Abstract

Nowadays, NoSQL databases have been rapidly becoming the popular data platform for big data and real-time web applications. Simpler horizontal scaling, flexible schema designing, high performance data access have made NoSQL databases to be alternative approaches for traditional relational databases . However, there are some disadvantages in NoSQL, among which the lack of effective suppprt for access control and privacy protection is the most serious ones. The huger data we have, the more challenge in data protection we have to face. In this thesis, we address this issue by implemeting a comprehensive framework for enforcing attribute-based security policies stored in JSON document. We use Polish notation for modeling conditional expressions which are the combination form of subject, resource, and environment attributes so that the policies are flexible, dynamic and fine grained. Moreover, with the approach of attribute-based access control, we have proposed a flexible model struture for privacy protection so that it can be evaluated not only by access purpose but also by subject, resource, environment attributes. We also build a web application which interacts to our framework so that administrators can easily define and review policies. The experiment is carried out to illustrate the relationship between the processing time for access decision and the complexity of policies.

1. Overview
   1. Introduction

Nowadays, the quanity of data is increasing exponentially by the development of social media appications, sensor for data acquisitions and smart phone utilization. NoSQL databases is the most popular approach to handle those semi and unstructured data for a scalable application. Like relational database, security is highly considered in NoSQL database, especially when working with huge volume data. For the last decade, Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role Based Access Control (RBAC) have been used almostly to handle security. However, because of the rapid development of large scale dynamic systems, those traditional access controls have gradually reveal their disadvantages, for example, applied for only closed system, role explosion and inflexibility in specifying dynamic policies and contextual conditions. To overcome those problems, Attribute Based Access Control models have been recently investigated and according to Gartner‘s prediction: “By 2020, 70% of all businesses will use attribute based access control as the dominant mechanism to protect critical assets.”[]

Based on the Hibert and Lopez assessment of global information storage over time, Figure[] shows the relative between the increasement of stored information from 1986 to 2007 and the trend of access control. Even when access control systems are successful in restricting unauthorized and unauthenticated users, they are ineffective as privacy protection for a large, decentralized system like the World Wide Web, where it is easy to copy or aggregate information. Most previous studies have considered privacy protection in access control models as constraints on purpose of data usage. Inheriting that approach, we have extended our privacy policy model so that the condition can be evaluated by user, resource, environment attributes (purpose value is included in environment attributes).



Figure[]

* 1. Related work

Our thesis is related to several topics in the area of access control and privacy protection for data management, namely policy specification, privacy-preserving data management systems. We now survey the most relevant approaches in these areas and point out the difference of our work with respect to these approaches.

Hua Wang, Lili Sun, and Vijay Varadharajan[] have proposed a purpose-based framework for supporting privacy preserving access control policies and mechanisms. They have also developed algorithms to help a system to detect and analyze the conflicts when adding new policies. However they don’t mention much about how to model the conditional expression and the algorithms they proposed just focused only on simple attributes lacked of evaluating conditional expression.

Prosunjit Biswas, Ravi Sandhu, and Ram Krishman[] have presented an attribute based protection model for JSON documents. Their approach is to add a new attibute called “security-label” to JSON elements and specify access control policies using these values. The advantage of the seperation of labeling and authorization policies is that they can be specified and administered independently possibly by different level of administrators. However, the number of label assignments can be very large because it is calculated by the exponential function. Therefore the space storage is considered to be a potential problem when the system is expanded.

Pietro Colombo, Elena Ferrari[] have proposed a systematic approach to the automatic development of a monitor that regulates the execution of SQL queries based on purpose based privacy policies. Their proposed solution does not require programming, it is general, platform independent and usable with most of the existing relational database management systems.

Ji-Won Byun, Ninghui Li[] have presented a comprehensive approach for privacy preserving access control based on the notion of purpose. A key feature of their approach is that it allows multiple purposes to be associated with each data element and also supports explicit prohibitions, thus allowing privacy officers to specify that some data should not be used for certain purposes.

Haibo Shen[] has proposed a semantic-aware attribute based access control model (SABAC) by combining the Semantic Web technologies with the attribute based access control. SABAC use the Web Ontology Language standard to represent the ontology of the resources and users and uses eXtensible Access Control Markup Language as the policy language.

* 1. Purpose and Scope

In this thesis, our access control model is built on the principle of NIST Standard ABAC that an access decision is permitted only if the request satisfies conditions on attributes of subject, resource and environment specified in policies. Moreover, with the approach of attribute-based access control, we have proposed a flexible model struture for privacy protection so that it can be evaluated not only by access purpose but also by subject, resource, environment attributes and function defined by user. We use Polish notation for modeling conditional expressions so as to describe complex policies such as user, data, environment, driven policies. We also build a web application which interacts to our framework so that administrators can easily define and review policies. The experiment is carried out to illustrate the relationship between the processing time for access decision and the complexity of policies.

* 1. Thesis Structure

1. Background:
   1. Attribute-based access control definition:

Attribute-based access control is a logical access control methodology where authorization to perform a set of operations is determined by evaluating attributes associated with the subject, resource, requested operations, and, in some cases, environment conditions against policy, rules, or relationships that describe the allowable operations for a given set of attributes[].

Subject is the entity (it can be the user, requestor, or mechanism acting on behalf of the user or requestor).

Resource is the entity to be accessed (e.g.file, database record, Store Information, …).

Action is the operation to be carried on the resource (e.g.read, create, delete,..)

Environment is any information regarding the context of the access that might be used in making the access decisiion (eg.time, network, location,…).

Policy is the presentation of rules or relationships that defined the set of allowable operations a subject may perform upon an object in permitted environment conditions.

* 1. Privacy protection:

Data privacy refers to the evolving relationship between technology and the legal right to, or public expectation of privacy in the collection and sharing of data about one’s self. Privacy concerns exist wherever uniquely identifiable data relating to a person or persons are collected and stored, in digital form or otherwise. In order to protect the privacy of individuals, a number of work has showed that the notion of purpose used for specifying privacy policies. In access management systems, purpose is considered as the reason to collect or to access private data. Private or sensitive information can be preserved by restricting the intended purpose of data access. According to “Data protection principles” [] based on Data Protection Act Organization , purpose is considered as the second principle: “Personal data shall be obtained only for one or more specified and lawful purposes, and shall not be further procecssed in any manner incompatible with that purpose or those purposes.” Extending those approaches, we have added user, resource, environment attributes when evaluating privacy policies so that we can specify privacy policies more dynamically and flexibly.

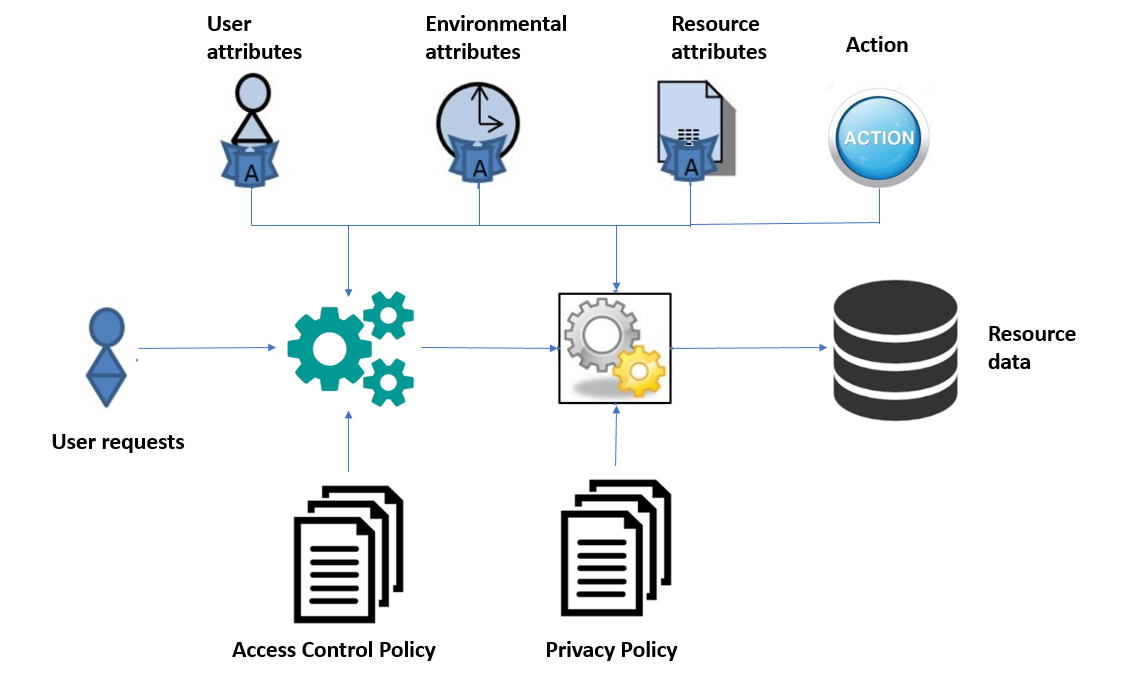
1. …Model

3.1) Case study:

3.2) Components:

In this section, we describe the base theory of this thesis. When a subject access an object, the authorization process is carried out though two stages called as 2-stage authorization:

* First stage: access control policy authorization verifies that the request is legitimate with rights for the subject to access data.
* Second stage: request is transfer to this stage for checking privacy compliance based on privacy policies.

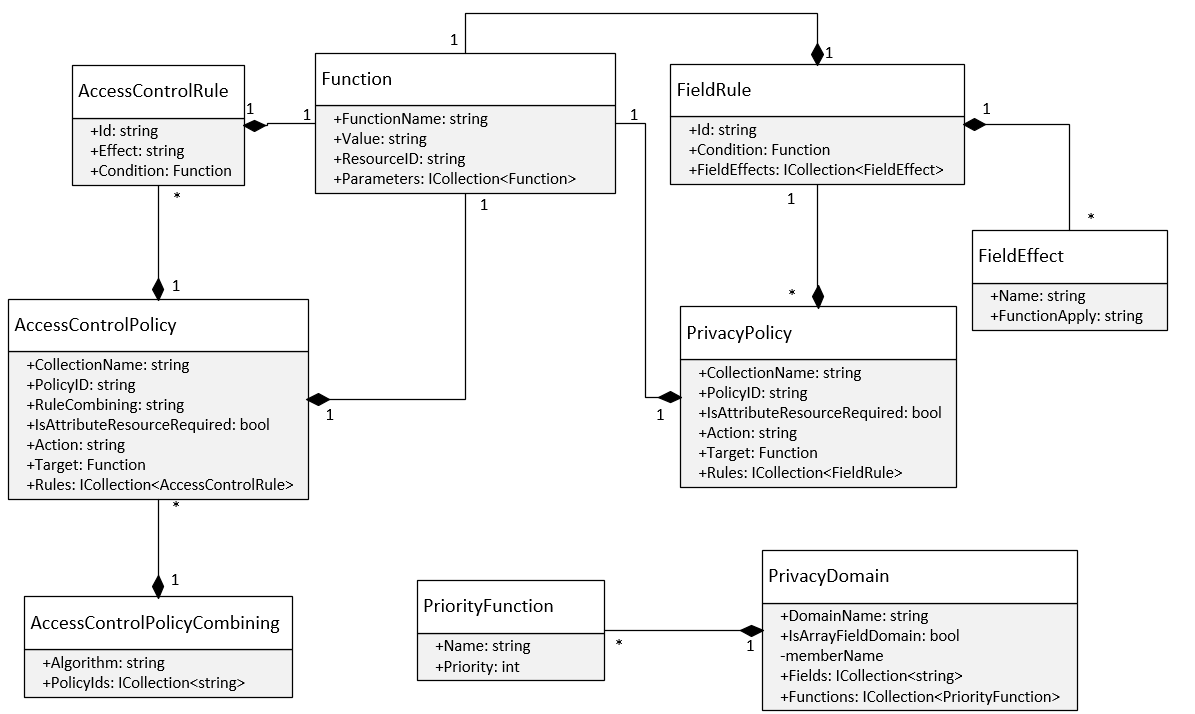


Figure[]

**Access Control Policies**: contain policies which are used to determine whether a subject can access resources. The decision is made based on rules inside policies which are the boolean expressions evaluated by user’s defined function, subject, resource, environment attribute. Those policies are specified and managed by administrators.

**Privacy Policies**: contain policies which are used to determine whether some fields of a record in resources should be shown, hidden or blurred when a user access to it. The privacy protection is made based on rules inside policies which are the boolean expressions evaluated by user’s defined function, subject, resource, environment attribute. Those policies are specified and managed by administrators.

1. Policy Structure:
   1. General Structure:



Figure[] Class Diagram

We will explain more about how we model the conditional expression of policies in *Function* class. It is a recursive structure. If the value of “Value” field is defined so it means that this function represents for a constant value. If the value of “Value” is null it means that this is the function and the value “FunctionName” field must be defined.

|  |  |
| --- | --- |
| Fields | Description |
| FunctionName | The name of function. |
| Value | The constant value or a path json |
| ResourceID | The identifer or name of the resource |
| Parameters | Its value can be a constant value or another function |

For example we have a conditional expression:

Equal (Subject.role, intern) And GreaterThan (Subject.age, 18)

Then the result will be:

{

"function\_name" : "And",

"parameters" : [

{

"function\_name" : "Equal",

"parameters" : [

{

"value" : "role",

"resource\_id" : "Subject"

},

{

"value" : "intern",

"resource\_id" : null

}

]

},

{

"function\_name" : "GreaterThan",

"parameters" : [

{

"value" : "age",

"resource\_id" : "Subject"

},

{

"value" : "18",

"resource\_id" : null

}

]

}

]

}

It can also be visualized in expression tree:

And

Equal GreaterThan

Subject.role intern Subject.Age 18

* 1. Access Control Policy Structure:

We specify access control policy structure as follows:

|  |  |
| --- | --- |
| **Fields** | **Description** |
| policy\_id | identifier of policy |
| collection\_name | name of collection or table containing resource data |
| Action | the action performed by subject |
| rule\_combining | to solve the conflict of rules |
| is\_attribute\_resource\_required | a derived field which is used to determine that whether the policy need attribute resource to evaluate condition of target or rules. The necessary of this field will be mentioned in the next section. |
| Target | The conditional expression specifies when the policy should be applied to. |
| Rules | an array field with each element in it is a rule which contains “id” field, “effect” field (value of this field can only be “Permit” or “Deny”) and condition. |

Example:

{

"policy\_id" : "Policy 1",

"collection\_name" : "Department",

"action" : "read",

"rule\_combining" : "permit-overrides",

"is\_attribute\_resource\_required" : true,

"target" : { //Equal (Subject.active, True)

"function\_name" : "Equal",

"parameters" : [

{

"value" : "active",

"resource\_id" : "Subject"

},

{

"value" : "True",

"resource\_id" : null

}

]

},

"rules" : [

{

"id" : "rule 1",

"effect" : "Permit",

"condition" : { // Equal (Resource.dept\_name, Subject.department)

"function\_name" : "Equal",

"parameters" : [

{

"value" : "dept\_name",

"resource\_id" : "Resource"

},

{

"value" : "department",

"resource\_id" : "Subject"

}

]

}

}

]

}

* 1. Privacy Policy Structure:

We specify access control policy structure as follows:

|  |  |
| --- | --- |
| **Fields** | **Description** |
| policy\_id | identifier of policy |
| collection\_name | name of collection or table containing resource data |
| is\_attribute\_resource\_required | a derived field which is used to determine that whether the policy need attribute resource to evaluate condition of target or rules. The necessary of this field will be mentioned in the next section. |
| Target | The conditional expression specifies when the policy should be applied to. |
| Rules | an array field with each element in it is a rule which contains “id” field, “field\_effects” field and condition. “field\_effects” field is an array field with each element specifies which privacy function will be used with the value of “name” field when condition is satisfied. |

Example:

{

"collection\_name" : "Department",

"policy\_id" : "policy 2",

"action" : "read",

"is\_attribute\_resource\_required" : true,

"target" : {

"function\_name" : "Equal",

"parameters" : [

{

"value" : "role",

"resource\_id" : "Subject"

},

{

"value" : "intern",

"resource\_id" : null

}

]

},

"rules" : [

{

"rule\_id" : "rule 1",

"condition" : {

"function\_name" : "Equal",

"parameters" : [

{

"value" : "dept\_name",

"resource\_id" : "Resource"

},

{

"value" : "OPERATIONS",

"resource\_id" : null

}

]

},

"field\_effects" : [

{

"name" : "dept\_id",

"effect\_function" : "DefaultDomainPrivacy.Hide"

},

{

"name" : "dept\_no",

"effect\_function" : "DefaultDomainPrivacy.Show"

},

{

"name" : "dept\_name",

"effect\_function" : "DefaultDomainPrivacy.Show"

}

]

}

]

}

* 1. Conflict resolving approach:

Because an access control policy may contain multiple rules and we have many policies. Each rule, access control policy may evaluate to different decisions (Permit, Deny). Therefore our approach is to use combining rule algorithm inherited from XACML[].

A new structure named “AccessControlPolicyCombiningConfiguration” is added to resolve conflict between multiple access control policies.

|  |  |
| --- | --- |
| **Fields** | **Description** |
| policies\_id | A list identifiers of policies |
| algorithm | The name of algorithm is used to solve conflict when multiple policies are contained in “policies\_id” field |

Example:

{

"\_id" : "58f24565de2b68f43464287a",

"policies\_id" : [

"Policy 1", "Policy 2"

],

"algorithm" : "permit-overrides"

}

In privacy policies, the conflict situation occurred when there are multiple rules in a privacy policy which are satisfied the condition. It results to that we have many privacy functions to be applied to one field of object. Therefore we have added a new structure named “PrivacyDomain” to solve conflict. We also specify a constraint that a field in resource can only belong to at most two domains, one is default domain which contains two basic privacy functions: show and hide, another domain is configured by administrator.

|  |  |
| --- | --- |
| **Fields** | **Description** |
| domain\_name | Name of domain. |
| fields | The names of fields in resource which are belong to this domain. |
| hierarchy | To configurate the priority for each privacy function or sub-privacy policy. |
| is\_sub\_policy | To check whether this is domain for privacy function or sub-privacy policy. |

Example:

{

"domain\_name" : "DefaultDomain ",

"fields" : [],

"is\_sub\_policy" : false,

"hierarchy" : [

{

"name" : "Hide",

"priority" : 1

},

{

"name" : "Show",

"priority" : 2

}

]

},

Let consider below example to see how conflict resolving process work:

Employee Resource:

{

"name": "John",

"personal\_info": {

"birth\_date": "15/01/1994",

"ssn": "457-55-5462"

}

}

Privacy policy:

{

"policy\_id": "policy 1",

….

"rules" : [

{

"rule\_id" : "rule 1",

"field\_effects" : [

{

"name" : "name",

"effect\_function" : "Optional"

},

{

"name" : "personal\_info.birth\_date",

"effect\_function" : "DateTimeDomain.ShowYear"

},

{

"name" : "personal\_info.ssn",

"effect\_function" : "SsnDomain.SerialNumber"

}

],

"condition" : {//assume that this condition is satisfied}

},

{

"rule\_id" : "rule 2",

"field\_effects" : [

{

"name" : "name",

"effect\_function" : "DefaultDomain.Show"

},

{

"name" : "personal\_info.birth\_date",

"effect\_function" : "DateTimeDomain.ShowMonthAndYear"

},

{

"name" : "personal\_info.ssn",

"effect\_function" : "SsnDomain.AreaNumber"

}

],

"condition" : {//assume that this condition is satisfied }

},

{

"rule\_id" : "rule 3",

"field\_effects" : [

{

"name" : "name",

"effect\_function" : "DefaultDomain.Show"

},

{

"name" : "personal\_info.birth\_date",

"effect\_function" : "DefaultDomain.Show"

},

{

"name" : "personal\_info.ssn",

"effect\_function" : "Optional"

}

],

"condition" : {//assume that this condition is satisfied }

}

]

}

Privacy Domain:

{

"domain\_name" : "DateTimeDomain",

"fields" : ["Employee.personal\_info.birth\_date"],

"is\_sub\_policy" : false,

"hierarchy" : [

{

"name" : "ShowYear",

"priority" : 1

},

{

"name" : "ShowMonthAndYear",

"priority" : 2

}

]

},

{

"domain\_name" : "SsnDomain",

"fields" : ["Employee.personal\_info.ssn"],

"is\_sub\_policy" : false,

"hierarchy" : [

{

"name" : "AreaNumber",

"priority" : 1

},

{

"name" : "GroupNumber",

"priority" : 2

},

{

"name" : "SerialNumber",

"priority" : 3

}]}

First we will explain more detail about the “field\_effects” field in privacy policy structure. It is an array field and the number of elements in this field is equal to the number of single value field in resource. Each element has the following structure:

"name": is the path to the single value field.

"effect\_function": This field has only 2 value patterns. First is "Optional" value, second is "X.Y" value where X is privacy domain, and Y is name of privacy function in that domain.

Back to the example, we have the conflict privacy table.

|  |  |
| --- | --- |
| **Fields** | **Conflict Privacy Functions** |
| name | Optional, DefaultDomain.Show |
| personal\_info.birth\_date | DateTimeDomain.ShowMonthAndYear,  DateTimeDomain.ShowYear,  DefaultDomainPrivacy.Show |
| personal\_info.ssn | SsnDomain.AreaNumber, SsnDomain.SerialNumber, Optional |

The privacy function will be chosen using the following rule:

P(“Optional”) < P(“DefaultDomain.Show”) < P(X.Y1) < … < P(X.Yn) < P (“DefaultDomain.Hide”)

where P(X.Y) stands for priority of privacy function Y in domain X. The priority is configured by admininstrator in “PrivacyDomain” structure.

Applying this rule to above conflict table, we will have the following result:

|  |  |
| --- | --- |
| **Fields** | **Privacy Function Chosen** |
| name | DefaultDomain.Show |
| personal\_info.birth\_date | DateTimeDomain.ShowYear |
| personal\_info.ssn | SsnDomain.AreaNumber. |

Applying those privacy function chosen, the result data will be:

{

"name": "John",

"personal\_info": {

"birth\_date": "1994",

"ssn": "457"

}

}

* 1. Advanced privacy policy support for array of objects field:

JSON object can contain not only single-value field (string, number, boolean value) but also contain an array of objects field which makes the JSON object more complex. To support privacy for array of objects field, we extend the value of “effect\_function” field in Privacy Policy structure so that its value can be an identifer of another privacy policy which target to this array field.

For example:

A complex JSON Object:

"Project Bonus" Resource:

{

"project\_name" : "E-learning system",

"bonuses": [

{

"name": "John",

"bonus": 600

},

{

"name": "Bob",

"bonus": 900

}

]

}

We have a privacy rule: "Employee can only view his/her bonus for each project. "

The equivalent sub-privacy policy will be specified as followings:

{

"policy\_id": "Policy 3",

**"collection\_name" : "ProjectBonus.bonuses",**

….

"rules" : [

{

"rule\_id" : "rule 1",

"field\_effects" : [

{

"name" : "name",

"effect\_function" : "DefaultDomain.Show"

},

{

"name" : "bonus",

"effect\_function" : " DefaultDomain.Hide"

},

],

"condition" : {

"function\_name" : "Equal",

"parameters" : [

{

"value" : "name",

"resource\_id" : "Subject"

},

{

"value" : "name",

"resource\_id" : "Resource"

}

]

}

},

]

}

Then the main privacy policy for "ProjectBonus" collection will use the value of "policy\_id" of this sub-privacy policy for the "effect\_function" field:

{

"policy\_id": "Policy 1",

**"collection\_name" : "ProjectBonus",**

….

"rules" : [

{

"rule\_id" : "rule 1",

"field\_effects" : [

{

"name" : " project\_name",

"effect\_function" : "DefaultDomain.Show"

},

{

**"name" : "bonuses",**

**"effect\_function" : " Policy 3"**

},

],

"condition" : { //assume that this condition is satisfied }

},

]

}

Now, assuming that John want to view "ProjectBonus" collection, the result will be:

{

"project\_name" : "E-learning system",

"bonuses": [

{

"name": "John",

"bonus": 600

},

{

"name": "Bob",

"bonus": ""

}

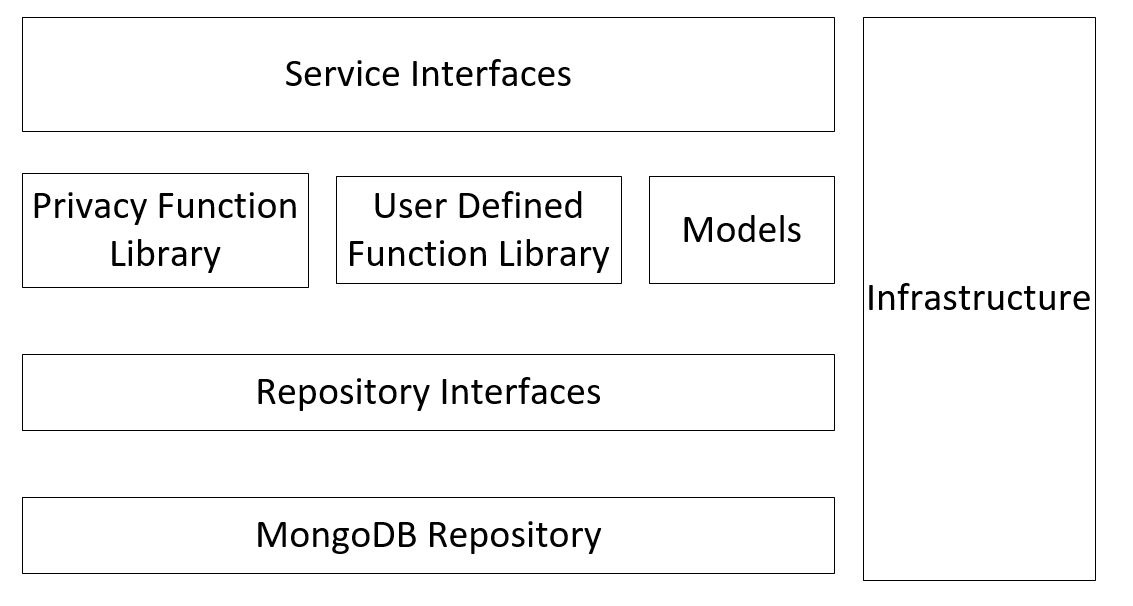
]

}

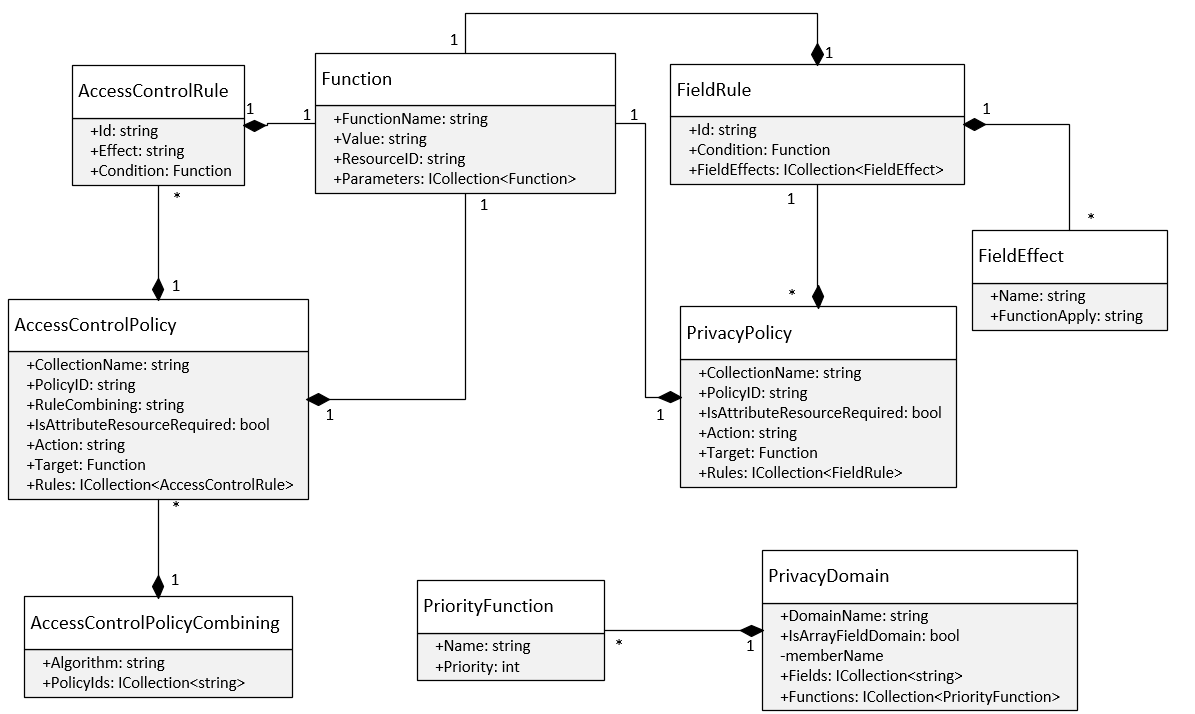
Bonus for Bob has been hidden due to sub-privacy policy.

1. Proposed Framework
   1. Architecture

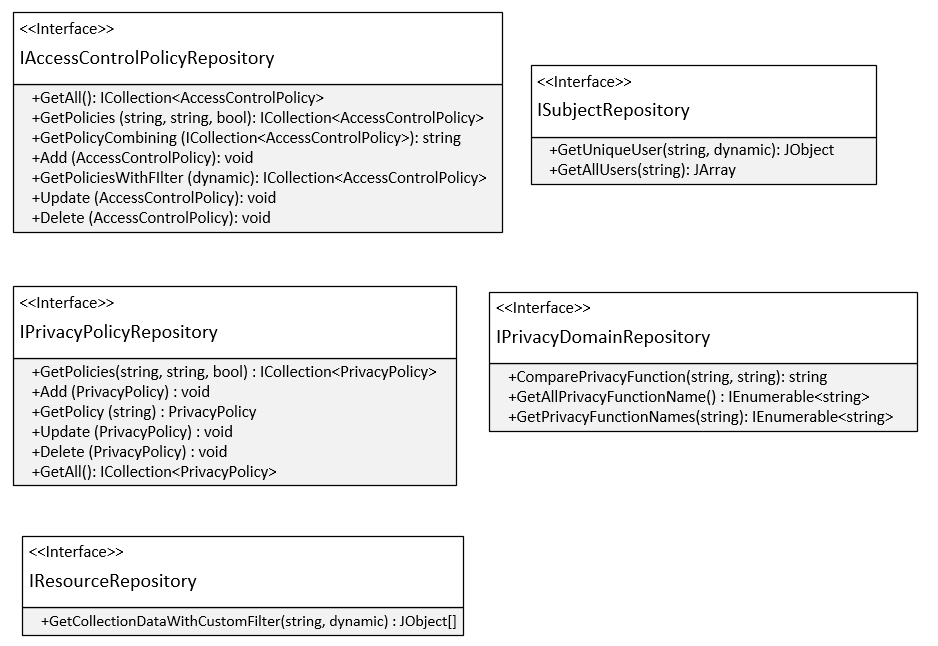
Framework architecture is divided into sub-components as followings:



* Models



* Repository Interfaces:



Figure[]

This component is designed following the Repository Pattern. Basically, it provides an abstraction of data, so that our framework can work with a simple abstraction that has an interface approximating that of a collection. Tight coupling of the database logic in the service make framework tough to test and extend further. Direct access of the data in the service logic may cause problems such as:

+ Difficulty applying Unit Test to the service logic.

+ Business logic cannot be tested without the dependencies of external systems like database.

+ Duplicate data access code throughout the business layer.

This component separates the data access logic and maps it to the models in the service component. It hides the details of data access from the service component. In other words, service component can access the data object without having the knowledge of the underlying data access architecture. In the future, underlying data sources or architecture can be changed without affecting the service component.

There are various advantages of the Repository Pattern including:

+ Service logic can be tested without need for an external source.

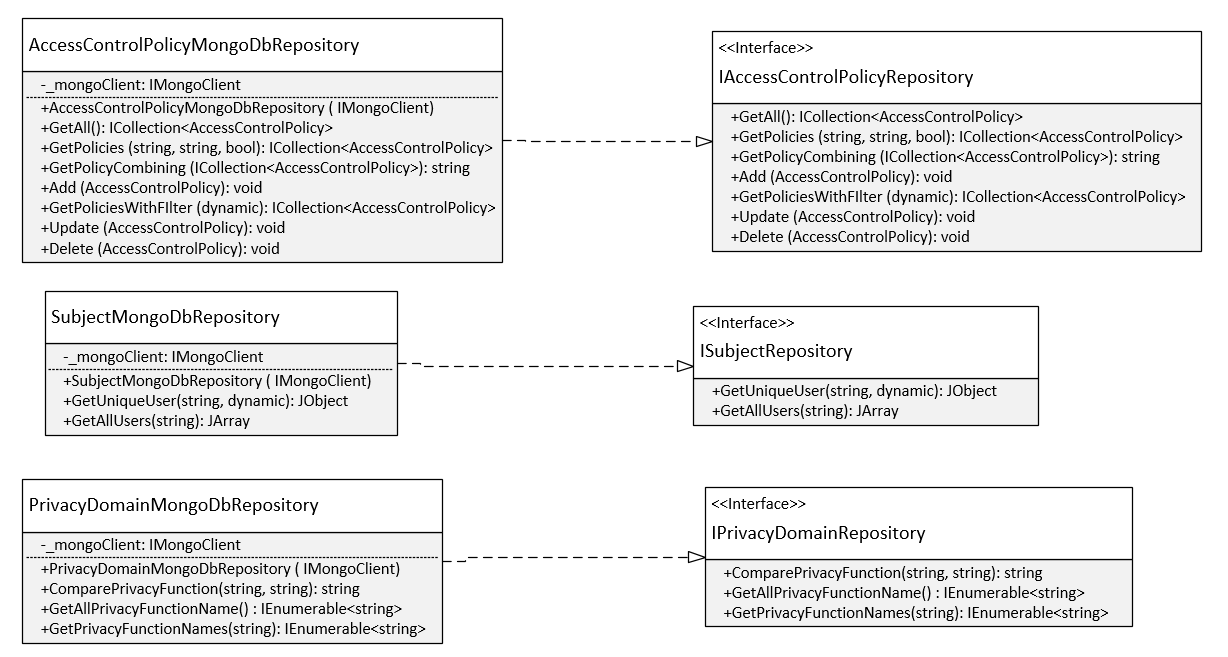
+ Database access logic can be tested separately.

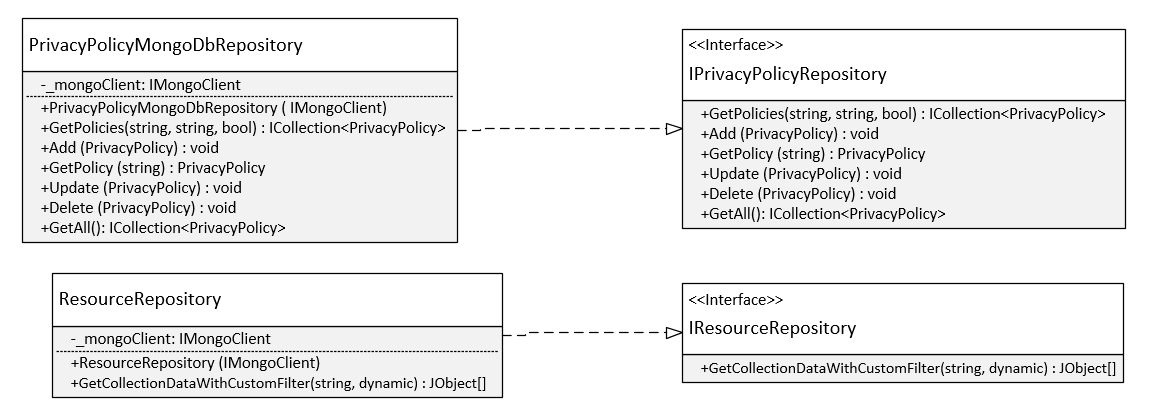
+ No duplicate of code.

+ Caching strategy for the datasource can be centralized.

+ Centralizing the data access logic, so code maintainability is easier.

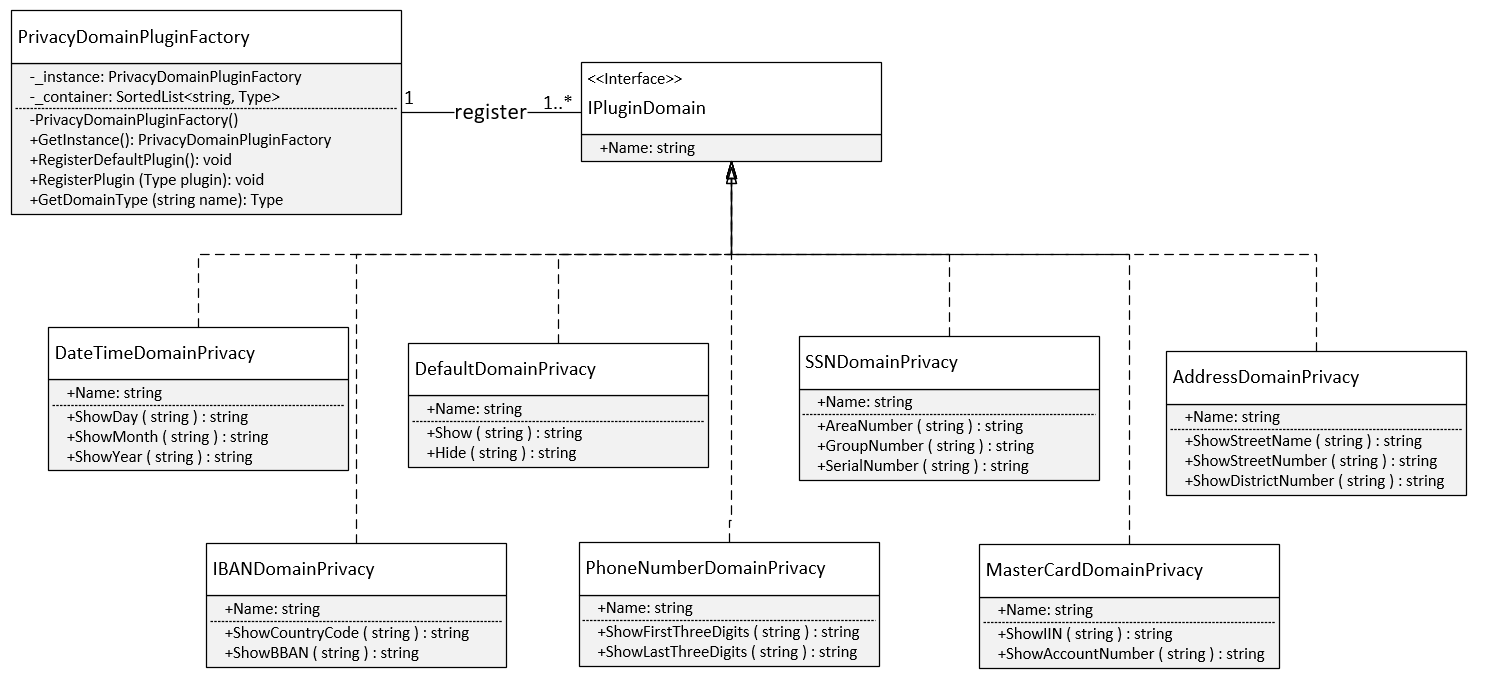
* MongoDB Repository (implemented from Repository Interfaces)





Figure[]

* Privacy Function Library:

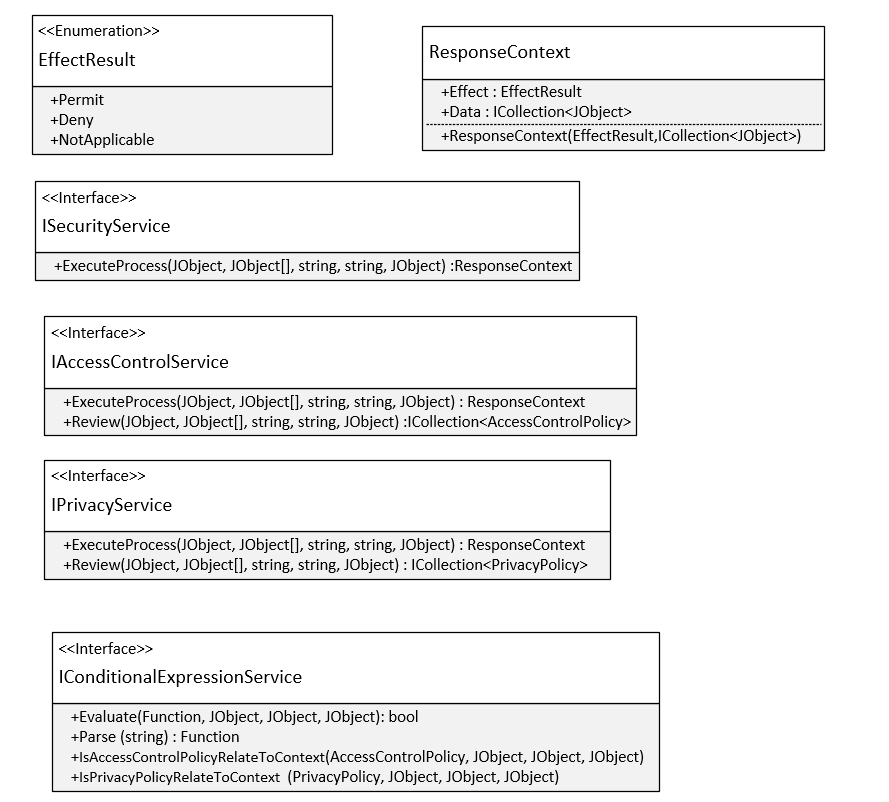


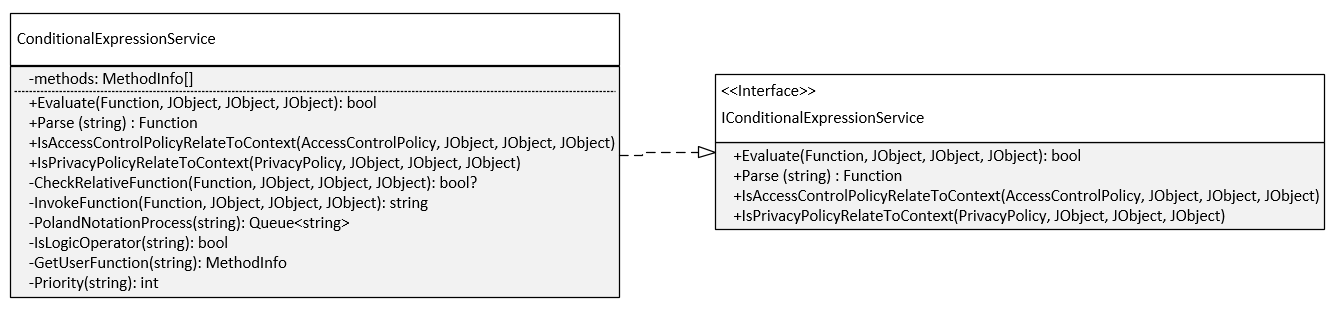
Privacy Function Library is designed based on these patterns: Singleton, Abstract Factory, Plugin Pattern. Plug-in is an extrnal piece of functionality that may be add to an existing system by abiding by a contract pre-defined by that system. The main reasons we apply this pattern in the design of Privacy Function Library:

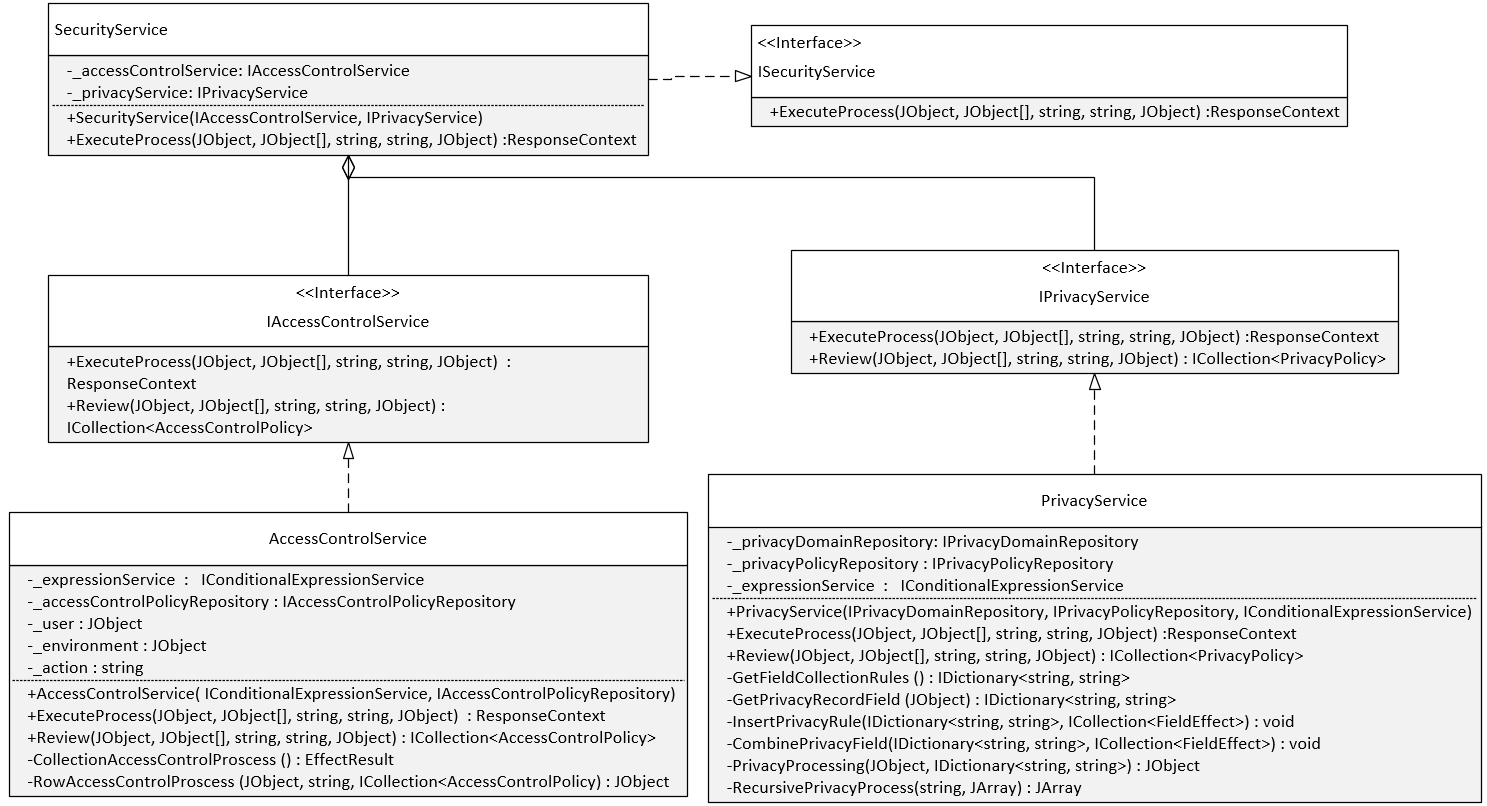
* Easily adding new features.
* Reducing the size of library.
* Other systems can extend this library in our framework without modifying the existing source code.

*PrivacyDomainPluginFactory* class provides a level of indirection that abstracts the creation of plugins without directly specifying their concrete classes. The *PrivacyDomainPluginFactory* object has the responsibility for providing creation of registered plugins for the entire framework. The other classes never create instance of plugin directly, they ask the factory to do that for them. This mechanism makes exchanging registered plugins easy because the specific class of the factory object appears only once in the framework - where it is instantiated. Because the registered plugin provided by the factory object is so pervasive, we have implemented it with Singleton pattern. It ensures that only one instance of *PrivacyDomainPluginFactory* class is created and provides a global point of access to the object.

* Service Interfaces:







1. Application Demonstration:
   1. Architecture:
   2. Prototype:
2. Experiment:

References:

<https://www.avatier.com/products/identity-management/resources/gartner-iam-2020-predictions/>

<https://www.tripwire.com/state-of-security/security-data-protection/security-controls/rbac-is-dead-now-what/>

<https://www.researchgate.net/publication/221312943_A_Semantic-Aware_Attribute-Based_Access_Control_Model_for_Web_Services>

<https://ico.org.uk/for-organisations/guide-to-data-protection/data-protection-principles/>