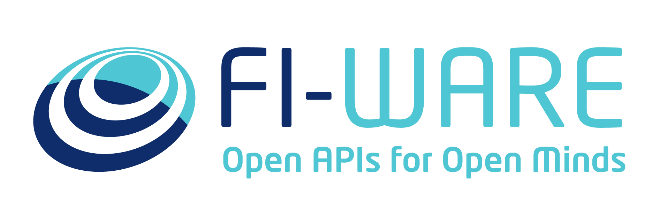
**MNCA Smart City**

****

****

**DSI Métropole Nice Côte d’Azur**

**IOT Management Application**

|  |  |
| --- | --- |
| VERSION: draft | REVISION DATE: 02/11/2017 |

L'approbation de l'architecture technique indique une compréhension de l'objectif et du contenu décrits dans ce livrable. En signant ce livrable, chaque personne est d'accord avec son contenu.

|  |  |  |  |
| --- | --- | --- | --- |
| **Approver Name** | **Title** | **Signature** | **Date** |
| Stéphane Roux | ICT Quality Assurance Engineer |  |  |
| Lionel Chaudanson | Responsable technique, IOT & Smart City |  |  |
| Serge Massiera | Directeur des Systèmes d'Information |  |  |
|  |  |  |  |
|  |  |  |  |

[1 Section 1 DOCUMENT SCOPE 3](#_Toc506222017)

[1.1 Contexte 3](#_Toc506222018)

[1.2 Glossaire 3](#_Toc506222019)

[2 Section 2 ARCHITECTURE TECHNIQUE 5](#_Toc506222020)

[2.1 Diagramme de contexte d'architecture de l’application 5](#_Toc506222021)

[2.2.1 *Considérations générales sur l'architecture* 5](#_Toc506222022)

[2.3 Définitions des composants de l’application 7](#_Toc506222027)

[2.3.1 Interface avec le Composant IDM GE 7](#_Toc506222028)

[2.3.2 Interface avec le Context Broker Orion 8](#_Toc506222029)

[*2.3.3* Interface avec les composants Agents IOT IDAS 8](#_Toc506222030)

[*2.3.4* Composant IOT Broker 9](#_Toc506222031)

[*2.3.5* Composant IOT Discovery 9](#_Toc506222032)

[2.3.6 Interface ETL (données importées de fichiers) 9](#_Toc506222033)

[*2.3.7* Interface avec Complex Event Processing 9](#_Toc506222034)

[*2.3.8* Accès Composant Application Mashup 9](#_Toc506222035)

[*2.3.9* Accès Composant Business Ecosystem 10](#_Toc506222036)

[2.3.10 Accès Composant Développement 10](#_Toc506222037)

[2.3.11 Accès Composant Monitoring 10](#_Toc506222038)

[2.3.12 API Management Component 10](#_Toc506222039)

[3 Section 3 Description de l’interface utilisateur 11](#_Toc506222040)

[3.1 Présentation globale 11](#_Toc506222041)

[3.2 Organisation de l’interface 11](#_Toc506222042)

# Section 1 DOCUMENT SCOPE

## Contexte

***Document Scope*** *décrit le contexte et les objectifs de ce document dans un récit.*

This document describes the IOT Management application for the MNCA smart

Ce document présente et décrit l’application de gestion de la plateforme IOT smartcity de la Métropole Nice Côte d’azur (MNCA).

La plateforme IOT Smart city MNCA est basée sur Fiware.

Fiware est basée sur une architecture micro-services, les composants présentent une interface basée sur une api RESTful, mais il n’existe pas d’interface utilisateur sauf pour le composant IDM, et la partie Business Ecosystem.

Afin de pouvoir gérer de façon ergonomique et simple les entités qui ne sont adressées qu’au travers d’API RESTful, il est nécessaire d’implémenter des interfaces homme machine pour les gérer au travers d’un application web.

## Glossaire

**SOA**: Service Oriented Architecture, est un style de conception de logiciel où les services sont fournis aux autres composants par des composants d'application, via un protocole de communication sur un réseau.

**Micro Services Architecture**: Microservices est une variante du style architectural SOA (Service-Oriented Architecture) qui structure une application en une collection de services faiblement couplés. Dans une architecture de microservices, les services doivent être affinés et les protocoles doivent être légers, cela impose naturellement une structure modulaire.

**FIWARE**: est une communauté ouverte et indépendante dont les membres s'engagent à concrétiser la mission FIWARE, à savoir: "construire un écosystème durable ouvert autour de standards de plate-forme logicielle publique, libre de redevances et d'implémentation qui facilitera le développement de nouvelles applications intelligentes dans plusieurs secteurs ".

**Plate-forme Fiware:** plate-forme fournissant un ensemble simple mais puissant d'API facilitant le développement d'applications intelligentes dans plusieurs secteurs verticaux: https://www.fiware.org/, elle repose principalement sur une architecture de services micro, où les composants exposent API RESTful.

**IOT**: Internet des objets

**OMA**: Open Mobile Alliance: OMA est une organisation à but non lucratif qui fournit des spécifications ouvertes pour créer des services interopérables qui fonctionnent à travers toutes les frontières géographiques, sur n'importe quel réseau support.

**TM Forum**: TM Forum est l'association industrielle mondiale qui pilote la transformation numérique de l'industrie des communications grâce à la collaboration.

**GE**: Generic Enabler, Fiware a utilisé la terminologie pour concevoir un module essentiel de la plate-forme

**API**: Interface de programmation d'application

**RESTful API**: également appelé service Web RESTful, il repose sur la technologie REST (Representational State Transfer), un style architectural et une approche des communications souvent utilisés dans le développement de services Web comme alternative à SOAP.

**Context Broker**: Selon Gartner:

"Un broker de contexte est un service conçu pour rassembler des données contextuelles accessibles d'une variété de types, de sources et de vitesses. Il applique ensuite le conditionnement, l'intégration, les règles et l'analyse pour dériver les données de contexte préparées réduites, exploitables par un système ou un humain au moment de la décision de l'entreprise.

D'un point de vue fonctionnel, le contexte permet aux données d'être renforcées par l'interaction avec d'autres éléments de données. En un mot, il peut être considéré comme un bus d'échange de données, fournissant un accès aux informations de contexte par le biais du paradigme de publication / abonnement (c'est un peu "MQTT" pour les données de contexte, via l'API RESTful).

**NGSI**: interface de services de nouvelle génération, OMA référencée API (NGSI V9 et V10) et utilisée avec des modifications par Fiware, pour utilisation via Context Broker, comme dorsale de la plateforme: NGSI V1 et maintenant NGSI V2 qui devient la version api de référence .

**Docker**: est une technologie logicielle de virtualisation fournissant des conteneurs pour exécuter des applications logicielles isolément.

**Swarm**: Docker Swarm fournit une fonctionnalité de cluster natif pour les conteneurs Docker, ce qui transforme un groupe de moteurs Docker en un seul moteur Docker virtuel.

**VRF**: Composant de réseau virtuel de routage et de transfert.

**RGPD**: Règlement général sur la protection des données

**RIA**: application Internet enrichie

**UI/IHM**: interface utilisateur (interface homme machine)

**Ontologie**: spécification formelle d'une conceptualisation, utilisée pour capturer explicitement la sémantique d'une certaine réalité

**SPOF**: ce dit d’un composant informatique qui est point unique d'échec (Single point of failure)

# Section 2 ARCHITECTURE TECHNIQUE

## 2.1 Diagramme de contexte d'architecture de l’application

L’application doit permettre la manipulation des services et entités gérés par plusieurs des composants de la plateforme Fiware, le schéma suivant, présente ce contexte global

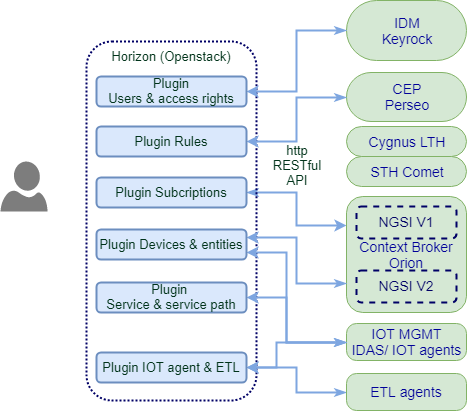


Figure : Schéma logique de l'application de gestion IOT

L’application est composée de plusieurs modules, développés sous forme de plugins pour Horizon

Il doit être possible de gérer les entités suivantes au sein de l’application de gestion IOT :

* Les utilisateurs, les droits d’accès vis-à-vis du composant IDM
* Les services (Fiware service) et sous services (service path) vis-à-vis des IOT agents
* Les devices (IOT agents) et les entités (Context broker)
* Les subcriptions vis-à-vis du context broker (pour configurer l’historisation vers Comet/Cygnus)
* Les Rules (règles), vis-à-vis du CEP
* Les agents IOT (si manager OK)
* Les modules ETL pour l’import des données à partir de sources fichiers

### 2.2.1 *Considérations générales sur l'architecture*

*La section Considérations générales sur l'architecture définit la manière dont les exigences techniques supplémentaires ont été traitées par l'architecture. Les éléments de cette section incluent :*

* *Stratégie de déploiement*
* *Stratégie de sécurité*
* *Exigences de disponibilité*
* *Exigences de performance*
* *Accessibilité*
* *Gestion base de données*
* *Politique de stockage*
* *Volumes de transactions*
* *Nombre d’utilisateurs concurrents*
* *Import, export et modèles de données*
* *Reprise après sinistre*
* *RGPD*

#### Stratégie de déploiement

Comme tous les autres composants de la plateforme, l’application sera déployée dans le swarm Docker.

Plus exactement comme les différents modules de cette application seront basés sur Horizon, ils seront déployés en tant que plugin dans un container Horizon



#### Stratégie de sécurité

L’application sera sécurisée au travers d’Horizon et du composant IDM

Même si l’application est utilisée dans un contexte d’administration, comme elle sera possiblement accédée de l’extérieur, il sera nécessaire de l’exposer en HTTPS, ce au travers d’un NGINX ou d’un HAProxy.

#### Exigences de disponibilité

Le composant Horizon sera déployé sous forme d’un service Docker qui permettra la haute disponibilité de l’application de management

#### Exigences de performance

Cette application sera utilisée pour l’administration de la plateforme, elle n’est pas destinée à être utilisée par beaucoup d’utilisateurs simultanément.

Néanmoins si nécessaire elle peut être déployée avec plus d’une instance.

#### Accessibilité

Rien de spécifique n'est inclus dans Fiware pour l'accessibilité, cet élément doit être pris en compte au niveau de l'interface utilisateur de l'application.

Les fonctionnalités d'accessibilité des systèmes d'exploitation ou des navigateurs Internet hébergeant l'application cliente seront utilisées.

#### Gestion des bases de données

L’application n’utilise pas de base de données, elle consomme uniquement des APIs RESTful, pour agir sur une partie des composants de la plateforme, comme un orchestrateur des différents services et entités qu’elle manipule.

#### Volumes de transactions

C’est une application d’administration, qui ne génère pas de volume de transaction conséquent.

#### Dimensionnement du stockage

Il n’y a pas de besoin spécifique de stockage, autre que l’image du container contenant le logiciel et la configuration.

Les logs du container Horizon sont redirigés à l’extérieur du container, ils seront capturés par le dispositif de surveillance qui sera mis en place (ElasticSearch/Kibana avec file Beat ou Logstash), c’est donc la base ElasticSearch qui supportera le besoin en stockage.

#### Stockage persistent

Pas de besoin spécifique pour l’application

#### Nombre d’utilisateurs et appareils connectés

C’est une application d’administration, qui n’a donc pas besoin de pouvoir supporter des centaines d’utilisateurs.

#### Import, export et modèles de données

Cette application étant dévolue à la configuration de la gestion des données qui entrent dans la plateforme, qu’elles proviennent d’appareils (devices) distants, de fichiers, d’une API, voire de base de données.

Comme la cible [Fiware data models](https://www.fiware.org/data-models/)

les “guidelines” [How To Contribute](https://github.com/fiware/dataModels#how-to-contribute).

#### Reprise après sinistre

Le composant doit être répliqué sur les 3 data centers pour assurer la meilleure disponibilité possible

#### RGPD

Ne s’applique pas dans le cas de cette application d’administration, qui ne stocke pas d’information personnelle concernant les utilisateurs.

## Définitions des composants de l’application

L’architecture de l’application est basée sur Horizon, provenant d’Openstack.

Horizon est l’implémentation « canonique » du projet « [OpenStack’s Dashboard](https://github.com/openstack/horizon) »

Horizon fournit une interface utilisateur basée sur le Web aux services OpenStack, comme Nova, Swift, Keystone, etc.

Pour un aperçu plus détaillé d'Horizon et de son architecture, consultez les Principes de base d'Horizon :

[Horizon Basics](https://docs.openstack.org/horizon/latest/contributor/intro.html#contributor-intro)

Pour apprendre ce que vous devez savoir pour commencer, consultez la page démarrage rapide. : [Quickstart](https://docs.openstack.org/horizon/latest/contributor/quickstart.html#quickstart).

L’application de gestion IOT doit pouvoir interagir avec les composants suivants de la plateforme Fiware au travers de leurs apis RESTful respectives :

* Le module IDM, en particulier le logiciel Keyrock basé sur Keystone, cette partie est déjà incluse dans Horizon (différemment selon le projet source).
* Les modules IOT Agents déployés sur la plateforme, y compris l’IOT manager lorsqu’il sera releasé
* Le module context broker Orion (le réservoir des données temps réel, cad les dernières rapatriées)
* Le module CEP en l’occurrence Perseo qui a été choisi par rapport à Proton (cf doc d’architecture générale)
* Les modules ETL d’import des données à partir de fichiers

Il existe 2 projets github celui d’openstack : <https://github.com/openstack/horizon>

Et celui forké pour Fiware : <https://github.com/ging/horizon> auquel ont été ajoutés des extensions

Les interfaces seront ajoutées en tant que plugins (cad hors du projet Horizon), qui intégreront les dashboards/panels nécessaires en fonction du besoin.

Pour le moment envisageons un plugin IDM et sécurité

et un plugin dédié à l’IOT, pour gérer les entités nécessaires, en s’appuyant sur la composante sécurité apportée par l’IDM.

### Interface avec le Composant IDM GE

Ce plugin présente les interfaces de gestion des identités et des droits d’accès sur les services offerts au travers de la plateforme,

Ce module existe déjà et est intégré dans le composant Horizon, il s’interface avec le module Keystone, l’IDM d’OpenStack.

Il faudrait cependant peut-être partir du Horizon (du repo ging), inclus dans le composant Keyrock du catalogue Fiware, qui intègre déjà les interfaces (dashboards) de l’IDM Fiware, alors que celui d’Openstack n’intègre pas les extensions pour Fiware.

2 solutions :

* Soit on part de la version Horizon OpenStack, et l’on ajoute les plugins nécessaires d’Horizon ging pour les extensions Keyrock
* Soit on part de la version ging, dans laquelle il faut aussi retirer/cacher certains liens vers des pages inutiles dans le contexte.

Le mieux serait d’extraire ou de trouver le plugin Keystone/Keyrock, pour le conserver à part, et reposer sur un Horizon vide comme base de l’application, auquel on vient ajouter des plugins dont celui pour Keystone/Keyrock.

Le module de gestion doit gérer au moins les utilisateurs, les applications/services, les rôles, les droits d’accès, et la génération des tokens, clés d’API.

### Interface avec le Context Broker Orion

Le context broker est à la fois le réservoir des données temps réel des entités et le chef d’orchestre de leur rediffusion vers des abonnés.

Il est principalement alimenté par les données provenant du terrain soit disponibles sous forme de fichier à importer périodiquement, soit provenant de dispositifs distants (passerelles, ou objets communicants) au travers des agent IOT.

Il est aussi responsable de redistribuer les données d’entités reçues aux abonnés qui ont souscrit (les subscriptions), ce mécanisme est notamment utilisé pour gérer l’historisation des données d’entités via Cygnus/Comet, ou la notification contrôlée du composant CEP si nécessaire (pour l’instant Perseo examine la base de données du contex broker).

Il n’est pas pour l’instant utile de pouvoir créer des « entities » directement à partir de l’application, car elles sont créées automatiquement par les agents IOT, lors de la création d’un device.

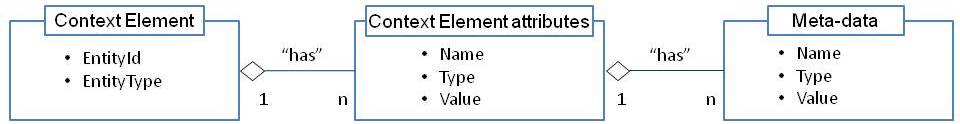


Figure : modèle d'une entité NGSI dans le context broker

Cependant pour certaines catégories d’entités il peut y avoir besoin d’ajouter des attributs qui ne sont pas gérés dans la source de données, donc il pourrait être utile de pouvoir modifier une entité pour lui ajouter ne serait que des méta-data, qui ne sont pas propagées par les agents IOT.

L’interaction de l’application avec le Context Broker Orion doit donc permettre :

* L’affichage de la liste des **entities** NGSI contenues dans sa base, avec une possibilité de filtrage sur le type d’entité, le service, le sous service/service path,
* La possibilité de modifier une entité, pour lui ajouter des attributs, des méta-données, .
* L’affichage de la liste des « **subcriptions** », la possibilité d’en créer de nouvelles, de modifier une existante, de les supprimer (CRUD).

Pour l’instant l’interface présentera une simple liste dans la moitié haute de la page, et la zone inférieure pour afficher le détail de la ligne sélectionnée dans la liste.

Le context broker propose 2 versions d’interface dérivées de la norme OMA NGSI-9 et NGSI-10 :

La V1 : <https://ngsi9.docs.apiary.io/>

La V2 : <https://orioncontextbroker.docs.apiary.io>

La V2 est plus simple d’utilisation, sera donc utilisée en priorité, par rapport à la V1, dès lors que cela est possible.

### Interface avec les composants Agents IOT IDAS

Les agents IOT sont en charge de convertir les données qui leur arrivent en entrée en objets entités NGSI et les transmettre au contexte broker.

Pour cela les agents IOT disposent d’une interface de configuration, permettant de provisionner des services et si besoin des devices.

Les

L’application doit donc permettre d’interagir avec les agents IOT au travers de leur [API de provisioning](https://telefonicaiotiotagents.docs.apiary.io/#reference/configuration-api) pour les entités suivantes : Les services (et sous-services) et les Devices.

* Les services et sous services (CRUD) :
  + Un service représente un tenant (par exemple une commune de la métropole, une direction métier dépendante de la métropole, ou un prestataire extérieur.
  + Il y a équivalence entre le service défini au niveau des agents IOT et la notion de « Fiware Service » du context broker Orion.
  + De même un **sous service** est **équivalent à** la notion de **service path** au niveau du context broker Orion
  + La combinaison service/sous service permet de définir les entités de destination dans le context broker cible, pour la famille entière de devices lui appartenant.
  + Entre autres le mapping entre les attributs des données source et les attributs des entités de destination peut être défini au niveau du service/sous service avec l’API des agents IOT, pour s’appliquer à tous les devices appartenant à la même famille.
  + Si l’API service permet aussi de définir des attributs statiques, ils ne sont pas répercutés vers les entités dans le context broker, on se demande à quoi cela sert !
* Les devices (CRUD) :
  + Un device est un dispositif qui diffuse des données issues de capteurs à destination de la plateforme IOT.
  + Si

[L’IOT agent manager](https://github.com/telefonicaid/iotagent-manager) qui joue le rôle de proxy vis-à-vis des IOT agents n’est toujours pas releasé dans le catalogue Fiware.

Le gestionnaire d'agents IoT fonctionne comme un proxy pour les scénarios dans lesquels plusieurs agents IoT proposent différents protocoles pour communiquer avec les objets, ce qui sera fort probablement le cas.

La documentation de l’API n’est pas complète, pour l’instant, la seule méthode documentée est « protocols » qui permet la déclaration d’un agent IOT.

Si le rôle du composant est de faire passerelle, il doit pouvoir recevoir les requêtes API destinées aux agent IOT, et les dispatcher en fonction du protocole, cela reste à tester…

Pour l’instant un seul IOT agent JSON est suffisant, donc le manager n’est pas nécessaire.

L’application de management s’adresse directement à l’agent IOT json pour le provisionning des services et des devices.

Il est aussi possible de faire pointer tous les agents IOT vers la même base de données Mongo, et ne s’adresser qu’à un seul pour la configuration des services et devices.

### Composant IOT Broker

Il n’est pas prévu de déployer ce composant pour l’instant

### Composant IOT Discovery

Il n’est pas prévu de déployer ce composant pour l’instant

### Interface ETL (données importées de fichiers)

De la même façon que l’on gère Services et Devices, et le mapping des attributs du device source vers les attributs de l’entité de destination, il serait judicieux de fournir une interface pour effectuer le mapping des attributs du document source vers les attributs destination de l’entité dans le context broker.

Un outil qui permet de charger un fichier csv, excel, xml et de configurer vers quels attributs entités ventiler les attributs source.

Pour simplifier la tâche et éviter les erreurs, il serait intéressant de pouvoir charger un modèle d’entité NGSI (à partir d’un schéma JSON, ou d’une entité modèle)

### Interface avec Complex Event Processing

Il est nécessaire d’interagir avec le CEP Perseo afin de pouvoir configurer des règles (rules de l’API Perseo).

Une règle s’applique à une entité ou à un groupe d’entité, elle comprend la partie condition (exemple : si température > un seuil) et la partie action si la condition est remplie (exemple : envoyer un email)

Une règle doit pouvoir être appliquée à une entité ou a un groupe d’entités NGSI.

### Accès Composant Application Mashup

L’application Horizon incluse dans le composant Keyrock, comporte un lien d’accès au portail de mashup Wirecloud, nous pourrons éventuellement offrir cette possibilité dans l’application de management.

### Accès Composant Business Ecosystem

De même un lien d’accès à l’interface de management Business Ecosystem pourrait être intégré dans le menu de l’application de management.

### Accès Composant Développement

Un lien d’accès au service de développement basé sur Eclipse Che pourrait aussi être ajouté au menu principal

### Accès Composant Monitoring

L’application pourrait aussi proposer un lien vers le système de monitoring de la plateforme au niveau du menu principal

### API Management Component

Un lien d’accès au service d’API management pourrait aussi être ajouté

# Section 3 Description de l’interface utilisateur

## Présentation globale

L’application de management est une application web, qui s’utilise dans un navigateur internet (Browser)

Une page de login permet à l’utilisateur de se connecter (login et mot de passe) de façon sécure à l’application (https oblige)

## Organisation de l’interface

Une fois l’utilisateur connecté, l’interface de l’application se présente sous la forme suivante :

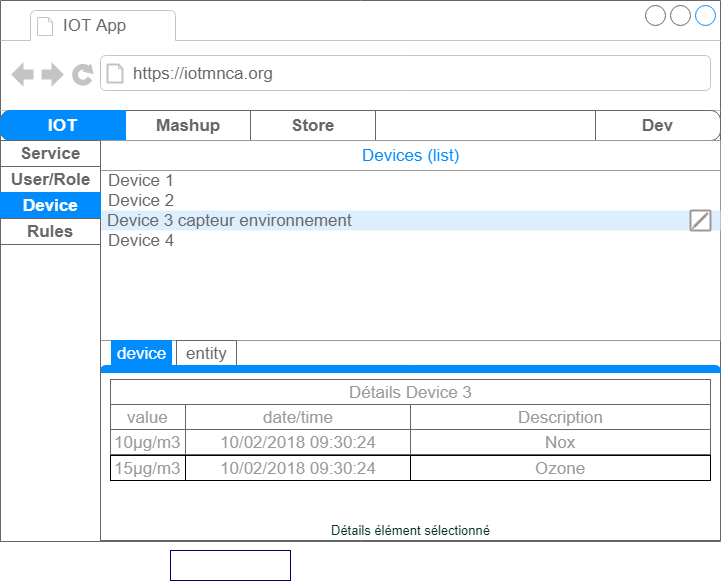


Figure : Organisation de lIHM de l'application