La spécification *OpenPS*

1. **Introduction** :

Aujourd’hui de nombreuses applications utilisent des modèles fortement orientés contenu et plus complexes que le modèle Web (client-serveur) pour répondre à des exigences de communication souhaité. En effet, les applications nécessitant ces modèles sont de plus en plus utilisées dans l’Internet d’aujourd’hui et sont par exemple celles de l’Internet des Objets (IoT) telles que les systèmes de gestion des bâtiments ou des usines connectées. Ou encore des nouvelles applications de management des réseaux informatiques d’entreprises. Un des modèles utilisant des communications orientées contenu est le publication/abonnement (publish/subscribe, ou pub/sub). Il est déjà largement utilisé dans les réseaux et applications d'entreprise, principalement en raison de son évolutivité et de la prise en charge de la topologie de réseau dynamique. Des protocoles pub/sub sont déjà très utilisés tels que AMQP, et MQTT, ou encore des systèmes complets de messagerie tels que Kafka et RabbitMQ. La technologie des réseaux de données nommées (Named Data Networking), proposée pour l’internet du futur supporte efficacement le modèle pub/sub avec des protocoles tels que PSync, Syncps, et pub/sub dans NDN-Lite.

A ce jour, aucune spécification n’a encore été définie pour uniformiser la description des APIs pub/sub à la manière du modèle Web.

Dans ce chapitre nous décrivons une spécification open source pub/sub appelé OpenPS similaire à celle de OpenAPI permettant d’uniformiser les APIs REST 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 web en se basant sur le modèle Publish/Subscribe. Elle assure l’interopérabilité et la lisibilité des code par les producteurs, consommateurs ainsi que pour les machines. Elle se réfère sur la spécification OpenAPI pour la description et la mise en place des APIs REST.

Avec les outils de documentation de génération de code une spécification OpenPS permet de générer le code du producteur et du consommateur dans divers langages de programmation. Le document OpenPS peut être un seul ou divisé en plusieurs partie connectées en référençant ces parties. Dans ce cas il est recommandé de nommer le document racine openps.json ou openps.yaml.

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 (7) objet racine. Parmi ces objets la présence de quatre (4) sont exigé pour un minimum description d’un document 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 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 représente l’objet Path de la spécification OpenAPI. Il exige que le nom des topics 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.

1. **publish** :

Il définit l’opération publish (de publication) d’un message sur topic. C’est un objet qui décrit les résumés sur un topic à publier, l’entités 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** :
* **QoS** : c’est une chaine indiquant comment le message est distribué et 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 et à la valeur false par défaut.

1. **components** : c’est un objet conteneur qui permet la description pour éviter la duplication du code des opérations qui ont des paramètre 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 objet est le même comme dans la spécification OpenAPI.
2. **security** :

Il respectera toutes les règles de celle OpenAPI mais pas encore défini.

1. **tags** : le même que la spécification OpenAPI.
2. **externalsDocs** : le même que la spécification OpenAPI.
3. **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 1**

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 et tout le reste.

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 par l'homme et par la machine, et 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 capabilités d’une API.

Dans le prochain chapitre nous décrirons l’architecture de la spécification OpenPS, nous présenterons quelque outil utilisé pour la génération automatique correspondant à un document de la spécification OpenPS et un cas d’utilisation de OpenPS.

https://mina7.net/pin/