26/10/2021

IO 6: ТУТОРИЈАЛ ЗА ПРОЈЕКТОВАЊЕ И ТЕХНИЧКУ ИМПЛЕМЕНТАЦИЈУ ВЕБ ЛАБОРАТОРИЈА – ИМПЛЕМЕНТАЦИЈА СЕРВЕРА (PYTHON)

Co-funded by
the European Union

1. УВОД

Интерфејс удаљене лабораторије развијен са EjsS може лако комуницирати са лабораторијским хардвером/софтвером коришћењем *Remote Interoperability Protocol* (RIP). За ово су потребне две ствари: RIP елемент EjsS-а (погледајте упутство за имплементацију клијента) и имплементација RIP сервера.

Ово упутство описује како користити RIP сервер који је имплементиран у Python-у. Постоји још једно упутство за имплементацију RIP сервера у LabVIEW-у (погледајте упутство за имплементацију сервера (LabVIEW)). Имплементација Python-а, која је обрађена у овом документу, је изузетно корисна када је лабораторијска опрема управљана са MATLAB/SIMULINK, Node.js, Arduino, Red Pitaya плоче, итд. Глава 2 овог приручника објашњава како конфигурисати RIP сервер и користити га да бисте објавили своје управљачке програме као веб сервисе који могу бити коришћени путем EjsS апликације са RIP елементом.

Али претходно треба знати где да преузмете Python имплементацију RIP сервера. Софтвер је доступан на https://github.com/UNEDLabs/rip-python-server. За најнапредније кориснике (оне који желе да се баве развојем), документ који представља спецификацију Remote Interoperability Protocol може се наћи овде: https://github.com/UNEDLabs/rip-spec .

2. КОРИШЋЕЊЕ PYTHON ИМПЛЕМЕНТАЦИЈЕ RIP СЕРВЕРА

Python имплементација RIP сервера је дистрибуирана као изворни код. Последњу верзију можете подићи са *github* репозиторијума (https://github.com/UNEDLabs/rip-python-server). Преузмите га као зип датотеку, или, ако имате инсталиран *git*, клонирајте спремиште отварајући терминал и откуцајте следећу команду:

```
git clone https://github.com/UNEDLabs/rip-python-server
```

Овај водич претпоставља да већ имате инсталиран Python 3, а уколико немате, посетите https://www.python.org/ и преузмите најновију верзију.

Следећа поглавља ће вас упутити како да: 1) подесите Python виртуелно окружење и проверите да ли ваш *rip-python-server* ради (2.1), 2) конфигуришете вашу апликацију (2.2), и 3) проширите сервер да одговара вашим специфичним потребама (2.3).

2.1. Подешавање виртуелног окружења

Rip-server-python зависи од библиотека *cherrypy*, *ujson*. Зависно од ваше апликације, ви ћете можда имати потребу да инсталирате *oct2py* или *MATLAB engine* за Python. Можете користити инсталацију Python-а на свом систему, али ми сматрамо да је предност користити виртуелну околину. Иако постоје различити алати које можете користити, овај водич користи *virtualenv*. Овај алат можете инсталирати следећом командом:

```
pip3 install virtualenv
```

Затим, креирате ново виртуелно окружење:

```
virtualenv -p python3 venv
```

Опција `-p python3` је да осигура да користимо коректну верзију, и `venv` је директоријум у коме се чува окружење. Од сада запамтите да морате да додате префикс `venv/bin/` да бисте извршили команде унутар виртуелног окружења, а не на инсталацији вашег система. Следећи корак је инсталирање зависности:

```
venv/bin/pip install cherrypy ujson
```

У зависности од модула за имплементацију који желите да користите за своју апликацију (Matlab, Arduino, итд.), можда ћете морати да инсталирате друге потребне модуле. На пример, за Arduino, такође би требало да покренете:

```
venv/bin/pip install serial pyserial
```

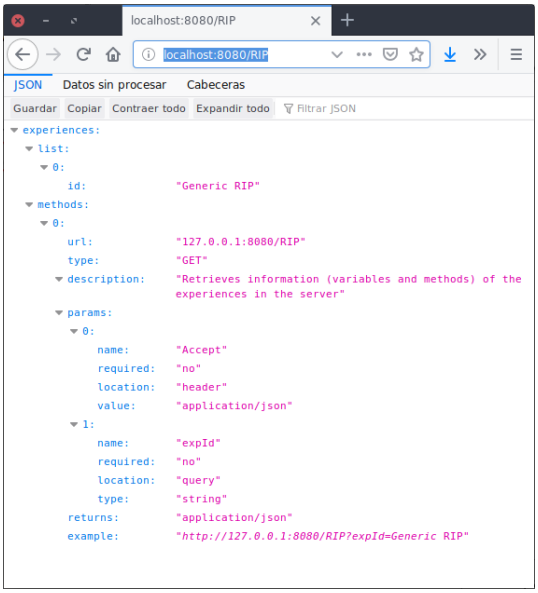
Код пројекта је подразумевано конфигурисан за Матлаб. Према томе, следеће неће радити осим ако немате инсталиран Matlab. Да бисте променили имплементациони модул у други (Arduino, на пример) погледајте одељак 2.2.

Коначно, можете покренути RIP сервер:

```
venv/bin/python3 App.py
```

RIP сервер би требало да се чита са порта 8080. Да би проверили да ли то заиста ради, отворите ваш прегледач и унесите URL адресу: http://localhost:8080/RIP. Требало би да добијене JSON

одговор као што је показано на Слици 1. Одзив садржи метаподатке RIP сервера, а који описују могућности сервера и историју стања која се на њему налази. Можете да прочитате RIP спецификацију за детаље имплементације.



Слика 1. Тестирање одзива RIP сервера.

2.2. Дефинисање ваше апликације

Rip-python-server је грубо подељен на два нивоа: први који имплементира РИП функционалност, а други који имплементира приступ хардверу. Пошто ниво комуникације обично није подложан променама, да бисте подесили сервер, потребно је само да повежете апликацију са одређеном имплементацијом хардвера у конфигурационој датотеци `AppConfig.py`, и опционо да пружите неке додатне информације о дотадашњим стањима података на серверу: подаци који су доступни за читање и писање путем сервера, аутори који су створили или променили неки од података на серверу, кључне речи које се користе за описивање садржаја података на серверу, итд. Ви можете нажи неке примере како то да урадите у директоријуму `config-examples`. Следећи листинг програма одговара основној конфигурацији за тестирање:

```
# This file contains the configuration of the RIP server application.
config = {
  'server': {
    'host': '127.0.0.1',
    'port': 8080,
  },
  'control': {
    'impl_module': 'RIPGeneric',
    'info': {
      'name': 'Generic RIP',
      'description': 'A generic implementation of RIP',
      'authors': 'J. Chacon',
      'keywords': 'Raspberry PI, RIP',
      'readables': [{
        'name':'time',
        'description':'Server time in seconds',
        'type':'float',
```

```
            'min':'0',
            'max':'Inf',
            'precision':'0'
          }],
          'writables': []
      }
    }
}
```

Слика 2. Конфигурација RIP сервера.

Потребно је да специфицирате следећа поља:

1. `config.control.impl_module`: садржи име модула који имплементира приступ хардверу. Можете да користите било коју од уграђених имплементација (`RIPMatlab`, `RIPOctave`, ...) или дефинисати сопствени управљачки модуо како ће бити објашњено касније.
2. `info`: садржи дефиницију искуства (о дотадашњим стањима података на серверу), укључујући назив искуства, кратак опис, ауторе и кључне речи и листу елемената који се могу читати (*readables*) и елемената који се могу уписивати (*writables*).

Као и раније, можете покренути РИП сервер да бисте тестирали нову конфигурацију, са следећом командом:

```
venv/bin/python3 App.py
```

Апликација ће очитавати *host* – рачунар и специфицирани порт у конфигурацији. Отворите нов прозор прегледача и верификујте да сервер ради.

2.3. Креирање новог хардверског интерфејса

У случају да уграђени хардверски адаптери не одговарају вашој апликацији, можете написати сопствени код и интегрисати га са RIP сервером. У ту сврху ћете морати да креирате фасаду која ће открити ваш хардверски приступ ниског нивоа преко RIP API-ја (погледајте документ RIP спецификација). Да би ваш задатак био лакши, класа `RIPGeneric` (дефинисана у `rip/RIPGeneric.py`) обезбеђује заједничку функционалност RIP-а, тако да можете да је подкласирате да додате свој код. У најмању руку, потребно је да урадите следеће задатке:

1. Увезите класу `RIPGeneric` и било коју другу библиотеку коју треба да користите.
2. Креирајте потокласу која проширује `RIPGeneric`.
    a. Дефинишите конструктор `__init__()` са кодом за иницијализацију.
    b. Дефинишите како читати и писати серверске објекте, надјачавајући методе `set` и `get`.
    c. Пријавите променљиве које треба да се шаљу у периодичним ажурирањима, замењујући метод `getValuesToNotify`.
3. Следећи листинг садржи пример минималне имплементације обезбеђене као шаблон (имајте на уму да морате да ставите свој код у фасциклу "rip"). Код креира сервер који обезбеђује два читљива објекта: `time`, које враћа време непрекидног рада сервера, и `random`, који враћа насумични број, и један уписив: `seed`, да би се модификовало семе произвољног генератора. Сервер прихвата SSE везе и периодично пријављује насумичне бројеве повезаним клијентима.

```
import random
from rip.RIPGeneric import RIPGeneric
```

```
class RIPAdapterTemplate(RIPGeneric):
  '''
  RIP Adapter Template
  '''

  def default_info(self):
    return {
      'name':'RIPAdapterTemplate',
      'description':'A template to extend RIP Generic',
      'authors':'J. Chacon',
      'keywords':'Adapter Template',
      'readables':[{
        'name':'time',
        'description':'Server time in seconds',
        'type':'float',
        'min':'0',
        'max':'Inf',
        'precision':'0',
      },

      {
        'name':'random',
        'description':'Random value generator',
        'type':'float',
        'min':'0',
        'max':'1',
        'precision':'0'
      }],
      'writables': [{
        'name':'seed',
        'description':'Random seed',
        'type':'float',
        'min':'0',
        'max':'1',
        'precision':'0'
      }],
    }
```

Слика 3. RIPAdapterTemplate - Дефиниција.

Код на слици 3 је једноставан. Прво увози стандардни пакет насумично који ће се користити за генерисање насумичних бројева и `RIPGeneric`, и онда креира класу под називом `RIPAdapterTemplate` која проширује `RIPGeneric`. Метод `default_info` обезбеђује подразумевану дефиницију искуства, која се може заменити у in AppConfig.py.

```
  def set(self, expid, variables, values):
    '''
    How to write server variables
    '''
    n = len(variables)
    for i in range(n):
      try:
        n, v = variables[i], values[i]
        if v in self._get_writables():
          self.n = v
      except:
        pass

  @property
  def seed(self):
    return self._seed

  @seed.setter
  def seed(self, value):
    random.seed(value)


  def get(self, expid, variables):
    '''
    How to read server variables
    '''
    toReturn = {}
```

```
    n = len(variables)
    for i in range(n):
      name = variables[i]
      if v in self._get_readables():
        toReturn[name] = random.rand
    return toReturn

  @property
  def random(self):
    return random.random()

  @random.setter
  def random(self, value):
    pass
```

Слика 4. RIPAdapterTemplate – Методе get и set.

Слика 4 приказује дефиницију метода set() и get(). Ове методе само проверавају да ли је одговарајућа променљива уписана (читљива) и уписује (чита) вредност. Користећи Python својства, објекти се мапирају на методе које даје пакет насумучно.

Коначно, метод `getValuesToNotify` враћа променљиве које ће периодично бити пријављиване кориснику (користећи SSE канал). Метод треба да врати листу где је први елемент листа која садржи називе верификованих променљивих, а други елемент је друга листа са одговарајућим вредностима.

```
  def preGetValuesToNotify(self):
    pass

  def getValuesToNotify(self):
    '''
    Variables to include in periodic SSE updates
    '''
    return [['time', 'random'], [self.sampler.lastTime(), self.random]]

  def postGetValuesToNotify(self):
    pass
```

Слика 5. RIPAdapterTemplate – Периодична ажурирања.

Опционо, можете заменити метод `preGetValuesToNotify` и `postGetValuesToNotify` у случају да треба да извршите неке радње пре и после читања променљивих, респективно. На пример, у имплементацији која користи MATLAB сесију, ви можете користити `preGetValuesToNotify` да покренете неки MATLAB код који ажурира радни простор, пре читања вредности променљивих.