**CNT5517/CIS4930 Mobile Computing**

**Fall 2021**

**Professor Sumi Helal**

**Group Project – Atlas IoT Application IDE Development & Demonstration**

**Due Date:**

* **Part One:** 14 November 2021, 11:55pm
* **Part Two:** 30 November 2021, 12:00 noon

In this group project, using your Raspberry Pi, IoT-DDL builder, and the experience and knowledge you gained in Lab 3 and Lab 4, you will be developing and demonstrating an IoT Application IDE. Each **group of 4 students** will use its hardware. All members of a group should be engaged and putting forth an equal amount of effort. The final report must include a section dedicated to explaining who did what and what % of effort each has put into the project. There are two deadlines for this assignment, one for the initial design, and the other for the final completed project.

You will develop an IDE to create and manage Atlas IoT Applications hosted within your VSS. A simple way to know you succeeded in this assignment is to try to develop the same “demo” IoT Application in Part II of Lab4, only difference is redoing it through YOUR IDE and seeing if the IDE makes it easier for you to develop the same app.

The IDE should constantly listen to tweets within your VSS to quickly learn about the IoT in your space. This will require checking and processing the different types of tweets (e.g., thing identity, service, thing language, relationship, entity identity) paying attention to the structure of each tweet (as JSON key-value formatted messages).

The IDE should present what it learned –available things, offered services, and possible relationships, in an organized way that includes space meta information (space where the IDE is launched), and tabs for each of “Things”, “Services”, “Relationships”, and “Apps”. The **Things** tab includes info about each thing (the 5 or 4 RPi’s of the group), whereas the **Services** tab contains a filter to show services of all or a subset of the things (that is shows the union of services belonging to 5, 4, 3, 2 or 1 things). Each service presented under this tab should contain an identity of its owner thing. The **Relationships** tab should present (graphically if you are able to) all relationships also through a filter similar to that of the services. However, the design of this tab is more involved providing features for the developer to edit the relationships and to select which ones s/he intends to use in developing an application. The IoT-DDL builder allows you to define relationship(s) linking services belonging to the same thing defined in an IoT-DDL. It also allows you to define unbound services that could be bound by the developer at development time to other services belonging to other things. Under this tab, you should allow the user to bind an unbound relationship by linking it to an appropriate service available under the Services’ tab.

The **Recipe** tab is an app editor which allows the developer to drag and drop services (S) and relationships (R) into the Recipe tab to work with and to further shape them into an application according to the following simple grammar:

APP = {evaluate(R) | evaluate(S) | Cond\_Eval ;}+

Cond\_Eval = IF {evaluate(R) | evaluate (S)} THEN {evaluate (R) | evaluate(S)}

The Recipe tab should implement “clear” operation to clear the editor. It should also implement “finalize” which converts the recipe into an app that gets added in the Apps tab. The conversation from recipe to app amounts to taking an app specification and actually creating and deploying (even though not activating) an actual app. Code would need to be generated and deployed in this process. Finalizing also involves naming the app that is the finalization of the recipe. Once finalized successfully, the new app appears among other apps in the **Apps** tab.

A key feature to consider is images for things, services, and even apps. While IoT-DDL does not include images as metadata at the time being, you should allow the developer to add images to these elements as it will make the IDE easier to use.

You will also implement an **Application Manager** using appropriate gadget/widget. The manager is also invokable within the Apps tab. The manager implements the following basic functionalities:

* **Save** a developed IoT app into a local file (use the format you find appropriate: XML, Text, JSON) based on a set but changeable “working directory”. The App name should be short and contains no special characters. This feature should be accessible only when the IDE is in an “app development mode” For instance, if the developer is browsing services and recipes but has not started an app, there should be nothing to save, and hence the Save function should not be accessible or allowed.
* **Upload** a developed app into the IDE for further inspection, modifications, or in preparation to run the app. This feature should be allowed from inside the Apps tab and is invoked by double clicking the App on interest. When the IDE starts, it should tune to and scan the “working directory” to discover and list available Apps.
* **Activate** an app from the list of apps in the Apps tab by loading it if it is not already loaded. Activation starts an execution life cycle of the app and updates a status panel on the IDE showing all running apps, their start time, and their status (“active”). Clicking on any app in the status panel should pop-up a window showing a log of all the interactions involving this app so far.
* **Stop** is to end the activation of an app. It can be invoked from the status panel or the Apps tab. It simply stops the execution of the app by properly stopping all of its services, doing all necessary clean-up and garbage collection tasks, and reflecting the status in the app status panel showing “inactive”. An app can remain inactive for 5 minutes before it is completely removed from the status panel. Some apps, when activated, will execute and end (Stop) on their own, usually taking a short time in the execution. Those apps cannot be stopped on command, their status should show (“Completed”) in the status panel for a period of 5 minutes before they disappear.
* **Delete** removes the application from the IDE and from local storage and can be invoked only from the Apps tab by an appropriate means.

Use of **open source i**s permitted with compliance of necessary licenses, and clear documentation about what the open source provided and a justification for its use.

**Web-based or not**, you are free to make all choices for what environment, sandbox, or languages you should use to implement your IDE.

**Setting up** for development of your IDE is crucial. Obviously, you want to make sure all your RPi’s are fired up and running correctly before you launch your IDE under development. You may need to develop a small tool to test that this is indeed the case. Another feature you may consider is freezing the IDE as it starts until it hears enough tweets and sets its tabs up correctly. Allowing the developer to engage while these tabs are being “learned” and constructed is obviously problematic and hence going on **“Smart Space Scanning” progress bar** to implement the freeze may simplify matters.

The IDE needs to be **user friendly, easy to use**, and most importantly should only include features that work. Buggy features will likely reduce your grade, so you need to make sure you test and submit working features. Testing will be of paramount importance and the group should designate someone responsible for testing and reporting through Slack.

**Deliverables** **Details**

Part I: you will deliver a one-page design/progress plan by 11:55pm on Friday 12 November. You will receive a grade for this part so you need to show some progress, at least in your thinking and planning.

Part II: Each group should start a GitHub as a private repository (other groups should not be able to see your changes while you are working). Once completed (second deadline) the repository folder should be zipped and uploaded to Canvas. It should remain private but after the deadline, access should be given to Prof Helal, Aashish, and Prof Helal’s volunteers: Dr. Ahmed Khaled and Mr. Wyatt Lindquist.

The zipped upload should also include a detailed report in Word that clarifies the code as much as needed, description and justification of any third-party open source used, screen shots of the IDE and its dialogue boxes, and anything else your group feels keen to report on part one.