26/10/2021

IO 6: TUTORIAL DE DISEÑO TÉCNICO E IMPLEMENTACIÓN DE UN WEBLAB – IMPLEMENTACIÓN DE SERVIDOR (PYTHON)

Co-funded by the European Union

1. INTRODUCCIÓN

Una interfaz de laboratorio remoto desarrollada con EjsS puede comunicarse fácilmente con el hardware / software del laboratorio utilizando el Protocolo de interoperabilidad remota (RIP). Para ello, se necesitan dos cosas: el elemento RIP de EjsS (consulte el manual **de implementación del cliente**) y una implementación del servidor RIP.

Este manual describe cómo utilizar el servidor RIP implementado en Python. Hay otro para la implementación de LabVIEW del servidor RIP (consulte el manual **de implementación del servidor (LabVIEW)**. La implementación de Python abordada en este documento es extremadamente útil cuando el equipo de laboratorio se controla con placas MATLAB/SIMULINK, Node.js, Arduino, Red Pitaya, etc., ya que presenta interfaces para todas estas soluciones. En el capítulo 2 de este manual se explica cómo configurar el servidor RIP y utilizarlo para publicar los programas de control como servicios web que pueden ser consumidos por las aplicaciones EjsS con el elemento RIP.

Pero primero, necesita saber dónde obtener la implementación de Python del servidor RIP. El software está disponible en https://github.com/UNEDLabs/rip-python-server. Para los usuarios más avanzados (aquellos que quieran hacer algún desarrollo), un documento que presenta la especificación del Protocolo de Interoperabilidad Remota se puede encontrar aquí: https://github.com/UNEDLabs/rip-spec.

2. USO DE LA IMPLEMENTACIÓN DE PYTHON DEL SERVIDOR RIP

La implementación de Python del servidor RIP se distribuye como código fuente. Puede obtener la versión más reciente en el repositorio de github (https://github.com/UNEDLabs/rip-python-server). Descárgalo como un archivo zip o, si tienes git instalado, clona el repositorio abriendo un terminal y escribiendo el siguiente comando:

https://github.com/UNEDLabs/rip-python-server de clonación git

Esta guía asume que ya tiene instalada una distribución de Python 3. De lo contrario, visite https://www.python.org/ y descargue una versión reciente.

Las siguientes secciones lo guiarán sobre cómo: 1) s et up un entorno virtual Python y verificar si su servidor rip-python está funcionando (2.1), 2) configurar su aplicación (2.2) y 3) extender el servidor para satisfacer sus necesidades específicas (2.3).

2.1. Configuración del entorno virtual

El rip-server-python depende de las bibliotecas de terceros *cherrypy*, *ujson*. Dependiendo de su aplicación, es posible que también necesite instalar *el motor oct2py* o *MATLAB* para Python. Puede usar la instalación de Python en todo el sistema, pero consideramos ventajoso usar un entorno virtual. Aunque hay varias herramientas que puede usar, esta guía usa *virtualenv*. Puede instalar la herramienta con el siguiente comando:

pip3 instalar virtualenv

A continuación, cree un nuevo entorno virtual:

virtualenv -p python3 venv

La opción -p python3 es para asegurarnos de que estamos usando la versión correcta, y venv es la carpeta donde se almacena el entorno. A partir de ahora recuerde que debe agregar el prefijo venv / bin / para ejecutar los comandos dentro del entorno virtual, y no en la instalación de su sistema. El siguiente paso es instalar las dependencias:

venv/bin/pip instalar cherrypy ujson

Dependiendo del módulo de implementación que desee utilizar para su aplicación (Matlab, Arduino, etc.), es posible que deba instalar otros módulos necesarios. Por ejemplo, para Arduino, también necesitaría ejecutar:

venv/bin/pip instalar pyserial serie

El código del proyecto está configurado para Matlab de forma predeterminada. Por lo tanto, lo siguiente no funcionará a menos que tenga instalado Matlab. Para cambiar el módulo de implementación a uno diferente (Arduino, por ejemplo), consulte la Sección 2.2.

Finalmente, puede iniciar el servidor RIP:

venv/bin/python3 App.py

El servidor RIP debe comenzar a escuchar en el puerto 8080. Para comprobar si realmente está funcionando, abra su navegador e ingrese la URL: http://localhost:8080/RIP. Debe obtener una respuesta JSON como se muestra en la Figura 1. La respuesta contiene los metadatos del servidor RIP que describen las capacidades del servidor y las experiencias alojadas en él. Puede leer la especificación RIP para obtener detalles de implementación.

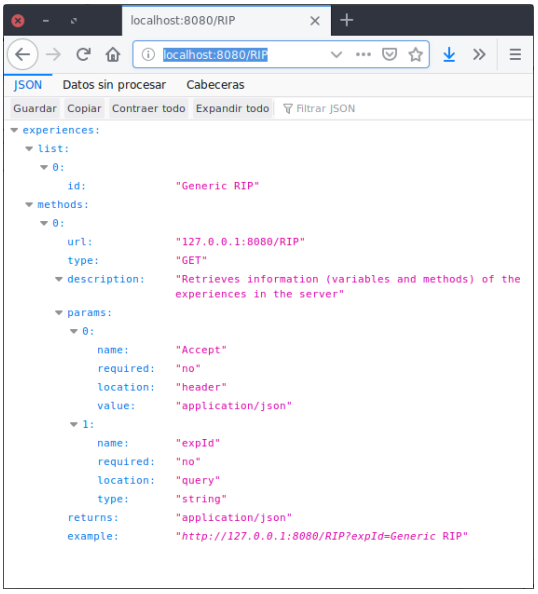

Figura 1. Probar la respuesta del servidor RIP.

2.2. Definición de la aplicación

El servidor rip-python se divide aproximadamente en dos niveles: el primero que implementa la funcionalidad RIP y el segundo que implementa el acceso al hardware. Como el nivel de comunicación no suele estar sujeto a cambios, para configurar el servidor solo es necesario enlazar la aplicación a una implementación de hardware específica en el archivo de configuración AppConfig.py, y opcionalmente proporcionar información adicional sobre la experiencia alojada en el servidor: variables *leíbles y escriturables expuestas*, autores, palabras clave, etc. Puede encontrar algunos ejemplos de cómo hacerlo en los ejemplos de configuración de folder. Las siguientes cifrascorresponden a una especificación básica para las pruebas:

```
# Este archivo contiene la configuración de la aplicación del servidor RIP.
config = {
 'servidor': {
  'host': '127.0.0.1',
  «puerto»: 8080,
 },
 'control': {
  «impl_module»: «RIPGeneric»,
  'info': {
   'nombre': 'RIP genérico',
   «descripción»: «Una aplicación genérica del PIR»,
   «autores»: «J. Chacón»,
```

```
    'palabras clave': 'Raspberry PI, RIP',
    'legibles': [{
      'nombre':'tiempo',
      'descripción':'Tiempo del servidor en segundos',
      'tipo':'flotar',
      'min':'0',
      'max':'Inf',
      'precisión':'0'
    }],
    'escriturables': []
  }
 }
}
```

Figura 2. Configuración de la experiencia del servidor RIP.

Debe especificar los siguientes campos:

1. config.control.impl_module: contiene el nombre del módulo que implementa el acceso de hardware. Puede utilizar cualquiera de las implementaciones integradas (RIPMatlab, RIPOctave, ...) o definir su propio módulo de control, como se explicará más adelante.
1. info: contiene la definición de la experiencia, incluyendo el nombre de la experiencia, una breve descripción, autores y palabras clave y la lista de *legibles*, elementos que se pueden leer, y *escriturables*, elementos que se pueden escribir.

Como antes, puede iniciar el servidor RIP para probar la nueva configuración, con el siguiente comando:

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

La aplicación escuchará el host y el puerto especificados en la configuración. Abra una nueva ventana del navegador y compruebe que el servidor está funcionando.

2.3. Creación de una nueva interfaz de hardware.

En caso de que los adaptadores de hardware integrados no se ajusten a su aplicación, puede escribir su propio código e integrarlo con el servidor RIP. Para ello tendrás que crear una fachada que exponga tu acceso de hardware de bajo nivel a través de la API RIP (consulta el documento de especificación RIP). Para facilitar su tarea, la clase RIPGeneric (definida en rip/RIPGeneric.py) proporciona la funcionalidad común de RIP, por lo que puede subclasificarla para agregar su código. Como mínimo, debe realizar las siguientes tareas:

2. Importe la clase RIPGeneric y cualquier otra biblioteca que necesite utilizar.
3. Cree una subclase que extienda RIPGeneric.
    a. Defina el constructor __init__() con el código de inicialización.
    b. Defina cómo leer y escribir objetos de servidor, anulando los métodos establecidos y obtenidos.
    c. Informe de las variables que deben enviarse en las actualizaciones periódicas, anulando el método getValuesToNotify.
4. La siguiente lista contiene un ejemplo de una implementación mínima proporcionada como plantilla (tenga en cuenta que debe colocar su código dentro de la carpeta 'rip'). El código crea s un servidor que proporciona s dos objetoslegibles: time, que devuelve

el tiempo de actividad del servidor, y random, que devuelve un número aleatorio, y uno escribible: seed, para modificar la semilla generadora aleatoria. El servidor aceptalas conexiones SSEe informa periódicamente de los números aleatorios a los clientes conectados.

```
importar aleatoriamente
de rip. RIPGeneric importación RIPGeneric

clase RIPAdapterTemplate(RIPGeneric):
 '''
 Plantilla de adaptador RIP
 '''

 def default_info(self):
  volver {
   'nombre':'RIPAdapterTemplate',
   'description':'Una plantilla para extender RIP Generic',
   'autores':'J. Chacón',
   'palabras clave':'Plantilla de adaptador',
   'legibles':[{
    'nombre':'tiempo',
    'descripción':'Tiempo del servidor en segundos',
    'tipo':'flotar',
    'min':'0',
    'max':'Inf',
    'precisión':'0',
   },

   {
    'nombre':'aleatorio',
    'descripción':'Generador de valores aleatorios',
    'tipo':'flotar',
    'min':'0',
    'máx.':'1',
    'precisión':'0'
   }],
   'escriturables': [{
    'nombre':'semilla',
    'descripción':'Semilla aleatoria',
    'tipo':'flotar',
    'min':'0',
    'máx.':'1',
    'precisión':'0'
   }],
  }
```

Figura 3. RIPAdapter Template - Definición.

El código de la Figura 3 es sencillo. Primero, importa el paquete estándar aleatorio que se utilizará para generar los números aleatorios y RIPGeneric, y luego crea una clase llamada RIPAdapterTemplate que extiende RIPGeneric. El método default_info proporciona la definición predeterminada de la experiencia, que se puede anular en AppConfig.py.

```
 def set(self, expid, variables, values):
  '''
```

```
  Cómo escribir variables de servidor
  '''
  n = len(variables)
  para i en el rango(n):
    probar:
      n, v = variables[i], valores[i]
      si v en self._get_writables():
        self.n = v
    exceptuar:
      pasar

@property
def semilla (self):
  self._seed de retorno

@seed.setter
def semilla (self, valor):
  random.seed(valor)

def get(self, expid, variables):
  '''
  Cómo leer variables de servidor
  '''
  toReturn = {}
  n = len(variables)
  para i en el rango(n):
    nombre = variables[i]
    si v en self._get_readables():
      toReturn[nombre] = random.rand
  volver aReturn

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

@random.setter
def random(self, valor):
  pasar
```

Figura 4. RIPAdapter Template – Los métodos se obtienen y se establecen.

La figura 4 muestra la definición del métodos set() y get(). Estos métodos simplemente comprueban si la variable correspondiente es grabable (legible) y escribe (lee) el valor. Usando *las propiedades* de Python, los objetos se asignan a los métodos proporcionados por el paquete aleatorio.

Finally, el método getValuesToNotify devuelve las variables que se informarán periódicamente al usuario (utilizando el canal SSE). El método debe devolver una lista donde el primer elemento es una lista que contiene el nombre de las variables notificadas y el segundo elemento es otra lista con los valores correspondientes.

```
def preGetValuesToNotify(self):
  pasar

def getValuesToNotify(self):
```

```
    '''
    Variables a incluir en las actualizaciones periódicas de la ESS
    '''
    return [['time', 'random'], [self.sampler.lastTime(), self.random]]

  def postGetValuesToNotify(self):
    pasar
```

Figura 5. RIPAdapter Template - Actualizaciones periódicas.

Opcionalmente, puede invalidar el método preGetValuesToNotify y postGetValuesToNotify en caso de que necesite ejecutar algunas acciones antes y después de leer las variables, respectivamente. Por ejemplo, en una implementación que utiliza una sesión de MATLAB, puede utilizar preGetValuesToNotify para ejecutar código de MATLAB que actualice el espacio de trabajo, antes de leer los valores de las variables.