Spectrum Characterization and Occupancy Sensing (SCOS) Policy Architecture

*Nilesh Khambekar, Apurva Mody, Roger Hislop*

Date: Feb 16th, 2017

# Overview

The purpose of the Spectrum Characterization and Occupancy Sensing (SCOS) network is to acquire and make available data from networks of sensors. It is intended to establish a platform that enables “spectrum sensing as a service” and collective measurement efforts.

**<Purpose>**

Offer the ability to provision, orchestrate, and operate RF sensor network.

**<Goals>**

* Simplified occupancy sensing and spectrum characterization using a network of RF-sensors.
* Precise, Configurable, Automated, and Secure sensing operations
* Precise, Configurable, Automated Secure sensor data management
* Precise, Configurable, Automated Secure RF-sensor management

**<Objectives>**

1. Provide the ability to specify rules, constraints, and behavior for sensing tasks.
2. Provide the ability to control and prioritize the access to sensing devices.
3. Provide the ability to specify rules, constraints, and behaviors for how managing(accessing, distributing, storing) the sensing data.

### **SCOS Policy Architecture**

SCOS Policy

Sensor Management Policy

Sensing Data Management Policy

Sensing Operations Management Policy

Figure 1. SCOS Policy Architecture

(This is aligned with various SCOS usage models/ stakeholder-interests)

### **SCOS Policy Primitives**

### Actor

An actor is an entity that wishes to use the SCOS platform. An actor could be specified in terms of role, user, or a user-group.

### User

An user is an individual actor with specified name and access-credentials.

### User-group

A user-group is a logical collection of users. A user-group is specified with name and access-credentials.

### Role

A role is specified with a name. The role could be associated with specific SCOS services/functionality. A role could also be associated with privilege of various users of SCOS platform.

An user (or a user-group) is associated a role.

### Resource

SSD and Sensing data are two prime resources within SCOS platform.

### Resource-group

Multiple SSDs could be grouped together to jointly specify policies for using the SSDs. SSDs could possibly be grouped based on various attributes such as location, SSD-hardware-type, SSD-software-type.

### Namespace

Actors or resources are associated with a namespace. This avoids name collisions and enables to identify actors or resources uniquely (when the names have been reused across namespaces).

### Task

A task represents an action within SCOS platform. A task could be towards sensing task, sensor-management task, or sensing-data-management task.

### Task-group

Multiple tasks could be grouped together for convenience in specifying policies. For example, various tasks that can be performed towards sensor-management for a particular SSD operator could be grouped together and referred to in the SCOS policy. Similarly, sensing data management related tasks could be grouped together for precisely and conveniently specifying sensing-data-management related policies.

## Policy Description

Sensing management , Sensor management, and sensing data management functions are accomplished using SCOS policies. SCOS policy is expressed using JSON (JavaScript Object Notation).

There are three separate JSON files associated with three different functions.

|  |  |
| --- | --- |
| Function | Policy description |
| Sensing management | Sensing\_policy.json |
| Sensor management | Sensor\_policy.json |
| Sensing data management | Sensing\_data\_policy.json |

### SCOS Policy-types

There are two types of SCOS policies:

1. Actor-centric: which specifies which resources/functions a role, a user, or a group-of-users is permitted to exercise. The default case with user-centric policies is deny I.e. unless a matched user-policy is found, user is denied access to resource/services.
2. Resource-centric: which specifies what resource operations can be permitted and conditions/rules that must be satisfied to perform these operations. There can be a default-permit or default-deny policy to restrict access to SCOS platform resources/APIs.

### SCOS Policy Structure

Each SCOS policy is associated with a name, namespace, policy-type, scope, optional description, and one or more statement(s). A policy could be added, modified, removed, queried, and enabled/distabled.

Following figure shows the structure of a SCOS policy.

**SCOS Policy**

Top-level attributes

Statement

Statement

**Statement**

StatementId

Action

Actor

Resource

Task

Condition-Block

Figure 2. SCOS Policy Structure

Example:

{

# "Version": "2017-02-15",

 "Policy": {

 "namespace": "OperatorFoo",

 "name": "Calibration-access-control",

 "description": "This policy added by FooAdmin On this date."

 "type": "resource-centric",

 "Action": "permit",

"Resource": "Foo:Sensors::\*"

"Task": "Calibration"

"Scope": "Sensor-management:"

 "Condition": {

 "equals" : {

 "footag" : "bar"

 }

 }

}

### SCOS Policy Statement

Each statement from a actor-centric policy identifies resource(s) that the user or user-group is allowed to access under specified rule/condition is matched. Each statement from a resource-centric policy identifies task(s) permitted on a resource or resource-group when a specified rule/condition is matched. A statement is associated with a statement-id. Actor is optional within a resource-centric policy statement (All actors authorized to use the resource per actor-centric policy can execute the tasks specified in the resource-centric policy).

### Conditions

A condition is specified with a triplet of field(key), conditional-operator, and value. Condition is optional within a statement.

A condition evaluates whether a field meets certain criteria. Following table identifies various conditional operators.

|  |  |
| --- | --- |
| **Conditional-operator Name** | **Syntax** |
| equals | "equals" : "<value>" |
| Like |  "like" : "<value>"  |
| Contains |  "contains" : "<value>" |
| In | "in" : [ "<value1>","<value2>" ]  |
| Exists |  "exists" : "<bool>"  |
| LessThan |  "lessthan" : "<value>"  |
| GreaterThan | "greaterthan" : "<value>"  |
| LessThanEquals |  "lessthanequals" : "<value>"  |
| GreaterThanEquals | "greaterthanequals" : "<value>"  |

### Logical Operators

Logical operators enable to manipulate or combine multiple conditions. Following table specifies the logical operators.

|  |  |
| --- | --- |
| **Logical operator**  | **Syntax** |
| Not | “not”: {<condition>}  |
| AllOf |  "allOf" : [ {<condition>},{<condition>}]  |
| AnyOf |  "anyOf" : [ {<condition>},{<condition>}]  |

### Aliases

Aliases add convenience. Using aliases, multiple users can be combined together or multiple resources can be combined together to be referred in the SCOS policy . Furthermore, multiple tasks can be combined using task-groups.

Furthermore, locations could be specified using aliases to capture latitude, longitude, and altitude. A group of frequencies could also be combined using aliases. A group of time-slots also could be combined using aliases.

### SCOS Policy Dictionary

**Keywords**

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| Namespace |  |
| Name |  |
| Type |  |
| Resource |  |
| Service |  |
| User |  |
| Role |  |
| Action |  |
| StatementId |  |

**Actions**

|  |  |
| --- | --- |
| **Action** | **Description** |
| Permit |  |
| Deny |  |

**Policy-Scope**

|  |  |
| --- | --- |
| **Scope** | **Description** |
| Sensing-management | Sensing-scoped-policy |
| Sensing-data-management | Sensing-data-scoped-policy |
| Sensor-management | sensor-scoped-policy |

**Policy-Type**

|  |  |
| --- | --- |
| **Type** | **Description** |
| Actor-centric |  |
| Resource-centric |  |

**Fields**

|  |  |
| --- | --- |
| Field | Description |
| Location |  |
| Date |  |
| Frequency |  |
| Time |  |
| Altitude |  |
| Latitude |  |
| Longitude |  |
| SensingDuration |  |
| SamplingRate |  |
| ScanBandwidth |  |
| ScanResolution | SamplingRate divided by FFTSize |
| Calibration |  |
| Region | A region is specified based on a center location and radius in kilometers. |
|  |  |

### Examples

1. Users with role FooAdmin can reboot all SSDs within namespace OperatorFoo
2. User Sam can request sensing-data from Region SFLibrary in frequency-band 3.5GHz
3. User Dave cannot receive SensingData from SSDs within namespace OperatorFoo
4. Resource-group FooSDR permits software-update if role is FooAdmin and imageAuthenticationTag contains bar-signature.

## Policy Evaluation

Whenever an SCOS API needs to be executed, SSM needs to confirm if the action is permitted by evaluating related policies.

There exist three scopes for SCOS policies: Sensing management scope, Sensing-data management scope, and Sensor-management scope. Depending on the API, policies in the appropriate scope are looked up.

The second step is ensure that the actor is authorized to perform tasks on the resource. A specific accept policy or default-accept policy should be match for the user, user-group, or role.

The final step is ensure if the resource permits the intended task. A specific accept policy or default-accept policy should be match for the resource, or resource-group.

## Appendix: Example JSON Policy Files

## Appendix: Implementation Guidelines

### Sensor Management

Discovery Management

Power Management

Reporting Management

Health Management

Calibration Management

Software Management

Sensing Management

System Management

Networking Management

Figure 2: Sensor Management

### Sensing Operations Management

Spectrum-Scan Management

Sensor-Mapping Management

Sensor-schedule Management

 Sensor-capability Management

Calibration Management

Spectrum Characterization Management (Next release)

(one of the operations under schedule could be synchronization)

(Sensor mapping management includes managing sensor mobility)

### Sensing Data Management

Data Distribution/Storage Management

User Management

Access Management

 Database System Management

### Creating and Managing Actors and Resources Profiles

SCOS Policy evaluation requires creation of actor and resource profiles. An SCOS system may have root user which has the privilege to add other actors and resources.

**Actor-Profiles**

Profiles for Roles, Users, and/or User-groups could to be specified under the file scos\_actor\_profile. CLI, REST API or alternate interface could be used for managing the scos\_actor\_profile.

**Resource-Profiles**

Profiles for sensor-resource, sensing-data-resource could to be specified under the file scos\_resource\_profile. CLI, REST API or alternate interface could be used for managing the scos\_resource\_profile. The tasks that can be performed on a resource are specified in the resources’ profile. The tasks could be specified collectively using a task-group.