

# Relazione Elaborato Big Data Architectures

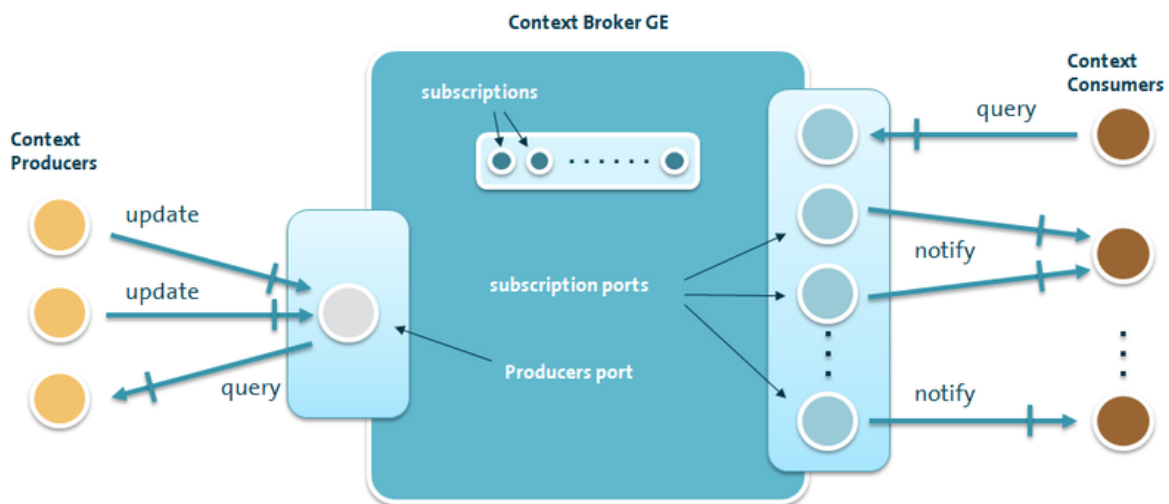
Il suddetto elaborato è relativo all'inserimento delle funzionalità di Multi-Service/Tenant e di Service-Path, fornite dall'Orion Context Broker, all'interno di IoT Directory, una web application che fa parte del sistema Snap4City ([www.snap4city.org](http://www.snap4city.org)), sviluppata dal laboratorio DISIT ([www.disit.org](http://www.disit.org)) dell'università degli studi di Firenze.

Questa relazione contiene:

- un'introduzione all'Orion Context Broker ([fiware-orion.readthedocs.io](http://fiware-orion.readthedocs.io)), un broker sviluppato da FIWARE ([www.fiware.org](http://www.fiware.org)), e alle funzionalità di Multi-Service/Tenant e di Service-Path che esso fornisce;
- un'introduzione all'IoT directory e alle sue funzionalità principali;
- una breve analisi dei casi d'uso da implementare e le principali attività svolte per la loro implementazione.

## Introduzione a Orion Context Broker

Possiamo definire un **context broker** come un servizio che è progettato per raccogliere dati di contesto (*context data*) di diverse tipologie, provenienti da sorgenti diverse e a velocità diverse. In particolare, il **FIWARE Context Broker** permette la pubblicazione di informazioni di contesto da parte dei **Context Producer**, in modo che le informazioni pubblicate siano disponibili ai **Context Consumers**, interessati all'elaborazione di tali informazioni.



Il FIWARE Context Broker supporta due tipologie di comunicazione:

- *Push*: il Context Producer invia le informazioni di contesto al Context Broker, che a sua volta le invia ai Context Consumer interessati (interessamento viene esplicitato tramite le *subscription*) a quelle informazioni.
- *Pull*: Il Context Broker richiede le informazioni dai Context Producers tramite query (in questo caso i Context Producers prendono il nome di **Context Providers**); similmente i Context Consumers richiedono le informazioni al/ai Broker (modalità on-request).

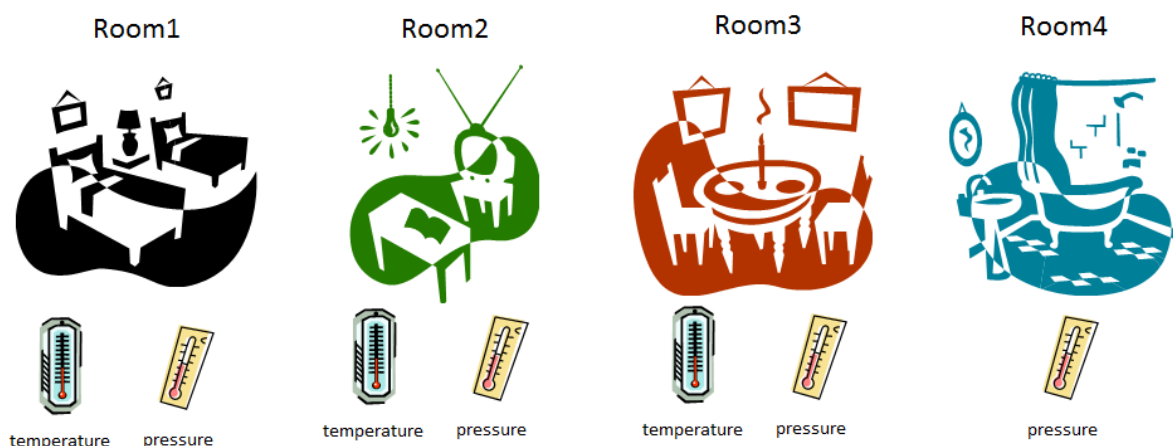
Una funzionalità fondamentale fornita dal FIWARE Context Broker consiste nel completo disaccoppiamento tra producers e consumers; da una parte i Context Producers pubblicano dati senza sapere quali, dove, e quando i Context Consumers utilizzeranno quei dati, dall'altra i Context Consumers utilizzano le informazioni di contesto senza sapere quale Context Producer le ha pubblicate, dato che sono interessati alle informazioni e non a "chi" le ha pubblicate. Il FIWARE Context Broker rappresenta quindi un ponte che permette alle IoT app di gestire gli eventi in maniera semplice, nascondendo la complessità della raccolta dei dati dai sensori.

Tutte le comunicazioni che avvengono tra i vari elementi avvengono tramite l'interfaccia RESTful **NGSI**. Un'implementazione di tale specifica e del FIWARE Context Broker viene fornita dall'**Orion Context Broker**, che permette di gestire il ciclo di vita delle informazioni di contesto. Tramite l'Orion Context Broker è possibile creare e gestire elementi di contesto; inoltre è possibile sottoscrivere a certe informazioni in modo da venire notificati quando si verificano determinati eventi.

Le principali funzionalità di **Context Management** fornite dall'Orion Context Broker verranno descritte di seguito tramite un caso di esempio.

## Caso di esempio

Supponiamo che il nostro Orion Context Broker debba gestire delle stanze; al loro interno sono presenti dei sensori che misurano la temperatura (in °C) e la pressione atmosferica (in mmHg). Il Broker dovrà interagire con le applicazioni che producono le informazioni e con quelle che le utilizzano.



Le richieste inviate al broker verranno fatte utilizzando *PostMan* (<https://www.getpostman.com/>), che consente di effettuare richieste *http* tramite una *GUI* intuitiva anche per utenti non esperti. È possibile utilizzare strumenti come *curl* se si desidera utilizzare una *CLI*.

In questo esempio, il context broker è raggiungibile all'indirizzo `localhost:1026`

## Context Management

Le operazioni di context management, che comprendono le operazioni creazione, query e modifica delle entità sono i blocchi principali della gestione del contesto.

### Creazione delle Entità

La prima cosa da fare è quella di rendere Orion "consapevole" dell'esistenza delle entità. La "creazione" (i.e. la registrazione) di una singola entità viene fatta tramite una richiesta `POST /v2/entities`. In questo esempio, vengono create la `Room1` e la `Room2`.

- **Room 1**

- Richiesta

```
1 POST localhost:1026/v2/entities .
2 Content-Type: application/json
```

```
1 {
2   "id": "Room1",
3   "type": "Room",
4   "temperature": {
5     "value": 23,
6     "type": "Float"
7   },
8   "pressure": {
9     "value": 720,
10    "type": "Integer"
11  }
12 }
```

Il body della *POST* contiene l'*id* e il *tipo* dell'entità, che verranno utilizzati per identificarla, e un insieme di attributi, ciascuno dei quali contiene un tipo, un valore ed eventuali metadati.

N.B.: Orion non effettua nessun controllo sui tipi.

- Risposta: se tutto va a buon fine, si riceve una risposta di tipo `201 Created`

- **Room 2**

- Richiesta

```
1 POST localhost:1026/v2/entities .
2 Content-Type: application/json
```

```
1 {
2   "id": "Room2",
3   "type": "Room",
4   "temperature": {
5     "value": 21,
6     "type": "Float"
7   },
8   "pressure": {
9     "value": 711,
10    "type": "Integer"
11  }
12 }
```

## Query sulle Entità

Per ottenere le informazioni contenute sul broker è possibile utilizzare `GET /v2/entities` (per tutte le entità presenti sul broker), `GET /v2/entities/{id}` per ottenere le informazioni sulla singola entità, e `GET /v2/entities/{id}/attrs/{attribute}` per ottenere le informazioni su un singolo attributo.

- **Tutte le entità:**

- Richiesta

```
1 GET localhost:1026/v2/entities .
2 Accept: application/json
```

- Risposta

```
1 [
2   {
3     "id": "Room1",
4     "type": "Room",
5     "pressure": {
6       "type": "Integer",
7       "value": 720,
8       "metadata": {}
9     },
10    "temperature": {
11      "type": "Float",
12      "value": 23,
13      "metadata": {}
14    }
15  },
16  {
17    "id": "Room2",
18    "type": "Room",
19    "pressure": {
20      "type": "Integer",
21      "value": 711,
22      "metadata": {}
23    },
24    "temperature": {
25      "type": "Float",
26      "value": 21,
27      "metadata": {}
28    }
29  }
30 ]
```

- Singola entità:

- Richiesta

```
1 GET localhost:1026/v2/entities/Room1 .
2 Accept: application/json
```

- Risposta

```
1 {
2   "id": "Room1",
3   "type": "Room",
4   "pressure": {
5     "type": "Integer",
6     "value": 720,
7     "metadata": {}
8   },
9   "temperature": {
10    "type": "Float",
11    "value": 23,
```

```
12     "metadata": {}
13   }
14 }
```

- **Singolo attributo:**

- Richiesta

```
1 GET localhost:1026/v2/entities/Room1/attrs/temperature .
```

- Risposta

```
1 {
2   "type": "Float",
3   "value": 23,
4   "metadata": {}
5 }
```

È possibile ottenere i dati in forma *key-value* utilizzando il parametro `options=keyvalue`. Si possono effettuare anche delle operazioni di filtraggio. Si rimanda a [https://fiware-orion.readthedocs.io/en/master/user/walkthrough\\_apiv2/index.html#query-entity](https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html#query-entity) e a [https://fiware-orion.readthedocs.io/en/master/user/walkthrough\\_apiv2/index.html#getting-all-entities-and-filtering](https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html#getting-all-entities-and-filtering) per ottenere ulteriori informazioni al riguardo.

## Aggiornamento delle Entità

Ci sono due modi per aggiornare le entità presenti sul context broker: il primo prevede l'invio di una richiesta `PATCH`, mentre il secondo prevede l'invio di una richiesta `PUT`.

- **Update tramite PATCH:**

- Richiesta: La sua struttura è molto simile a quella fatta per creare l'entità

```
1 PATCH localhost:1026/v2/entities/Room1/attrs .
2 Content-Type: application/json
```

```
1 {
2   "temperature": {
3     "value": 26.5,
4     "type": "Float"
5   },
6   "pressure": {
7     "value": 763,
8     "type": "Float"
9   }
10 }
```

- Risposta: `204 No Content`

- **Update tramite PUT:**

- Richiesta

```
1 PUT localhost:1026/v2/entities/Room1/attrs/temperature/value .
2 Content-Type: text/plain
```

- Risposta: 204 No Content

## Sottoscrizioni

L'Orion Context Broker ha una feature da cui trarre vantaggio, ossia l'abilità di sottoscrivere verso delle informazioni di contesto (i.e. verso delle entità) in modo che quando "avviene qualcosa" l'applicazione riceva dal broker notifiche asincrone. In questo modo non è necessario effettuare pooling. Il broker "consegnerà" le informazioni di interesse quando arriveranno.

### Creazione di una Sottoscrizione

Le sottoscrizioni vengono inviate tramite una richiesta `POST /v2/subscriptions`. Di seguito si fornisce un esempio.

- Richiesta

```
1 POST localhost:1026/v2/subscriptions .
2 Content-Type: application/json
```

```
1 {
2   "description": "A subscription to get info about Room1",
3   "subject": {
4     "entities": [
5       {
6         "id": "Room1",
7         "type": "Room"
8       }
9     ],
10    "condition": {
11      "attrs": [
12        "pressure"
13      ]
14    },
15  },
16  "notification": {
17    "http": {
18      "url": "http://192.168.128.139:1028/accumulate"
19    },
20    "attrs": [
21      "temperature"
22    ]
23  },
24  "expires": "2040-01-01T14:00:00.00Z",
25  "throttling": 5
26 }
```

- Il sottocampo `entities` all'interno del campo `subject` e quello `attrs` (lista di elementi) all'interno del campo `notification` definiscono il contesto dei messaggi di notifica.
- L'elemento `condition` definisce i "trigger" per le sottoscrizioni. il sottocampo `attrs` contiene gli attributi che quando creati/modificati innescano l'invio di una notifica. Se la lista `condition.attrs` contiene più elementi, esso va considerato come una

- condizione di OR, i.e. la variazione di uno dei vari elementi della lista causa l'invio di una notifica. Se lasciato vuoto, la notifica viene inviata ad ogni cambiamento. Nell'esempio presentato sopra, viene inviata una notifica ad ogni variazione della pressione.
- L'URL da utilizzare per inviare le notifiche viene definito tramite il sottocampo `url`. È possibile utilizzare un solo URL per sottoscrizione, anche se è possibile effettuare più sottoscrizioni sullo stesso elemento di contesto (i.e. entità o attributo).
  - Il sottocampo `expires` definisce il tempo di scadenza della sottoscrizione, oltre il quale viene semplicemente ignorata dal broker, ma non viene eliminata dal database. La durata della sottoscrizione può essere modificata successivamente. Se il campo viene omissso, la sottoscrizione è permanente.
  - L'elemento `throttling` viene utilizzato per specificare un intervallo di tempo (in secondi) minimo tra una notifica e l'altra, indipendentemente da quanti cambiamenti sono avvenuti.
- Risposta: Si riceve una risposta del tipo `201 Created`. Questa risposta contiene l'header `Location`, che contiene un ID a 24 cifre esadecimali della sottoscrizione (e.g. `57458eb60962ef754e7c0998`), che può essere utilizzato per la modifica e/o la rimozione della sottoscrizione. Utilizzando l'accumulator server disponibile su <https://github.com/telefonicaid/fiware-orion/blob/master/scripts/accumulator-server.py>, è possibile notare che, quando viene inviata la sottoscrizione, viene stampato il seguente risultato:

```
1 POST http://192.168.1.137:1028/accumulate .
2 Fiware-Servicepath: /
3 Content-Length: 141
4 User-Agent: orion/2.3.0-next libcurl/7.29.0
5 Ngsiv2-Attrsformat: normalized
6 Host: 192.168.1.137:1028
7 Accept: application/json
8 Content-Type: application/json; charset=utf-8
9 Fiware-Correlator: 71536e6e-21bf-11ea-84d9-0242ac120003
```

```
1 {
2   "data": [
3     {
4       "id": "Room1",
5       "temperature": {
6         "metadata": {},
7         "type": "Float",
8         "value": 28.5
9       },
10      "type": "Room"
11    }
12  ],
13  "subscriptionId": "5dfa67d78bf86a9f1f4441bd"
14 }
15
```

## Ottenimento Informazioni sulle Sottoscrizioni

È possibile ottenere informazioni su tutte le sottoscrizioni tramite una richiesta `GET /v2/subscriptions`. La risposta (o meglio, il suo body) che si ottiene è di questo tipo:

```
1 [
2   {
```

```

3      "id": "5dfa67d78bf86a9f1f4441bd",
4      "description": "A subscription to get info about Room1",
5      "expires": "2040-01-01T14:00:00.00Z",
6      "status": "active",
7      "subject": {
8          "entities": [
9              {
10                 "id": "Room1",
11                 "type": "Room"
12             }
13         ],
14         "condition": {
15             "attrs": [
16                 "pressure"
17             ]
18         }
19     },
20     "notification": {
21         "timesSent": 1,
22         "lastNotification": "2019-12-18T17:54:31.00Z",
23         "attrs": [
24             "temperature"
25         ],
26         "onlyChangedAttrs": false,
27         "attrsFormat": "normalized",
28         "http": {
29             "url": "http://192.168.1.137:1028/accumulate"
30         },
31         "lastSuccess": "2019-12-18T17:54:31.00Z",
32         "lastSuccessCode": 200
33     },
34     "throttling": 5
35 }
36 ]

```

## Cancellazione e Modifica delle Sottoscrizioni

Le sottoscrizioni possono essere cancellate con una richiesta del tipo `DELETE`

`/v2/subscriptions/{subId}`

Le modifiche alle sottoscrizioni avvengono tramite delle richieste `PATCH`, in maniera analoga alle modifiche fatte alle entità.

## Operazioni Batch

La specifica *NGS/v2* include delle operazioni "batch", tra le quali abbiamo un *batch update* (`POST /v2/op/update`) e una *batch query* (`POST /v2/op/query`).

Il **batch update** consente di creare o aggiornare più entità con una richiesta singola. Per esempio, la richiesta

```

1 POST localhost:1026/v2/op/update .
2 Content-Type: application/json

```

```

1 {
2     "actionType": "append",

```



```

3      "entities": [
4          {
5              "type": "Room",
6              "id": "Room3",
7              "temperature": {
8                  "value": 21.2,
9                  "type": "Float"
10             },
11             "pressure": {
12                 "value": 722,
13                 "type": "Integer"
14             }
15         },
16         {
17             "type": "Room",
18             "id": "Room4",
19             "temperature": {
20                 "value": 31.8,
21                 "type": "Float"
22             },
23             "pressure": {
24                 "value": 712,
25                 "type": "Integer"
26             }
27         }
28     ]
29 }

```

crea due nuove entità, `Room3` e `Room4`. Se imposta il parametro `"actionType": "update"` si esegue un aggiornamento delle entità.

La **batch query** è simile alla `GET /v2/entities`, ma può esprimere delle query che quest'ultima non supporta. Per esempio, la richiesta:

```

1 POST localhost:1026/v2/op/query .
2 Content-Type: application/json

```

```

1 {
2     "entities": [
3         {
4             "idPattern": ".*",
5             "type": "Room"
6         }
7     ],
8     "attrs": [
9         "temperature",
10        "pressure"
11    ],
12    "expression": {
13        "q": "temperature>27"
14    }
15 }

```

restituisce tutte le entità la cui temperatura ha un valore maggiore di 27°C. Sebbene una query simile sia fattibile anche con `GET /v2/entities`, la batch query è uno strumento molto più flessibile. Ad esempio, è possibile effettuare una richiesta del tipo

```
1 POST localhost:1026/v2/op/query .
2 Content-Type: application/json
```

```
1 {
2   "entities": [
3     {
4       "idPattern": ".*",
5       "type": "Room"
6     },
7     {
8       "id": ".*",
9       "type": "Car"
10    }
11  ],
12  "attrs": [
13    "temperature",
14    "pressure"
15  ],
16  "expression": {
17    "q": "temperature>40",
18    "georel": "near;maxDistance:20000",
19    "geometry": "point",
20    "coords": "40,31,-3.75"
21  }
22 }
```

che restituisce la temperatura e la pressione delle entità di tipo `Room` o `Car` la cui temperatura è maggiore di 40°C e che si trovano entro 20 km dalle coordinate `40.31, -3.75`. Una query simile non è fattibile con una `GET /v2/entities`

## Context availability management

Il Context availability management si occupa delle *sorgenti delle entità e degli attributi*. Il concetto di base è la **registrazione**, che contiene informazioni su una sorgente di informazioni (i.e. un context provider) e quali entità/attributi sono forniti da quella sorgente.

- Esempio: La richiesta

```
1 POST localhost:1026/v2/registrations .
2 Content-Type: application/json
```

```
1 {
2   "description": "Registration for Room5",
3   "dataProvided": {
4     "entities": [
5       {
6         "id": "Room5",
7         "type": "Room"
8       }
9     ],
10    "attrs": [
11      "temperature",
12      "pressure"
13    ]
14  },
```

```

15     "provider": {
16         "http": {
17             "url": "http://mysensors.com/Rooms"
18         }
19     }
20 }

```

afferma che gli attributi `temperature` e `pressure` di `Room5` sono forniti da un context provider raggiungibile all'URL <http://mysensors.com/Rooms>.

## Multi Tenancy

L'Orion Context Broker implementa un semplice modello multitenant/multiservice basato sulla separazione logica del database, in modo da facilitare le politiche di autorizzazione service-based/tenant-based fornite da altri componenti FIWARE o da applicazioni software di terze parti. Il Multitenant/multiservice assicura che le entità, gli attributi e le sottoscrizioni ad un servizio/tenant siano "invisibili" agli altri servizi/tenants; per esempio, le richieste sullo spazio `tenantA` non restituiranno mai informazioni provenienti dallo spazio `tenantB`.

Questa funzionalità viene attivata tramite l'opzione `-multiservice`; quando questa opzione viene utilizzata, Orion utilizza l'header HTTP `Fiware-Service` nelle richieste per identificare il servizio/tenant. Se questo header non è presente, viene utilizzato il servizio/tenant di default.

- Esempio: nella richiesta

```

1  POST http://127.0.0.1:9977/notify .
2  Content-Length: 725
3  User-Agent: orion/0.13.0
4  Host: 127.0.0.1:9977
5  Accept: application/json
6  Fiware-Service: t_02
7  Content-Type: application/json
8
9  {
10 ...
11 }
12

```

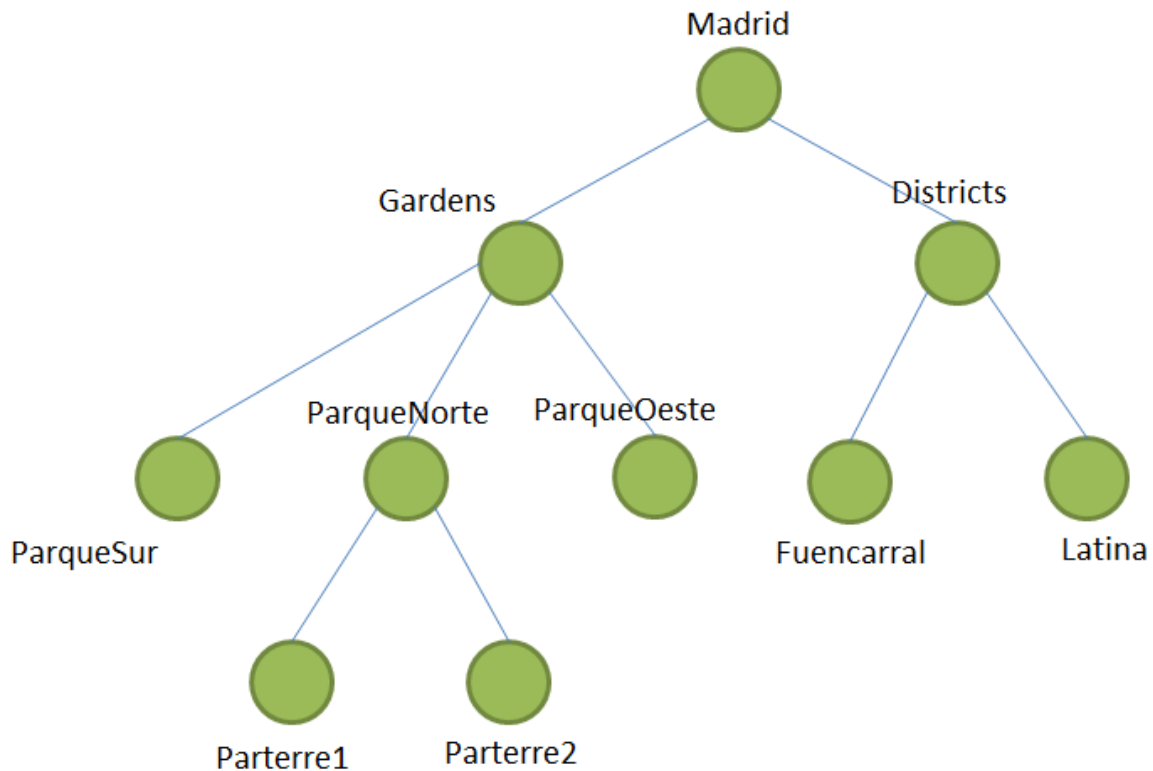
`t_02` è il nome del servizio/tenant. Se questa è la prima richiesta che coinvolge questo service, esso viene creato "al volo".

Per quanto riguarda la sintassi del nome del servizio/tenant, esso deve essere una stringa di caratteri alfanumerici (più il simbolo "\_", non sono quindi consentiti numeri e caratteri speciali ) di lunghezza minore o uguale a 50 caratteri. È importante ricordare che l'Orion Context Broker interpreta il nome in minuscolo, quindi è obbligatorio utilizzare nomi che non contengano lettere maiuscole.

## Service Path

### Entity Service Path

L'Orion Context Broker supporta scope gerarchici, quindi le entità possono essere assegnate ad uno scope al momento della creazione. Le query e le sottoscrizioni possono essere utilizzate per localizzare le entità negli scope corrispondenti.



L'utilizzo dello scope viene specificato utilizzando l'header `Fiware-ServicePath` nelle richieste di update/query. Per esempio, con questo header

```
1 | Fiware-ServicePath: /Madrid/Gardens/ParqueNorte/Parterre1
```

è possibile creare e/o modificare entità nello scope o interrogare lo scope per ottenere informazioni circa le entità che contiene.

Gli scope sono gerarchici e può essere fatta una ricerca gerarchica tramite il carattere "#". Per esempio, al percorso `/Madrid/Gardens/ParqueNorte/#` corrisponde il nodo `ParqueNorte` e i nodi figli (i.e. `Parterre1` e `Parterre2`).

È possibile fare delle query su scope disgiunti utilizzando una comma-separated list come valore dell'header `Fiware-ServicePath`. Per esempio, a questo header

```
1 | Fiware-ServicePath: /Madrid/Gardens/ParqueNorte, /Madrid/Gardens/ParqueOeste
```

corrispondono i path `/Madrid/Gardens/ParqueNorte` e `/Madrid/Gardens/ParqueOeste`.

Ci sono alcune cose da tenere a mente:

- **Limitazioni:**

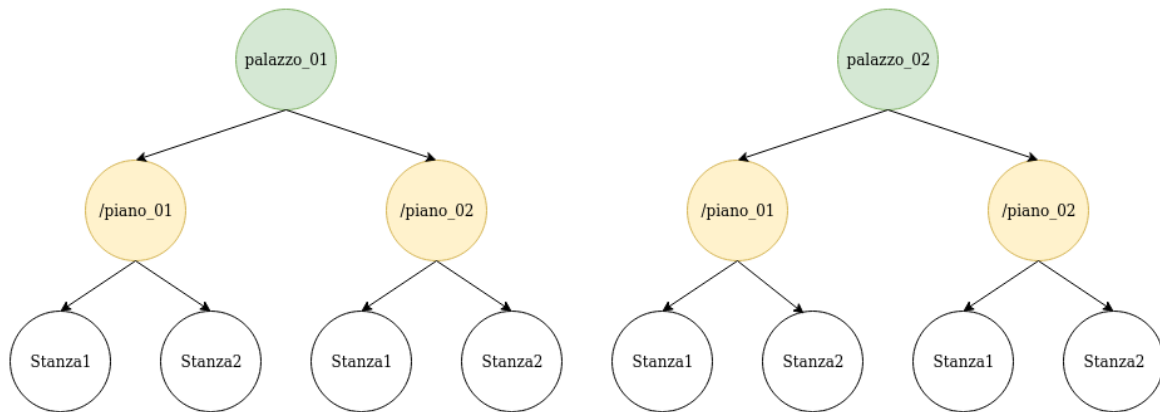
- Lo scope deve cominciare con "/", quindi possono essere utilizzati soltanto scope assoluti.
- Sono consentiti al massimo 10 livelli in un path.
- Per ogni livello, sono consentiti al massimo 50 caratteri.
- (valido per le get) La lista dei path disgiunti può contenere al massimo 10 elementi.
- Gli slash finali vengono scartati.

- `Fiware-ServicePath` è un header opzionale. Si assume che le entità create senza esso appartengano allo scope radice `/`. Tutte le query che non lo contengono si riferiscono implicitamente a `/#`. In questo modo si ha la retro-compatibilità con versioni di Orion che non supportano Service Path.
- È possibile avere un'entità con lo stesso ID e lo stesso tipo in scope differenti.
- Le entità appartengono ad un solo scope.
- `Fiware-ServicePath` viene inserito nelle richieste di notifica fatte da Orion.
- Il meccanismo di scope può essere combinato con le funzionalità multi-service/multi-tenant. In quel caso, ogni "scope tree" si trova su un servizio/tenant diverso con un isolamento database-based.

## Caso di Test

In modo da testare le funzionalità di Multi-Service e Service Path, si propone un caso di test in cui sono presenti due service/tenant che rappresentano due edifici e due service path che rappresentano i piani del singolo edificio. Ad ogni piano corrispondono due entità, che rappresentano due stanze, che a loro volta ospitano un sensore per la temperatura.

La struttura è quindi la seguente:



## Creazione delle Entità

La creazione delle entità avviene in modo analogo a quello documentato nel caso di esempio. Viene usata l'header `Fiware-Service` per definire il servizio/tenant e l'header `Fiware-ServicePath` per definire il service path. Vengono mostrate come esempio le creazioni di qualche entità

- `palazzo_01/piano_01/Stanza1`

```

1 POST localhost:1026/v2/entities .
2 Content-Type: application/json
3 Fiware-Service: palazzo_01
4 Fiware-ServicePath: /piano_01
  
```

```
1 {
2   "id": "Stanza1",
3   "type": "Room",
4   "temperature": {
5     "value": 20.1,
6     "type": "Float"
7   }
8 }
```

- palazzo\_01/piano\_01/Stanza2

```
1 POST localhost:1026/v2/entities .
2 Content-Type: application/json
3 Fiware-Service: palazzo_01
4 Fiware-ServicePath: /piano_01
```

```
1 {
2   "id": "Stanza2",
3   "type": "Room",
4   "temperature": {
5     "value": 20.2,
6     "type": "Float"
7   }
8 }
```

- palazzo\_01/piano\_02/Stanza1

```
1 POST localhost:1026/v2/entities .
2 Content-Type: application/json
3 Fiware-Service: palazzo_01
4 Fiware-ServicePath: /piano_02
```

```
1 {
2   "id": "Stanza1",
3   "type": "Room",
4   "temperature": {
5     "value": 20.3,
6     "type": "Float"
7   }
8 }
```

- palazzo\_02/piano\_01/Stanza1

```
1 POST localhost:1026/v2/entities .
2 Content-Type: application/json
3 Fiware-Service: palazzo_02
4 Fiware-ServicePath: /piano_01
```

```

1 {
2   "id": "Stanza1",
3   "type": "Room",
4   "temperature": {
5     "value": 20.5,
6     "type": "Float"
7   }
8 }

```

## Sottoscrizione a palazzo\_01/piano\_01/Stanza1

Per testare l'effettiva separazione fornita dall'utilizzo del multiservice e del service path, viene effettuata la sottoscrizione alle informazioni fornite da palazzo\_01/piano\_01/Stanza1. La separazione garantisce che verranno notificate le variazioni che avvengono solo in palazzo\_01/piano\_01/Stanza1, e non quelle che avvengono in altre entità, sebbene esse abbiano alcune "cose" in comune, (e.g. modifiche a palazzo\_02/piano\_01/Stanza1 o a building\_01/floor\_02/Room1 non vengono notificate).

Per mostrare le notifiche inviate dal broker si utilizza un echo-server eseguito localmente e in ascolto sulla porta 1028.

- Richiesta

```

1 POST localhost:1026/v2/subscriptions .
2 Content-Type: application/json
3 Fiware-ServicePath: /piano_01
4 Fiware-Service: palazzo_01

```

```

1 {
2   "description": "A subscription to get info about Stanza1",
3   "subject": {
4     "entities": [
5       {
6         "id": "Stanza1",
7         "type": "Room"
8       }
9     ],
10    "condition": {
11      "attrs": [
12        "temperature"
13      ]
14    },
15  },
16  "notification": {
17    "http": {
18      "url": "http://10.131.1.65:1028/accumulate"
19    },
20    "attrs": [
21      "temperature"
22    ]
23  },
24  "expires": "2040-01-01T14:00:00.00Z",
25  "throttling": 5
26 }

```

- Notifiche al server
  - Creazione Sottoscrizione

```
1 POST http://10.131.1.65:1028/accumulate .
2 Fiware-Servicepath: /piano_01
3 Content-Length: 143
4 User-Agent: orion/2.3.0-next libcurl/7.29.0
5 Ngsiv2-Attrsformat: normalized
6 Host: 10.131.1.65:1028
7 Accept: application/json
8 Fiware-Service: palazzo_01
9 Content-Type: application/json; charset=utf-8
10 Fiware-Correlator: 1200dfc4-391f-11ea-ae5b-0242ac120003
```

```
1 {
2   "data": [
3     {
4       "id": "Stanza1",
5       "temperature": {
6         "metadata": {},
7         "type": "Float",
8         "value": 20.1
9       },
10      "type": "Room"
11    }
12  ],
13  "subscriptionId": "5e219eb3507881eafda41175"
14 }
```

- Modifica alle informazioni (i.e. temperatura)

```
1 POST http://10.131.1.65:1028/accumulate .
2 Fiware-Servicepath: /piano_01
3 Content-Length: 145
4 User-Agent: orion/2.3.0-next libcurl/7.29.0
5 Ngsiv2-Attrsformat: normalized
6 Host: 10.131.1.65:1028
7 Accept: application/json
8 Fiware-Service: palazzo_01
9 Content-Type: application/json; charset=utf-8
10 Fiware-Correlator: 6d3ff00a-391f-11ea-ae5b-0242ac120003
```

```
1 {
2   "data": [
3     {
4       "id": "Stanza1",
5       "temperature": {
6         "metadata": {},
7         "type": "Float",
8         "value": "21.1"
9       },
10      "type": "Room"
11    }
12  ]
13 }
```



```

12     ],
13     "subscriptionId": "5e219eb3507881eafda41175"
14 }

```

# Introduzione a IoT Directory

L'IoT Directory è una web-app per il monitoraggio e la gestione dei device IoT (sensori/attuatori) disponibili tramite i loro Context Broker (tra i quali l'Orion Context Broker). La web app permette di vedere i device IoT sia a livello di dispositivo che a livello dei singoli sensori/attuatori con il loro `valueType` e rispettiva `valueUnit`: il `valueType` è un concetto legato alla semantica del dato, mentre `valueUnit` è un concetto che descrive come il valore viene codificato e rappresentato.

In base al Context Broker utilizzato, un device viene identificato tramite un campo `Id` (come avviene con Orion CB), o tramite il nome del canale/topic in base al quale vengono pubblicate le informazioni (come in MQTT, NGSI, AMQP e altri).

L'IoT Directory permette allo sviluppatore di:

- visualizzare e gestire la lista di sensori e attuatori, in base al loro `valueType`, il loro `valueName`, il loro criterio di `healthiness` e così via, fornendo inoltre informazioni circa il loro stato, la loro posizione sulla mappa (se è presente) e descrittori legati al tipo di device e dei sensori.
  - È possibile aggiungere un nuovo sensore/attuatore ad un dispositivo
- visualizzare e gestire la lista dei device associati ad un CB e aggiornare le sue proprietà tramite un web form. Inoltre, il sistema identifica la presenza di possibili attributi che costituiscono un *device schema* dei sensori. Per ogni attributo il sistema inferisce automaticamente il suo tipo, quando i valori sono organizzati in base ad una lista di modelli di base (disponibili in CSV, JSON e XML). È inoltre possibile inserire device custom tramite un web form.
- Etichettatura semantica sugli attributi nello schema del device. È possibile utilizzare la grande varietà di concetti presenti nell'ontologia Km4city per descrivere il concetto che l'attributo vuole esprimere. In questo modo, è possibile facilitare l'ottenimento dei sensori che si basano su tali concetti. Il sistema fornisce degli strumenti per l'inferenza automatica dell'etichettatura semantica.
- vedere i CB disponibili, indipendentemente dal protocollo di comunicazione utilizzato

- attivare il monitoraggio di un CB, con aggiornamento dei device attualmente disponibili

Link alla piattaforma: <https://iotdirectory.snap4city.org/management/ssoLogin.php>

Link al codice sorgente: <https://github.com/disit/iot-directory>

## Casi D'Uso

---

Per quanto riguarda l'inserimento delle funzionalità di Multi-Service e di Service-Path fornite dall'Orion Context Broker all'interno di IoT Directory, sono stati identificati i seguenti casi d'uso da analizzare e implementare

### Registrazione/Modifica di Context Broker

---

Durante la registrazione di un nuovo CB (o durante la modifica di un CB esistente), che ricordiamo avviene tramite un web form, l'utente deve poter dichiarare che tale CB supporta la funzionalità di Multi-Service/Tenant e inserire, tramite lo stesso web form, i Service/Tenant utilizzati dal CB; i Service/Tenant sono memorizzati in un DB.

### Registrazione/Modifica di un IoT Device

---

Durante la registrazione di un nuovo IoT Device (o durante la modifica di un device esistente), che ricordiamo avviene tramite un web form, l'utente deve poter dichiarare che tale device utilizza un CB che supporta la funzionalità di Multi-Service/Tenant e inserire, tramite lo stesso web form, i Service/Tenant utilizzati dal CB (che sono memorizzati in un DB) e definire una stringa che rappresenta il ServicePath che il device andrà ad utilizzare per l'invio dei dati.

Il sistema permette all'utente di registrare un nuovo IoT Device tramite l'utilizzo di modelli "già pronti" e presenti sul sistema. L'utilizzo di un modello solleva l'utente dall'onere della compilazione integrale del form dedicato alla registrazione di un nuovo device, in quanto la selezione di un modello di device causa il riempimento di alcuni dei campi del form. L'utente può comunque inserire i valori che desidera, e può inoltre ricorrere all'utilizzo del cosiddetto modello

`custom`

### Creazione/Modifica di un Modello

---

Come accennato sopra, l'utente può, durante la creazione di un nuovo IoT Device, ricorrere all'utilizzo di modelli di device già pronti, in modo che alcuni campi del form da riempire vengano compilati dal sistema, in base alle informazioni contenute nel modello. L'utente può comunque inserire i valori che desidera, e può inoltre ricorrere all'utilizzo del cosiddetto modello `custom`.

Il sistema deve permettere la creazione e la modifica di un modello che utilizzi un CB provvisto della funzionalità di Multi Service/Tenant.

## Implementazione

---

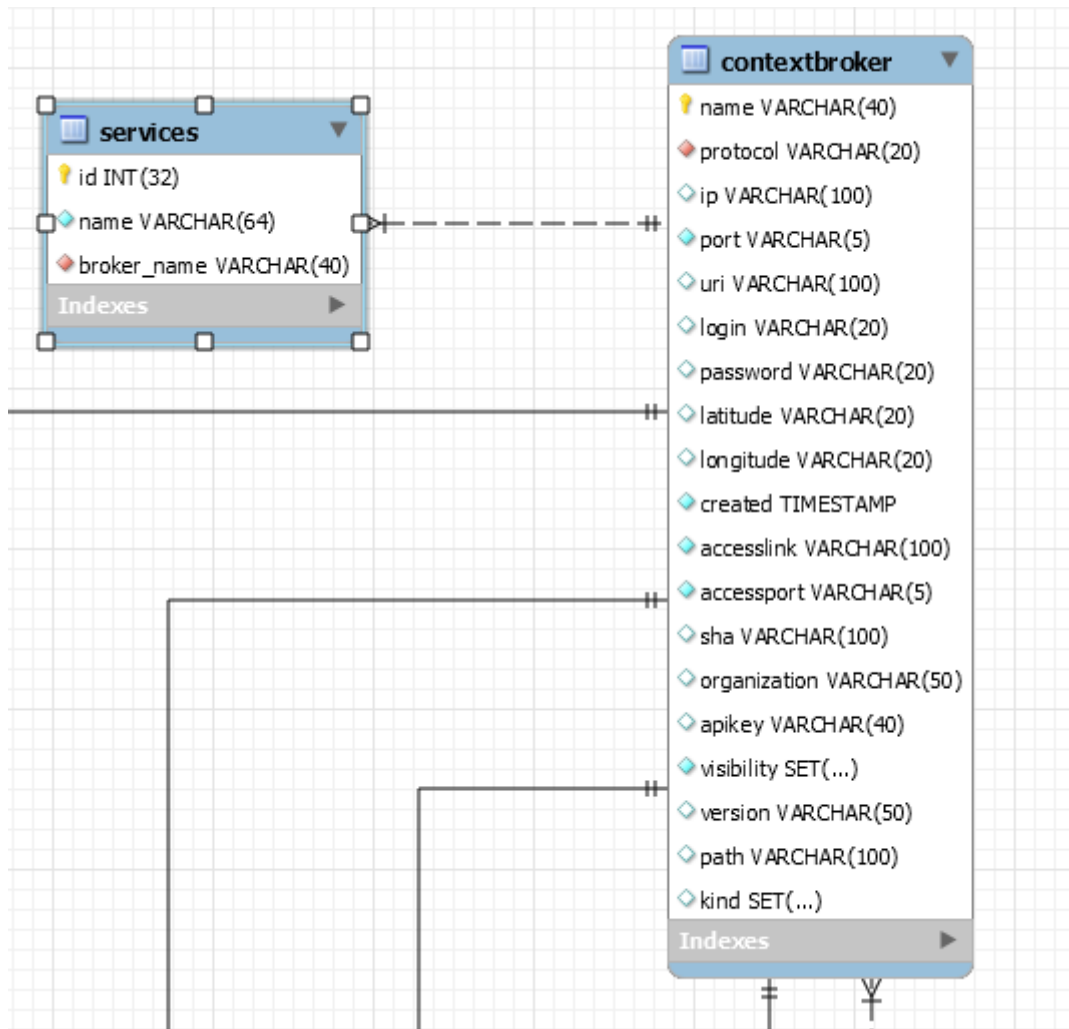
### Modifiche effettuate al DB

---

Al fine di poter implementare le funzionalità di MultiService e ServicePath all'interno del sistema Snap4City, risulta necessario attuare delle modifiche all'interno del database utilizzato per memorizzare i dati presenti all'interno del sistema.

## Modifiche a livello di Context Broker

È stata aggiunta una tabella, chiamata `services`, il cui compito è quello di memorizzare i dati relativi ai vari Servizi/Tenants e il loro CB di riferimento. La tabella `services` ha infatti una relazione multi-a-uno con la tabella `contextbroker`, già presente nel sistema e che memorizza i dati relativi ai broker. Per mettere in pratica tale relazione, la tabella `services` contiene la colonna `broker_name` che costituisce una chiave esterna, con riferimento alla colonna `name` (i.e. la chiave primaria) della tabella `contextbroker`. Questa relazione viene mostrata graficamente di seguito:



Affinchè ci sia consistenza nei valori di entrambe le tabelle, nel caso di modifica o rimozione di un CB (identificato univocamente tramite il nome), si ritiene opportuno utilizzare il vincolo `CASCADE` sia nel caso di modifica (`ON UPDATE`) che nel caso di rimozione (`ON DELETE`) di un CB. In questo caso, il campo `broker_name` verrà modificato/rimosso quando il relativo CB verrà modificato/rimosso (N.B: l'unico caso di modifica "coperto" dalla clausola è quello in cui viene cambiato il nome del CB ed eventuali altri parametri non correlati al MultiService).

L'inserimento e gli altri casi di modifica verranno gestiti tramite le transazioni (per ulteriori dettagli, si rimanda all' [Appendice A](#)).

## Modifica a livello di Modelli e Device

In questo caso, sono state aggiunte due colonne nelle tabelle relative ai device, ossia le tabelle `devices`, `deleted devices` e nella tabella `model`. Le due colonne sono state chiamate rispettivamente `service` e `servicePath` e servono per memorizzare il Service/Tenant e il ServicePath utilizzato dal modello o dal device.

## Inserimento di un Context Broker

Per quanto riguarda il caso d'uso relativo alla registrazione di un nuovo CB all'interno del sistema, è stato inserito una nuovo tab all'interno form dedicato. L'attivazione di questo tab è vincolata alla selezione, da parte dell'utente, del valore `ngsi w/MultiService` come valore del campo `protocol`; se tale valore viene selezionato, il tab risulta visibile e mostra una sezione aggiuntiva del form che altrimenti non sarebbe raggiungibile.

La sezione, denominata `Services/Tenants` contiene un primo campo di testo dedicato all'inserimento del primo Service/Tenant e un bottone che consente l'inserimento di ulteriori Service/Tenant, che possono essere rimossi cliccando il bottone `Remove` che compare alla loro destra. Se l'utente decide di inserire un CB che supporta la funzionalità di Multi-Service/Tenant, è poi tenuto ad inserire almeno un valore relativo ai Service/Tenant.

Al fine di rendere l'esperienza utente il meno frustrante possibile, è stata inserita una funzionalità che permette di "ricordare" i valori inseriti quando la sezione viene nascosta a causa della selezione di un altro protocollo (che potrebbe essere stata effettuata per errore); questi valori (al netto di eventuali stringhe vuote, che non verranno memorizzate) verranno ripristinati quando la sezione tornerà ad essere visibile.

### Add new context broker

Info

Geo-Position

Security

▼

Kind

amqp

▼

Protocol

IP

IP is mandatory

Access Link

Private

▼

Ownership

Name

Context Broker name is mandatory

Port

Port is mandatory

Version

Access Port

Cancel

Confirm

## Add new context broker

Info

Services/Tenants

Geo-Position

Security

Kind

Name

Context Broker name is mandatory

IP

IP is mandatory

Port

Port is mandatory

ngsi w/MultiService

Protocol

Version

Access Link

Access Port

Private

Ownership

Cancel

Confirm

## Add new context broker

Info

Services/Tenants

Geo-Position

Security

Check your values

- white spaces are not allowed
- use only lower case letters
- special characters are not allowed (except for "\_")
- service/tenant name must not be longer than 50 characters

topolino

Service/Tenant

Add Service/Tenant

Remove

Service/Tenant

Cancel

Confirm

## Add new context broker

Info

Services/Tenants

Geo-Position

Security

Ok

topolino

Service/Tenant

Add Service/Tenant

Remove

pippol

Service/Tenant

Cancel

Confirm

È stata inserita una funzione che controlla se l'utente ha inserito correttamente i valori relativi ai Service/Tenant (i vincoli legati ai nomi da utilizzare per i Service/Tenants vengono definiti nella sezione [Multi Tenancy](#)); in particolare, il controllo sulla sintassi del nome del singolo servizio viene effettuato, sia lato client che lato server, tramite un'espressione regolare (`/^[a-z]_[0-9]{1,50}$/`). Questa funzione, chiamata `checkcbServices`, va ad aggiungersi alle altre funzioni di controllo già presenti, le quali determinano il blocco o lo sblocco del bottone `confirm`. Di seguito viene riportato il codice della funzione:

```

1 function checkCbServices(){
2     // feedback message to the user
3     var message = null;
4     // service values
5     var values = [];
6     // check if the tab is hidden or not
7     var isHidden = $('#multiServiceTabSelector').hasClass('hidden');
8
9     // insert first row value
10    values.push(document.getElementById("inputServiceCB").value);
11    // get values of all the additional rows
12    $('#serviceTenantTabCB
div[name="additionalRow"]').find('input[name="inputServiceCB"]').each(function()
13        values.push($(this).val());
14    });
15
16    // check if the MultiService tab is hidden
17    if(isHidden){
18        addCbConditionsArray['inputServicesCB'] = true;
19        return;
20    }else{
21        var serviceRegex = /^[a-z]_|{1,50}$/;
22        for(const value of values){
23            if(!serviceRegex.test(value)){
24                message = `Check your values <br>
25                <ul>
26                    <li>white spaces are not allowed</li>
27                    <li>use only lower case letters</li>
28                    <li>special characters are not allowed (except for
29                    "_"</li>
30                    <li>service/tenant name must not be longer than 50
31                    characters</li>
32                </ul>`;
33                addCbConditionsArray['inputServicesCB'] = false;
34                $("#inputServiceCBMsg").removeClass("alert alert-info");
35                $("#inputServiceCBMsg").addClass("alert alert-danger");
36                $("#inputServiceCBMsg").html(message);
37                break;
38            }else{
39                message = 'ok';
40                addCbConditionsArray['inputServicesCB'] = true;
41                $("#inputServiceCBMsg").removeClass("alert alert-danger");
42                $("#inputServiceCBMsg").addClass("alert alert-info");
43                $("#inputServiceCBMsg").html(message);
44            }
45        }
46    }
47 }

```

Questa funzione viene “agganciata” ai seguenti eventi:

- click sul pulsante `Add new device`
- variazione del valore di una riga qualsiasi del tab `Services/Tenants`
- creazione di una riga aggiuntiva per un Service/Tenant
- rimozione di una riga aggiuntiva per un Service/Tenant
- variazione del valore del campo `Protocols`

Gli eventi di creazione e rimozione di una riga aggiuntiva per un Service/Tenant, che in sostanza consistono in una variazione dell'insieme degli elementi contenuti nel tab `Services/Tenants` (o meglio, l'insieme degli elementi figli dell'elemento `<div id="serviceTenantTabCB" class="tab-pane fade">`) vengono monitorati utilizzando un oggetto di tipo `MutationObserver`, in questo modo:

```
1 // Observe the Multi-Service/Tenant Tab for child element creation/removal
2 const targetNode = document.getElementById('serviceTenantTabCB');
3 // Options for the observer (which mutations to observe)
4 const config = {childList: true};
5 // Callback function to execute when mutations are observed
6 const callback = function() {
7     checkCbServices();
8     checkAddCbConditions();
9 };
10 // Create an observer instance linked to the callback function
11 const observer = new MutationObserver(callback);
12 // Start observing the target node for configured mutations
13 observer.observe(targetNode, config);
```

Quando viene premuto il tasto `Confirm`, il client invia i dati relativi al nuovo CB al server tramite la funzione `$.ajax` di JQuery. In questo caso, sono state aggiunte delle modifiche che consentono la gestione e l'invio dei dati relativi agli eventuali Services/Tenants inseriti dal client, che vengono salvati in un array e inviati al server in formato JSON tramite la funzione `JSON.stringify`.

Quando il server riceve la richiesta di inserimento da parte del client, avvia una transazione SQL (per approfondire: [Appendice A](#)) che consiste nell'inserimento del nuovo CB nella tabella `contextbroker` e, se il protocollo scelto è `ngsi w/MultiService`, dei vari Service/Tenant. Si ricorre all'utilizzo delle transazioni in modo che l'inserimento del CB risulti valido solo se tutte le query al DB hanno avuto un esito positivo; se una qualsiasi delle query ha avuto un problema tale da non essere stata eseguita correttamente, l'intera transazione fallisce. In questo modo si evita di avere il DB in uno stato di inconsistenza.

## Modifica di un Context Broker

### Modifiche preliminari alla gestione delle richieste di tipo `get`

Per poter implementare appieno la nuova funzionalità di Multi Service/Tenancy per la modifica di un broker, è stato necessario apportare alcune modifiche alle funzioni che gestiscono la richiesta, da parte del client, di informazioni sui CB presenti sul sistema, e le rispettive funzioni lato server che si occupano di soddisfare tali richieste.

È possibile definire tre diverse situazioni, che prendono il nome del valore del campo `action` usato durante le richieste: `get_all_contextbroker`; `get_subset_contextbroker` e `get_all_contextbroker_latlong`.

Quando viene utilizzata `get_all_contextbroker`, vengono richieste al server le informazioni circa tutti i CB presenti sul sistema; il server restituirà al client tutti i CB che è autorizzato a vedere. Lato server, durante la gestione delle informazioni dei CB che verranno inviate al client, è stata inserita una porzione di codice che si occupa di richiedere le informazioni dei Services legati al CB. In particolare, le informazioni di un CB vengono gestite tramite un array; se il CB utilizza il



MultiService, uno degli elementi dell'array sarà un ulteriore array, contenente i valori dei Services correlati al CB. Di seguito viene mostrato il codice inserito per svolgere tale compito (N.B.: la variabile `$row` indica l'array che, in un dato istante di tempo, contiene le informazioni su un singolo CB):

```
1 // Author: Antonino Mauro Liuzzo
2 if ($row["protocol"] == "ngsi w/MultiService") {
3     // the CB supports MultiServices
4     $brokerName = $row["name"];
5     $servicesQueryString = "SELECT * FROM services WHERE broker_name =
    '$brokerName'";
6
7     // query to services table
8     $sqr = mysqli_query($link, $servicesQueryString);
9
10    if ($sqr) {
11
12        dev_log($servicesQueryString . " OK");
13        $row["services"] = array();
14
15        while ($servicesRow = mysqli_fetch_assoc($sqr)) {
16            array_push($row["services"], $servicesRow["name"]);
17        }
18    } else {
19
20        dev_log($servicesQueryString . " ERROR");
21        $output= format_result($_REQUEST["draw"], 0, 0, null, 'Error: errors
    in reading data about IOT Broker. <br/>' . generateErrorMessage($link),
    '\n\r Error: errors in reading data about IOT Broker.' .
    generateErrorMessage($link), 'ko');
22
23        logAction($link,$username,'contextbroker','get_all_contextbroker','', $organi
    zation,'Error: errors in reading data about IOT Broker.','faliure');
24    }
25 }
```

Lato client, questa richiesta viene utilizzata all'interno della funzione `fetch_data`, una funzione di utilità che viene utilizzata per creare la tabella dei CB e mostrarla al client. In particolare, `get_all_contextbroker` viene utilizzata quando il client desidera ricevere le informazioni di tutti i CB a lui visibili, senza alcun tipo di condizione, cosa che viene esplicitata quando la funzione `fetch_data` viene chiamata impostando il parametro `selected` a `null`. Questa funzione viene, per esempio, chiamata in questo modo quando viene fatto il refresh della pagina e al termine delle operazioni di inserimento, modifica e rimozione di un CB.

Le richieste di tipo `get_all_contextbroker_latlog` vengono effettuate dal client nel caso in cui si voglia vedere la disposizione geografica dei CB, cosa che lato client viene fatta tramite la funzione `drawMapAll`. Tale disposizione viene mostrata all'utente tramite una mappa sulla quale vengono segnate, tramite dei puntini, le posizioni dei CB e alcuni parametri come, ad esempio, il nome. Poiché le informazioni relative al protocollo non vengono mostrate all'utente, non si ritiene necessario effettuare modifiche al codice sorgente dell'applicazione. Si ritiene inoltre che inserire sulla mappa le informazioni sul protocollo e sui vari services/tenants utilizzati dai singoli broker renderebbe scomoda la fruizione della mappa stessa, in quanto l'utente dovrebbe cercare le informazioni a lui utili in uno spazio piuttosto piccolo (inserire dimensione del frame con la mappa) che sarebbe saturo di informazioni.

Le richieste di tipo `get_subset_contextbroker` vengono effettuate dal client quando, dopo che il sistema mostra ad esso la mappa (si rimanda al paragrafo precedente), il client effettua delle operazioni di filtraggio/selezione sulla suddetta mappa come, ad esempio il tracciare su di essa un cerchio o un poligono che racchiuda un sottoinsieme dei CB mostrati sulla mappa. Quest'operazione viene fatta lato client tramite la funzione `fetch_data`; questa volta il parametro `selected` ha un valore non nullo. In questo caso è stata effettuata la stessa modifica applicata nel caso di `get_all_contextbroker`, il cui codice sorgente è riportato sopra. Si segnala, tuttavia, che se durante l'esecuzione della funzione (lato server), si entra nel seguente blocco di codice:

```
1  if (!empty($accessToken)) {  
2      getOwnershipObject($accessToken, "BrokerID", $result);  
3      getDelegatedObject($accessToken, $username, $result);  
4  }
```

la funzione non ha un “vero e proprio completamento” e, all'utente, non viene restituito alcun risultato. Si sospetta che ci sia qualche bug all'interno delle due funzioni richiamate nel blocco.

## Modifiche effettive

Al netto delle modifiche citate pocanzi, il caso d'uso relativo alla modifica di un CB esistente richiede l'esecuzione di modifiche molto simili a quelle messe in atto nel caso d'uso precedente. Anche in questa occasione è stata inserita una sezione al form di modifica del CB che risulta visibile solo se il campo `protocols` contiene il valore `ngsi w/MultiService`. Analogamente al caso di inserimento di un nuovo CB, sono state inserite le funzionalità di “ricordo e ripristino” e i controlli di validità (anche in questo caso il controllo sulla sintassi sul nome del singolo service viene fatto sia lato client che lato server).

## Edit Context Broker - iotobsf

Info

Geo-Position

Security

Internal



iotobsf

Kind

Name

iotobsf

1026

IP

Port

Ok

Ok

ngsi



Protocol

Version

iotobsf

8080

Access Link

Access Port

Ownership

2019-11-11 15:36:28

Created

null

API Key

Path

Cancel

Confirm

## Edit Context Broker - iotobsf

Info

Services/Tenants

Geo-Position

Security

Internal



iotobsf

Kind

Name

iotobsf

1026

IP

Port

Ok

Ok

ngsi w/MultiService



Protocol

Version

iotobsf

8080

Access Link

Access Port



2019-11-11 15:36:28

Ownership

Created

null

API Key

Path

Cancel

Confirm

## Edit Context Broker - iotbsf

Info

Services/Tenants

Geo-Position

Security

Ok

topolino

Service/Tenant

Add Service/Tenant

paperino

Service/Tenant

Remove

Cancel

Confirm

Anche in questo, è stata inserita una funzione che controlla se l'utente ha inserito correttamente i valori relativi ai Service/Tenant (i vincoli legati ai nomi da utilizzare per i Service/Tenants vengono definiti nella sezione [Multi Tenancy](#)). Questa funzione, chiamata `checkEditCbServices`, va ad aggiungersi alle altre funzioni di controllo già presenti, le quali determinano il blocco o lo sblocco del bottone `confirm`. Il codice sorgente è simile a quello utilizzato nel caso di creazione di un CB, pertanto non viene riportato.

Gli eventi di creazione e rimozione di una riga aggiuntiva per un Service/Tenant, che in sostanza consistono in una variazione dell'insieme degli elementi contenuti nel tab `Services/Tenants` (o meglio, l'insieme degli elementi figli dell'elemento `<div id="editServiceTenantTabCB" class="tab-pane fade">`) vengono monitorati utilizzando un oggetto di tipo `MutationObserver`, in modo analogo a quello che avviene quando viene creato un nuovo CB.

Quando viene premuto il tasto `Confirm`, il client invia i dati aggiornati relativi al CB al server tramite la funzione `$.ajax` di JQuery. In questo caso, sono state aggiunte delle modifiche che consentono la gestione e l'invio dei dati relativi agli eventuali Services/Tenants inseriti dal client, che vengono salvati in un array e inviati al server in formato JSON tramite la funzione `JSON.stringify`.

Quando il server riceve la richiesta di inserimento da parte del client, avvia una transazione SQL (per approfondire: [Appendice A](#)) che consiste nella modifica del CB nella tabella `contextbroker` e, se il protocollo scelto è `ngsi w/MultiService`, nella rimozione dei vecchi Service/Tenant e di quelli nuovi. Si ricorre all'utilizzo delle transazioni in modo che la modifica del CB risulti valido solo se tutte le query al DB hanno avuto un esito positivo; se una qualsiasi delle query ha avuto un problema tale da non essere stata eseguita correttamente, l'intera transazione fallisce. In questo modo si evita di portare il DB in uno stato di inconsistenza.

## Rimozione di un Context Broker

Per quanto riguarda la rimozione di un CB, non sono state effettuate modifiche rilevanti sul Front-End o sul Back-End. L'inserimento del vincolo `ON DELETE CASCADE`, aggiunto alla chiave esterna della tabella `services` (maggiore dettaglio in [Modifiche effettuate al DB](#)), è sufficiente affinché la rimozione di un CB dal DB causi la rimozione di tutte i Service/Tenant ad esso

correlato.

## Inserimento di un Modello

Per quanto riguarda il caso d'uso relativo alla registrazione di un nuovo modello all'interno del sistema, sono state inseriti un elemento di tipo `select` e un elemento di tipo `input` all'interno form dedicato. L'attivazione di questi campi è vincolata alla selezione, da parte dell'utente, del valore `ngsi w/MultiService` come valore del campo `protocol`; se tale valore viene selezionato, i due elementi vengono "sbloccati" e possono essere utilizzati dall'utente.

**Add New Model**

General Info | **IOT Broker** | Values

iotbsf ▼ ContextBroker

Format

Service/Tenant  
only ngsi w/MultiService supports Service/Tenant selection

coap ▼ Protocol

ServicePath  
only ngsi w/MultiService supports ServicePath

Cancel Confirm

**Add New Model**

General Info | **IOT Broker** | Values

iotbsf ▼ ContextBroker

Format

Service/Tenant  
select one Service/Tenant

ngsi w/MultiService ▼ Protocol

ServicePath  
servicePath preview: /

Cancel Confirm

Una volta che i due elementi sono stati sbloccati, l'utente può scegliere quale Service/Tenant utilizzare, tra quelli che sono supportati dal broker scelto (l'utente può scegliere l'opzione "vuota" se non vuole utilizzare un Service/Tenant), e può definire quale ServicePath verrà utilizzato dal modello (l'utente può lasciare vuoto questo campo e usare il servicePath di default, ossia `/`).

Il "blocco" e lo "sblocco" di questi due elementi viene fatto tramite una funzione di controllo sul valore del campo `protocol` e che viene richiamata quando questo campo subisce variazioni e quando viene aperto il form per l'inserimento del nuovo modello.

È stata inserita inoltre una funzione per il controllo della sintassi del campo di input relativo al ServicePath. Il valore di questo campo è soggetto ad alcuni vincoli sintattici (definiti con maggiore dettaglio in [Service Path](#)) che devono essere rispettati. Al di sotto di questo campo è presente un messaggio che fornisce un feedback circa la validità sintattica del valore inserito: se il valore non ha errori nella sintassi, il sistema mostra all'utente un'anteprima del valore che verrà

effettivamente inserito, altrimenti verrà mostrato un messaggio in cui viene specificato l'errore commesso.

### Add New Model

General Info

IOT Broker

Values

iotbsf

ContextBroker

ngsi w/MultiService

Protocol

Format

Test/ServicePath

ServicePath

servicePath preview: /Test/ServicePath

select one Service/Tenant

Cancel

Confirm

### Add New Model

General Info

IOT Broker

Values

iotbsf

ContextBroker

ngsi w/MultiService

Protocol

Format

Test/Service Path

ServicePath

you can't use whitespaces

select one Service/Tenant

Cancel

Confirm

È stata inserita inoltre una funzione adibita all'ottenimento dei Service/Tenant che l'utente può scegliere in base al broker scelto. Questa funzione invia una richiesta al server con il quale richiede tutti i Service legati ad un dato broker, che viene identificato tramite il nome. I dati che vengono forniti in risposta vengono poi utilizzati per inserire dinamicamente gli elementi `option` interni al `select`. La funzione è la seguente:

```
1  /**
2   *
3   * @param name: context broker's name
4   * @param mode: add or edit
5   */
6  function getServicesByCBName(name, mode){
7      console.log('getServicesByCBName');
8
9      // data to send to server
10     var data = {
11         action : "get_services_by_cb_name",
12         brokerName : name
13     };
14
15     // send POST request to server and manage its result
16     $.post('../api/contextbroker.php', data).done(function(data){
```

```

17
18     var servicesObj = data['content'];
19     var services = [];
20
21     for (let i = 0; i < servicesObj.length; i++){
22         services.push(servicesObj[i]['name']);
23     }
24
25     var selectService = null;
26
27     if (mode == 'add'){
28         console.log('getServicesByCBName : add case');
29         selectService = $('#selectService');
30     } else if (mode == 'edit') {
31         console.log('getServicesByCBName : edit case');
32         selectService = $('#editSelectService');
33     } else {
34         console.log('getServicesByCBName : ERROR');
35         return;
36     }
37
38     // remove "old" services
39     selectService.find('.extraService').remove();
40
41     // option creations
42     for(let i = 0; i < services.length; i++){
43         var option = document.createElement('option');
44         $(option).attr('class', 'extraService');
45         $(option).attr('value', services[i]);
46         $(option).html(services[i]);
47
48         selectService.append(option);
49     }
50     return;
51 }).fail(function(){
52     alert("Something wrong during getting services");
53 });
54 }

```

N.B.: le funzioni scritte in questa fase sono state progettate in modo da essere utilizzabili sia a livello di gestione dei modelli, sia a livello di gestione dei device. Per questo motivo queste funzioni si trovano in un file comune, chiamato `commonModelDevice.js`, e utilizzano dei parametri che consentono alle funzioni di “capire” il contesto in cui sono state chiamate.

Quando l'utente preme sul tasto `Confirm`, viene fatto una sorta di pre-processing del valore del `servicePath`; se necessario, infatti, viene inserito uno `/` all'inizio della stringa (si ricorda che si possono utilizzare soltanto dei path assoluti, che quindi iniziano con `/`) e viene rimosso lo `/` finale, nel caso sia stato aggiunto per sbaglio.

Lato server, si verifica il valore del protocollo utilizzato: se il protocollo utilizzato è `ngsi w/MultiService`, il server effettua una query di inserimento che include anche i dati relativi al service e al servicePath (che possono avere i valori di default `""` e `/`, rispettivamente), dopo aver effettuato lo stesso controllo sintattico che viene fatto lato client e verificato che la sintassi del valore del servicePath sia corretta; altrimenti viene utilizzata la query che era già presente nel sistema prima di queste modifiche.



```

1 // Author: Antonino Mauro Liuzzo
2 $service = mysqli_real_escape_string($link, $_REQUEST['service']);
3 $servicePath = mysqli_real_escape_string($link, $_REQUEST['servicePath']);
4
5 // perform different queries based on protocol value
6 if ($protocol == 'ngsi w/MultiService'){
7
8     $syntaxRes = servicePathSyntaxCheck($servicePath);
9
10    if($syntaxRes == 0){
11        // valid servicePath value
12        $q = "INSERT INTO model(name, description, devicetype, kind,
13            producer, frequency, contextbroker, protocol, format, healthiness_criteria,
14            healthiness_value, kgenerator, attributes, edgeway_type, organization,
15            visibility, service, servicePath ) " .
16            "VALUES('$name', '$description', '$type', '$kind', '$producer',
17            '$frequency', '$contextbroker', '$protocol', '$format', '$hc', '$hv',
18            '$kgenerator', '$listAttributes', '$edgeway_type', '$organization',
19            'private', '$service', '$servicePath')";
20    } else {
21        // invalid servicePath value
22        $result["status"]='ko';
23        $result["error_msg"] = $servicePath . " is NOT a valid servicePath";
24    }
25 } else {
26     $q = "INSERT INTO model(name, description, devicetype, kind, producer,
27         frequency, contextbroker, protocol, format, healthiness_criteria,
28         healthiness_value, kgenerator, attributes, edgeway_type, organization,
29         visibility ) " .
30         "VALUES('$name', '$description', '$type', '$kind', '$producer',
31         '$frequency', '$contextbroker', '$protocol', '$format', '$hc', '$hv',
32         '$kgenerator', '$listAttributes', '$edgeway_type', '$organization',
33         'private')";
34 }

```

## Modifica di un Modello

Lato client, le modifiche effettuate sono del tutto simili a quelle effettuate nel caso d'uso precedente. Anche in questo caso, sono stati inseriti due elementi all'interno del form apposito, in modo che l'utente possa definire il Service e il ServicePath che il modello andrà ad utilizzare, e con essi tutte le funzioni di controllo e pre-processing.

Merita una particolare menzione l'inserimento di una funzione, chiamata `fillMultiTenancyFormSection`, che si occupa di compilare in automatico i campi relativi al service e al servicePath del modello da modificare. Questa funzione effettua una richiesta `POST` al server per ottenere i valori dei Service/Tenant che appartengono al broker usato dal modello, per poi impostare il valore attualmente utilizzato, prelevato dagli attributi del bottone Edit e passato come parametro, insieme al valore assunto dal servicePath. Questi campi vengono popolati solo se il protocollo usato è `ngsi w/MultiService`.

Lato server, le modifiche effettuare sono simili a quelle effettuate per il caso d'uso relativo all'inserimento di un nuovo modello. In questo caso, oltre alle modifiche già documentate in [Inserimento di un Modello](#), è stata modificata anche il testo della query di "default", in modo che essa imposti come `NULL` il valore di service e servicePath: in questo modo viene coperta la

situazione dove, modificando un modello, viene rimosso l'utilizzo del multitenancy.

## Inserimento di un Device

Per quanto riguarda il caso d'uso relativo alla registrazione di un nuovo device all'interno del sistema, sono state inseriti un elemento di tipo `select` e un elemento di tipo `input` all'interno form dedicato. L'attivazione di questi campi è vincolata alla selezione, da parte dell'utente, del valore `ngsi w/MultiService` come valore del campo `protocol`; se tale valore viene selezionato, i due elementi vengono "sbloccati" e possono essere utilizzati dall'utente.

### Add new device

IOT Broker	Info	Position	Values
iotbsf		sensor	
ContextBroker		Kind	
Ok		Ok	
ngsi		json	
Protocol		Format	
Ok		Ok	
Service/Tenant		ServicePath	
only ngsi w/MultiService supports Service/Tenant selection		only ngsi w/MultiService supports ServicePath	

CancelConfirm

### Add new device

IOT Broker	Info	Position	Values
testName		sensor	
ContextBroker		Kind	
Ok		Ok	
ngsi w/MultiService		json	
Protocol		Format	
Ok		Ok	
paperino		/Test/Uno/Due	
Service/Tenant		ServicePath	
select one Service/Tenant		servicePath preview: /Test/Uno/Due	

CancelConfirm

Una volta che i due elementi sono stati sbloccati, l'utente può scegliere quale Service/Tenant utilizzare, tra quelli che sono supportati dal broker scelto (l'utente può scegliere l'opzione "vuota" se non vuole utilizzare un Service/Tenant), e può definire quale ServicePath verrà utilizzato dal modello (l'utente può lasciare vuoto questo campo e usare il servicePath di default, ossia `/`).

Il "blocco" e lo "sblocco" di questi due elementi viene fatto tramite una funzione di controllo sul valore del campo `protocol` e che viene richiamata quando questo campo subisce variazioni, quando viene aperto il form per l'inserimento del nuovo device e quando il campo relativo al modello di riferimento subisce delle variazioni.

È stata inserita inoltre una funzione per il controllo della sintassi del campo di input relativo al ServicePath. Il valore di questo campo è soggetto ad alcuni vincoli sintattici (definiti con maggiore dettaglio in [Service Path](#)) che devono essere rispettati. Al di sotto di questo campo è presente un messaggio che fornisce un feedback circa la validità sintattica del valore inserito: se il valore non ha errori nella sintassi, il sistema mostra all'utente un'anteprima del valore che verrà effettivamente inserito, altrimenti verrà mostrato un messaggio in cui viene specificato l'errore commesso.

### Add new device

IOT Broker	Info	Position	Values
testName	▼	sensor	▼
ContextBroker		Kind	
Ok		Ok	
ngsi w/MultiService	▼	json	▼
Protocol		Format	
Ok		Ok	
paperino	▼	/Test/Uno/Due	
Service/Tenant		ServicePath	
select one Service/Tenant		servicePath preview: /Test/Uno/Due	
<div>CancelConfirm</div>			

### Add new device

IOT Broker	Info	Position	Values
testName	▼	sensor	▼
ContextBroker		Kind	
Ok		Ok	
ngsi w/MultiService	▼	json	▼
Protocol		Format	
Ok		Ok	
paperino	▼	/Test/U no/Due	
Service/Tenant		ServicePath	
select one Service/Tenant		you can't use whitespaces	
<div>CancelConfirm</div>			

La funzione `getServicesByCBName`, il cui codice sorgente è riportato nella sezione [Inserimento di un Modello](#), viene chiamata non solo quando il campo relativo al broker subisce delle variazioni, ma anche quando il campo relativo al modello ne subisce.

Quando l'utente preme sul tasto `Confirm`, viene fatto un pre-processing del valore del `servicePath`; se necessario, infatti, viene inserito uno `/` all'inizio della stringa (si ricorda che si possono utilizzare soltanto dei path assoluti, che quindi iniziano con `/`) e viene rimosso lo `/` finale, nel caso sia stato aggiunto per sbaglio.

Lato server, le modifiche necessarie sono state più numerose, se paragonate a quelle effettuate nei casi d'uso di inserimento di un CB o di un modello; questa situazione è dovuta al fatto che l'inserimento di un device nella piattaforma coinvolge più funzioni, alcune dedicate all'inserimento del device nel db, altre all'inserimento del device all'interno del broker utilizzato (che in questa fase di sviluppo è in esecuzione all'interno di una macchina virtuale che si trova nel

dispositivo del sottoscritto). Queste funzioni sono definite all'interno del file `api/device.php` e `api/common.php`.

All'interno di `api/device.php` sono state effettuate delle modifiche che consentono al server di prendere anche i dati relativi al service e al servicePath, che vengono mandati dal client. Il codice all'interno di questo file rimanda l'inserimento del device alla funzione `insert_device` (e a quelle richiamate da quest'ultima), che si trova all'interno di `api/common.php`. Queste funzioni sono state modificate in modo da poter gestire i due nuovi parametri.

La modifica più importante fatta all'interno della funzione `insert_device` è relativa alla "scelta" del testo della query da effettuare per inserire il device all'interno del database. Questa modifica è molto simile a quella effettuata durante l'inserimento di un modello. Viene controllato il valore del protocollo: se esso corrisponde a `ngsi w/MultiService`, il testo della query che verrà fatta conterrà anche i valori di `service` e `servicePath`, altrimenti questi valori non verranno considerati. Inoltre, se si utilizza il multiService, il campo relativo al `servicePath` subirà un controllo sintattico identico a quello effettuato durante l'inserimento di un modello (viene usata la stessa funzione).

È stata modificata anche la funzione `registerKB`, una funzione utilizzata per inserire il device all'interno di una **Knowledge Base** e, indirettamente, per registrare il device all'interno del broker, in quanto richiama una delle funzioni dedicate a tale scopo. All'interno di questa funzione è presente un workaround che rende possibile registrare nella KB un device il cui broker è di tipo `ngsi w/MultiService`. Per fare in modo che non ci siano errori durante l'inserimento di un device del genere, il tipo del broker viene modificato in `ngsi`, se necessario.

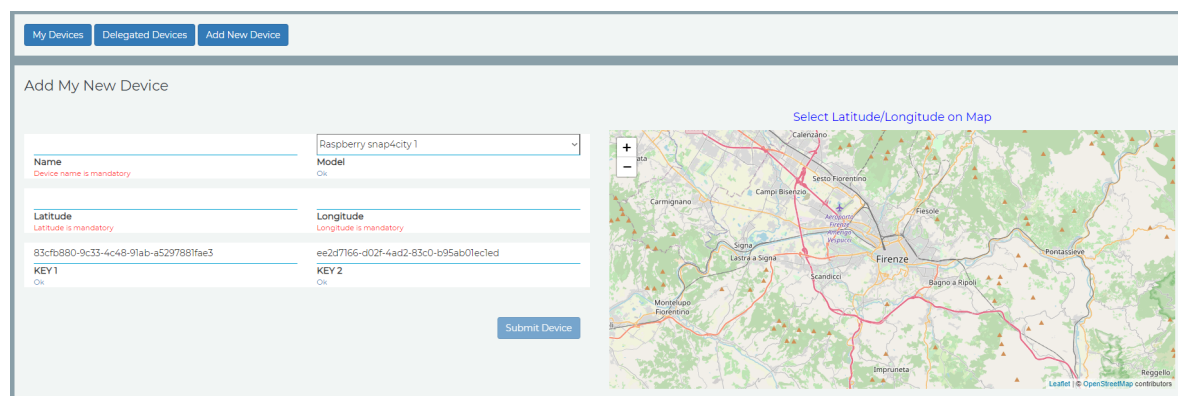
```
1 // this workaround MUST be fixed -> ngsi w/MultiService, for now, is not a
  valid broker type
2 $msg["broker"]["type"]=$row["protocol"];
3 if ($rowCB["protocol"] == "ngsi w/MultiService") $msg["broker"]["type"] =
  "ngsi";
```

È stata inserita, infine, una funzione, chiamata `insert_ngsi_multiservice`. Il cui codice è molto simile alla funzione `insert_ngsi` già presente. Questa funzione si occupa di registrare il device all'interno dell'apposito broker. La differenza maggiore sta nell'utilizzo dei parametri `service` e `servicePath`, che vengono utilizzati per costruire le righe `Fiware-Service` e `Fiware-ServicePath`, ossia le due righe da aggiungere all'header delle richieste HTTP fatte al broker per utilizzare le funzionalità di Multi Tenancy e Service Path. Di seguito viene riportata la costruzione della richiesta HTTP fatta al broker, tramite `CURL`.

```
1 // Setup CURL
2 $ch = curl_init($url_orion);
3 $authToken = 'OAuth 2.0 token here';
4 curl_setopt_array($ch, array(
5     CURLOPT_POST => TRUE,
6     CURLOPT_RETURNTRANSFER => TRUE,
7     CURLOPT_HTTPHEADER => array(
8         'Authorization: '.$authToken,
9         'Content-Type: application/json',
10        'Fiware-Service: ' . $service,
11        'Fiware-ServicePath: ' . $servicePath),
12     CURLOPT_POSTFIELDS => json_encode($msg_orion)
13 ));
```

# Inserimento di un Device attraverso schermata guidata

Oltre alla modalità di inserimento descritta in [Inserimento di un Device](#), è possibile inserire un device utilizzando una schermata “più guidata”, raggiungibile nella sezione `My IOT Devices`. Questa schermata consente un inserimento rapido e semplice, ed il suo utilizzo è consigliato agli utenti più inesperti. Tuttavia, questa schermata da meno possibilità rispetto a quella presente all'interno della sezione `IOT Devices`, in quanto l'utente può solo inserire il nome e le coordinate del device, mentre le altre caratteristiche (comprese quelle relative al multitenancy) vengono ereditate dal modello utilizzato.



In questo caso, le uniche modifiche apportate riguardano l'invio al server delle informazioni sul service e sul servicePath ereditate, come citate poc'anzi, dal modello utilizzato (se il modello utilizza tali funzionalità). Non è stato necessario effettuare ulteriori modifiche, in quanto vengono utilizzate, sia lato client che lato server, le stesse funzioni utilizzate in [Inserimento di un Device](#).

## Modifica di un Device

La sezione relativa alla modifica di un device non ha subito le stesse modifiche effettuate per i gli altri casi di modifica (i.e. Modifica di un Broker e Modifica di un Modello); infatti i parametri di un device relativi al multitenancy (i.e. service e servicePath) non sono modificabili dall'utente, tanto che gli elementi di interfaccia associati a questi parametri vengono impostati in sola lettura. Questa scelta è dettata dal fatto che non è possibile (questa conclusione viene tratta dopo la consultazione della documentazione di Orion) modificare il service e il servicePath di un device registrato all'interno di un context broker, in quanto le informazioni relative a queste funzionalità sono contenute all'interno delle intestazioni HTTP e vengono utilizzate per identificare il device da modificare.

Tuttavia, sono state fatte delle piccole modifiche alla sezione del server che si occupa della modifica di un device, in modo che possa utilizzare le informazioni relative al multitenancy (se presenti) per identificare un device che utilizza tale funzionalità e modificarlo. In particolare, sono state modificate le funzioni `update_ngsi` e `updateKB` contenute nel file `api/common.php`, in modo che utilizzino i due nuovi parametri. All'interno della funzione `updateKB` è presente lo stesso workaround sul tipo utilizzato in [Inserimento di un Device](#), e che andrebbe rimosso.

## Rimozione di un Device

Per quanto riguarda il caso d'uso relativo alla rimozione di un device, non sono state effettuate modifiche in termini di GUI nelle due pagine web in cui tale operazione è supportata (i.e. `management/device.php` e `management/allddevices.php`). Le modifiche fatte lato client sono relative all'invio al server delle informazioni relative al service e al servicePath. Queste

informazioni verranno poi utilizzate dal server per identificare, sul context broker, il device da eliminare.

```
1 var service = $("#deleteDeviceModal span").attr('data-service');
2 var servicePath = $("#deleteDeviceModal span").attr('data-servicepath');
3 if(service === "null") service = "";
4 if(servicePath === "null") servicePath = "";
5
6 // codice non modificato e non riportato...
7
8 $.ajax({
9     url: "../api/device.php",
10    data:{
11        action: "delete",
12        username: loggedUser,
13        organization : organization,
14        dev_organization : dev_organization,
15        id: id,
16        uri : uri,
17        contextbroker : contextbroker,
18        token : sessionToken,
19        // Author: Antonino Mauro Liuzzo
20        service: service,
21        servicePath: servicePath
22    },
23    type: "POST",
24    // codice non modificato e non riportato...
```

Lato server, sono state modificate le funzioni che si occupano della rimozione del device; in particolare, sono state modificate le funzioni `delete_ngsi` e `deleteKB` contenute nel file `api/common.php`, in modo che utilizzino le informazioni relative al service e al servicePath (se presenti) per identificare, all'interno del context broker, il device da eliminare. All'interno della funzione `deleteKB` è presente lo stesso workaround sul tipo utilizzato in [Inserimento di un Device](#), e che andrebbe rimosso.

```
1 function deleteKB($link, $name, $contextbroker, $kburl="", &$result,
2 $service="", $servicePath="") {
3     $query = "SELECT d.organization, d.uri, d.id, d.devicetype AS
4     entityType, d.kind, d.format
5     d.macaddress, d.model, d.producer, d.protocol, d.longitude,
6     d.latitude, d.visibility,
7     d.frequency, d.service, d.servicePath, cb.name, cb.protocol as type,
8     cb.ip, cb.port,
9     cb.login, cb.password, cb.latitude as cblatitude, cb.longitude as
10    cblongitude, cb.created,
11    cb.kind as cbkind
12    FROM devices d JOIN contextbroker cb ON d.contextBroker = cb.name
13    WHERE d.deleted is null and d.contextBroker='$contextbroker' and
14    d.id='$name';";
15
16    // codice non modificato e non riportato...
17
18    $msg["broker"]["type"]=$row["protocol"];
19    if ($rowCB["protocol"] == "ngsi w/MultiService") $msg["broker"]["type"]
20    = "ngsi";
21
22    // codice non modificato e non riportato...
```

```

15 // codice non modificato e non riportato...
16
17 switch ($protocol){
18     case "ngsi":
19     case "ngsi w/MultiService":
20         $res = delete_ngsi($name, $type, $contextbroker, $kind, $protocol,
21             $format, $model,
22             $latitude, $longitude, $visibility, $frequency,
23             $listnewAttributes, $ip,
24             $port, $uri, $service, $servicePath, $result);
25 // codice non modificato e non riportato...

```

```

1 function delete_ngsi($name, $type, $contextbroker, $kind, $protocol,
2     $format, $model, $latitude,
3     $longitude, $visibility, $frequency,
4     $listnewAttributes, $ip, $port, $uri,
5     $service="", $servicePath="", &$result){
6
7     // codice non modificato e non riportato...
8
9     // Setup CURL
10    $ch = curl_init($url_orion);
11    $authToken = 'OAuth 2.0 token here';
12    curl_setopt_array($ch, array(
13        CURLOPT_CUSTOMREQUEST => 'DELETE',
14        CURLOPT_RETURNTRANSFER => TRUE,
15        CURLOPT_HTTPHEADER => array(
16            'Authorization: '.$authToken,
17            // Author: Antonino Mauro Liuzzo
18            'Fiware-Service: ' . $service,
19            'Fiware-ServicePath: ' . $servicePath
20        ),
21    ));
22 // codice non modificato e non riportato...

```

## Appendice A: Transazioni SQL

Una **transazione** è una successione di query che si conclude con un successo o un insuccesso. Nel primo caso gli effetti prodotti dalle query diventano permanenti, altrimenti il database torna nello stato precedente l'inizio della transazione.

Le transazioni sono possibili in MySQL solo con tabelle di tipo `InnoDB` e `BDB`.

Di default MySQL funziona in modalità `AUTOCOMMIT`; tutte le query che modificano il contenuto del database (`INSERT`, `DELETE`, `UPDATE`) hanno un effetto duraturo ed non possono essere annullate. Per effettuare una transazione è possibile disabilitare l'`AUTOCOMMIT` ed utilizzare i comandi `COMMIT` e `ROLLBACK` per confermare o annullare gli effetti delle query eseguite.

Di seguito viene riportato un esempio di transazione, eseguita in PHP tramite `mysqli`:

```

1 // Connessione al db
2 $conn = new mysqli('localhost', 'root', '', 'test_auto');

```

```

3
4 // Verifica Connessione
5 if ($conn->connect_error) {
6     die('Connection failed: ' . $conn->connect_error);
7 }
8 echo('Connected successfully<br>');
9
10 // Testo delle query
11 $qString1 = "INSERT INTO test_auto.persone VALUES
12 ($nickname, '$nomePersona', '$cognome')";
13 $qString2 = "INSERT INTO test_auto.auto (nome, proprietario) VALUES
14 ($nomeAuto1, '$nickname')";
15 $qString3 = "INSERT INTO test_auto.auto (nome, proprietario) VALUES
16 ($nomeAuto2, '$nickname')";
17
18 // Disattivazione autocommit -> Inizio della transazione
19 $conn->autocommit(FALSE);
20 $success = TRUE;
21
22 // Esecuzione delle query
23 if(!$conn->query($qString1)){
24     echo("Query 1 ERROR<br>" . $conn->error);
25     $success = FALSE;
26 }else{
27     echo("Query 1 success<br>");
28 }
29
30 if(!$conn->query($qString2)){
31     echo("Query 2 ERROR<br>" . $conn->error);
32     $success = FALSE;
33 }else{
34     echo("Query 2 success<br>");
35 }
36
37 if(!$conn->query($qString3)){
38     echo("Query 3 ERROR<br>" . $conn->error);
39     $success = FALSE;
40 }else{
41     echo("Query 3 success<br>");
42 }
43
44 // Commit o Rollback
45 if($success){
46     $conn->commit();
47 }else{
48     $conn->rollback();
49 }

```