Propriétés du Document

Source du Document	
Titre du Document	Procédure d'ajout d'un composant ACCORDS en Python
Module(s)	ACCORDS
Responsable	Prologue
Auteur(s) / contributeur(s)	Elyès Zekri & Hamid Medjahed
Statut du Document	Draft
Version	V0.5
Validé par **Date de la validation**	

Résumé

Ce document présente la procédure d'ajout d'un composant (fournisseur de service) à la plateforme ACCORDS

Mots Clefs

Fournisseur de service, composant, OCCI, ACCORDS

Table des Matières

1.Introduction

Ce document s'adresse aux développeurs souhaitant mettre en œuvre un composant (fournisseur de services) ACCORDS [1][2][3] en Python. Ce document présente donc une marche à suivre basée sur l'interface Python C/API [8].

D'un point de vue technique, un composant ACCORDS est susceptible d'être implémenté dans n'importe quel langage de programmation du moment que les deux conditions suivantes sont vérifiées:

1)Le composant doit être en mesure de gérer les messages du standard OCCI/REST [5][6] [7]

2)Une connaissance à *priori* de l'architecture et des fonctionnalités ACCORDS est nécessaire (Resolver, Publisher...)

Afin d'aider le développeur dans sa démarche de création d'un composant en Python, nous proposons une procédure générique et paramétrable dont l'objectif est de générer un script Python décrivant le comportement du module.

2.Pré-requis

Afin d'ajouter un nouveau composant CompatibleOne, il est nécessaire de disposer du noyau de la plateforme ACCORDS organisé autour des interfaces OCCI et CORDS, des outils de manipulation de données (xml,...), ainsi que des composants de base de la plateforme ACCORDS (publisher, coss...). Ces éléments sont décrits dans les livrables [1] et [2].

Les sources de la plateforme ACCORDS et la procédure d'installation sont accessibles par l'intermédiaire du lien donné par [4].

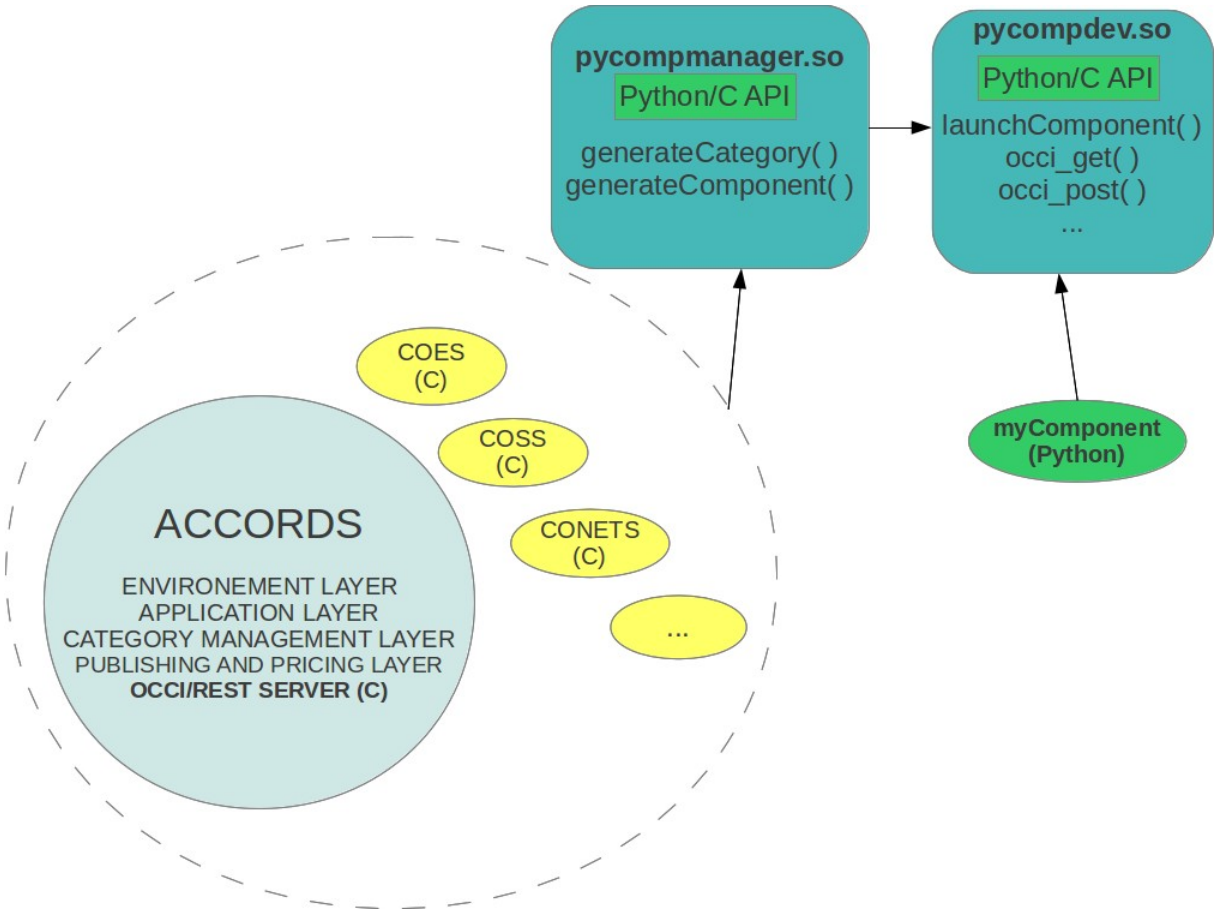
3.Procédure d'ajout d'un composant ACCORDS

Dans ce qui suit, seront décrites les étapes qui mèneront à la génération d'un composant ACCORDS sur la base de la définition d'un ensemble de catégories OCCI qui lui seront associées.

a)Principe

Le principe de cette procédure consiste à faire appel à des fonctions génériques compilées sous forme de bibliothèques partagées pour le paramétrage d'un module ainsi que pour sa création.

Le schéma synoptique suivant permet de mettre en évidence les différents composants mis en jeu dans cette procédure



La bibliothèque « pycompmanager.so » contient les fonctions de gestion des composants ACCORDS en langage Python.

La bibliothèque « pycompdev.so » contient les fonctions et les types Python permettant le développement d'un module : structures de données représentant les catégories, fonction de lancement d'un module...

Ces deux bibliothèques résultent d'un interfaçage avec des fonctions de gestion des composants codées en C. Cet interfaçage est réalisé par l'intermédiaire de l'API Python/C [8].

b)Installation

Pour générer la bibliothèque « pycompmanager.so » il faut taper les commandes suivantes au niveau du sous-répertoire « pyaccordsSDK/pycompmanager » :

```
python setup.py install
```

La bibliothèque « pycompdev.so » sera générée automatiquement suite aux manipulations qui seront effectuées dans ce qui suit.

c)Gestion des catégories OCCI

Une catégorie est une entité OCCI permettant d'organiser sémantiquement les ressources. Elle est caractérisée principalement par un ensemble d'attributs et par une liste d'actions [5].

Nous proposons deux fonctions de la bibliothèque « pycompmanager.so » permettant de générer une nouvelle catégorie ou de supprimer une catégorie existante :

generateCategory (categoryName, attributeList, accordsDirectory)

où

categoryName désigne le nom attribué à la catégorie

attributeList désigne la liste des attributs (séparé par un espace) qui seront associés à la catégorie

accordsDirectory désigne le répertoire du projet ACCORDS

Cette fonction permet de générer les fichiers sources liés à la catégorie et de les placer au niveau des sous-répertoires du projet ACCORDS.

Ces fichiers sont :

1) les fichiers C placés dans les sous répertoires « occi/src » et « cords/src » contenant :

- •les fonctions d'accès OCCI de base : POST, GET, PUT...
- •les fonctions de gestion de la mémoire spécifiques à la catégorie créée.

2) Les fichiers Python permettant l'interfaçage avec les fichiers C précédents. Parmi ces fichiers, celui qui nous intéresse le plus est le fichier « categoryInterface.py » se trouvant dans le sous-répertoire « pyaccords » et contenant la définition des méthodes d'accès CRUD (Create, Retrieve, Update, Delete). C'est ce fichier qui est censé être modifié par le développeur.

eraseCategory (categoryName, accordsDirectory)

Cette fonction permet de supprimer une catégorie existante en supprimant ses fichiers sources générés au niveau du répertoire du projet ACCORDS.

Remarque : à l'issue des opérations de génération/suppression de catégorie et afin que :

1) les bibliothèques ACCORDS soient mises à jour et
2) la bibliothèque « pycompdev.so » soit compilée,

il faut faire appel à la fonction « commit() » qui va procéder à la recompilation.

Exemple :

Voici l'exemple d'un script Pyhton permettant de créer une catégorie « mycategory » contenant deux attributs « attribute1 » et « attribute2 »:

```
>>>import pycompmanager
>>>pycompmanager.generateCategory (“myCategory”, “attrbute1 attribute2”, “/home/accords-platform/”)
>>>pycompmanager.commit()
```

le fichier Python contenant l' interface CRUD liée à la catégorie « mycategory » générée à la suite de l'appel de la fonction « generateCategorie() » a la structure suivante :

```
import sys
import pycompdev
sys.path.append("/home/accords-platform/pyaccords/pysrc")
from mycategoryClass import *
""" Note: mycategory is a python class to interface the accords category :
        -Attributes of this category are members of this class.
        -List of attributes:
               - attribute1
                - attribute2
"""

def mycategory_create(mycategory):
        """Implement here your CRUD create function"""
        return  mycategory

def mycategory_retrieve(mycategory):
        """Implement here your CRUD retrieve function"""
       return mycategory

def mycategory_update(mycategory):
        """Implement here your CRUD update function"""

        return mycategory

def mycategory_delete(mycategory):
        """Implement here your CRUD delete function"""
       return mycategory
```

d)Création d'un composant ACCORDS

Les composants, tels qu'il sont implémentés dans ACCORDS, ont tous le même comportement de base, à savoir:

1)Parser les paramètres passés en argument (fichier de configuration, nom de l'agent associé au composant,...)

2)Charger les données de configuration (adresse du composant, adresse du publisher,...)

3)Publier les catégories au niveau du publisher

Afin de générer un composant Python mettant en œuvre ce comportement de base, il faut faire appel à la fonction suivante :

generateComponent (componentName, categoryList, accordsDirectory)

où

componentName désigne le nom attribué au composant

categoryList désigne la liste des catégories (séparées par des espaces) qui seront associées au composant (ces catégories doivent préalablement avoir été créés par la fonction « generateCategory() »

accordsDirectory désigne le répertoire du projet ACCORDS

Le résultat de l'appel de cette fonction est un fichier généré dans un sous-répertoire portant le même nom que le composant

Exemple:

Voici l'exemple d'un script python permettant de générer un module

```
>>>import pycompmanager
>>>pycompmanager.generateComponent(“myComponent”, “categ1 categ2”,
   “/home/accords-platform/”)
```

Remarque : l'appel de « commit() » n'est pas nécessaire suite à la generation d'un composant

Le fichier Python “mycomponent.py” généré dans le sous-répertoire “mycomponent” aura la structure suivante :

```
import sys
import pycompdev

def main():
        argc=len(sys.argv)
        return pycompdev.launchComponent( argc  ,sys.argv , "mycomponent",
          "categ1 categ2")
if __name__=="__main__":
        main()
```

Remarque : pour lancer le module, taper la commande suivante :

python mycomponent.py –config config_file.xml CO-MYCOMPO-AGENT/1.0

4.Références

[1] L01.1 (ACCORDS) - Distribution des ressources (Architecture, API, protocoles) : Spécifications

[2] L01.2 (ACCORDS) - Distribution des ressources (Architecture, API, protocoles) : Version V0

[3] L17.1 (CORDS) - Modele de description des ressources (CORDS) : Specifications et Documentation V0

[4] Accords Platform at OW2 : http://gitorious.ow2.org/ow2-compatibleone/accords-platform

[5] Open Cloud Computing Interface - Core
http://www.ogf.org/documents/GFD.183.pdf

[6] Open Cloud Computing Interface - Infrastructure
http://www.ogf.org/documents/GFD.183.pdf

[7] Open Cloud Computing Interface – Http Binding
http://www.ogf.org/documents/GFD.183.pdf

[8] Python/C API : http://docs.python.org/c-api/