La spécification *OpenPS*

1. **Introduction** :

Dans une architecture logicielle, une API a pour fonction de simplifier la vie du développeur en lui offrant des entrées et sorties de communication (Input/Output) standardisées et découplées de leur implémentation applicative. Ce rôle d’interaction pour assurer ces implémentations pose de problème aujourd’hui dans les architectures distribuées et plus complexe tel que publication/abonnement et même dans les IoT etc.

Les interfaces de programmation (API) étant une des parties intégrantes des systèmes distribués [1]. Pour simplifier leurs utilisations, un certain nombre de documentations sur les normes des API ont été proposées ; pour les services Web basés sur REST, La spécification OpenAPI [2] a été adopté. Les APIs basées sur http sont courantes sur le Web, résident entre le serveur et les clients, ne répondent qu’aux requête bien formulées, et fournissent des URLs vers les ressources. Néanmoins, La conception d’API pour l’architectures pub/sub comme on le fait pour des APIs d’architectures centralisées (client/serveur) serait surement une erreur de conception d’architecture logicielle. Car on passe d’une logique applicative centralisée à une logique applicative distribuée où la fonction de l’API s’en trouve transformée. Cette erreur est pourtant fréquente car on continue de la gérer au niveau de la conception d’application alors qu’elle devrait être gérée au niveau de l’interface de communication qu’est l’API [3] pour rendre l’interconnexion transparente.

Comparer aux applications web, IoT est encore nouveau et de nombreuses APIs IOT sont disponibles. Leur principal protocole est devenu un des protocole Pub/Sub (MQTT) qui leur permettra probablement une forte croissance dans le monde des APIs. Car leurs principaux fournisseurs comme Amazon, Azure et IBM fournissent à la fois des API http et MQTT pour accéder à leurs services [5].

A ce jour, étant donné l’existence de quelques APIs de type pub/sub [4], aucune spécification, ni outil d'installations ni test en ligne simples disponibles pour apprendre à utiliser les API IOT n’a encore été définie pour uniformiser la description des APIs pub/sub à la manière du modèle Web alors que, le développement d’application IoT ne cesse de croître. Ces APIs Pub/Sub bien qu'il existe plusieurs standards pour les systèmes pub/sub [6, 7], communiquent à travers des canaux, fournissent des abonnements topics permettant aux clients la réception des informations à tout moment et intéressantes dans le monde des IoT. Cela implique que chaque projet d'API est unique et dépend des cas d'usages spécifiques. Pour le partage des bonnes pratiques d’implémentation, des ressources, et l’interconnexion d’application ou API, nécessite de mettre en place des règles communes qui seront respecter par chaque API d’où la bienvenue d’une spécification.

Dans ce chapitre nous décrivons une spécification open source pub/sub appelée *OpenPS* similaire à celle de *OpenAPI* permettant d’uniformiser les APIs, en utilisant le modèle pub/sub afin de minimiser l’intervention des développeurs dans la communication des applications.

1. **Définition** :

La spécification *OpenPS* définie une interface standard pour les APIs en se basant sur le modèle Publish/Subscribe. Elle assure l’interopérabilité et la lisibilité des codes par les producteurs, consommateurs ainsi que pour les machines. Elle se sert de quelques objets de la spécification *OpenAPI* et d’autres nouvel objet pour la description et la mise en place des APIs REST.

Avec les outils de documentation et de génération de code, une spécification *OpenPS* permet de générer le code du producteur et du consommateur dans divers langages de programmation.

1. **Constitution de la spécification *OpenPS***:

Tout comme la spécification *OpenAPI*, la spécification *OpenPS* est constituée d’un ensemble d’objet suivant une structure hiérarchique. Une spécification *OpenPS* est composé de sept (9) objet racine. Parmi ces objets la présence de quatre (4) sont exigé pour un minimum description d’un document *OpenPS*.

protocol

string

broker

object

info

object

openps

string

tags

object

security

object

components

object

topic

object

externalsDocs

object

**Figure 3.1** : différents objets de *OpenPS*

1. **openps** :

C’est une chaîne de caractère décrivant la version utilisée par les documents *OpenPS* permettant aussi leurs interprétations par les outils de cette spécification et fait partie des objets exigé par la spécification. Cette spécification est à la version « 1.0.0 »

1. **info** :

C’est un objet dont sa présence dans la spécification est obligatoire. Il fournit les informations sur l’entête de l’API, il est constitué d’autre objet dont les plus important sont :

* **title** : le nom de l’API.
* **version** : version du document *OpenPS* « 1.0.0 ».

1. **broker** :

C’est un objet exigé par la spécification appelé aussi courtier qui offre les services Publish/Subscribe. Sa valeur fournie des informations des entités cibles. Il exige la présence de certains objets conteneurs tels que :

* **name** : est une unique chaîne de caractère qui représente le nom du broker.
* **hostname** : représente le nom d’hôte du broker cible (ici localhost).
* **port** : un nombre entier qui désigne le port du broker cible.

1. **protocol** :

Il précise le protocole à implémenter pour la communication Publish/Subscribe par exemple le protocole AMQP.

1. **topics** :

Cet objet décrit le nom des topics disponible ainsi que les opérations de publication/abonnement pour l’API. Il correspond à l’objet « Path » de la spécification OpenAPI. Il exige que le nom d’un topic commence par un slash (/) sur lequel il n’y a que deux opérations disponibles et un objet message exprimant le format de l’information échangée sur ce topic. Il est le dernier objet exigé par cette spécification. Ces trois composantes constitutives sont les suivants :

1. **publish** :

Il définit l’opération publish (de publication) d’un message sur un topic. C’est un objet qui décrit les résumés sur un topic à publier, l’entité chargé de publier ce topic ainsi que les qualités de service pour la distribution de ce topic. Il définit ces informations à travers d’autres objets qu’il supporte. Ces objets sont présentés ci-dessous :

* **tags** : c’est un tableau de chaine de caractère dont l’item i correspond à la description de l’opération de catégorie i.
* **summary** : c’est une chaine de caractère désignant un petit résumé de ce que fait l’opération publish.
* **description** : c’est la description de l’opération publish en chaine de caractère.
* **entities** : définis le points de terminaison ou l’entité qui publie les messages.
* **QoS** : c’est une chaine indiquant comment le message est distribué ; les valeurs autorisées sont : *none, at-most-once, at-least-once, exactly-once*.

1. **subscribe** :

Il définit l’opération subscribe (abonnement ou récupération d’un message) sur un topic. C’est un objet qui décrit les résumés de l’abonnement sur un topic et l’entités chargé de récupérer les messages sur ce topic. Il fournit ces informations à travers les mêmes objets que publish mais ne prend pas en charge la qualité de services QoS.

1. **message** :

Il contient trois (3) composants permettant de définir le format et le contenu d’un message.

* ***description*** : un petit résumé en chaîne de caractère décrivant le message.
* ***content*** : c’est un tableau *map*(*string*, *object*) permettant de définir le contenu d’un message
* ***Required*** : un booléen qui indique si le corps du message est exigé, il a la valeur false par défaut.

1. **components** : c’est un objet conteneur qui permet à la description d’éviter la duplication du code des opérations qui ont des paramètres en commun, et tout en le référençant dans l’opération en question avec la propriété $*ref* qui a un chemin comme valeur depuis l’objet racine. Il correspond au même objet « *components »* de la spécification *OpenAPI*.
2. **security** : il respectera toutes les règles de celle de *OpenAPI* mais pas encore défini.
3. **tags** : le même que la spécification *OpenAPI*.
4. **externalsDocs** : le même que la spécification *OpenAPI*.
5. **Les langages utilisés** :

Comme la spécifications *OpenAPI*, la spécification *OpenPS* supporte deux langages de description des APIs qui sont : JSON et YAML.

1. **L’outil utilisé par *OpenPS* :**

La spécification *OpenPS* utilise les mêmes outils que la spécification *OpenAPI*, il s’agit de Swagger UI, Swagger Editor et Swagger Codegen. *OpenPS* modifie les fichiers Codegen pour subvenir à ses fins et utilise la manière d’ajout des extensions de la spécification OpenAPI (x-) pour définir les différentes opérations. Avec Swagger Editor et Swagger UI nous décrivons, éditons et générons nos documents APIs et avec Swagger Codegen générons des clients permettant d’interagir avec notre API.

1. **La syntaxe de la spécification *OpenPS*** :

Cette syntaxe indique comment les objets de cette spécification se combines et comment spécifier les valeurs des différentes propriétés.

Les objets qui n’ont pas encore été définis tels que *security* ne seront pas présenté dans la syntaxe ci-dessous de cette spécification.

openps: 1.0.0

info:

  version: 1.0.0

  title: Multiverse Telemetry API

  description: A simplified version of the Controller-Agent pub-sub API for telemetry.

brokers:

  - name: broker1

    description: First AMQP broker.

    hotsname: 'broker1.multiverse-nms.com'

    port: 5672

protocol: 'AMQP 1.0'

topics:

  '/capability':

    publish:

      entity: agent

      description: Agent advertizes its capabilities.

      qos: 'none'

    subscribe:

      entity: controller

      description: Controller collects agents measurement capabilities.

    message:

      required: true

      content:

        application/json:

          schema:

            $ref: '#/components/schemas/Capability'

components:

  schemas:

    Capability:

      description: A capability provided by the agent.

      allOf:

        - type: object

          required:

            - capability

          properties:

            capability:

              description: Verb of the capability message.

              type: string

              enum:

                - measure

externalDocs: ‘C:\Users\OASC\Desktop\Projet\chapitres\openps- specification.doc’

**Figure 3.2** : extrait de la spécification *OpenPS*

1. **Conclusion** :

Les API connectent les machines et les humains. Ce sont les éléments constitutifs de l'écosystème numérique connecté du 21e siècle, travaillant en arrière-plan pour tout créer, des jeux vidéo aux appareils médicaux etc.

La spécification *OpenPS* crée un cadre commun pour l'ensemble du cycle de vie de l'API afin de définir les fonctionnalités des service RESTful, y compris leurs ressources. *OpenPS* est lisible à la fois pour l'homme et pour la machine. Elle agit comme le modèle de l'ensemble du développement et de la livraison des APIs permettant de communiquer la valeur et la fonctionnalité des APIs, à la fois aux parties prenantes internes et aux consommateurs externes. Elle est open source offrant avec ses outils plus de compréhensibilité pour aider les développeurs à prendre plus d’avantage de toutes les capacités d’une API.

Dans le prochain chapitre, nous décrirons l’architecture de la spécification *OpenPS*, nous présenterons quelques outils utilisés pour la génération automatique de code. En utilisant comme entré (input) un document de la spécification *OpenPS* (en YAML) et un cas d’utilisation de *OpenPS*.

[1] <https://eudatasharing.eu/fr/technical-aspects/api-guidance-partie-1>

[2] Koren, István, and Ralf Klamma. "The exploitation of openapi documentation for the generation of web frontends." *Companion Proceedings of the The Web Conference 2018*. 2018.

[3] <http://web.archive.org/web/20210124225408/http://www.christian-faure.net/2016/02/07/architecture-des-apis-dans-les-systemes-distribues/>

[4] <https://zato.io/docs/pubsub/api/rest.html>

[5] <http://www.steves-internet-guide.com/iot-apis/>

[6] SJÖBERG, Pekka. The Java Message Service 1.0. 2. *Retrieved on August*, 2007, vol. 10, p. 2011

[7] CURBERA, Francisco, FERGUSON, Donald, GRAHAM, Steve, *et al.* Web services eventing (ws-eventing). 2004.