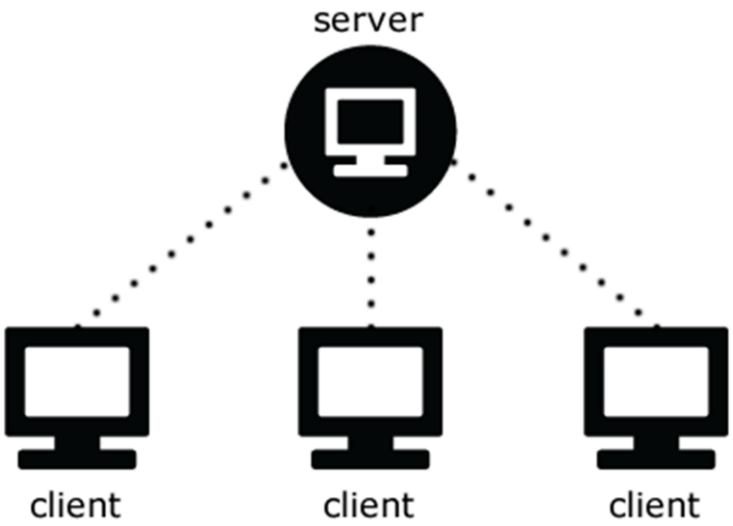
2016

SAFAX

Architecture Document



TU/e Security Group

Contents

Chapter 1

Introduction

1.1 Purpose

Cloud storage services have become increasingly popular in recent years. Users are often registered to multiple cloud storage services that suit different needs. However, the ad-hoc manner in which data sharing between users is implemented leads to issues for these users. For instance, users are required to define different access control policies for each cloud service they use and are responsible for synchronizing their policies across different cloud providers. Users do not have access to a uniform and expressive method to deal with authorization. Current authorization solutions cannot be applied *as-is*, since they cannot cope with challenges specific to cloud environments.

In order to address these challenges we have developed SAFAX [1]. SAFAX is a novel XACML-based architectural framework tailored to the development of extensible authorization services for distributed and collaborative systems. The key design principle underlying SAFAX is that all components are loosely coupled services, thus providing the flexibility, extensibility, and scalability needed to manage authorizations in complex environments.

This document provides a comprehensive architectural overview of SAFAX. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 List of Acronyms and Definitions

1.2.1 List of Acronyms

CH: Context Handler

PAP: Policy Administration Point

PEP: Policy Enforcement Point

PIP: Policy Information Point

SAFAX: eXtensible Authorization Framework As a Service

UDF: User Defined Function

XACML: eXtensible Access Control Markup Language

DO: the Domain Owner

DC: Domain Controller

1.2.2 List of Definitions

SAFAX: An extensible authorization framework offered as a service.

Domain: a data set managed by a user, e.g. files stored at a specific cloud storage service.

DO: the user to whom the domain belongs.

DC: a host of multiple domains, possibly belonging to different DOs. A cloud storage service provider is a typical example of DC. The DC is responsible for guaranteeing the confidentiality, integrity and availability of the data stored within the domains it hosts.

1.3 Outline

We first introduce the baseline authorization framework reference for SAFAX in Chapter 2. In Chapter 3, we implement SAFAX as an extensible authorization framework as a service. Chapter 4 gives an overview of database schema underneath SAFAX. Chapter 5 outlines a list of web services in SAFAX.

Chapter 2

XACML-based Architectural Framework

2.1 Introduction

As a baseline for the development of an authorization framework that meets the needs in a collaborative environment, we rely on Extensible Access Control Markup Language (XACML) [7]. XACML is becoming the de facto standard for the specification and enforcement of access control policies; thus its adoption would facilitate policy interoperability across domains controllers.

XACML defines an attribute-based access control policy language implemented in XML, as well as a reference architecture for policy evaluation and enforcement. In the remainder of the section we present an overview of the XACML reference architecture, and review existing XACML implementations against the identified requirements. The XACML policy language is not the focus of this deliverable; we refer to the XACML Core specification [5] for its complete description.

2.2 XACML Reference Architecture

The XACML reference architecture is shown in Figure 1. An access request from an application is sent to the Policy Enforcement Point (PEP). The PEP forwards the request to the Context Handler (CH). If the request is specified in the application's native format, the CH constructs a XACML request and sends it to the Policy Decision Point (PDP) for evaluation. The PDP retrieves the policies from the Policy Administration Point (PAP). If additional attributes are required for the evaluation of the access request, the PDP queries the CH for such attributes. The CH obtains these attributes from the Policy Information Point (PIP) and sends them to the PDP. The PDP evaluates the request against the policies and returns a response specifying the access decision (and possibly a set of obligations to be fulfilled) to the CH. The CH sends the response to the PEP which is responsible for the enforcement of the decision.

EXTENSIBILITY OF XACML: XACML also provides a number of extensibility points to customize policy evaluation with respect to the needs of the application domain. The most noteworthy extensibility point is the possibility to augment the PDP with User Defined Functions (UDFs) for the specification of custom constraints in the policies.

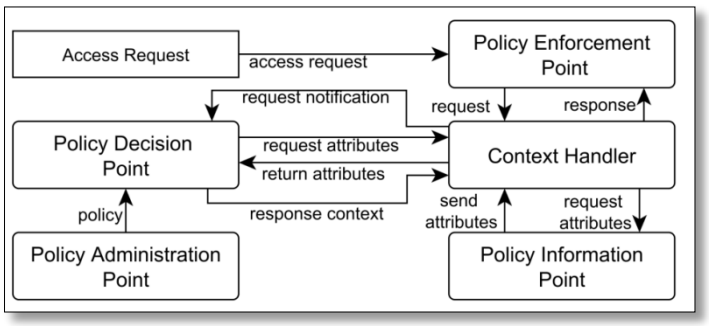


Figure 1 XACML Reference Architecture

2.3 XACML Implementations

The XACML reference architecture provides a blueprint which needs to be followed by any implementation to be compliant with the standard. However, the standard is underspecified, and hence different implementations of XACML may have slight variations between them based on implementation choices.

We have reviewed several open-source XACML implementations including SUN-XACML1, HERAS-AF [11], XEngine [12], enterprise-java-xacml2, and WSO2 Balana3. These implementations support different versions (and in some cases, multiple versions) of XACML. Some of these implementations, such as HERAS-AF and WSO2 Balana, are actively developed compared to other implementations. In addition, some implementations have been used as a backbone for XACML-based frameworks. For instance, Lazouski et al. [13] extend WOS2 Balana for the enforcement of usage control policies in distributed systems, while TAS3 Trust PDP [14] extends SUN-XACML for the evaluation of trust policies.

A major issue of existing XACML implementations is that they are often implemented as a monolithic component. Although the choice of implementing the XACML reference architecture as a monolithic component may be suitable for enterprise environment, it cannot fully address the key requirements for collaborative environments such as scalability, extensibility and flexibility. For instance, UDFs are typically implemented within the PDP, thus limiting the extensibility of existing implementations. Hence, there is a clear need for a completely different approach for XACML implementations tailored for collaborative environments.

Chapter 3

Authorization as a Service

3.1 Introduction

To overcome the limitations of existing XACML implementations in collaborative scenarios (see Chapter 2), we propose SAFAX, an eXtensible Authorization Framework as a Service. This chapter presents an overview of SAFAX and its architecture.

3.2 SAFAX Overview

SAFAX is an XACML compliant policy evaluation engine which aims to satisfy the requirements for enforcing access control requests in collaborative environments. To this end, SAFAX components are designed as loosely coupled services. As the XACML standard does not constrain how communication between components happens, we can still remain compliant with the standard.

In this section, we present a high level overview of SAFAX and discuss how it overcomes the challenges faced in cloud environments, while the next section provides a detailed view of the main components within the SAFAX framework.

An overview of the SAFAX framework is shown in Figure 2.

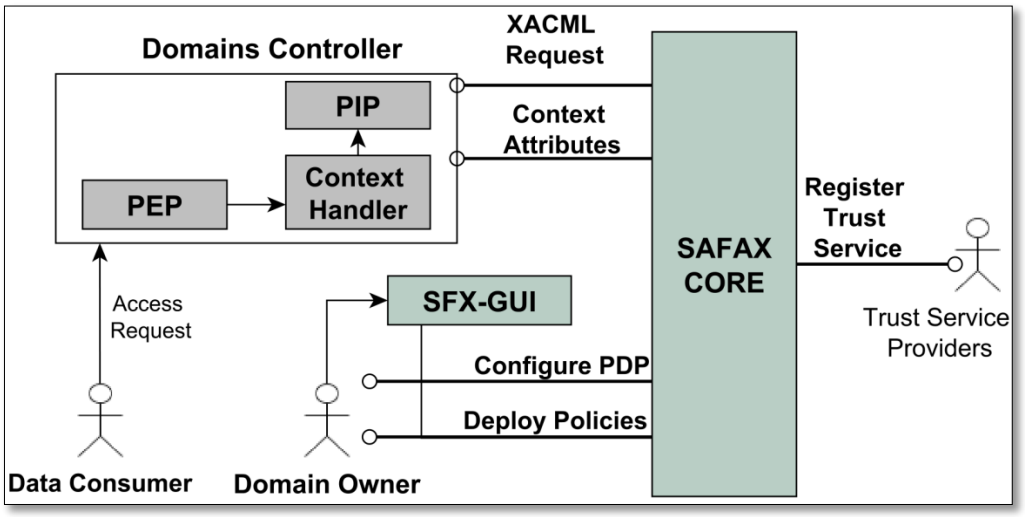


Figure 2 SAFAX Framework

The SAFAX framework consists of three main blocks: domain-specific components (i.e., PEP, CH and PIP); the SAFAX-CORE, which represents the baseline of the authorization service; and trust services, which can be used to evaluate custom constraints in *DO*'s policies. All components forming these blocks are designed as loosely coupled services. SAFAX does not prescribe who should provide these services.

Deployment configurations of SAFAX are explained in Section (?). Here, for description purposes, we assume a `default' configuration in which domain-specific components are under the control of *DC*'s, SAFAX-CORE under the control of an authorization service provider and trust services under the

control of some providers independent from the authorization service provider. Domain-specific components depend on the application environment. For instance, they should handle the conversion between the attribute representation in the application environment and the attribute representation in the XACML format.

The only requirements for domain-specific components within SAFAX is that they adhere to the XACML specification and are offered as services and thus we do not discussed them further. Two major problems *DO*s have to face when defining their policies are the limited expressiveness of the policy languages currently offered by cloud providers and the heterogeneity of policy specification. SAFAX adopts XACML for policy specification. This standard defines a canonical representation for the inputs and outputs of the PDP (which is the main component of SAFAX-CORE).

In particular, XACML provides an attribute-based policy language in which subjects, actions and resources are characterized through attributes, thus addressing the inherent limitation of traditional access control models like Access Control Lists (ACLs).

In addition, the XACML specification provides policy authors a rich set of predefined attributes as well as the possibility to use custom attributes, allowing for the definition of fine-grained policies. The use of XACML addresses some of the challenges for cloud environments and, in particular, the ones concerning policy specification. However, a key aspect is to determine whether a data consumer has the required attributes during policy evaluation. In other words, the relevant attributes of the data consumer requesting the data should be made available to the PDP during policy evaluation. SAFAX anticipates this need and allows *DO*s to constrain the access to their data on the basis of additional trust information that is fetched from external trust services during policy evaluation, for instance attributes certified by a trusted authority. Intuitively, *DO*s can delegate part of the authorization decision making to trust service providers external to the authorization service provider.

The task of connecting external trust sources with SAFAX is hidden from the *DO*s, thus empowering them with the capability of defining fine-grained access policies and at the same time relieving them from the complexity of maintaining trust sources manually. In order to achieve this flexibility, we extend the PDP of XACML through UDFs.

However, in a major departure from current XACML implementations, we design UDFs as self-configuring clients that consume external trust services. These clients are responsible for generating valid requests for the external trust services and handling the data parsing between the external services and the XACML PDP. We stress that the authorization service can be outside the control of *DC*s and can be operated instead by an independent provider trusted by *DO*s. Intuitively, SAFAX framework allows each *DO* to choose an authorization service and to require the *DC*s controlling its domains to consume this service for authorization decision making. This is similar to the authentication mechanism offered by many websites, which allows users to authenticate using the credentials provided by an external identity provider.

Thus, SAFAX mitigates the need for DOs to define access control policies for each cloud service to which they are subscribed by providing a single point for deploying and maintaining their policies (i.e., a single PAP) and thus addressing challenge. SAFAX-CORE assigns a dedicated PDP to every *DO*, which is identified with a unique URL (hereafter, we refer to such a URL as PDP-URL). When a *DC* receives an

access request for a piece of data, it forwards the request to the authorization service along with the *DO*'s PDP-URL.

The authorization service uses the PDP-URL to identify the pertinent PDP for evaluation. This implies that the *DC*s should provide a way for the *DO* to specify this PDP-URL as part of their domain configuration settings.

Trust services can be used for a range of purposes such as retrieving trust information from external sources, relocating the computation of complex functions relieving the burden on the PDP and providing additional functionalities. An important example of functionality is enabling policy interoperability in distributed systems. Every DC and trust service provider may use a different vocabulary for attribute representation. While XACML unifies policy specification syntactically, it does not solve differences in semantic such as different naming or interpretation of attributes between the entities involved in policy evaluation. While existing PDP implementations have no features built in to resolve this issue, the extensibility of the SAFAX framework using UDFs allows resolving such issues. By connecting to external services that provides semantic alignment for the relevant attributes SAFAX can resolve semantic misalignment issues without the need of any human intervention (at evaluation time).

The *DC*s, who provide the cloud storage services, can greatly benefit from using SAFAX as a trusted third party authorization service. In fact, SAFAX reduces *DC*s efforts of creating, maintaining and monitoring their custom authorization solutions and, thus, they can focus on their core business. *DC*s simply need to consume the SAFAX service. In order to share data with consumers who are not registered with their service, the *DC*s do not have to establish a web of trust with other service providers. Using SAFAX, the *DC*s are completely relieved from this burden and can share data with any data consumer, as long as the properties of the data consumers are captured by the services used for policy evaluation. The data consumers also benefit from this approach as they do not have to register to multiple cloud storage providers in order to be able to access data.

3.3 SAFAX Components

In this section we focus on the SAFAX-CORE component. SAFAX-CORE exposes several interfaces that can be invoked by different entities interacting with the policy engine. The main interfaces provided by SAFAX-CORE are:

- **Policy Deployment:** allows *DO*s to deploy their access control policies.
- **Configure PDP:** allows *DO*s to configure their PDPs.
- **Register Service:** allows the authorization service provider, *DC*s and external trust service providers to register their services with SAFAX.
- **XACML Request:** handles valid XACML requests from *DC*s.
- **Context Attributes:** allows *DC*s to augment access requests with additional application-specific attributes.

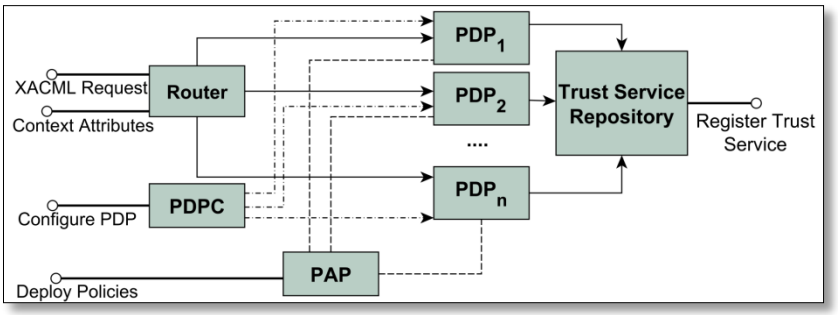


Figure 3 SAFAX Core Architecture

The components of SAFAX-CORE, as shown in Figure 3, are designed as loosely coupled services thus breaking away from the monolithic component structure often adopted by existing XACML implementations. We provide a brief explanation of the different services that are part of SAFAX-CORE.

Router:

This service is responsible for forwarding access requests from DCs to the proper PDP based on the DO's unique PDP-URL.

Service Repository:

This service allows any service registered within SAFAX to discover, bind and consume other services in a dynamic manner. In addition, this service allows external service providers to register their services with SAFAX. Service registration requires the following details

a) Service Identifier
b) Service Provider's Identity;
c) Service Description
d) Service endpoint
e) HTTP Method to invoke the service
f) Request parameters data type;
g) Response parameters data type
h) Request and Response Messaging format
i) Service Type - which indicates whether the service being offered is a PDP service, PAP service, an External Trust Service registered as a UDF and so on.

These details are necessary for services to discover other registered services, configure clients to generate valid requests and parse the responses.

Policy Administration Point (PAP):

This service facilitates *DO*s to store and manage their access control policies regardless of the location in which the data to be protected are stored. This service also enables the PDP service to fetch the policies of a specific *DO*. In addition, whenever a policy associated to a DO is updated or a

new policy is added, the PAP service forces the PDP service to reinitialize the *DO*'s PDP in order to reflect the updated policies.

PDP Configuration (PDPC):

SAFAX-CORE can support several PDP services; intuitively, every XACML implementation can be seen as a different PDP service in SAFAX. The PDPC service allows *DO*s to select the PDP service to be used for policy evaluation. Moreover, the PDPC allows *DO*s to configure the selected PDP service by setting a number of parameters including the root combining algorithms[1]. These configuration settings can be changed at any time and pushed to the PDP service which in turn initializes (or reinitializes) the *DO*'s PDP to reflect the changes.

Policy Decision Point (PDP):

SAFAX assigns a dedicated PDP to each *DO*. Every PDP is identified by a unique URL (PDP-URL). The PDP assigned to a *DO* handles all the access requests for its data regardless of the *DC* which sends them. The PDP fetches the policies associated to the *DO* from the PAP service. In addition, this service allows the PDPC service to push the PDP configuration settings specified by the *DO*. In SAFAX we decouple UDFs from the XACML PDP component to allow the framework to be extensible without disrupting existing components. As shown in Figure 4, the PDP consists of a XACML PDP and the *External Service Extensions* component that handles the invocations to external trust service. The *External Service Extensions* component contains, for each UDF, a self-configuring client that consumes the external trust service corresponding to the UDF. These clients are generic and self-configure themselves based on the information related to a given UDF that has been registered by a trust service provider with the SAFAX-CORE service. They read the external trust service description from the Service Repository and, accordingly, build a valid request and communicate with the service using the specified protocol. Moreover, they parse the response from external trust services to retrieve the trust information requested for policy evaluation.

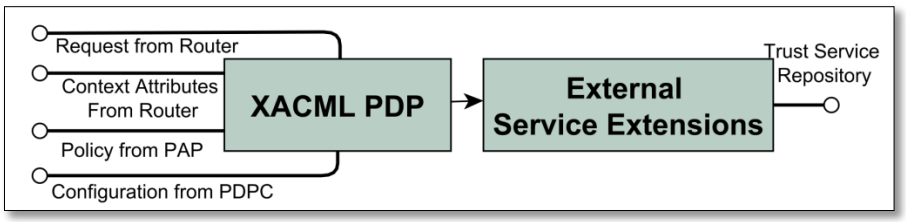


Figure 4 PDP Architecture

External Trust Services:

We decouple UDFs from the PDP and implement them as external, but pluggable services, as shown in Figure 5. This pluggable architecture allows SAFAX to be *extensible* in order to serve access requests that require complex processing in a *scalable* manner as well as consuming information from external sources. This way, the processing of trust information can be outsourced to external services while

[1] A root combining algorithm is needed to resolve policy conflicts that can arise when more than one policy is deployed in the PDP

SAFAX acts as an orchestrator for these service calls based on the *DO*'s policies. External service providers that want to plug-in their services with SAFAX need to comply with two main constraints: *i)* register their service through the Service Repository of SAFAX along with a UDF Identifier which will be used by the *DO*s in their access control policies; and *ii)* external service APIs must conform to implementation dependent specifications of SAFAX.

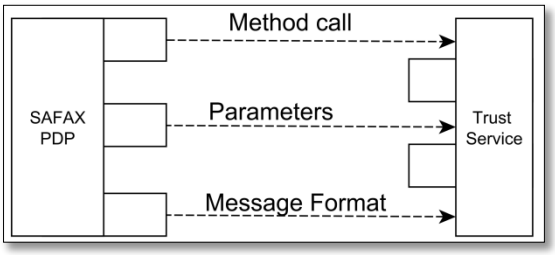


Figure 5 SAFAX Plugins

As shown in Figure 6, each provider that registers their service as an external trust service in the SAFAX framework must specify: *a)* the *Method call* through which the service can be invoked; *b)* the parameters that must be supplied to the service; and *c)* the message format that indicates how the parameters will be packaged.

3.4 Message Flow within SAFAX

The services presented in the previous section give an indication of the functionality and capabilities of each component. In this section, we present the overall integrated view of the SAFAX architecture and its usage.

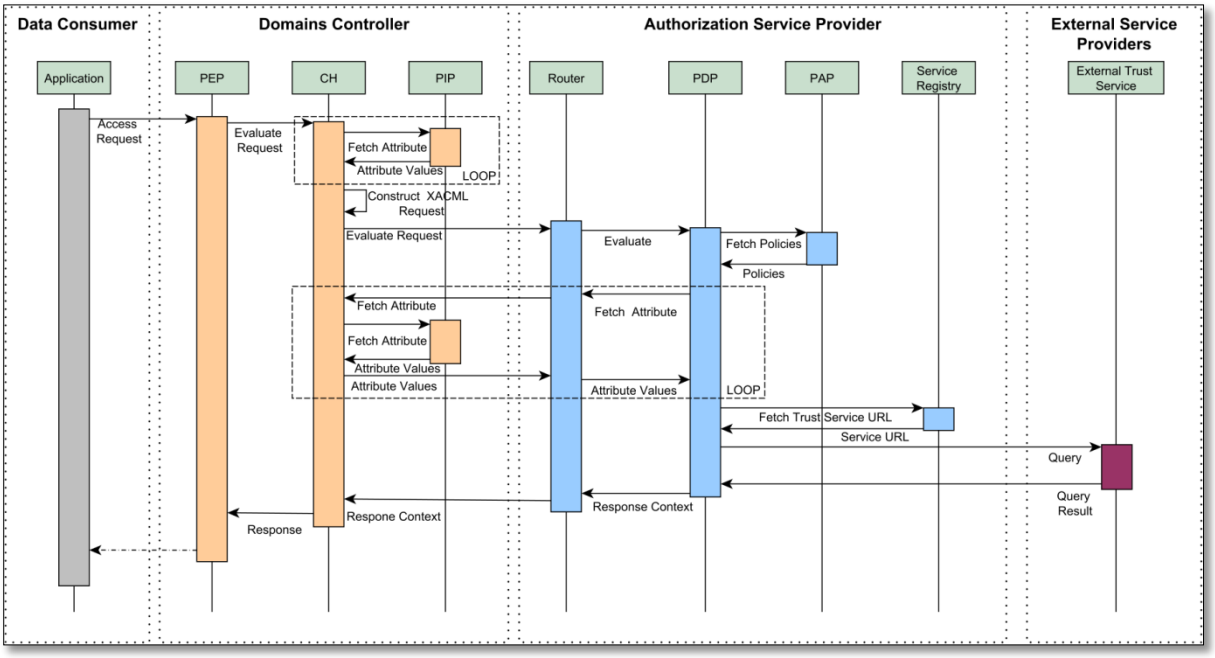


Figure 6 Message Flow for Policy Evaluation using SAFAX

*DO*s specify access control policies for their data, which can be stored across multiple cloud providers *DC*s. *DO*s use the PAP to deploy the specified policies and the PDPC to configure their PDP by selecting a default root combining algorithm, the PDP service to be used for policy evaluation, etc. SAFAX provides the *DO* a unique PDP-URL to invoke the assigned PDP. All access requests for data belonging to the *DO* are forwarded to the assigned PDP through the unique PDP-URL. As a consequence, *DO*s should be able to configure the PDP-URL at every *DC* they are subscribed to. Moreover, each *DC* should implement the domain-specific components (i.e., PEP, CH and PIP) and register the Context Handler service with the SAFAX framework. This is necessary since during policy evaluation the PDP might need additional attributes from the CH of the *DC* such as the IP addresses of the data consumer, system time, and other attributes that can be required for policy evaluation. In particular, the CH interface to fetch the attributes should be registered with the service registry of the SAFAX framework.

Figure 6 shows the interaction of services within SAFAX. Whenever a data consumer sends a request to access data belonging to a *DO* (Alice) to a *DC*, the access request is intercepted by the PEP of the *DC*. The PEP forwards the request to the CH service, which first fetches additional information from its PIP, enriching the original request, and then constructs a valid XACML request. The CH service forwards the XACML request to the PDP assigned to the *DO* by invoking the SAFAX service. The Router intercepts the access request coming from the CH and forwards it to the proper PDP based on the PDP-URL. During policy evaluation, if the PDP needs additional attributes (e.g. timestamp), it contacts the *DC*'s Context Handler service. When the *DO* specifies policies that require additional (external) trust information to make a decision, the PDP service contacts the external trust services. After receiving the required information, the PDP computes the decision and informs the *DC*'s Context Handler. The CH parses the XACML response and sends the decision to the PEP. The PEP enforces the decision by allowing or denying access to the *DO*'s data to the consumer based on the given decision.

Chapter 4

Database Schema

4.1 Introduction

The data model is a key aspect of SAFAX service. It provides a means to have persistence storage of data and act as data source for various services within the SAFAX framework. In this chapter we present the underlying database schema of SAFAX.

4.2 SAFAX Resources

The key elements of SAFAX framework is the manner in which resources are organized. The central part of the SAFAX framework are the users. Each user is organized into groups with privilages associated to different user groups. The permissions themselves are not hardcoded into SAFAX, but rather defined through XACML policies (see User Manual document).

The various resouce types that are stored within SAFAX are:

- User
  - Account
  - Groups
  - Actions
  - Sessions
- Projects
  - Project Details
  - Project Demos
  - Project Users
- Demos
  - Demo Details
  - Demo PDP
  - Demo Policies (XACML and Trust Policies)
  - Demo Requests
  - Demo Attributes (PIP)
- Log
  - Activity log
  - Session log

These resources are organized heirarchically in the database as shown in Figure 8. We use a relational database schema that has strong bindings between the different data elements. The SQL file to generate the schema will be provided to you by the SAFAX administrator.

This file is formatted for MySQL databases, however the structure and the model can be easily and readily adapted to any relational database.[2]

[2] Several online converting tools are readily available (such as the one provided here)

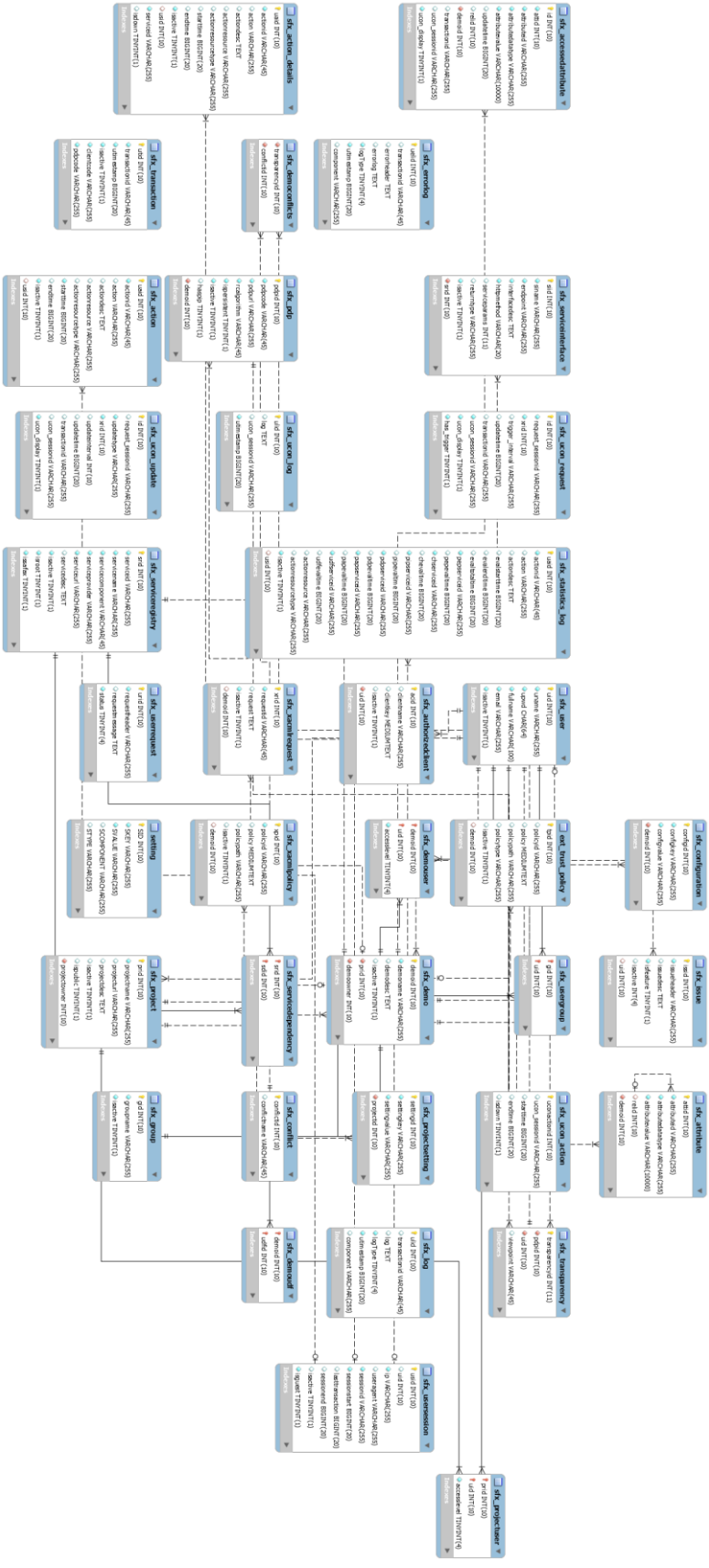


Figure 7 SAFAX Main Database Layout

4.3 SAFAX Log

In addition to the relational database, to improve performance NoSQL database is used for containing the log of SAFAX. The schema of this database is shown in Figure 8. The log contains the userid, start time of the evaluation, end time of the evaluation, evaluation duration, the service component id, start time of each service component, end time of each service component, and a list of user defined functions.

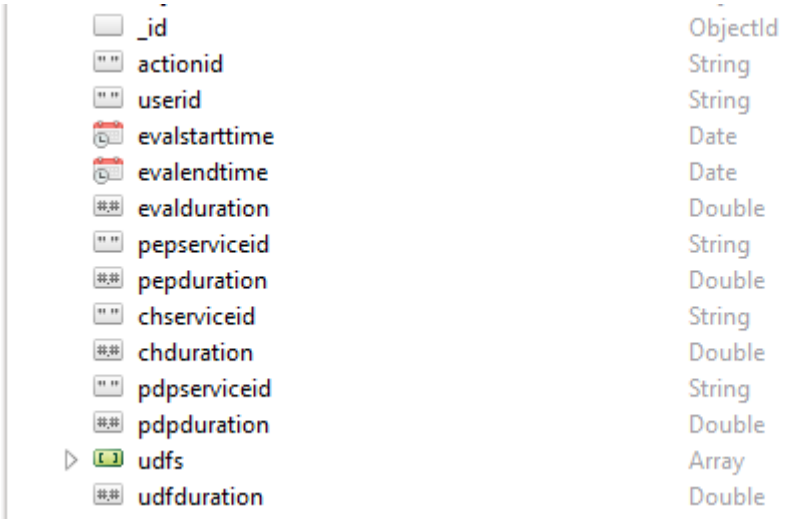


Figure 8 SAFAX NoSQL Database Layout

Chapter 5

Web Services

5.1 Introduction

SAFAX comprises of web services that are loosely coupled and communicate with each other via messages. The various services that are part of SAFAX are explained in this chapter. First of all, the general guidelines of how these services are organized in a package are discussed.

5.2 General Guidelines

A key aspect of the development process within SAFAX is that code should be organized, understandable and modular. Within SAFAX, we organize the code within the following packages.

5.2.1 nl.tue.sec.safax.servicename.db

This package contains three standard classes:

a) DBParameters – this class contains the parameters to connect to the database.
b) DBAbstraction – this class interacts with the underlying database and exposes several methods for other classes to interact with the underlying database. This class makes use of the DBParameters class to connect and perform queries on the database.
c) DBFns – this is the richest class in this package, this class contains all the functions that will be used by the rest of the framework to interact with the database.

The whole purpose of this organization is to shield the rest of the service from any changes that might occur with the database. At any stage, changes are just contained to one class – for example, if a developer wants to use Postgre SQL instead of MySQL – the changes to connect and query the database are localized to DBAbstraction class (and any changes to the connection parameters – username, password etc. are localized to DBParameters class). In this manner we enable developers to extend SAFAX within multiple scenarios without having to change different parts of the service.

5.2.2 nl.tue.sec.safax.servicename.ds

This package will contain classes that contain data structures or complex data types. For several services a developer might need to define their own data structures and this package contains all such data structures. It should be noted that the functionality to parse, convert or modify different data structures should not be contained in this package. In this manner, it becomes easier for other developers to find all the custom data structures defined by the developer to be used within their service.

5.2.3 nl.tue.sec.safax.servicename.client

This package contains classes that are essentially clients consuming external services. This prevents many different versions of the client to be created in random classes whenever there is a need to consume external information. Furthermore, when the external service (external to the current service being developed) changes its interfaces with regards to input variables, output format, or the messaging format – this can be contained within the classes contained in this package, shielding the rest of the classes from any ripple effect.

5.2.4 nl.tue.sec.safax.servicename.factory

This package contains classes that define interfaces or abstract classes. This makes it easier for developers to a consolidated view of the different interfaces that they must conform to or abstract classes that can be extended.

5.2.5 nl.tue.sec.safax.servicename.init

This package contains classes that will are used for initializing complex objects that require certain non-trivial actions. For example, to initialize the PDP object in the HERAS-AF PDP service, the policies must be fetched and the configurations of the PDP object must be read from the database and then the PDP object can be initialized. Once again, separating the implementation logic from the initializing logic, the developer has a clear understanding where he might need to extend or modify the control flow with the service.

5.2.6 nl.tue.sec.safax.servicename.util

This class contains various utility classes that are necessary during the service execution. SAFAX services have two standard utility classes:

a) DataUtil – this class contains utilities for formatting data, modifying data from one data structure to another, parsing the data among other data related functions.
b) LogUtil – this class contains several functions that allow a developer to write logs to the database. Based on the nature of the log – such as logging an exception which will be made available only to the SAFAX administrator or to log messages that are aimed at the end user the developer can make of different functions.

In addition, other utilities can be grouped to this package. This greatly enhances the understandability and readability of the implementation logic.

5.2.7 nl.tue.sec.safax.servicename.impl

This package contains classes that have the implementation logic, they are the core orchestrators of each service call by performing all the different operations – calling the database functions, calling the data utility functions, initializing functions among others. It should be noted that any exceptions that may be raised by classes within this package MUST be handled and a user understandable message must be sent back in case of an exception.

5.2.8 nl.tue.sec.safax.servicename.ws

This package will contain classes that expose web interfaces. These classes do not have any implementation logic nor do they have any data formatting embedded into them. They just validate the request and, if it is a valid request, it forwards the request to the classes contained in the implementation package.

5.3 SAFAX Servcies

The various services within SAFAX framework are listed here:

- Context handler service (ch)
- Context handler service with well-known text supported (wtk-ch)
- Credential service (credential)
- HERAS-AF PDP service (herasaf)
- HERAS-AF PDP service with geolocation supported (herasaf-geolocation)
- Geolocation service (geolocation)
- PAP service (pap)
- PEP service (pep)
- PIP service (pip)
- Evidence based reputation service (reputation-eb)
- Flow based reputation service (reputation-fb)
- SAFAX UI (sfx)
- SAFAX backend service (sfxservice)
- Similarity service (similarity)
- UCON services (UCON)
- Transparency service (transparency)

5.3.1 Context Handler Service

This is a web service developed in JAVA and packaged as a WAR file. The Context Handler service exposes several interfaces accessible through the network (web APIs). These APIs allow various components to interact with the data sources contained within SAFAX. The APIs allow the retrieval of attributes, modification of attributes and construction of valid XACML requests.

Every request to the context handler must contain the PDP code, which is assigned to each user along with a unique transaction ID – which is necessary for logging purposes. Based on the PDP code, the context handler forwards the access requests to the assigned PDP. Furthermore, the context handler uses the PDP code to retrieve the attributes assigned to a particular user (if the user makes use of the SAFAX PIP) *(see PIP Service for more details)*.

5.3.2 Context Handler Service with Well-known Text supported

This is a web service developed in JAVA and packaged as a WAR file. It has all the functionalities of the previous Context handler service (see section 5.3.1). In addition, it supports converting a request containing well-known text geometry to a GeoXACML [9] request with Geography Markup Language (GML) 2.0 definition [3].

5.3.3 Credential Service

This is a web service developed in JAVA and packaged as a WAR file. This web service exposes interfaces where a consumer can query to a) verify whether a user has certain credential issued by a certain entity; b) find all the credentials issued to a user by a certain entity; and c) find all the users who have certain credential issued by a certain entity. This information is computed based on Credential policies represented in Prolog. The credential service has a custom Prolog emulator which parses the policies and solves the queries similar to how a prolog service would do. It should be noted

that the prolog emulator is custom designed for the credential service and thus cannot be seen as a standalone prolog emulator that can solve any kind of Prolog query.

This service needs the unique transaction ID along with the PDP code assigned to a user. The Credential service contacts the “sfxservice” to fetch the credential policies based on the pdp code. Furthermore, the credential service makes use of a feature provided by SAFAX framework for remote logging. Thus, the credential service contacts the “service registry” to write a log message for a particular transaction.

SAFAX supports credential-based trust management as an external service. In particular, SAFAX uses a credential-based trust management service based on GEM [8], a distributed goal evaluation algorithm for trust management systems.

5.3.4 HERAS-AF PDP Service

This is a web service developed in JAVA and packaged as a WAR file. The HERAS-AF PDP service uses the PDP implementation provided by HERAS-AF. It encapsulates the JAR file that contains the PDP implementation and has a wrapper that exposes the various interfaces of HERAS-AF XACML PDP as a service. Since the HERAS-AF PDP is designed as a monolithic block, the PAP and PIP are all subsumed into the JAR file. However, in SAFAX we decouple these various components as independent services. However, during policy evaluation time, we fetch the policies from the PAP service of SAFAX and deploy it on the PAP provided by HERAS-AF. Similarly, the PIP of HERAS-AF is extended to consume information from the PIP service of SAFAX. These additions and modifications make it possible for SAFAX to have modular, interacting services for each component.

Furthermore, we make use of the UserDefinedFunctions (UDFs) feature of XACML that is supported by HERAS-AF and extend the implementation of the UDFs with a self-configuring client. These clients read information about UDFs from the service repository and auto-configure themselves to parse the output and validate the inputs. This feature is a core feature of SAFAX that allows a PDP to consume external information during policy evaluation.

5.3.5 HERAS-AF PDP Service with Geolocation Supported

This is a web service developed in JAVA and packaged as a WAR file. It has all the functionalities of the previous Herasaf-Af PDP services (see section 5.3.4). In addition, it supports evaluating a GeoXACML [9] request with Geography Markup Language (GML) 2.0 definition [3].

5.3.6 Geolocation Service

This is a web service developed in JAVA and packaged as a WAR file. It is an external trust service that can establish relationships between geometries.

5.3.7 PAP Service

This is a web service developed in JAVA and packaged as a WAR file. The PAP service provides interfaces for users to deploy their access control policies. The PAP service not only allows users to deploy XACML policies but also the trust policies that are used by external trust services such as the Credential Service and the Reputation services.

5.3.8 PEP Service

This is a web service developed in JAVA and packaged as a WAR file. The PEP service provides interfaces for consumers to send an access request to SAFAX for evaluation. The PEP interface typically is a domain dependent component, but for the sake of completeness (and testing) within SAFAX, we provide the PEP service to users. The PEP service generates a unique transaction ID along with a request and sends it to the context handler of a consumer's choice. It also is responsible for handling and enforcing obligations that accompany the result of policy evaluation.

5.3.9 PIP Service

This is a web service developed in JAVA and packaged as a WAR file. The PIP service has similar interfaces to that of the context handler service, except the "evaluate request" interface. The PIP service is used by the context handler to access and manipulate the data contained in the data sources.

5.3.10 Reputation Services (Evidence based and Flow based)

The reputation services are implemented in JAVA and packaged as WAR files. SAFAX supports two types of reputation: flow based reputation [6] and evidence based reputation [5].

These web services are external trust services that compute the reputation of a user based on reputation policies that are represented in Prolog format.

5.3.11 SAFAX UI

SAFAX UI is an HTML, CSS, JavaScript service and it is packaged as a WAR file as well. It consists of various HTML pages that set up the static html contents of the various pages. Jquery library are used along with the MetroUI library that is used to provide a uniform UI layout and effects. Metro UI also has certain CSS styles that provide SAFAX UI its uniform look. AJAX is used extensively to create the dynamicity required for the web pages. This is primarily designed as a front end and interacts with the SFX backend service to populate and provide data to SAFAX framework.

5.3.12 SFX Backend Service

SFX backend service is developed in JAVA and packaged as a WAR file. It performs various actions such as logging users, establishing guest sessions, creating projects, demos among many other features. This service uses SAFAX to authorize the various actions performed by users. Each action by a user goes through the SAFAX PEP, Context Handler, interacting with a dedicated SAFAX PIP to enrich the original access request from a user with other user specific information such as user roles etc. In addition, it validates policies to ensure that they are of the right format among other things.

5.3.13 Similarity Service

Similarity service is developed in JAVA and packaged as a WAR file. This web service is an external trust service that can be invoked during policy evaluation to find the semantic similarity between two different terms. Within SAFAX, this service is primarily used for policy alignment where policy and the access request might use different terms but with the same semantic meaning.

5.3.14 UCON Service

UCON service is developed in JAVA and packaged as a WAR file. This web service can be used as a PEP component. The UCON service allows control over a resource during the entire lifetime of its usage. It is based on two distinctive characteristics: attribute mutability and access decision continuity.

5.3.15 Transparency Service

Transparency service is developed in JAVA and packaged as a WAR file. This web service can be used as a PEP or PDP component. It aims to detect mismatches between the decision enforced by the authorization system and the user policies [2].

References

[1] S. P. Kaluvuri, A. I. Egner, J. den Hartog, and N. Zannone. SAFAX -- an extensible authorization service for cloud environments. Frontiers in ICT 2(9), 2015.

[2] R. Mahmudlu, J. den Hartog, and N. Zannone. Data Governance & Transparency for Collaborative Systems. In Proceedings of the 28th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 2016), 2016. Springer.

[3] Open Geospatial Consortium Inc. Geospatial eXentsible Access Control Markup Language (GeoXACML) Extension A – GML2 Encoding [Internet]. 2007 Nov [cited 2016 May 20]. Available from http://www.opengeospatial.org/standards/geoxacml.

[4] Open Geospatial Consortium Inc. OpenGIS Implementation Standard for Geographic information – Simple feature access – Part 1: Common architecture [Internet]. 2011 May [cited 2016 May 20]. Available from http://www.opengeospatial.org/standards/sfa.

[5] B. Skoric, S. de Hoogh, and N. Zannone. Flow-based Reputation with Uncertainty: Evidence-Based Subjective Logic. International Journal of Information Security, 2015.

[6] A. Simone, B. Skoric, and N. Zannone. Flow-based reputation: more than just ranking. International Journal of Information Technology & Decision Making, 11(3):551-578, 2012.

[7] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0 [Internet]. 2005 Feb [cited 2016 May 20]. Available from https://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[8] D. Trivellato, N. Zannone, and S. Etalle. GEM: a Distributed Goal Evaluation Algorithm for Trust Management. Theory and Practice of Logic Programming, 2014.

[9] Open Geospatial Consortium Inc. Geospatial eXentsible Access Control Markup Language (GeoXACML) version 1 corrigendum [Internet]. 2007 Nov [cited 2016 May 20]. Available from http://www.opengeospatial.org/standards/geoxacml.

[10] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0 [Internet]. 2005 Feb [cited 2016 May 20]. Available from https://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[11] HERAS-AF [Internet]. [cited 2016 August 3]. Available from https://bitbucket.org/herasaf/herasaf-xacml-core.

[12] A. Liu, F. Chen, J. Hwang, and T. Xie. XEngine [Internet]. 20087 [cited 2016 August 3]. Available from http://xacmlpdp.sourceforge.net/.

[13] A. Lazouski, G. Mancini, F. Martinelli, and P. Mori. Architecture, workflows, and prototype for stateful data usage control in cloud. International Workshop on Data Usage Management, 2014.

[14] K. Bohm, S. Etalle, J. Hartog, C. Hutter, S. Trabelsi, D. Trivellato, et al. A flexible architecture for privacy-aware trust management. J. Theor. Appl. Electron. Commer. 2010

Appendix 1

This section outlines the XACML input and output interface for using SAFAX.

XACML Request

SAFAX takes a standard XACML request [7] as an input. A sample XACML request is depicted in Figure 9.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os http://docs.oasis-open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">

        <Subject>
                <Attribute
                        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                        DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>student</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>grade list</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>write</AttributeValue>
                </Attribute>
        </Action>

        <Environment />
</Request>
```

Figure 9 XACML Request Sample

XACML Response

After evaluation, response can be: Permit, Deny, NotApplicable, and Indeterminate. A sample XACML response is depicted in Figure 10.

Response

```
<Response xmlns:ns2="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
    <Result>
        <Decision>Deny</Decision>
        <Status>
            <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
        </Status>
    </Result>
</Response>
```

Figure 10 XACML Response Sample

Appendix 2

This section outlines a list of common external web service interfaces provided by SAFAX.

Credential

UDF	urn:nl:tue:sec:pdp:1.0:udf:credential:user:has:credential
Description	This function verifies whether a given user has a given credential issued by a given issuer.
Input	userid: String credential: String issuer: String
Outputs	Type: Boolean *True* if userid has credential; *False* otherwise

UDF	urn:nl:tue:sec:pdp:1.0:udf:credential:find:user:credentials
Description	This function takes as input a user and an issuer and returns the credentials of the user issued by a given issuer.
Input	userid: String issuer: String
Outputs	Type: Bag of String A bag of credentials

UDF	urn:nl:tue:sec:pdp:1.0:udf:credential:users:with:credential
Description	This function takes as an input the Credential and Issuer and returns a bag of users
Input	credential: String issuer: String
Outputs	Type: Bag of String The set of users

Geolocation

UDF	urn:ogc:def:function:geoxacml:1.0:geometry-contains
Description	This function checks whether a geometry instance contains another geometry instance.
Input	geometry1: WKT String geometry2: WKT String
Outputs	Type: Boolean *True* if geometry1 contains geometry2; *False* if geometry1 does not contains geometry2

UDF	urn:ogc:def:function:geoxacml:1.0:geometry-distance
Description	This function calculates the distance between two geometry instances.
Input	geometry1: WKT String geometry2: WKT String
Outputs	Type: Double

	Distance between geometry1 and geometry2

UDF	urn:ogc:def:function:geoxacml:1.0:geometry-one-and-only
Description	This function checks whether a geometry collection contains only one geometry instance.
Input	geometryCollection: WTK String
Outputs	Type: Geometry A *geometry* instance in geometryCollection if geometryCollection contains only one geometry instance *Null* in other cases

UDF	urn:ogc:def:function:geoxacml:1.0:geometry-overlap
Description	This function checks whether two geometry instances overlap.
Input	geometry1: WTK String geometry2: WTK String
Outputs	Type: Boolean *True* if geometry1 overlaps geometry2; *False* if geometry1 does not overlap geometry2

Evidence-based Reputation

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:eb:belief:on:user
Description	This function calculates the belief of a given user.
Input	userid: String
Outputs	Type: Double Belief of userid

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:eb:disbelief:on:user
Description	This function calculates the disbelief of a given user.
Input	userid: String
Outputs	Type: Double Disbelief of userid

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:eb:uncertainty:on:user
Description	This function calculates the uncertainty of a given user.
Input	userid: String
Outputs	Type: Double Uncertainty of userid

Flow-based Reputation

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:fb:count:above:rep
Description	This function counts the number of users whose reputation is above a given threshold.
Input	threshold: Float

Outputs	Type: Integer The number of users whose reputation is greater than threshold

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:fb:count:above:user
Description	This function counts the number of users whose reputation score above the reputation score of a given user.
Input	userid: String
Outputs	Type: Integer The number of users whose reputation is greater than the reputation of userid

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:fb:count:below:rep
Description	This function counts the number of users whose reputation is below a given threshold.
Input	threshold: Float
Outputs	Type: Integer The number of users whose reputation is lower than threshold

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:fb:count:below:user
Description	This function counts the number of users whose reputation is below the reputation of a given user.
Input	userid: String
Outputs	Type: Integer The number of users whose reputation is lower than the reputation of userid

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:fb:filter:users:above
Description	This function returns a list of users whose reputation is above a given threshold
Input	users: Bag of strings threshold: Float
Outputs	Type: Bag of strings The set of users in users whose reputation is greater than threshold.

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:fb:filter:users:below
Description	This function returns a list of users whose reputation is below a given threshold
Input	users: Bag of strings threshold: Float
Outputs	Type: Bag of strings The set of users in users whose reputation is less than threshold.

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:fb:user:rep
Description	This function calculates the reputation of a given user.
Input	userid: String

Outputs	Type: float *Reputation* of userid.

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:fb:users:above
Description	This function determines the set of users that have at least a given reputation.
Input	repscore: float
Outputs	Type: Bag of strings The set of users whose reputation is greater than or equal to repscore.

UDF	urn:nl:tue:sec:pdp:1.0:udf:reputation:fb:users:below
Description	This function determines the set of users that have at most a given reputation.
Input	repscore: float
Outputs	Type: Bag of strings The set of users whose reputation is lower than or equal to repscore.

Similarity

UDF	urn:nl:tue:sec:pdp:1.0:udf:similarity:check:attributes
Description	This function checks whether the similarity between two given attributes is greater than or equal to a given threshold.
Input	attribute1: String attribute2: String threshold: Double
Outputs	Type: Boolean *True* if the similarity score between attribute1 and attribute 2 is greater than or equal to threshold; *False* otherwise

UDF	urn:nl:tue:sec:pdp:1.0:udf:similarity:score
Description	This function returns the similarity score between two attributes.
Input	attribute1: String attribute2: String
Outputs	Type: Double *Similarity score* between attribute1 and attribute2

UDF	urn:nl:tue:sec:pdp:1.0:udf:similarity:bag:check
Description	This function checks whether there exists an attribute in a given list of attributes whose similarity with a given attribute is greater than or equal to a given threshold.
Input	attributeList: Bag of String attribute: String threshold: Double
Outputs	Type: Boolean

	True if there exists an attribute in attributeList such that the similarity score between attribute and the attribute in attributeList is greater than or equal to threshold; *False* otherwise.

UDF	urn:nl:tue:sec:pdp:1.0:udf:similarity:bag:max
Description	This function returns the highest similarity score between a given attribute and the attributes in a given list of attributes.
Input	attributeList: Bag of String attribute: String
Outputs	Type: Double The *highest similarity score* between attribute and the attributes in attributeList.

UDF	urn:nl:tue:sec:pdp:1.0:udf:similarity:bag:subset
Description	This function returns the subset of attributeList such that the similarity score between a given attribute and any value from this subset is greater than or equal to a given threshold
Input	attributeList: Bag of String attribute: String threshold: Double
Outputs	Type: Bag of Strings The set of attributes in attributeList whose similarity with attribute is greater than or equal to threshold.