensions/tree/main/sapl-ethereum. Figure 13 provides an overview of the API and attributes exposed.

This approach has the drawback, that the policies will be primarily of technical nature, and the domain language may be lost, making policies harder to write, understand and maintain. Ideally, an organization using smart contracts in a SAPL-driven authorization process should implement dedicated domain-specific PIPs, exposing attributes carrying the semantics of the smart contract itself. A prototype for such in integration has been developed and is also available on GitHub: https://github.com/heutelbeck/sapl-demos/tree/master/sapl-demo-ethereum. Here code for the domain-specific PIP is generated directly from the solidity source code and wrapped in a PIP service which can be linked to the PDP. This allows for a shared codebase between smart contract authors and authorization infrastructure, making maintenance of the PIPs more efficient and reliable.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

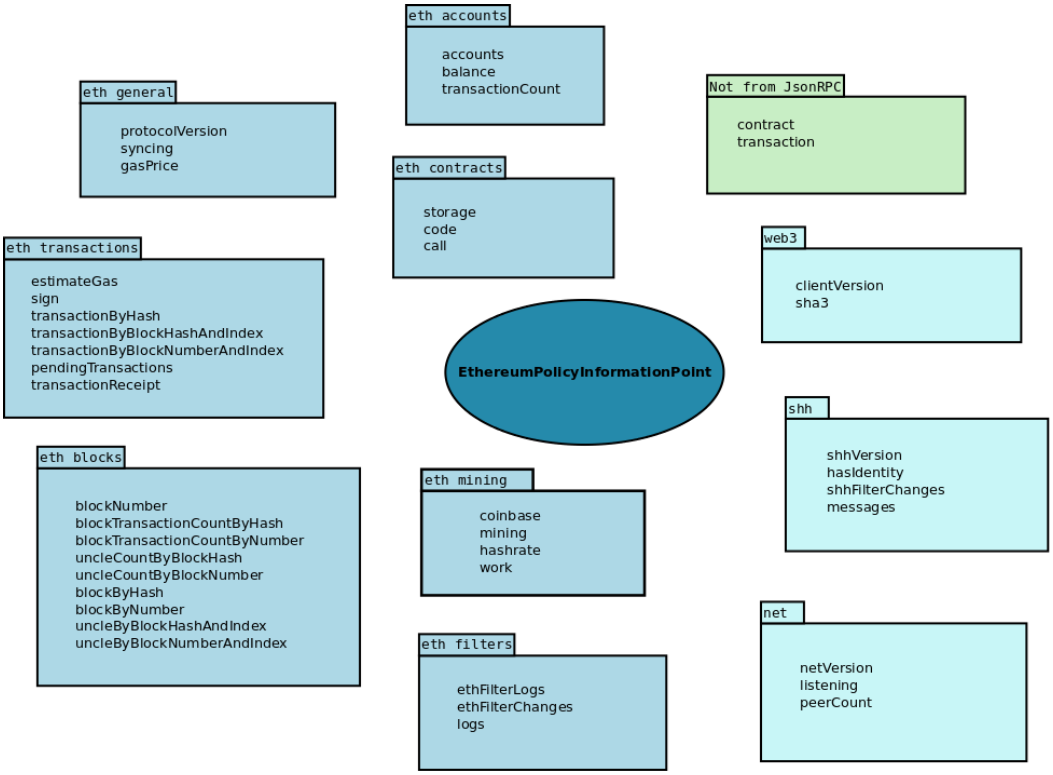

Figure 14: The Generic Ethereum PIP API

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

6. Conclusion

This deliverable described the overall authorization infrastructure of VPP4ISLANDS and the core APIs of the SAPL policy engine.
At time of writing, the APIs and Smart Contracts are not fully defined yet. However, it is important to first provide a stable platform for the cross-cutting authorization concerns which can be adopted during API specification and policy definition. The APIs and components of the hybrid authorization. Code examples and extensive documentation for all parts of the authorization environment have been produced to allow easy adoption by the partners.
In addition, the development of the proof-of-concept prototype realizing a true hybrid authorization bridging local authorization and potentially global DLTs with smart contracts on the base of Ethereum has underlined the feasibility of the approach laid out in the proposal.
VPP4I can nor realize dynamic process-driven authorization to comply with the data protection demands of all stakeholders by using the described approaches.
The next step is to train API developers of the project in the usage of the technologies (SAPL, DLT). A first training event has been held by FTK in October 2021 and is now available for all partners.
Furthermore, the Services of the VPP Node are beginning to take shape it is now important to train the respective developers and to identify the specific protection needs of the sub-domains. I.e., the Energy Management System Commitment Engine, Distributed Optimization Engine, Balancing Service Provider, User Interface Module, Data Analytics, the Local Forecasting Engine, and APIs exposing Node-level DB Information.
The key obstacle in achieving full integration of the authorization services and smart contracts will be the reluctance of developers to adopt external software, as often authorization and other security considerations are handled as an afterthought by developers which focus on implementing their core domain processes. Also, the “not invented here” phenomenon is often still prevalent. Therefore, in a first step the training efforts are key. Second, ensuring the actual adoption of the services is more of a management issue than it is a technical one which has to be addressed by establishing matching processes.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852

7. REFERENCES

Garner, R., Jansen, G., & Dehouche, Z. (2021). *D2.4 - Concept Definition.*

Heutelbeck, D. (2019). Attribute stream-based access control (ASBAC)-functional architecture and patterns. *International Conference on Security and Management (SAM)*, (pp. 182-188).

Heutelbeck, D. (2019). The Structure and Agency Policy Language (SAPL) for Attribute Stream-Based Access Control (ASBAC). *International Workshop on Emerging Technologies for Authorization and Authentication*, (pp. 52-68).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement n°957852