

Grant Agreement No.: 101070473
Call: HORIZON-CL4-2021-DATA-01
Topic: HORIZON-CL4-2021-DATA-01-05
Type of action: HORIZON-RIA



MIDTERM REPORT

Revision: v.0.2

Subproject Name	XADATU
Authors	Rafael Marín Pérez (ODINS), Alejandro Arias Jiménez (ODINS)
Submission date	dd/mm/yyyy
Reviewers	FLUIDOS Internal

DOCUMENT REVISION HISTORY

Version	Date	Description of change	List of contributor(s)
V0.1	13/08/2024	1st version of the template for comments	Rafael Marín Pérez (ODINS), Alejandro Arias Jiménez (ODINS)
V0.2	24/09/2024	Revised document with changes mentioned by UMU	Rafael Marín Pérez (ODINS), Alejandro Arias Jiménez (ODINS)



DISCLAIMER

The information, documentation and figures available in this deliverable are written by the Beneficiaries of the "Flexible, scaLable and secUre decentralizeD Operation" (FLUIDOS) project's Financial Support to Third Parties scheme under EC grant agreement 101070473 and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

COPYRIGHT NOTICE

© 2022 - 2025 FLUIDOS Consortium





TABLE OF CONTENTS

1 Introduction	7
1.1 Overview and objectives of xadatu project	7
1.2 Relevance and benefits to fluidos	8
1.3 Proposed solution	9
2 Technical Approach	10
2.1 Technology Selection	10
2.1.1 Extensible Access Control Markup Language	10
2.1.2 Hyperledger Fabric	13
2.2 Current status, developments and integrations	15
2.2.1 XACML Policies and Attributes	16
2.2.2 XACML Authorization Process	19
2.2.3 XACML Accounting	21
3 Conclusions and Future Work	22
4 References	23



LIST OF FIGURES

<i>Figure 1 FLUIDOS Overall architecture</i>	8
<i>Figure 2 XACML architecture</i>	11
<i>Figure 3 XACML Policy elements structure</i>	11
<i>Figure 4 Hyperledger Fabric</i>	13
<i>Figure 5 Verification and Authorization flow with ODINS updates</i>	15
<i>Figure 6 PAP main web page view</i>	16
<i>Figure 7 PAP attributes management web page view</i>	17
<i>Figure 8 PAP policies management web page view</i>	18
<i>Figure 9 JSON structure of policies stored in Blockchain</i>	19
<i>Figure 10 JSON structure of the authorization request stored in Blockchain for accounting</i>	21
<i>Figure 11 JSON structure for storing XACML Attributes in Hyperledger Fabric</i>	25
<i>Figure 12 JSON structure for storing XACML Policies in Hyperledger Fabric</i>	25
<i>Figure 13 Sequence diagram for registering XACML Policies/Attributes in Hyperledger Fabric</i>	26
<i>Figure 14 Sequence diagram for updating XACML Policies/Attributes in Hyperledger Fabric</i>	26
<i>Figure 15 Sequence diagram for querying XACML Policies/Attributes for a specific domain from Hyperledger Fabric</i>	27
<i>Figure 16 Sequence diagram for querying all XACML Policies/Attributes for all domains from Hyperledger Fabric</i>	28
<i>Figure 17 Sequence diagram for registering XACML authorization requests in Hyperledger Fabric</i>	29
<i>Figure 18 Sequence diagram for querying a specific XACML authorization request from Hyperledger Fabric</i>	29
<i>Figure 19 Sequence diagram for querying all XACML authorization requests between two dates from Hyperledger Fabric</i>	30

LIST OF TABLES

<i>Table 1 Example of a request to the XACML to obtain a Verdict</i>	20
<i>Table 2 Example of a successful Verdict response from XACML</i>	20

ABBREVIATIONS

ABAC	Attribute-Based Access Control
DLT	Distributed Ledger Technology
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
PAP	Policy Administration Point
PEP	Policy Enforcement Point
PDP	Policy Decision Point
REAR	Resource Acquisition Manager
VDR	Verifiable Data Registry
VP	Verifiable Presentation
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language

1 INTRODUCTION

This midterm-report compiles the ongoing contributions and current state of T.2 'Design and Developing' of the XADATU Project for the FLUIDOS Open Call. The information provided during this first phase of the project remains relevant for understanding the project's strategic approach and the foundational technologies that inform each component's development. Furthermore, this mid-report is a self-contained document that presents the current selection of technologies, an overview of the progress made in implementing key components, and an updated view of the project's overall status.

1.1 OVERVIEW AND OBJECTIVES OF XADATU PROJECT

XADATU is a project that aims to create, execute, and include decentralized authorization and accounting for resources dispersed geographically within the GAIA-X federated architecture of the FLUIDOS project. Moreover, to address a priority issue in the FLUIDOS Open Call, the XADATU Project is a Technology Extension (TE) that will improve the FLUIDOS architecture, shown in *Figure 1*, by introducing decentralized authorization and accounting functionality. Therefore, in a highly dynamic and dispersed Edge/Cloud situation, XADATU seeks to provide a safe and scalable solution for resource access control and accounting by imposing domain-specific access control policies by the FLUIDOS Project's current implementation of the zero-trust paradigm. To summarise, ODINS will specifically make use of the Verifiable Data Registry (VDR) to expand the present features with the accountability of the access control activities via the storage in distributed ledgers, e.g., Hyperledger Fabric. A distributed authorization system based on standards such as the Attribute-Based Access Control (ABAC) [1] model and Extensible Access Control Markup Language (XACML) [2] policy language will also be implemented by ODINS by extending the Privacy and Security Manager.

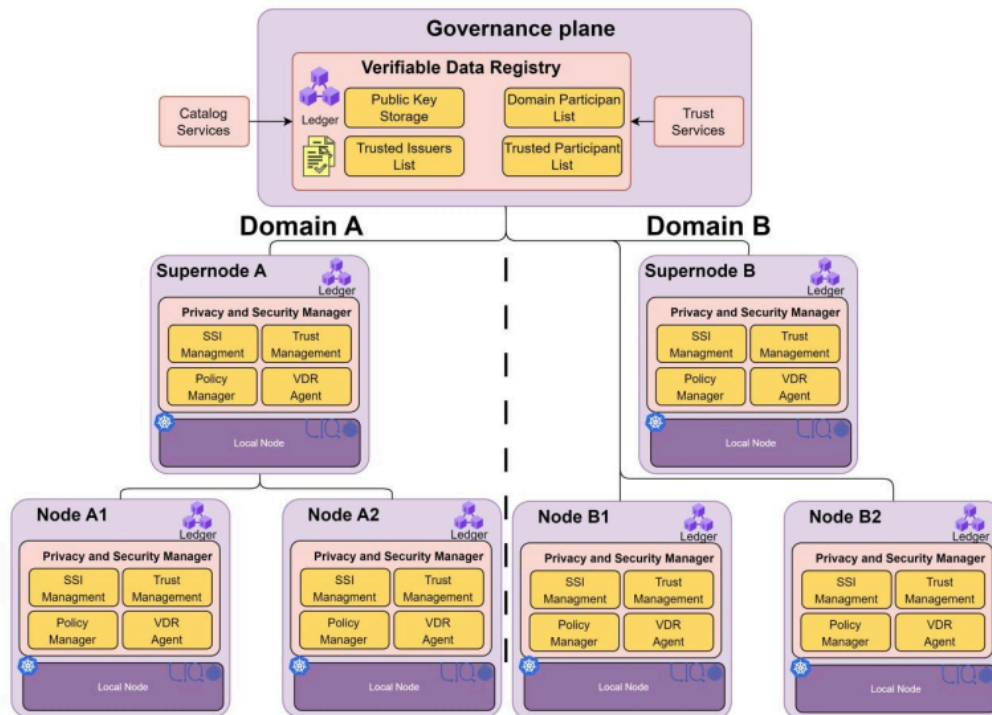


Figure 1 FLUIDOS Overall architecture

1.2 RELEVANCE AND BENEFITS TO FLUIDOS

The technical expansion of the FLUIDOS architecture with distributed authorization and accounting within the GAIA-X federated framework is the primary advantage of adopting the XADATU proposal. In practical terms, XADATU will offer fine-grained access control to improve security by limiting authorised user's access to dispersed resources, reducing risks, and guaranteeing adherence to laws like the General Data Protection Regulation (GDPR). Additionally, ABAC optimises resource use by enabling the accurate assignment of permissions according to roles and data sensitivity. Also, in accordance with the FLUIDOS project's present implementation of the zero-trust paradigm, domain-specific access control rules will be made possible using XACML-based policy access management. Furthermore, the XADATU proposal allows seamless interoperability with the current identity management mechanism, such as Decentralized Identifiers [3], Verifiable Credential [4], and OpenID Connect [5] implemented in the FLUIDOS ecosystem to extend the zero-trust paradigm to control efficiently and securely the authorization and accounting of highly distributed resources of dynamic Edge/Cloud deployments in Distributed Ledger Technologies (DLT) [6] systems, such as Hyperledger Fabric, which enables forensic analysis of resource utilisation and tracking of access events.

1.3 PROPOSED SOLUTION

This XADATU project will design, develop, and integrate the required components within the FLUIDOS architecture responsible for providing access control policies management, decentralized fine-grained security policy decisions, and distributed ledger accounting of highly distributed resources access.

The new features will be created and incorporated into the VDR, Resource Acquisition Manager (REAR), Privacy/Security Manager, and other FLUIDOS components. By extending the REAR, the FLUIDOS module in charge of the negotiation process, XADATU will provide access to resources and services from distant FLUIDOS nodes. Specifically, XADATU will feature a Distributed Authorization Engine that will act as a lightweight Policy Enforcement Point (PEP) to provide authorization in accordance with XACML and ABAC standards. Also, decisions made by the Security Manager will be translated into standards-based Extensible Markup Language (XML) [7] documents that can be consumed and transported across many different FLUIDOS architecture components, such as Edge nodes and Cloud supernodes. This enables a PEP to request a highly distributed authorization decision-making process from the Policy Decision Point (PDP). Furthermore, Policy Administration Points (PAP) for the management of security policies that can be generated independently in various FLUIDOS domains will be designed and integrated by XADATU. Moreover, the XADATU solution will make use of the decentralized storage of the distributed ledger, or Hyperledger Fabric, which is a component of the Governance plane of the FLUIDOS ecosystem, to store and account for the security policies and authorization decisions.

The XADATU solution allows extending the decentralized FLUIDOS architecture to ensure that distributed entities (i.e. nodes and supernodes) have access control over their own resources. The solution will enable evaluating access requests against predefined policies and generating any session-specific authorisation token used by a concrete client to access a specific resource. XADATU will use distributed ledgers as storage infrastructure to avoid using centralised systems in the cloud that would generate unwanted dependencies.

By leveraging the XADATU solution, the security user administrator will use the XADATU extension of VDR with the PAP to define the security authorisation policies linked with the distributed identifiers for the security policy management with DTL storage. XADATU will implement and integrate the PEP in the REAR. The PEP will be an access control system responsible for enforcing access decisions made by the PDP. Moreover, the PDP will be integrated into the Privacy/Security Manager to enable evaluating access requests against predefined policies and generating any session-specific authentication token or credential used by a client to access a resource.

2 TECHNICAL APPROACH

This section depicts the technological framework and developments under the FLUIDOS Open Call project. It includes a detailed examination of the XACML for defining and managing access control policies and the DLT used for secured and distributed storage of policies and authorization requests. An overview of the current developments, technologies in use, and their implementations will be provided, illustrating how these components are integrated and contribute to the advancement of the project's objectives.

2.1 TECHNOLOGY SELECTION

This subsection provides the State of the Art of the technologies used for the developments in the XADATU Project. Also, it outlines how these technologies are employed to address specific requirements of the project, highlighting their roles in enhancing access control and ensuring secure, transparent policy management.

2.1.1 Extensible Access Control Markup Language

XACML is a standard, declarative, and XML-based language to define access control policies, which allows specifying the set of subjects that can perform certain actions on a specific set of resources based on their attributes. As shown in *Figure 2*, the XACML architecture comprises:

- The PAP manages/configures access control policies and stores them, following the XACMLv2 specification, in the XACML Policies file. For this goal, this component offers a web frontend to the system administrator.
- The PDP will access the XACML Policies file when an authorization request is received and decide if it must respond with a positive or negative verdict.

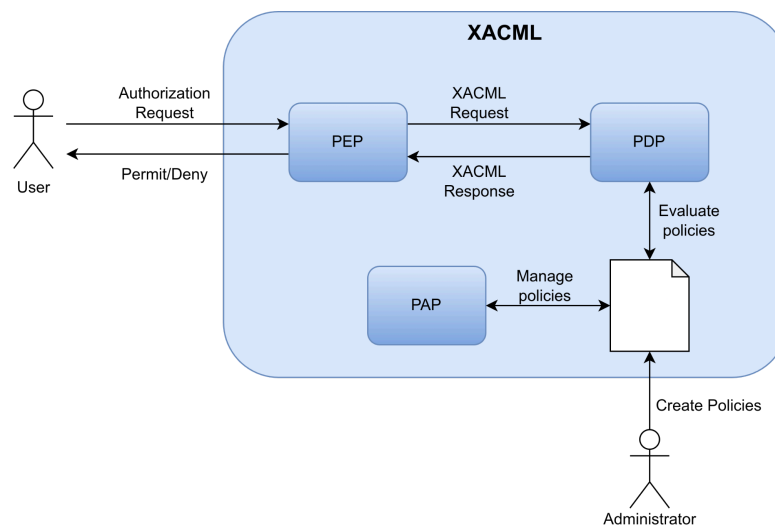


Figure 2 XACML architecture

The definition of access control policies is mainly based on three elements in the XACML data model: PolicySet, Policy, and Rule (*Figure 3*). A PolicySet may contain other PolicySets and Policies, whereas a Policy includes a set of Rules, specifying an Effect (Permit, Deny, or Not Applicable), because of applying that Rule for a particular request.

- **Rule:** contains a Boolean expression that can be evaluated in isolation, but that is not intended to be accessed in isolation by a PDP. So, it is not intended to form the basis of an authorization decision by itself. It is intended to exist in isolation only within an XACML PAP, where it may form the basic unit of management.
- **Policy:** contains a set of *Rule* elements and a specified procedure for combining the results of their evaluation. It is the basic unit of the policy used by the PDP, and so it is intended to form the basis of an authorization decision.
- **PolicySet:** contains a set of *Policy* or other *PolicySet* elements and a specified procedure for combining the results of their evaluation.

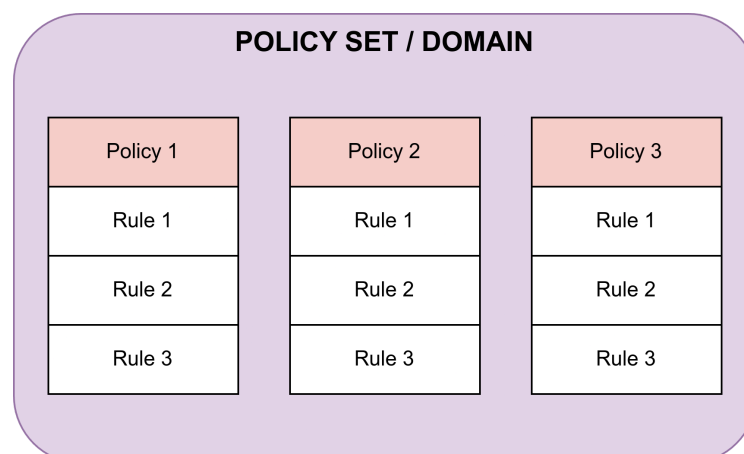


Figure 3 XACML Policy elements structure

The Target sections of these elements define the set of attributes from resources, subjects, actions, and environment to which the PolicySet, Policy, or Rule are applicable. Moreover, since different Rules might be applicable under a specific request, XACML defines Combining Algorithms to reconcile multiple decisions. In addition, a set of obligations (Obligations class) can be used to notify a set of actions to be performed related to an authorization decision.

In the XACML data model, rules are the fundamental building blocks used to determine whether a particular access request should be allowed or denied. Each rule has one of three possible effects: *Permit*, *Deny*, or *Not Applicable*.

The *Permit* effect is applied when an authorization request matches a rule that explicitly allows the action. This means that all the conditions specified within the rule are met, such as the attributes of the subject, the attributes of the resource being accessed, and the specific action being performed. When a rule's effect is *Permit*, the PDP concludes that the requested action is authorised and should be permitted. On the other hand, the *Deny* effect is triggered when an authorization request matches a rule that explicitly disallows or prohibits the action. This effect indicates that the conditions of the rule are met but the policy dictates that the action should not be allowed. A *Deny* decision prevents the requested action from taking place, ensuring that access is restricted according to the policy definitions.

The third possible effect, *Not Applicable*, occurs when an authorization request does not match any rule in the policy set. In this situation, the PDP cannot apply any of the rules because none of them meet the conditions specified in the request. Consequently, no explicit authorization decision is made, which can lead to a default behaviour being applied, such as denying access by default if no applicable rule is found.

In addition to these effects, XACML also supports combining algorithms, which are used to reconcile multiple rules' outcomes within a single policy. For instance, if multiple rules match a request with conflicting outcomes (some permitting and others denying), a combining algorithm will determine how these conflicts should be resolved — whether by giving precedence to *Permit* decisions, *Deny* decisions, or some other method (such as "deny-overrides" or "permit-overrides"). At this moment, the only algorithm that is implemented in the XACML component is the 'first-applicable' algorithm, which determines which rule to apply by selecting the first match that occurs while checking the rules. With this algorithm, the rules that you define first will have more priority than other rules defined later, so it is crucial to organise the order of the rules from more important to less important, so in the case there is a conflict between two or more rules, the rule that was defined first will be selected.

Together, these effects and combining algorithms make XACML a powerful and flexible framework for defining fine-grained access control policies, enabling organisations to precisely manage who can perform what actions on which resources under specific conditions.

2.1.2 Hyperledger Fabric

Hyperledger Fabric¹ is an open-source enterprise-grade permissioned DLT platform designed for use in enterprise contexts that delivers some key differentiating capabilities over other popular distributed ledger or blockchain platforms. One of the main features of Hyperledger Fabric is its modular and configurable architecture, enabling innovation, versatility, and optimization for a broad range of industry use cases. Another key feature is that Fabric is the first distributed ledger platform to support smart contracts authored in general-purpose programming languages such as Java, Go, and Node.js. The Fabric platform is permissioned, meaning that the participants are known to each other, rather than anonymous, and therefore fully untrusted as with Bitcoin or Ethereum. Another platform differentiator is its support for pluggable consensus protocols, such as crash fault-tolerant (CFT) when Fabric is deployed within a single enterprise or byzantine fault tolerant (BFT) when Fabric is deployed in a multi-party, decentralized use case. Also, Fabric can leverage consensus protocols that do not require a native cryptocurrency, thus reducing significant processing and transaction confirmation latency and the absence of cryptographic mining operations. In *Figure 4* the following elements can be observed:

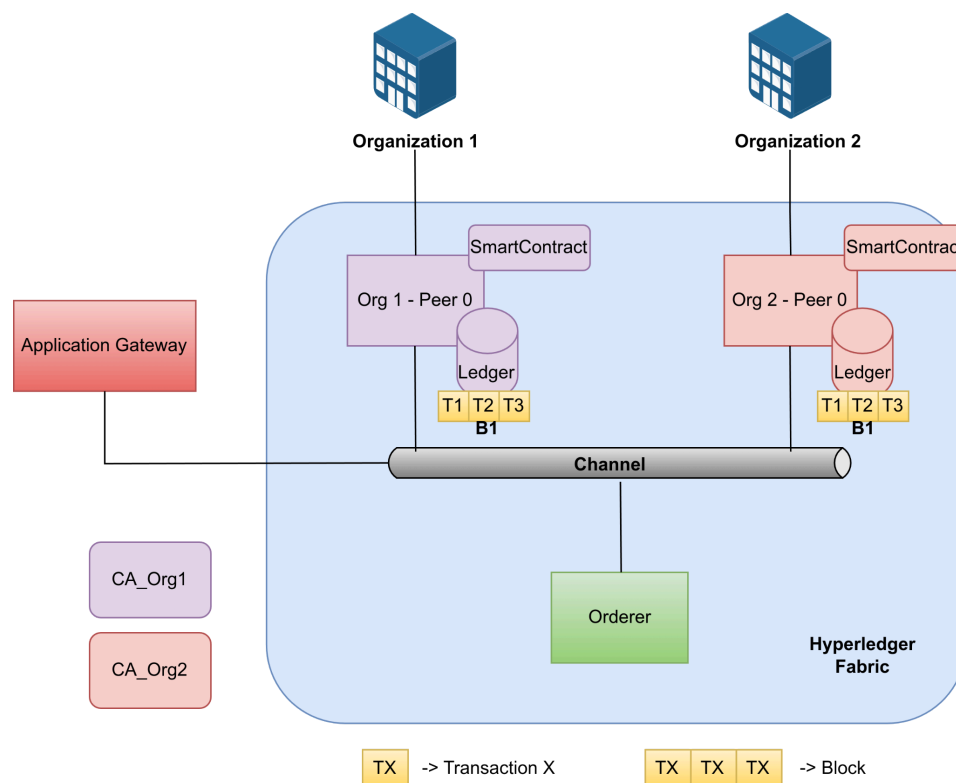


Figure 4 Hyperledger Fabric

- **Organisation:** An organisation is a member that has been added to the blockchain network invited by a blockchain network provider. An organisation is joined to the network by adding its Membership Service Provider (MSP) to the network. The MSP

¹ Hyperledger Fabric documentation | <https://hyperledger-fabric.readthedocs.io/en/release-2.5/whatis.html>

defines how other members of the network may verify that signatures were generated by a valid identity. The CAs are the entities responsible for issuing and managing cryptographic certificates for an organisation.

- **SmartContract/Chaincode:** A smart contract is code — invoked by a client application external to the blockchain network — that manages access and modifications to a set of key-value pairs in the World State via a Transaction. A smart contract defines the transaction logic that controls the lifecycle of a business object contained in the world state. In Hyperledger Fabric, smart contracts are packaged as chaincode. Then, the chaincode is installed on peers and then defined and used on one or more channels. Also, note that the smart contract can be defined within the same chaincode. When a chaincode is deployed, all smart contracts within it are made available for applications.
- **Channel:** A channel is a private blockchain overlay that allows for data isolation and confidentiality. Each channel has a completely separate world state, but applications and smart contracts can communicate between channels so that ledger information can be accessed between them. A channel-specific ledger is shared across the peers in the channel, and transacting parties must be authenticated to interact with it.
- **Ledger:** A ledger is formed by two different but related parts - a blockchain and the "state database", also known as "world state". Blockchains are immutable, e.g., once a block is added to the chain, it cannot be changed. In contrast, the "world state" is a database containing the current value of the set key-value pairs that have been added, modified, or deleted by the set of validated and committed transactions in the blockchain. Smart contracts primarily put, get, and delete states in the world state, and can also query the immutable blockchain record of transactions.
- **Peer:** Peers are one of the main elements of Hyperledger Fabric networks. This is because peers are network entities that maintain a ledger and run chaincode containers in order to perform read/write operations to the ledger. Also, peers are flexible and redundant elements that can be created, started, stopped, reconfigured, and deleted. Another key characteristic is the Fabric Gateway service, which allows peers to expose a set of APIs that enable client applications to interact with the services that peers provide.
- **Orderer:** The orderer is a node that receives and processes transactions, creating blocks, and broadcasting them to the network. Also, these nodes enforce basic access control for channels, restricting who can read and write data to them, and who can reconfigure them. Along with other orderer nodes, they form an ordering service.
- **Application:** An application is an external element of the Hyperledger Fabric network that can interact with the network by submitting transactions to a ledger or querying ledger content using the Fabric Gateway. The process that involves the interaction between a client application, the gateway service running on a peer, orderer nodes, and additional peers is divided into 3 phases: Phase 1 - Transaction Proposal and Endorsement, Phase 2 - Transaction Submission and Ordering, and Phase 3 - Transaction Validation and Commitment.

- **Transaction:** A transaction is created when a chaincode is invoked from a client application to read or write data from the ledger. The transaction represents an operation or a set of operations that are executed on the blockchain network. Transactions are very important to understanding how clients' applications interact with the blockchain ledger and how they can update the state of the ledger. Hyperledger Fabric uses the transaction model, which ensures that only valid transactions endorsed by the required parties are added to the blockchain. This allows the integrity and consistency of the ledger across the network while also providing privacy.

2.2 CURRENT STATUS, DEVELOPMENTS AND INTEGRATIONS

This subsection provides an overview of the ongoing progress, recent developments, and current integrations within the FLUIDOS Open Call project. It highlights the implementation and status of critical components such as the XACML and its interaction with the DLT. The integration of XACML authorization processes and blockchain storage not only enhances security and transparency but also supports efficient management and querying of authorization requests. This comprehensive update outlines how these elements contribute to the project's overall functionality and effectiveness, reflecting the advancements made and their impact on system performance and compliance. All developments have been made in the FLUIDOS WP5 GitHub repository². The green box of *Figure 5* shows the new integrations added by ODINS to the credential verification flow.

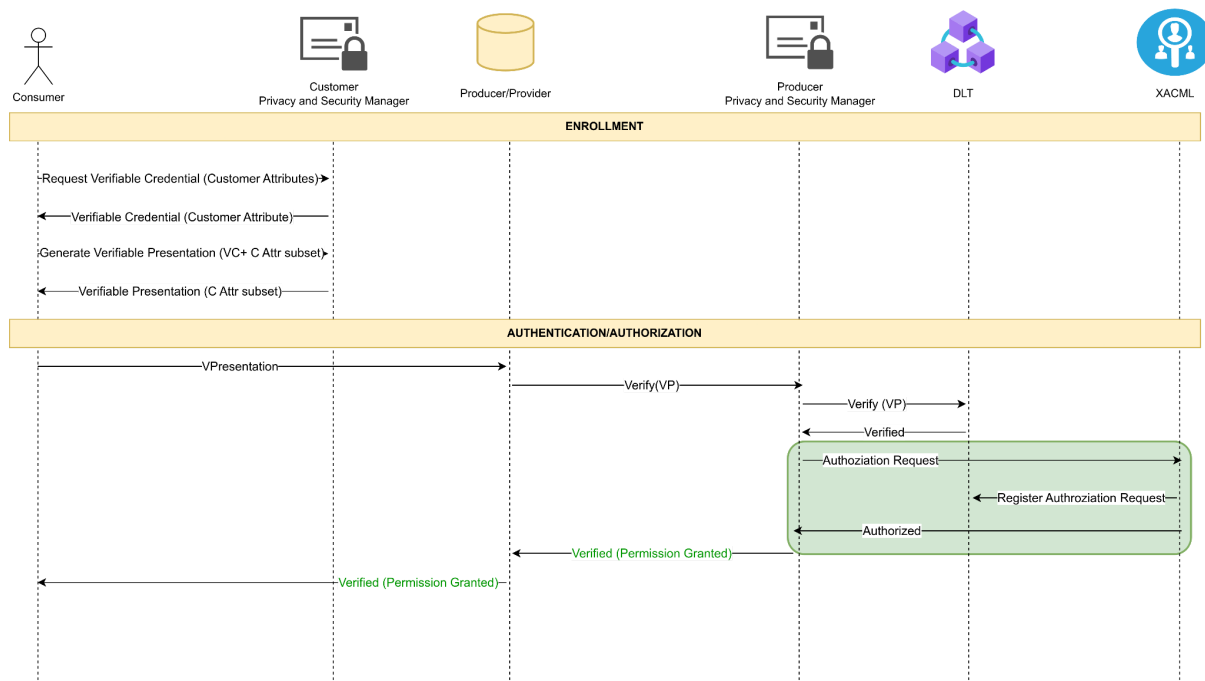


Figure 5 Verification and Authorization flow with ODINS updates

² <https://github.com/fluidos-project/idm-fluidos-aries-framework-go>

2.2.1 XACML Policies and Attributes

The XACML PAP provides a very simple web application from which it is possible to create, modify, and delete policies and attributes. The person responsible for interacting with the PAP should be the system administrator, and no one else should have access to manage the XACML policies and attributes. The frontend of the PAP can be divided into three parts or views:

1. The first part is the main page, as shown in *Figure 6*. In this view, domains can be created to separate groups of policies and attributes for different contexts and use cases, allowing for independent policies for each scenario. Additionally, creating a domain is the initial step required before attributes and policies can be defined. From this page, it is also possible to access the other two sections of the web application: attribute management and policy management.

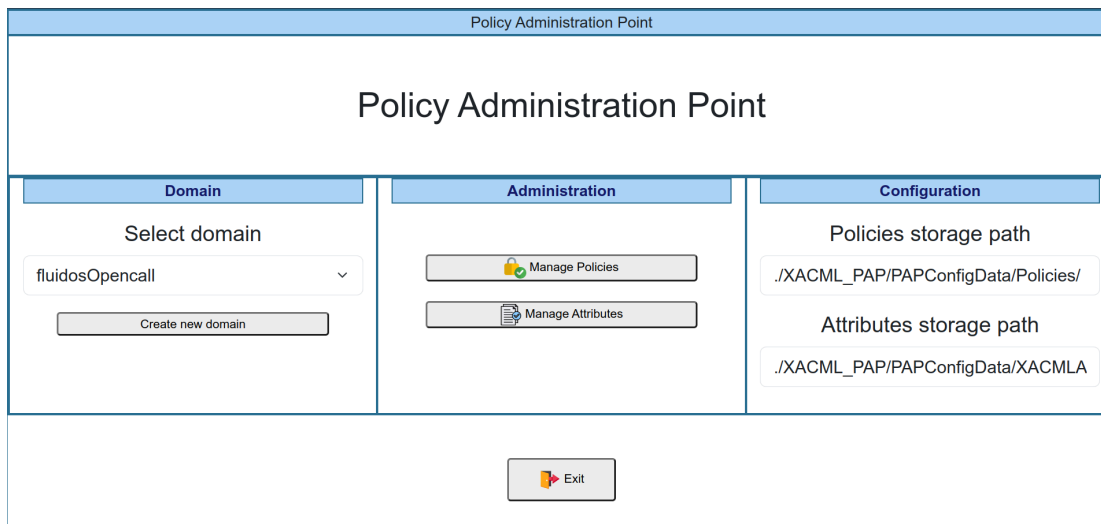


Figure 6 PAP main web page view

2. The second part of the web application is the attribute creation view. As shown in *Figure 7*, this section is divided into three parts, corresponding to each type of attribute supported by XACML. In this view, attributes to be used by the policies and rules can be created, modified, and deleted. All attributes are defined within a specific domain, as previously mentioned. Attribute creation is the second step in the process of defining policies.

Policy Administration Point

Policy Administration Point

Attributes Management | Domain: fluidosOpencall

Resources

```
test <()> urn:oasis:names:tc:xacml:1.0:resource:resource-id
resource1 <()> urn:oasis:names:tc:xacml:1.0:resource:resource-id
resource2 <()> urn:oasis:names:tc:xacml:1.0:resource:resource-id
```

New Resource
Delete Resource

Actions

```
GET <()> urn:oasis:names:tc:xacml:1.0:action:action-id
PUT <()> urn:oasis:names:tc:xacml:1.0:action:action-id
```

New Action
Delete Action

Subjects

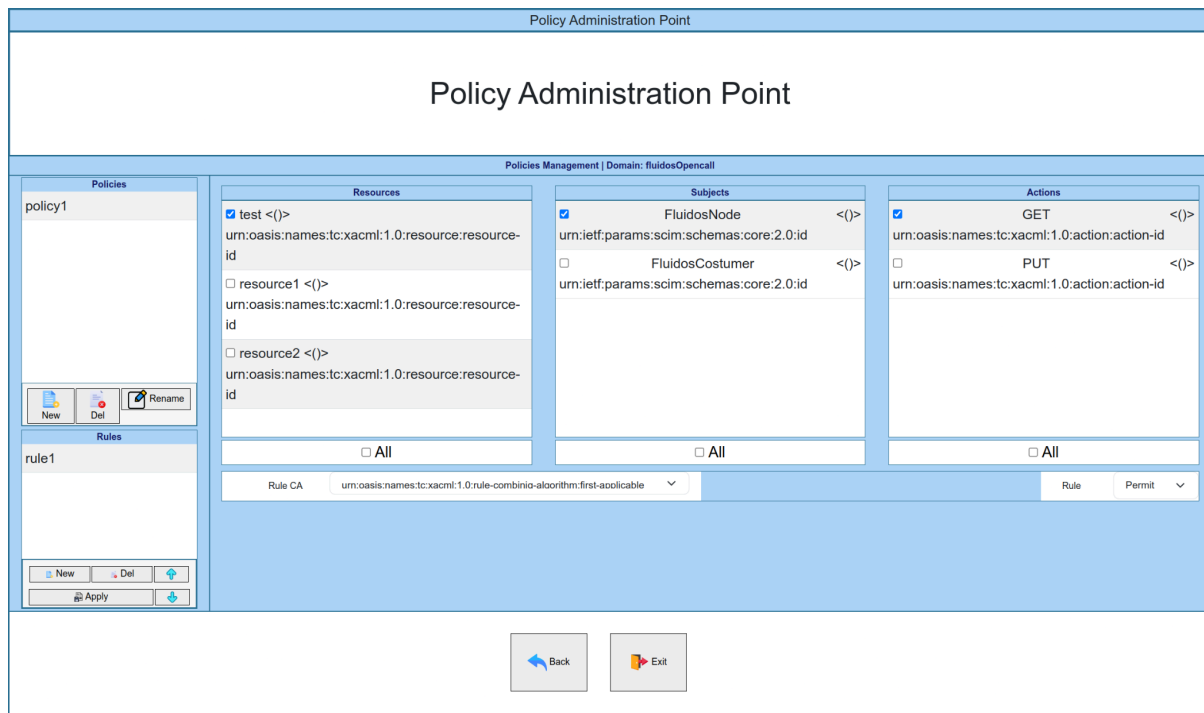
```
FluidosNode <()> urn:ietf:params:scim:schemas:core:2.0:id
FluidosCostumer <()> urn:ietf:params:scim:schemas:core:2.0:id
```

New Subject
Delete Subject
Save All Attributes

← Back
Exit →

Figure 7 PAP attributes management web page view

3. Finally, the third part of the PAP web application is the view for managing policies and rules. As previously mentioned, a policy consists of a group of rules, and multiple policies can be defined under a specific domain. The creation of policies is the final step in the process of defining them. *Figure 8* illustrates this part of the web application. To define rules, a policy must first be established, after which rules can be created by selecting one attribute for each group of attributes and specifying the policy decision, which indicates the verdict in the case there is a match with that rule, i.e., whether it is 'Permit' or 'Deny'. If there is no match with any of the defined rules, XACML will respond with 'Not Applicable,' which is neither permitted nor denied but is generally treated as denied.



Policy Administration Point

Policy Administration Point

Policies Management | Domain: fluidosOpenCall

Policies	Resources	Subjects	Actions
policy1	<input checked="" type="checkbox"/> test <()> urn:oasis:names:tc:xacml:1.0:resource:resource-id <input type="checkbox"/> resource1 <()> urn:oasis:names:tc:xacml:1.0:resource:resource-id <input type="checkbox"/> resource2 <()> urn:oasis:names:tc:xacml:1.0:resource:resource-id	<input checked="" type="checkbox"/> FluidosNode <()> urn:ietf:params:scim:schemas:core:2.0:id <input type="checkbox"/> FluidosCostumer <()> urn:ietf:params:scim:schemas:core:2.0:id	<input checked="" type="checkbox"/> GET <()> urn:oasis:names:tc:xacml:1.0:action:action-id <input type="checkbox"/> PUT <()> urn:oasis:names:tc:xacml:1.0:action:action-id

rule1

Rule CA urn:oasis:names:tc:xacml:1.0:rule-combinio-algorithm:first-aoplicable Rule Permit

Back Exit

Figure 8 PAP policies management web page view

The XACML leverages a DLT, Hyperledger Fabric, for the storage of policies and attributes, which ensures enhanced security and transparency, as Hyperledger Fabric's decentralized and immutable nature prevents unauthorised modifications and provides a verifiable, tamper-proof record of all policy changes. To be able to store policies and attributes in the Hyperledger Fabric network, a Smart Contract with the functionality to store these kinds of values has been developed and deployed in Hyperledger Fabric. *Figure 9* shows an example of the JavaScript Object Notation (JSON) structure that contains all the information about the policies defined for one domain, which is the JSON that is stored in the Blockchain. Also, the XACML checks every two minutes if the policies have been updated by checking the 'timestamp' value, and if this value does not match with the value stored in the blockchain, the XACML will update the policies by uploading the new JSON to the blockchain. It is important to note that after updating the policies in the PAP web application, the old policies may continue to be effective for up to two minutes, so this is something to be taken into account.

```

{
  "id": "urn:ngsi-ld:xacml:fluidosOpenCall",
  "type": "xacml",
  "version": {
    "type": "Property",
    "value": "2"
  },
  "xacml": {
    "type": "Property",
    "value": "<?xml version='1.0' encoding='UTF-8'?'><PolicySet xmlns='urn:oasis:names:tc:xacml:2.0:policy:schema:os' PolicyCombiningAlgId='urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable' PolicySetId='POLICY_SET'> <Target/> <Policy PolicyId='policy1' RuleCombiningAlgId='urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable'> <Target/> <Rule Effect='Permit' RuleId='rule1'> <Target> <Subjects> <Subject> <SubjectMatch MatchId='urn:oasis:names:tc:xacml:1.0:function:string-equal'> <AttributeValue DataType='http://www.w3.org/2001/XMLSchema#string'>FluidosNode</AttributeValue> <SubjectAttributeDesignator AttributeId='urn:ietf:params:scim:schemas:core:2.0:id' DataType='http://www.w3.org/2001/XMLSchema#string' /> </SubjectMatch> </Subject> </Subjects> <Resources> <Resource> <ResourceMatch MatchId='urn:oasis:names:tc:xacml:1.0:function:string-equal'> <AttributeValue DataType='http://www.w3.org/2001/XMLSchema#string'>test</AttributeValue> <ResourceAttributeDesignator AttributeId='urn:oasis:names:tc:xacml:1.0:resource:resource-id' DataType='http://www.w3.org/2001/XMLSchema#string' /> </ResourceMatch> </Resource> </Resources> <Actions> <Action> <ActionMatch MatchId='urn:oasis:names:tc:xacml:1.0:function:string-equal'> <AttributeValue DataType='http://www.w3.org/2001/XMLSchema#string'>GET</AttributeValue> <ActionAttributeDesignator AttributeId='urn:oasis:names:tc:xacml:1.0:action:action-id' DataType='http://www.w3.org/2001/XMLSchema#string' /> </ActionMatch> </Action> </Actions> </Target> </Rule> </Policy></PolicySet>"
  },
  "timestamp": {
    "type": "Property",
    "value": "2024-09-11 08:17:17"
  }
}

```

Figure 9 JSON structure of policies stored in Blockchain

2.2.2 XACML Authorization Process

The XACML authorization process is a process that consists of obtaining a verdict when trying to access a specific resource with a specific action. To obtain the verdict from the XACML, it is needed to send a specific request to the XACML component, which is responsible for making verdicts, the PDP. This functionality has been developed and integrated into the Privacy and Security Manager. It has been integrated right after the verification of a Verifiable Presentation (VP), as shown in *Figure 5*. This introduces an additional layer of security. This step ensures that not only the VP is authentic and valid, but also that access permissions are strictly evaluated against defined policies before granting authorization. By incorporating XACML-based authorization, the system benefits from fine-grained access control, allowing for dynamic decision-making based on context and user attributes. This added measure significantly reduces the risk of unauthorised access, providing a more robust and flexible security framework. The next box shows an example of the Hypertext Transfer Protocol (HTTP) request that is sent to the PDP to obtain a verdict:

Method: POST

URL: <http://xacml:8883/pdp/veredict>

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
  <Subject
    SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
      <Attribute AttributeId="urn:ietf:params:scim:schemas:core:2.0:id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue>FluidosNode</AttributeValue>
      </Attribute>
    </Subject>

    <Resource>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue>test</AttributeValue>
      </Attribute>
    </Resource>

    <Action>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue>GET</AttributeValue>
      </Attribute>
    </Action>

    <Environment/>
  </Request>
```

Table 1 Example of a request to the XACML to obtain a Verdict

After receiving the request, the PDP will check if there is a match with any of the rules defined under the domain specified in the 'domain' header of the request. As the only algorithm defined is the 'first-applicable', the PDP will iterate through every policy and rule defined for the specified domain until there is a match. If there is no match, it will reply with 'Not Applicable', but if there is a match, the PDP will reply with the decision included in the rule, which can be 'Permit' or 'Deny'. Following the example above from *Figure 8*, this request will match with 'rule1' and the PDP will reply this:

RESPONSE CODE: 200 OK

```
<Response>
  <Result ResourceID="test">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Table 2 Example of a successful Verdict response from XACML

2.2.3 XACML Accounting

Another functionality that has been developed and integrated into the FLUIDOS architecture is the registration of the XACML authorization request into the Blockchain. This process has been developed and integrated as a new XACML functionality, as the XACML has all the necessary information needed to register all the authorization requests it receives. Furthermore, a first version of the Smart Contract that allows storing authorization requests has been developed and deployed in the Hyperledger Fabric network of FLUIDOS. This first version of the Smart Contract offers the possibility to register, query-by-ID, and query-by-date authorization requests. The query-by-date functionality returns all the authorization requests between two dates. *Figure 10* shows the structure of the JSON that is stored in the Blockchain, which is formed by the following field:

- **Timestamp:** indicates the date in ISO 8601 format when the XACML receives the authorization request.
- **Action:** indicates the HTTP Method that the 'Subject' is attempting to use to access the resource indicated in 'Resource'.
- **Resource:** indicates the resource that the 'Subject' is trying to access.
- **ID:** indicates the unique identifier for each authorization request. It is also used as the key to store the authorization request in the Blockchain.
- **Subject:** indicates the identity of the requester of the authorization process.
- **Decision:** indicates the verdict of the XACML for this authorization request

```
{
  "Timestamp": "2024-09-12T07:50:21.054303",
  "Action": "GET",
  "Resource": "test",
  "ID": "ead88e2025937d27c753e5c82e0f98ec",
  "Subject": "FluidosNode",
  "Decision": "Permit"
}
```

Figure 10 JSON structure of the authorization request stored in Blockchain for accounting

The storage of XACML authorization requests on a blockchain offers several benefits. As Hyperledger Fabric is decentralized and immutable by nature, it ensures that authorization requests are securely recorded and cannot be altered. This provides a transparent and verifiable register of access control decisions, enhancing accountability and traceability. Moreover, the structured and permanent record on the blockchain facilitates efficient querying and consultation of authorization requests for accounting and compliance purposes, enabling organisations to easily track and verify historical access decisions and ensure adherence to regulatory requirements.

3 CONCLUSIONS AND FUTURE WORK

This Midterm-Report provides an overview of the XADATU Project's progress by the halfway point of the project as part of the FLUIDOS Open Call. The report summarises the contributions made so far to T.2 "Design and Developing" and focuses on the technological selection and recent developments of the project. In addition, as part of these developments, decentralized authorization and accounting functionalities, which are crucial for controlling access control in dynamic and geographically distributed Edge/Cloud environments, are added to the current FLUIDOS architecture. Additionally, XADATU improves the security and scalability of the FLUIDOS ecosystem by leveraging DLT technologies and advanced access control models like ABAC and XACML. The integration of these technologies not only addresses critical challenges within the FLUIDOS Open Call but also supports interoperability with existing identity management mechanisms and ensures alignment with the zero-trust paradigm. In conclusion, the methods suggested in this report are intended to optimise the use of resources, offer precise control over access, and guarantee an open and verifiable accounting of authorization operations.

Looking ahead, T.3 'Deploying and Evaluating' will focus on deploying and evaluating the Minimum Viable Products (MVPs) developed in T.2. This task will assess the MVPs against the factors defined in T.1, including the needs of the FLUIDOS project, open-source considerations, authorization/accounting functionalities, and key performance indicator (KPIs). The validation process will take place at testbed sites in OdinS and UMU (Spain), providing valuable feedback and expert opinions to refine the ongoing development in T.2. The expected outcome is a Technology Readiness Level (TLR) 5 demonstration fully integrated into the FLUIDOS architecture, with monitored KPIs to ensure the solution meets project goals. This next phase, scheduled from M5 to M8, will culminate in a final report, a TRL5 demo, and a comprehensive evaluation of the MVP's performance.

4 REFERENCES

- [1] Hu, V., Kuhn, D.R., Ferraiolo, D.F., & Voas, J.M. (2017). Attribute-Based Access Control. Computer, 48, 85-88.
- [2] eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. OASIS Standard. <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html>.
- [3] Sporny, M., Guy, A., Sabadello, M., and D. Reed, "Decentralized Identifiers (DIDs) v1.0", 19 July 2022, <<https://www.w3.org/TR/did-core/>>
- [4] Sporny, M., Noble, G., Longley, D., Burnett, D. C., Zundel, B., and K. Den Hartog "Verifiable Credentials Data Model v1.1", 03 March 2022, <<https://www.w3.org/TR/vc-data-model/>>
- [5] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>
- [6] Soltani R, Zaman M, Joshi R, Sampalli S. Distributed Ledger Technologies and Their Applications: A Review. Applied Sciences. 2022; 12(15):7898. <https://doi.org/10.3390/app12157898>
- [7] Garcia-Martin, M. and G. Camarillo, "Extensible Markup Language (XML) Format Extension for Representing Copy Control Attributes in Resource Lists", RFC 5364, DOI 10.17487/RFC5364, October 2008, <<https://www.rfc-editor.org/info/rfc5364>>.

APPENDIX A

This appendix will extend this midterm report with information about the Smart Contracts developed and deployed in the Fabric Network, the main characteristics of the new Smart Contracts, and how the XACML interacts with the Hyperledger Fabric to execute the Smart Contracts. Specifically, two new Smart Contracts have been developed and integrated into the Blockchain network, which will help to extend the functionality of the latest components integrated into the FLUIDOS architecture.

First of all, it is worth mentioning that a REST API has been developed and deployed with the functionality to interact with the Blockchain. This REST API offers different endpoints that allow the XACML to make requests to register/retrieve/update policies or attributes, which removes the need to implement and run a Gateway that connects to the Blockchain network and interacts with Smart Contracts for each security component added to the FLUIDOS architecture that requires interaction with the Blockchain. In this way, the REST API would be responsible for processing the request and executing the Gateway that connects to the Hyperledger Fabric network, which allows the REST API to interact with Smart Contracts.

The first Smart Contract deployed allows XACML to store policies and attributes defined in XACML. This chaincode has been developed in Go and includes an index that helps to query the database to obtain attributes or policies. The format in which the XACML stores policies and attributes is very similar, as both share most of the fields of the JSON structure stored in the Blockchain. This JSON structure can be shown in *Figure 11* for attributes and in *Figure 12* for policies, and includes the following fields:

- **Id:** indicates the domain in which these attributes/policies were defined. It also indicates if the JSON is for attributes by including 'attributes' before the domain name, or policies by including 'xacml' before the domain name.
- **Type:** indicates which information is included in the JSON, 'attributes' for attributes, and 'xacml' for policies.
- **Timestamp:** indicates the last time the attributes/policies were modified. This is useful to check if the attributes/policies JSON stored in the Blockchain is older than the local version, which means that the XACML will have to update the policies/attributes in the Blockchain.
- **Version:** indicates the actual version of the policies/attributes. It increases by one every time the policies/attributes are updated.
- **XACML/Attributes:** this is the only difference between the two elements. For the attributes, this field name is 'attributes' and indicates the attributes defined for a specific domain in XML format. Then, for policies, this field name is 'xacml' and it indicates the policies defined for a specific domain in XML format. This field is updated every time the policies/attributes are modified, so for each domain, there will be only two entries in the Fabric database, one that stores all policies defined, and the other one that stores all the attributes defined.


```
{
  "attributes": {
    "type": "Property",
    "value": "<?xml version='1.0' encoding='UTF-8'?>\n<attributes storingDate='2024-09-24T07:02:29.822449Z'>\n
    <attributes>\n  <attribute name='test' xacml_id='urn:oasis:names:tc:xacml:1.0:resource:resource-id'
    sortedValue='resource' xacml_DataType='#string'/>\n  <attribute name='GET' xacml_id='urn:oasis:names:tc
    :xacml:1.0:action:action-id' sortedValue='action' xacml_DataType='#string'/>\n  <attribute name
    ='FluidosNode' xacml_id='urn:ietf:params:scim:schemas:core:2.0:id' sortedValue='subject' xacml_DataType
    ='#string'/>\n </attributes>\n</attributes>\n"
  },
  "id": "urn:ngsi-ld:attributes:fluidosOpencall",
  "timestamp": {
    "type": "Property",
    "value": "2024-09-24 07:02:29"
  },
  "type": "attributes",
  "version": {
    "type": "Property",
    "value": "2"
  }
}
```

Figure 11 JSON structure for storing XACML Attributes in Hyperledger Fabric

```
{
  "id": "urn:ngsi-ld:xacml:fluidosOpencall",
  "timestamp": {
    "type": "Property",
    "value": "2024-09-24 07:03:03"
  },
  "type": "xacml",
  "version": {
    "type": "Property",
    "value": "2"
  },
  "xacml": {
    "type": "Property",
    "value": "<?xml version='1.0' encoding='UTF-8'?><PolicySet xmlns='urn:oasis:names:tc:xacml:2.0:policy:schema
    :os' PolicyCombiningAlgId='urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable'
    PolicySetId='POLICY_SET'> <Target/> <Policy PolicyId='Policy1' RuleCombiningAlgId='urn:oasis:names:tc:xacml
    :1.0:rule-combining-algorithm:first-applicable'> <Target/> <Rule Effect='Permit' RuleId='rul1'>
    <Target> <Subjects> <Subject> <SubjectMatch MatchId='urn:oasis:names:tc:xacml:1.0
    :function:string-equal'> <AttributeValue DataType='http://www.w3.org/2001/XMLSchema#string'>
    >FluidosNode</AttributeValue> <SubjectAttributeDesignator AttributeId='urn:ietf:params:scim:schemas
    :core:2.0:id' DataType='http://www.w3.org/2001/XMLSchema#string'/> </SubjectMatch>
    </Subject> </Subjects> <Resources> <ResourceMatch MatchId='urn:oasis
    :names:tc:xacml:1.0:function:string-equal'> <AttributeValue DataType='http://www.w3.org/2001
    /XMLSchema#string'>test</AttributeValue> <ResourceAttributeDesignator AttributeId='urn:oasis:names
    :tc:xacml:1.0:resource:resource-id' DataType='http://www.w3.org/2001/XMLSchema#string'/>
    </ResourceMatch> </Resources> </Resources> <Actions> <Action>
    <ActionMatch MatchId='urn:oasis:names:tc:xacml:1.0:function:string-equal'> <AttributeValue DataType
    ='http://www.w3.org/2001/XMLSchema#string'>GET</AttributeValue> <ActionAttributeDesignator
    AttributeId='urn:oasis:names:tc:xacml:1.0:action:action-id' DataType='http://www.w3.org/2001/XMLSchema#string'
    /> </ActionMatch> </Action> </Actions> </Target> </Rule> </Policy></PolicySet
    >"
  }
}
```

Figure 12 JSON structure for storing XACML Policies in Hyperledger Fabric

In terms of functionality, this Smart Contract also offers simple methods to manage policies and attributes. For registration, the 'id' field of the JSON mentioned before is used as the key for the storage in the Blockchain, and the value would be the JSON from the figures above. To do this, XACML will send a POST request to the specific endpoint of the REST API that offers this functionality, including in the request body the JSON the XACML wants to register in the Blockchain. *Figure 13* shows the process to register the attributes and policies in the Blockchain.

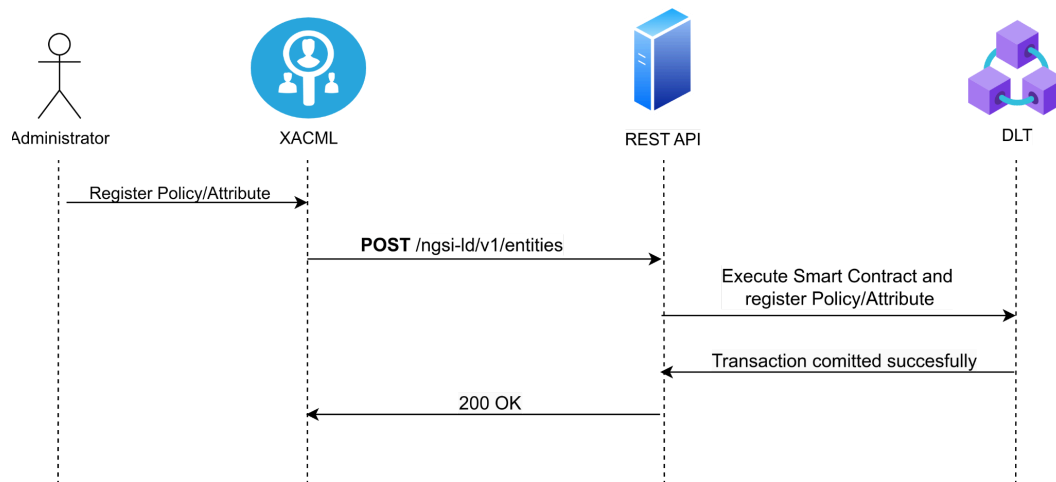


Figure 13 Sequence diagram for registering XACML Policies/Attributes in Hyperledger Fabric

Once the first entry of the policies and attributes for a specific domain has been registered for the first time, it can be modified as the Smart Contract implements the functionality to update values stored in the Blockchain, whether to add a policy, delete an attribute, or update a rule. In this case, the XACML sends a PATCH request to the specific endpoint of the REST API that provides this functionality, including in the request body the 'version', 'timestamp', and 'xacml/attributes' JSON fields with the updated values. The 'id' and 'type' fields are not included in the request body as they are embedded in the request's URL. The process and steps to update policies and attributes can be shown in *Figure 14*.

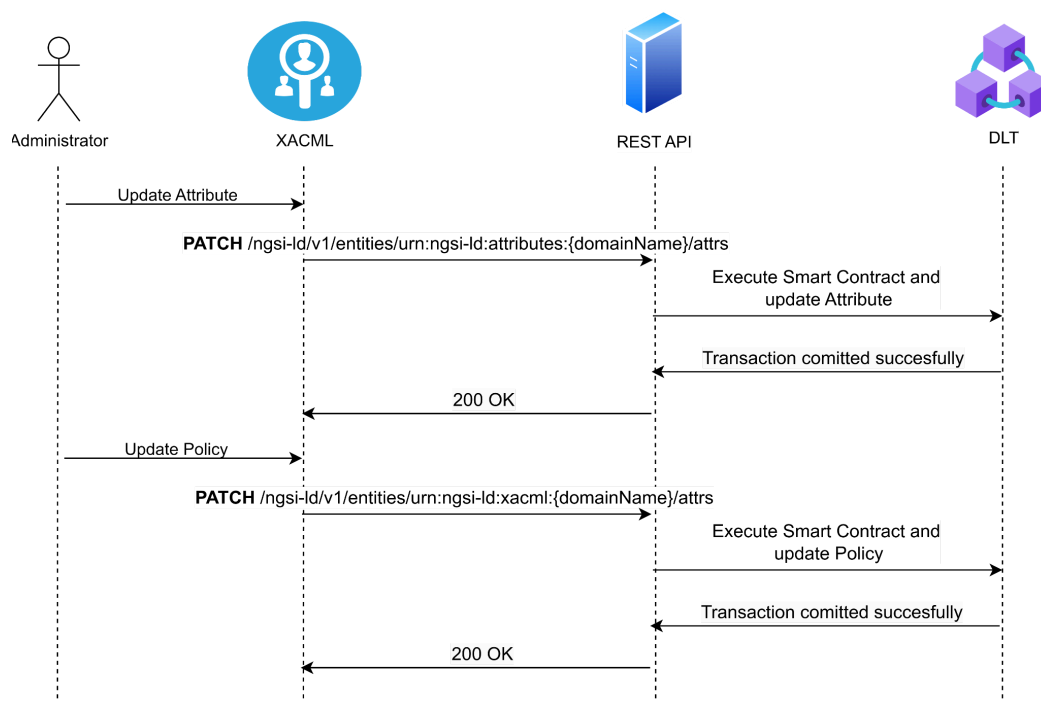


Figure 14 Sequence diagram for updating XACML Policies/Attributes in Hyperledger Fabric

There are two methods defined to retrieve policies and attributes stored in Blockchain, one for retrieving policies/attributes from a specific domain, and another one for retrieving all policies/attributes defined in the XACML. For the first case, the XACML will make a GET request to the specific endpoint in the REST API that offers that functionality, including in the URL the specific ID that the XACML wants to obtain, e.g., to retrieve the attributes from the 'fluidosOpencall' domain the Id would be 'urn:ngsi-Id:attributes:fluidosOpencall'. *Figure 15* shows the whole process of obtaining specific attributes or policies.

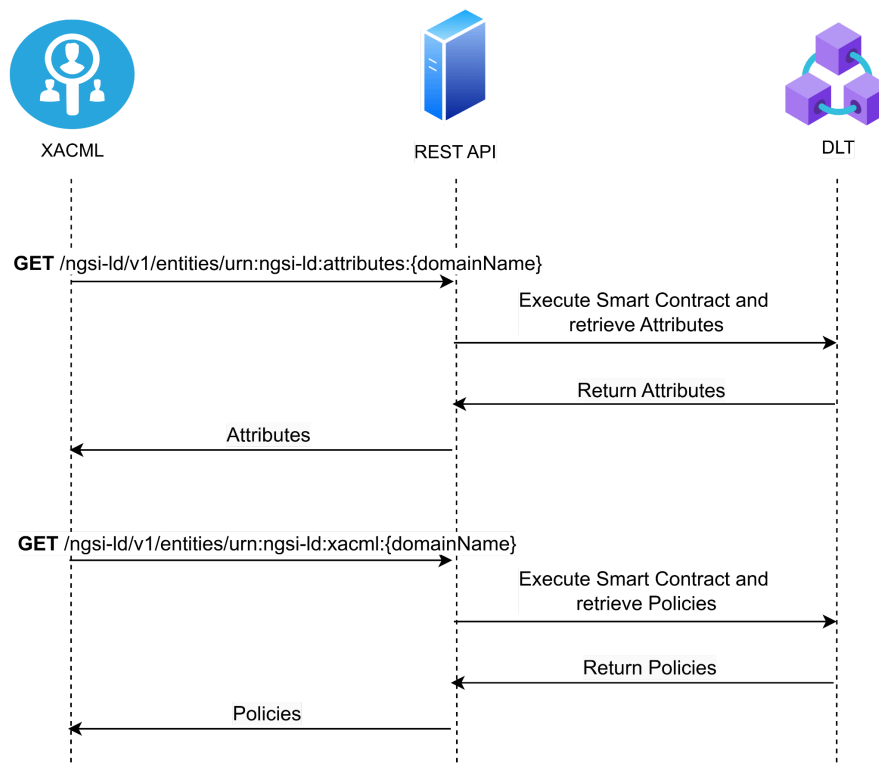


Figure 15 Sequence diagram for querying XACML Policies/Attributes for a specific domain from Hyperledger Fabric

For the second case, where the XACML wants to obtain all policies and attributes for all domains registered in XACML, it will make a GET request to the specific endpoint in the REST API that offers that functionality, including as query parameter in the URL the 'type' it wants to retrieve, where the possible values for 'type' are 'attributes' to retrieve the attributes and 'xacml' to retrieve the policies. The whole flow to retrieve all policies and attributes from all domains can be shown in *Figure 16*.

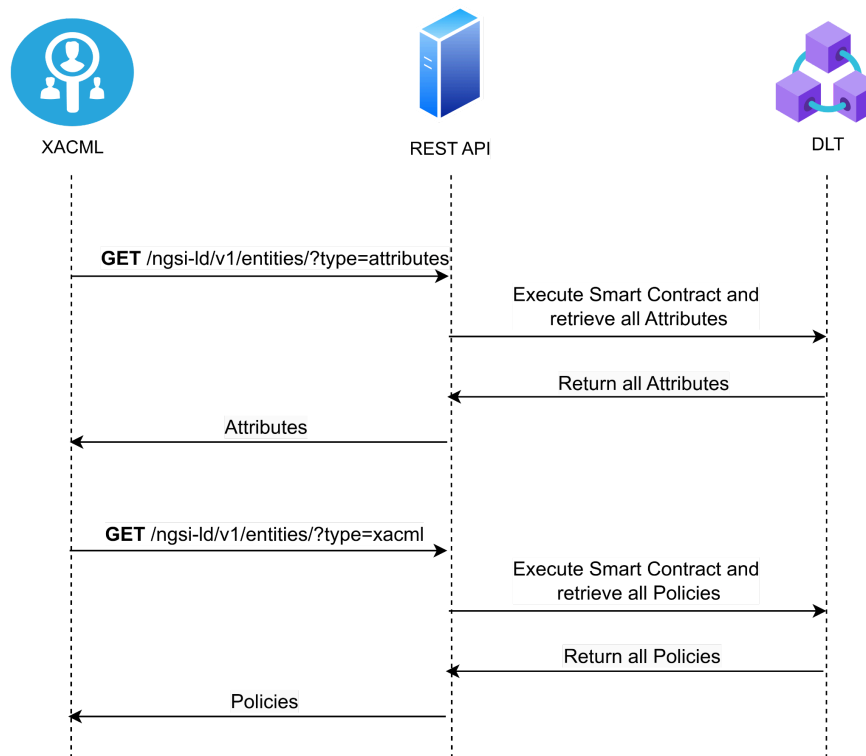


Figure 16 Sequence diagram for querying all XACML Policies/Attributes for all domains from Hyperledger Fabric

The second Smart Contract that has been deployed in the Hyperledger Fabric network allows XACML to register the authorization requests it receives from the Privacy and Security Manager on the Blockchain, which allows for the accounting of authorization requests. This Smart Contract also allows retrieving the stored requests, so it is possible to observe and analyse the historical record of the authorization requests. Furthermore, an index has also been deployed in this Smart Contract with the functionality to retrieve all authorization requests between two dates. This retrieval method has been added as an example of the possibilities the Blockchain and indexes offer for retrieving data, so it is possible to extend this by adding more retrieval methods in the future. The JSON structure that is stored in the Blockchain and contains all the information about the authorization request can be seen in *Figure 10*. Then, in terms of functionality, this Smart Contract also offers simple methods for the management of authorization requests, which are the registry and retrieval of requests. In this case, unlike what happened with the Smart Contract for attributes and policies, this Smart Contract does not offer the functionality to update stored values, as this would violate the integrity of the registered data, thus eliminating any validity of the authorization requests registered on the Blockchain. In the case of the registry, the XACML will send a POST request to the specific endpoint in the REST API, including in the request body the JSON that will be registered in the Blockchain. *Figure 17* shows the whole process of registering authorization requests in the Blockchain.

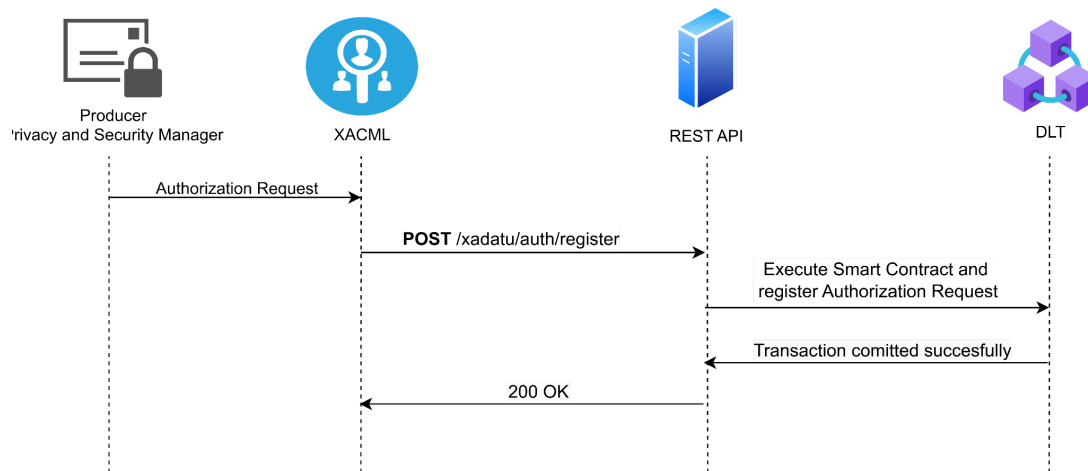


Figure 17 Sequence diagram for registering XACML authorization requests in Hyperledger Fabric

Next, for retrieving authorization requests stored in Blockchain, the functionality implemented includes two different methods, as mentioned before: one for retrieving a specific authorization request, and another one for retrieving all authorization requests between two dates. For the first case, the user must send a GET request to the specific endpoint in the REST API, including the ID of the specific authorization request to be retrieved in the URL as shown in *Figure 18*.

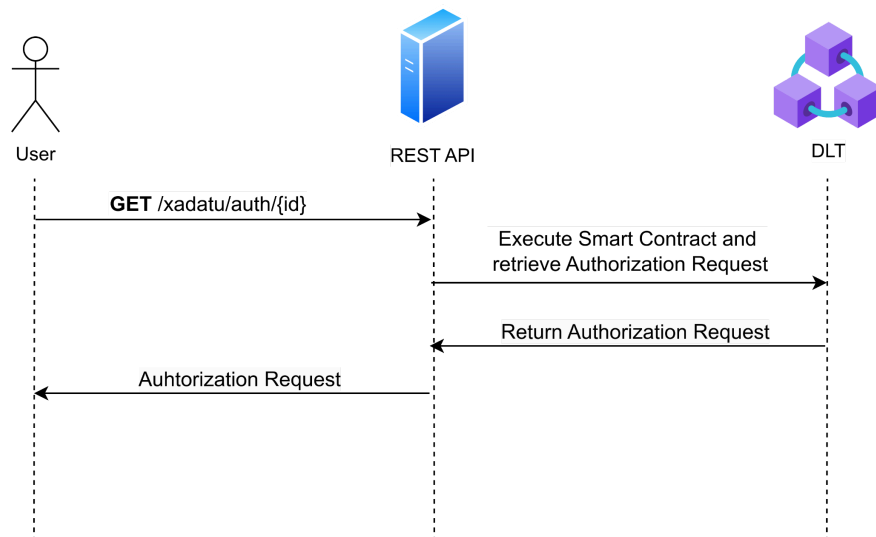


Figure 18 Sequence diagram for querying a specific XACML authorization request from Hyperledger Fabric

Finally, to retrieve all authorization requests between two dates, a GET request must be sent to the specific endpoint of the REST API that offers this functionality. The request body should include a JSON with two fields: 'startDate' containing the start date and time in ISO 8601 format, and 'endDate' with the end date and time in ISO 8601 format for the query. *Figure 19* shows the flow diagram of the process to retrieve all authorization requests between two specified dates. As mentioned before, there is the possibility to add new query

methods in the future, e.g., query all authorization requests from a specific 'Subject', query all denied requests, query all permitted requests, etc.

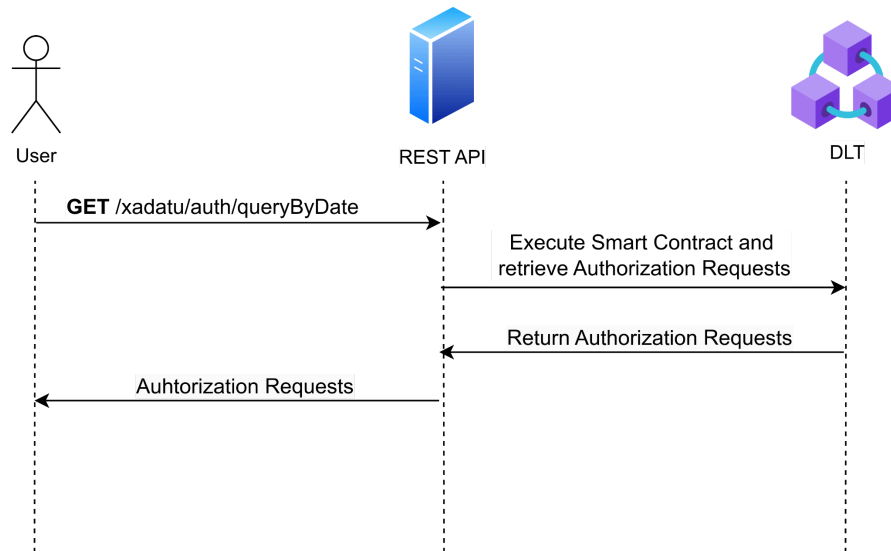


Figure 19 Sequence diagram for querying all XACML authorization requests between two dates from Hyperledger Fabric