CNT5517/CIS4930 Mobile Computing

Fall 2020

Professor Sumi Helal

Lab 3 – Build Microservices for your Smart Space and Use Atlas Thing Middleware and its IoT-DDL

Due Date: 11:55am Tuesday 26 October 2020.

Summary: In this assignment, you will be asked to design and develop C/C++ services to represent and trigger functionalities of the two sensor/actuator you have chosen in addition to the pushbutton and the LED (a total of 4 services). You will also import the Atlas Thing firmware on your Raspberry Pi 3, to turn your Raspberry Pi into an IoT thing that communicates with other things in the Virtual Smart Space (VSS) to advertise and offer its services. You will use the XML-based IoT device descriptive language (IoT-DDL) and its web-tool to describe your things and the services they offer to the VSS.

The same hardware, cabling, and software requirements you had in Lab 2 are required for this Lab assignment.

Part One — Design and Build Microservices

For the first part of this lab, you will use your four hardware components (let us call them bit-things) along with their online resources and datasheets to build the hardware setup and connections on your breadboard connected to your Raspberry Pi.

- Make sure to perform the essential hardware connections first before turning the Raspberry Pi on or connecting it to your laptop.
- Make sure to connect the VCC pin of the bit-things to either the Raspberry Pi 3.3V or 5V, according to the bit-thing datasheet. Mistakes in this step may burn out your bit-things.

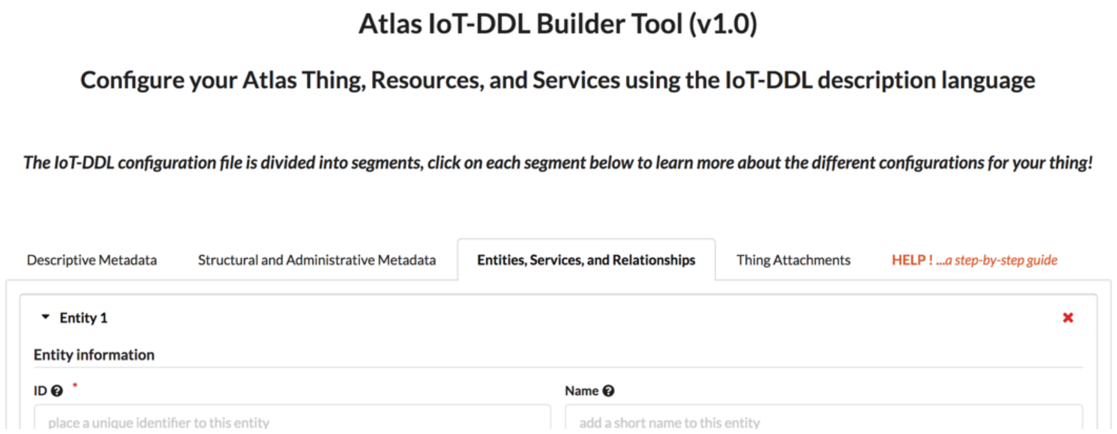
After this step, you will design and build 4 C/C++ meaningful and essential services that your bit-things can offer. As an example, if the bit-thing you have is a temperature sensor, the services could be: report the temperature in Celsius, and report the temperature in Fahrenheit. In this case, one bit-thing offers two services. For each service, design an appropriate API/signature (descriptive name, the required input(s) if any, along with the data type(s), and the expected output). You need to clearly comment the implemented services, and clearly cite the online articles/resources you used to implement this service, if any.

Finally test your services implementation by using a main program that calls the services.

Part Two — Describe and build your IoT-thing

In this part of the assignment you will use the Atlas thing middleware which is the key component of the Atlas framework project (https://github.com/AtlasFramework/Main). Atlas thing middleware is an operating layer embedded within an IoT thing that allows for the automatic integration of the IoT thing with the rest of the VSS smart space (with other IoT things). The middleware enables communication with other IoT things, users, and cloud platforms. The framework and its IoT Device Description Language (IoT-DDL) allow the users (e.g., device owners) to invoke and use the services provided by these things. They also allow IoT programmers to discover and combine available services to create IoT Applications for and in a smart space.

Step1: IoT-DDL is an XML-based language that describes a thing in a smart space in terms of inner components, identity, capabilities, resources, and services. Atlas thing middleware requires the use of IoT-DDL specification to configure the Atlas thing and enable it to perform a wide set of functionalities (e.g., self-discovery, announcing presence, dynamically create microservices, and enable users and other things to trigger such services through their respective APIs). In this step, you will use the IoT-DDL builder web-based tool (https://github.com/AtlasFramework/IoT-DDL) to build an IoT-DDL configuration file that describes your bit-things.



Atlas IoT-DDL Builder Tool (v1.0)

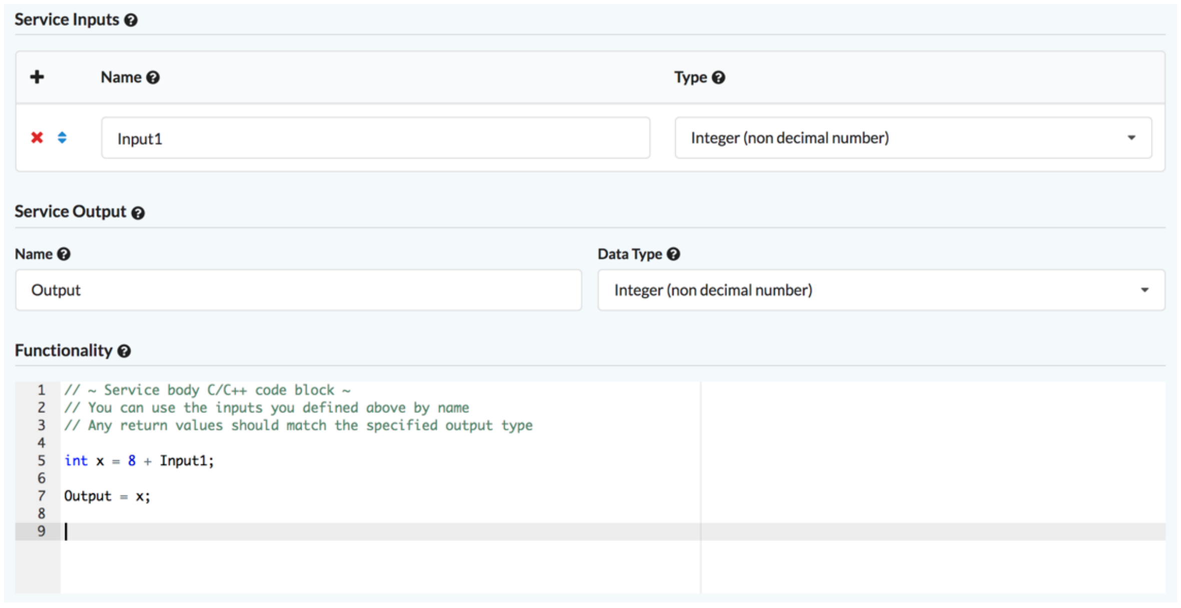
Configure your Atlas Thing, Resources, and Services using the IoT-DDL description language

The IoT-DDL configuration file is divided into segments, click on each segment below to learn more about the different configurations for your thing!

Descriptive Metadata
Structural and Administrative Metadata
Entities, Services, and Relationships
Thing Attachments
HELP ! ...a step-by-step guide
Entity 1
Entity information
ID
Name
place a unique identifier to this entity
add a short name to this entity

Consider the Raspberry Pi as a thing and the bit-thing(s) as its inner entities. Under "*Descriptive Metadata*" you can describe the main attributes and parameters of a thing. Under "*Entities, Services, and Relationships*" you can create inner entities and include the code of the services you designed in Part One provided by each entity. Create a separate entity for each bit thing (e.g., entity for each sensor and entity for each actuator).

For each service you create, you may specify the inputs expected for this service (the default is no inputs) as illustrated below, under "*Service Inputs*". Under "*service output*" you specify the expected output (the service returns void by default). Under "*Functionality*", you should develop the C/C++ code of this service (practically, you should bring and "reuse" the code you developed in Part I). As an example, you developed a service -as illustrated below- that takes single integer input (named Input1) and expected to return input value (named Output). In the functionality window, you can start to develop your C/C++ code, and use Input1 and Output within the code (no need to include the return statement, the middleware automatically includes this).


Service Inputs

Name	Type
Input1	Integer (non decimal number)

Service Output

Name: Output

Data Type: Integer (non decimal number)

Functionality

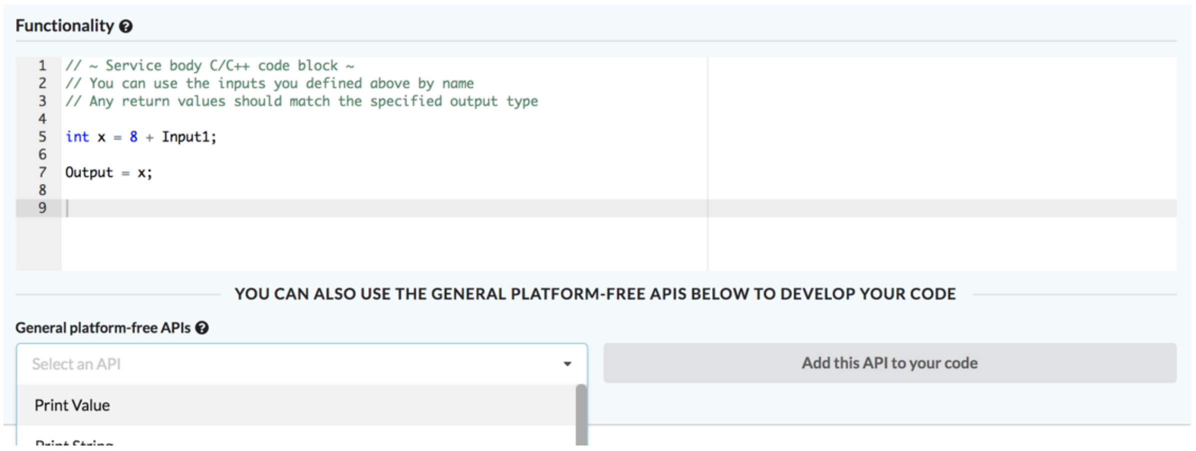
```
// ~ Service body C/C++ code block ~
// You can use the inputs you defined above by name
// Any return values should match the specified output type

int x = 8 + Input1;

Output = x;

```

The IoT-DDL builder tool also provides you with some ready, platform-free APIs and commands that you may use (select the API, fill the inputs, and click add this API to your code).


Functionality

```
// ~ Service body C/C++ code block ~
// You can use the inputs you defined above by name
// Any return values should match the specified output type

int x = 8 + Input1;

Output = x;

```

YOU CAN ALSO USE THE GENERAL PLATFORM-FREE APIS BELOW TO DEVELOP YOUR CODE

General platform-free APIs

At this point, you would have created your first IoT-DDL specification for all bit-things that you will attach to your Raspberry Pi thing. An example IoT-DDL for a Servo motor entity (attached to a RaspberryPi 3) offering a single service can be viewed using this link: https://github.com/AtlasFramework/IoT-DDL/blob/master/Resources/Servo_Motor.xml.

Step2: The current version of the middleware (V1.0) is available for Raspberry Pi platforms, thus in this step you will import the middleware to your Raspberry Pi and add the IoT-DDL file you developed in step 1 by following the instructions detailed in this link:
https://github.com/AtlasFramework/AtlasThingMiddleware_RPI

Important note: before you run the middleware, make sure that you placed the IoT-DDL in the appropriate directory inside the middleware, as detailed in the link above.

Step3: when you run the Atlas middleware (as mentioned in the last part of Step2), the middleware will parse the different parts of the IoT-DDL you included, generate microservice from the services you described, build JSON-based tweets (messages) summarizing the parameters of the IoT-DDL, and broadcast these tweets to the VSS smart space. Other things, users, and developers can capture these tweets and perform meaningful operators.

In this step, you will create a demo "IoT Application" in the VSS that simply invokes the services you described in the IoT-DDL through the appropriate API calls. Design and implement this demo IoT Application using the programming language you find appropriate (e.g., C/C++, Java) to unicast API calls (to the IP of your thing over port address 6668). The API call is a JSON-based message, structured as follows:

```
{ "Tweet Type"    : Service call,
  "Thing ID"      : the id of your smart thing as declared in the IoT-DDL,
  "Space ID"      : the id of your smart space as declared in the IoT-DDL,
  "Service Name"  : the name of the function you would like to call,
  "Service Inputs" : (list of expected inputs)};
```

Sending such call to the Atlas thing triggers the appropriate handles and the thing responds with JSON-based results back. Let us assume, in the IoT-DDL, you made *MySmartThing01* as your smart thing id, and *MySmartSpace* as your smart space id. You also developed two services, *Service1* that accepts no inputs and *Service2* that accepts two integer inputs. The structure of the API calls will be:

```
char Call1[] = "
{ \"Tweet Type\"   : \"Service call\",
  \"Thing ID\"     : \"MySmartThing01\",
  \"Space ID\"     : \"MySmartSpace\",
  \"Service Name\" : \"Service1\",
  \"Service Inputs\" : \"()\" }";

char Call2[] = "
{ \"Tweet Type\"   : \"Service call\",
  \"Thing ID\"     : \"MySmartThing01\",
  \"Space ID\"     : \"MySmartSpace\",
  \"Service Name\" : \"Service2\",
  \"Service Inputs\" : \"(3,9)\" }";
```

You should comment your code adequately to explain what demonstration it accomplishes.

The Virtual Smart Space

You will be able to call and test your thing services from your development machine as long as it is on the same network (direct Ethernet connection or WiFi other than the school network, as described in the previous lab document). However, we have created a "virtual smart space" (VSS) via an external VPN server that will allow you to interact with other student's thing services as well. We will use OpenVPN, so install this on your RPi with **`sudo apt install openvpn`**. Soon, you will receive **an OpenVPN configuration file** that will allow you to connect to the VSS. Put this file on your RPi, renaming it to **`vss.conf`** and moving it into the **`/etc/openvpn/client`** directory. To connect to the VSS, run the command **`sudo systemctl start openvpn-client@vss`**. After a second, you should see the VPN connection running successfully with **`sudo systemctl status openvpn-client@vss`**. If there was an error, ensure the configuration file has been named correctly and is in the right location.

You will now have an IP address within the VSS, likely in the `10.254.0.0/24` range. If you run `ip -br a`, you should see the IP under the `tap0` interface. Your device is now available to other student's devices, and you can call their services as well. To disconnect from the VSS, run **`sudo systemctl stop openvpn-client@vss`**.

NOTE: Please follow good IoT security practices and change the password (with `passwd`) for your RPi to something from the default, since it is now visible to other students.

Submission Details

You will submit a *.zip* file containing a folder for each part. For part one, you should include the C/C++ service implementations (one or more services for each one of your four bit-things). For part two, use a separate folder to include the IoT-DDL file you have generated using the builder tool. Also include the code for the demo IoT Application you have developed to use the services.