Orion Context broker

Orion Context Broker (OCB) permite administrar todo el ciclo de vida de la información en el conjunto de Fiware.

A través del OCB es posible registrar entidades de contexto y administrarlos a través de actualizaciones y consultas. Además, permite suscribirse a la información de contexto para que cuando se produzca alguna modificación se reciba una notificación.

Información de contexto

Es el valor del atributo que caracteriza una entidad en una aplicación. Esta información proviene de diferentes fuentes y protocolos.

Modelo NSGI

NSGI (Next Generation Service Interface Context Enabler) está basado en la definición de entidades y atributos.

Entidades son la representación virtual de todos los objetos físicos en el mundo.

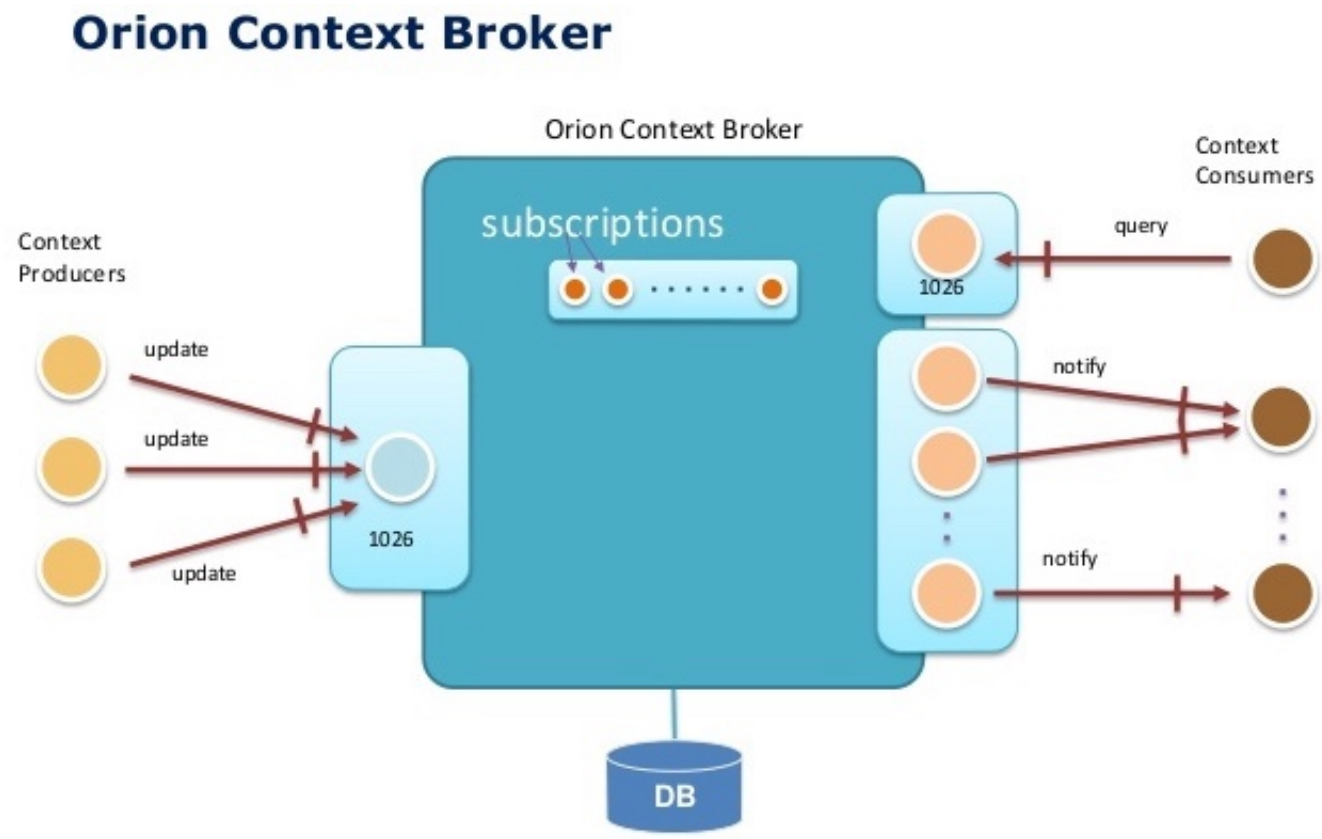
Cualquier información disponible sobre las entidades físicas son expresadas en forma de **atributos** de una manera virtual.

Orion Context Broker

Para iniciar el servidor de OCB es necesario realizar los siguientes pasos

```
# Utilizando la terminal.
#### Set up
# crear carpeta
mkdir curso
# cambiar de directorio
cd curso
# bajar estructura de las máquinas virtuales
git clone https://github.com/CarlosUrteaga/Fiware.git
# Cambiar a directorio Fiware
cd Fiware
cd vm-fiware-orion
# iniciar máquina virtual de Orion
vagrant up
# conectar a máquina virutal por medio de SSH
vagrant ssh
#iniciar docker de OCB
docker-compose up
```

La siguiente figura muestra de forma general el funcionamiento del OCB. Este es un intermediario entre los productores (dispositivos) y los consumidores (aplicaciones).



- Los productores (estaciones) son responsables de actualizar la información en la base de datos.
- Los consumidores realizan consultas para obtener el estado actual. Además se tiene la posibilidad obtener notificaciones de atributos particulares.

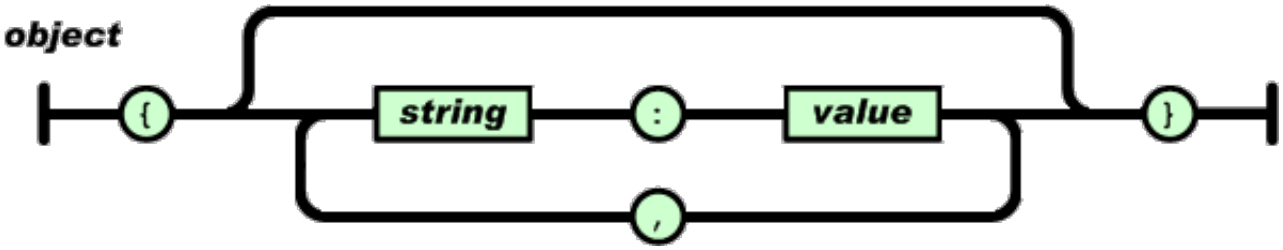
El formato de datos que utiliza el OCB es de tipo JSON.

JSON

El formato de datos JSON (Java Script Object Notation) es ligero para el intercambio de información, además de ser fácil de leer y de procesar.

El formato de JSON utilizado es una colección de nombres y valores. Estos valores pueden ser expresados como un objeto, estructura, registro, diccionario, tabla hash, etc.

Un Objeto JSON tienen las siguiente forma:




```
    { "entidad": atributo, "entidad":atributo, ...}
```

Es decir, se encuentran definidos entre llaves. *String* está definido como las propiedades de las entidades. *value* es el atributo de tipo booleano, objeto JSON, arreglo, número o nulo.

Ejemplo:

José tiene 21 años y está cursando con tres materias: Física, Matemáticas y Economía.

```
    {
        "nombre":"José",
        "edad":21,
        "Materias":
        {
            "Materias1":"Física",
            "Materias2":"Matemáticas",
            "Materias3":"Economía"
        }
    }
```

Consultas (RESTful)

OCB realiza consultas a la base de datos. Las típicas consultas son insertar, modificar y obtener.

OCB contiene una interfaz tipo web para realizar dichas consultas. Este es un servicio web de tipo REST (REpresentational state Transfer).

La forma de interactuar con este tipo de servicio web es a través de una URL, tipo de acción y un cuerpo con información.

Este tipo de servicios tiene la flexibilidad de obtener información de forma arborecente. Es decir, es posible obtener l actualizar l borrar información de una entidad completa o sólo valores de una entidad en especifico.

Ejercicio

Los siguientes ejercicios serán realizados utilizando el programa Insomnia.

Dentro de insomina se creará una carpeta con el nombre **Operaciones-Comunes**.

POST

Para generar una nueva petición que inserte una entidad en la base de datos se usará la operación *post*, el nombre sugerido es **inserta-entidad**.

```
En el campo de URL se pone la dirección donde se encuentra el OCB

    post http://url:1026/v2/entities
```

Ejemplo:

```
post: http://localhost:1026/v2/entities
Header:
    Content-type application/json

Body:
    {
        "id": "lugar01",
        "type": "ParkingSpot",
        "floor" :{
            "value":"PB",
            "type":"text"
        },
        "name" :{
            "value":"A-13"
        },
        "status" :{
            "value":"libre"
        }
    }
```

El **id**, representa el identificador del objeto mientras que **type** permite inferir el tipo atributo.

Para cada **id** existen varios **Type**, es decir, puede existir varios objetos con el mismo **id** siempre y cuando sean de distinto tipo.

Para los campos de atributos, muchas veces se puede inferir el tipo de acuerdo al valor, pero es una buena práctica incluir el tipo de dato (*type*).

GET

La operación utilizada para obtener información de la base de datos es el **get**. Es posible duplicar la consulta anterior y renombrarla, con el nombre **obten-todas-entidades**.

Para este caso, sólo se especifica el URL, sin body ni Content-type.

```
    get http://{url}:1026/v2/entities
```

ejemplo:

```
    get http://localhost:1026/v2/entities
```

No obstante si se está consultando un OCB en la nube, será necesario agregar en el campo X-Auth-Token el token que les fue asignado.

```
    get url   http://{url}:1026/v2/entities
headers X-Auth-Token    1O11Qj4FReeHTs0Rb5hVLYwKNHFbbu
```

Para consultar una sola entidad, el URL se especifica con el **id**.

```
    get http://{url}:1026/v2/entities/{id}
```

ejemplo:

```
    get http://localhost:1026/v2/entities/lugar01
```

Esta consulta permite notar que la base de datos infiere el tipo datos insertados.

Actualizar valores

Es posible actualizar varios valores o uno solo. En el caso de varios valores se utiliza el formato JSON mientras que en el otro se utiliza el *type* del valor a insertar.

```
    put http://{url}:1026/v2/entities/{id}
    Header:
    Body:
```

Ejemplo:

```
    put http://localhost:1026/v2/entities/lugar01/attrs
    Header
        Content-type application/json
Body:
{
    "name" :{
        "value":"A15"
    },
    "floor" :{
        "value":"PB"
    },
    "status" :{
        "value":"ocupado"
    }
}
```

```
    put http://{url}:1026/v2/entities/{id}/attrs/{value}/value
    Header:
    Body:
```

Ejemplo:

```
    put http://localhost:1026/v2/entities/lugar01/attrs/status/value
    Header
        Content-type text/plain
Body:
"free"
```

Delete

Es posible borrar atributos e identidades.

Para **borrar** un atributo se utiliza el comando **Delete**:

```
    delete http://url:1026/v2/entities/{id}/attrs/{value}
```

Para **borrar** una se utiliza la siguiente expresión:

```
    delete http://url:1026/v2/entities/{id}
```

Metadatos

Los metadatos se agregan en la nueva versión de NGSI. Simplemente se pone información adicional con la misma estructura. Por ejemplo, si se está manejando temperatura, se podría agregar algo así:

```
{
    "id":"lugar01",
    "type":"ParkingSpot",
    "temperature" :{
        "value":23,
        "metadata":{
            “precision”: {
                “type”:xxx,
                “value”: xxx
            }
        }
    ...
```

OCB Query Language

Para practicar con algunos de los comandos de OCB, primero poblaremos la BD con varias entidades. Se pueden agregar las entidades una a una con el método POST, pero también es posible enviar un conjunto de entidades especificando un APPEND con el URL de update, de lo contrario se tendrían que realizar post por cada entidad insertada.

```
{
    "actionType":"APPEND",
    "entities":[
    {
        "id": "CampusA01",
        "type": "ParkingSpot",
        "name" :{
            "value":"A-01"
        },
        "floor" :{
            "value":"PB"
        },
        "status" :{
            "value":"libre"
        }
    },
```

```
    {
        "id": "CampusA02",
        "type": "ParkingSpot",
        "name" :{
            "value":"A-02"
        },
        "floor" :{
            "value":"PB"
        },
        "status" :{
            "value":"libre"
        }
    },
    {
        "id": "CampusA13",
        "type": "ParkingSpot",
        "name" :{
            "value":"A-11"
        },
        "floor" :{
            "value":"PA"
        },
        "status" :{
            "value":"libre"
        }
    },
    {
        "id": "CampusB01",
        "type": "ParkingSpot",
        "name" :{
            "value":"A-01"
        },
        "floor" :{
            "value":"PB"
        },
        "status" :{
            "value":"ocupado"
        }
    },
    {
        "id": "CampusB02",
        "type": "ParkingSpot",
        "name" :{
            "value":"A-02"
        },
```

```
        "floor" :{
            "value":"PB"
        },
        "status" :{
            "value":"ocupado"
        }
    },
    {
        "id": "CampusB13",
        "type": "ParkingSpot",
        "name" :{
            "value":"B-13"
        },
        "floor" :{
            "value":"PA"
        },
        "status" :{
            "value":"desconocido"
        }
    },
    {
        "id": "CampusC01",
        "type": "ParkingSpot",
        "name" :{
            "value":"A-01"
        },
        "floor" :{
            "value":"PB"
        },
        "status" :{
            "value":"ocupado"
        }
    },
    {
        "id": "CampusC02",
        "type": "ParkingSpot",
        "name" :{
            "value":"A-02"
        },
        "floor" :{
            "value":"PB"
        },
        "status" :{
            "value":"ocupado"
        }
```

```
    },
    {
        "id": "CampusC13",
        "type": "ParkingSpot",
        "name" :{
            "value":"B-11"
        },
        "floor" :{
            "value":"PA"
        },
        "status" :{
            "value":"desconocido"
        }
    }
    ]
}
```

Al realizar consultas, el OCB entrega únicamente 20 elementos por defecto. Si se desea traer más, se agrega el parámetro limit al query.

```
En insomnia, se tiene una ventana especial para agregar los parámetros del query.
    limit=1
```

También se puede especificar el offset a partir del cual se leerán los valores.

```
oftset=5
```

Asimismo, si no deseamos recibir la información en formato NSGI, se puede especificar como opción keyValues como se muestra en el siguiente ejemplo:

```
options=keyValues
```

Otro caso es el obtener los lugares vacios, pero sólamente el nombre y el piso, y sin formato NSGI

```
q           status==desconocido
attrs       name,floor,status
options keyValues
```

En los siguientes ejemplos se muestran consultas para un determinado tipo, para combinar varios

identificadores, para elementos que coinciden con una expresión regular.

```
?
attrs   name,floor
options keyValues
idPattern   ^Campus[A-C]
```

Datos geo-referenciados

A una entidad puede tener el atributo de ubicación (*location*) para resolver los diferentes problemas.

- Lugares de interes
- Servicios cercanos
- Notificaciones

Ejemplo Insertar lugares interés de la ciudad de México para mostrar la capacidad de geolocalizacíon en OCB.

```
post http://localhost:1026/v2/op/update
Body: json

{
  "actionType": "APPEND",
    "entities": [
        {
            "id": "Catedral",
            "type": "PointOfInterest",
            "category": {
                "type": "Text",
                "value": "iglesia",
                "metadata": {}
            },
            "location": {
                "type": "geo:point",
                "value": "19.435433, -99.133072",
                "metadata": {}
            },
            "name": {
                "type": "Text",
                "value": "Catedral Metropolitana",
                "metadata": {}
            },
            "postalAddress": {
```

```
                "type": "StructuredValue",
                "value": {
                    "addressCountry": "MX",
                    "addressLocality": "México Ciudad de México",
                    "addressRegion": "Ciudad de México"
                }
            }
        },
        {
            "id": "Zocalo",
            "type": "PointOfInterest",
            "category": {
                "type": "Text",
                "value": "Plaza",
                "metadata": {}
            },
            "location": {
                "type": "geo:point",
                "value": "19.432579, -99.133287",
                "metadata": {}
            },
            "name": {
                "type": "Text",
                "value": "Zocalo",
                "metadata": {}
            },
            "postalAddress": {
                "type": "StructuredValue",
                "value": {
                    "addressCountry": "MX",
                    "addressLocality": "México Ciudad de México",
                    "addressRegion": "Ciudad de México"
                }
            }
        },
        {
            "id": "PalacioNacional",
            "type": "PointOfInterest",
            "category": {
                "type": "Text",
                "value": "Edificio",
                "metadata": {}
            },
            "location": {
                "type": "geo:point",
```

```
                "value": "19.432336, -99.131452",
                "metadata": {}
            },
            "name": {
                "type": "Text",
                "value": "Palacio Nacional",
                "metadata": {}
            },
            "postalAddress": {
                "type": "StructuredValue",
                "value": {
                    "addressCountry": "MX",
                    "addressLocality": "México Ciudad de México",
                    "addressRegion": "Ciudad de México"
                }
            }
        },
        {
            "id": "BellasArtes",
            "type": "PointOfInterest",
            "category": {
                "type": "Text",
                "value": "Edificio",
                "metadata": {}
            },
            "location": {
                "type": "geo:point",
                "value": "19.435180, -99.141207",
                "metadata": {}
            },
            "name": {
                "type": "Text",
                "value": "Palacio de Bellas Artes",
                "metadata": {}
            },
            "postalAddress": {
                "type": "StructuredValue",
                "value": {
                    "addressCountry": "MX",
                    "addressLocality": "México Ciudad de México",
                    "addressRegion": "Ciudad de México"
                }
            }
        },
        {
```

```
            "id": "TorreLatino",
            "type": "PointOfInterest",
            "category": {
                "type": "Text",
                "value": "Edificio",
                "metadata": {}
            },
            "location": {
                "type": "geo:point",
                "value": "19.433874, -99.140685",
                "metadata": {}
            },
            "name": {
                "type": "Text",
                "value": "Torre Latino",
                "metadata": {}
            },
            "postalAddress": {
                "type": "StructuredValue",
                "value": {
                    "addressCountry": "MX",
                    "addressLocality": "México Ciudad de México",
                    "addressRegion": "Ciudad de México"
                }
            }
        }
    ]
}
```


Ahora vamos a buscar puntos dentro de una figura geométrica. Para ello se utiliza la API V1, en la que se usan queries de contexto. Para especificar puntos fuera de una circunferencia, el método es POST y el body tiene lo siguiente:

```
POST localhost:1026/v1/queryContext

{
    "entities": [
        {
            "type": "PointOfInterest",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "attributes":[
        "location","name"
    ],
    "restriction": {
        "scopes": [
            {
                "type": "FIWARE::Location",
                "value": {
                    "circle": {
                        "centerLatitude": "19.432594",
                        "centerLongitude": "-99.133017",
                        "radius": "50",
                        "inverted":"true"
                    }
                }
            }
        ]
    }
}
```

Las restricciones para la búsqueda es un círculo con el centro de acuerdo a coordenadas geográficas y radio está expresado en metros. **Inverted: true** significa que recupera todos los que no están dentro del círculo.

Ahora hagamos un query de un polígono, el cual se de acuerdo a la siguiente consulta.

```
post http://localhost:1026/v2/op/update

{
    "entities": [
        {
            "type": "PointOfInterest",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "attributes": [
        "location"
    ],
    "restriction": {
        "scopes": [
            {
                "type": "FIWARE::Location",
                "value": {
                    "polygon": {
                        "vertices": [
                            {
                                "latitude": "19.432218",
                                "longitude": "-99.133836"
                            },
                            {
                                "latitude": "19.431994",
                                "longitude": "-99.132480"
                            },
                            {
                                "latitude": "19.433089",
                                "longitude": "-99.132380"
                            },
                            {
                                "latitude": "19.433254",
                                "longitude": "-99.133841"
                            }
                        ],
                        "inverted": "false"
                    }
                }
            }
        ]
    }
}
```

El polígono se forma por un conjunto de vértices.

Suscriptor

La siguiente parte mostrará un ejemplo de suscriptor de un cuarto con temperatura.

Insertar la siguiente estructura.

```
post http://localhost:1026/v2/entities

{
    "id":"Cuarto01",
    "type":"Room",
    "temperature": {
        "value":"23"
    },
    "pressure":{
        "value": 43
    }
}
```

Generar un método que actualice la temperatura

```
put     http://localhost:1026/v2/entities/Cuarto01/attrs/temperature/value

Header:
    text/plain
Body:
    20.1
```

Generar suscriptor utilizando el método POST

```
{
    "description": "Update average rating",
    "subject": {
        "entities": [
            {
                "id": "Cuarto01",
                "type": "Room"
            }
        ],
        "condition": {
            "attrs": [
                "id",
                "temperature"
            ]
        }
    },
    "notification": {
        "httpCustom": {
            "url": "http://192.168.83.2:8080/notifications",
            "headers": {
                "Content-Type": "text/plain"
            },
            "method": "POST",
            "qs": {
                "type": "${type}"
            },
            "payload": "La temperatura ${temperature} grados en ${id}"
        },
        "attrs": [
            "id",
            "temperature"
        ]
    },
    "expires": "2020-04-05T14:00:00.00Z",
    "throttling": 1
}
```

Iniciar máquina virtual de suscriptor

```
cd <path>/Fiware
cd vm-fiware-consumer
# iniciar máquina virtual de Orion
vagrant up
# conectar a máquina virutal por medio de SSH
vagrant ssh
export JAVA_HOME=/opt/jdk1.8.0_144

mvn -f fiware-orion-subscriber/pom.xml spring-boot:run
```

Realizar cambios en la variable y entrar a la siguiente página web

```
192.168.83.2:80
```