

COP 5615: Distributed Operating Systems Principles
Internet of Things Support in Xinu
Fall 2016
Term Project Report

Group 10

Mukesh Prasad, Saugat Chetry, Himanshu Vyas, Chanikya Mohan,
Reena Paranjape

mukesh.prasad@ufl.edu, saugatpchetry@ufl.edu, himanshuvyas@ufl.edu, konyala@ufl.edu,
rparanjape@ufl.edu

0. Important Links

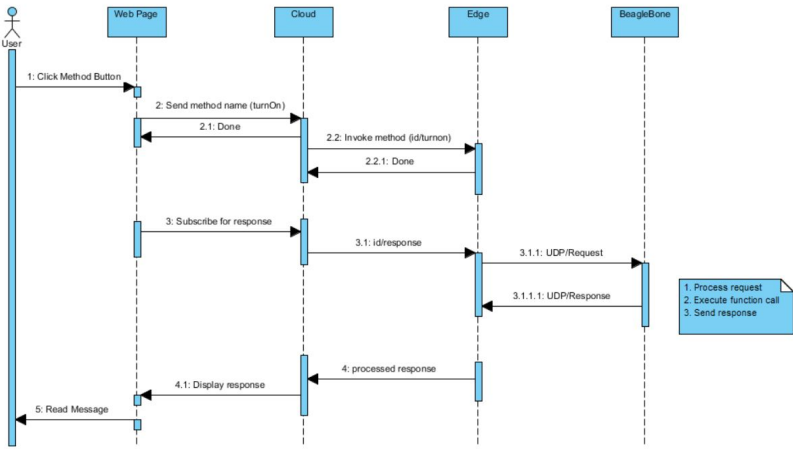
Description	URLs
Final Cloud Application	http://xinu-mukki.rhcloud.com/
Git Repository(will be made public after final presentation)	https://github.com/prime-optimus/xinu-drivers
Android apk	https://goo.gl/eUSdq9

1. Describe your project using this table

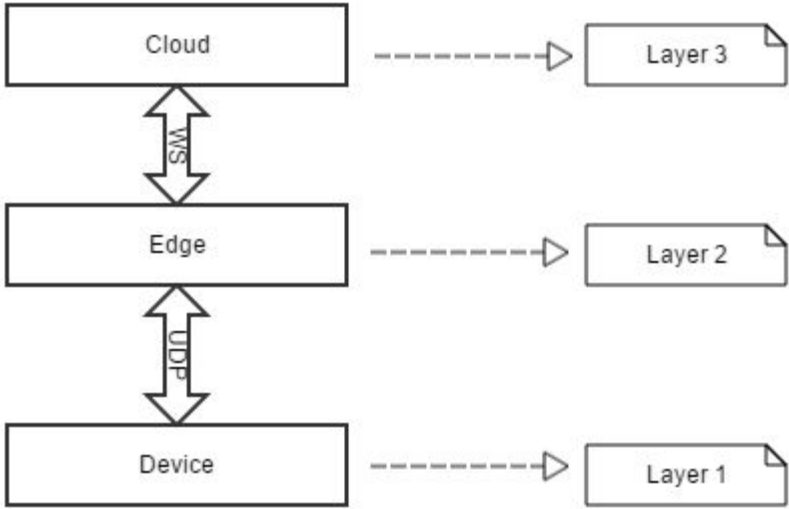
Part	Indicate Completeness (give a no. from 1-10), followed by Description
Xinu I/O Interface design	<p>Completeness - 10</p> <p>In our implementation we have divided the I/O interface into two parts, as described below,</p> <ol style="list-style-type: none">1. Low Level I/O Interface: In the low level I/O interface we are reading/writing data from/to the GPIO pins. We

	<p>have implemented methods that can directly access the values from the registers mapped to the GPIO pins.</p> <p>2. High Level I/O Interface: In the high level I/O interface we have implemented sensor specific methods and function calls. These methods are obtained from the sensor DDL file. For example, for a LED sensor we have methods such as “turnOn()”, “turnOff()” and “isOn()”.</p>
IoT-specific concerns your design addressed, including but not limited to Energy	<p>Completeness :- 10</p> <p>The concern we addressed was that of connecting various sensors from different vendors (using DDL) whose readings come from the physical world to the BBB and then controlling them via the cloud application that we developed. This overcomes the critical challenge of controlling the IOT environment from any location.</p> <p>The other concern, which we have addressed, is reducing latency of I/O operations by accessing (or setting) the values read by (or sent to) the sensors directly through the memory mapped to the pins. This make the I/O operation way more efficient and fast.</p> <p>The third concern we have addressed is the power and appropriateness of the I/O interface. We have used minimal synchronous UDP datagrams to communicate between the BBB and the edge. This helps our application to handle each request from the end-user appropriately, process the request and send the response. Furthermore UDP is connectionless protocol so no state is maintained which results in an energy efficient architecture.</p>
Xinu I/O Interface implementation and testing	<p>Completeness :- 10</p> <p>We created two files pindef.h and pinutils.c.</p> <p>pindef.h contains constants for various GPIO register addresses.</p> <p>pinutils.c has methods to read/write a GPIO pin. The generated C code from DDL uses these methods from pinutils.c to read or write from GPIO pins.</p> <p>For testing we have created separate shell commands which reads/writes on GPIO pins. For e.g. we created a command “led on” which turns on led and “led off” which turns off led attached to particular GPIO pins.</p>
Design of IoT Description Language, Language	<p>Completeness :- 10</p> <p>Indicate: XML</p> <p>Source: XSLT</p>

processing and code generation	<p>Design:</p> <p>A DDL file is obtained from the vendor for every sensor we connect to the BBB. This DDL is XML based. We used a high level language called XSLT which was used for transforming this DDL document into a C file. This C file contained the high level device driver code which was auto-generated by transforming DDL of that sensor as specified in their respective xsl.</p>
Implementation and testing of IoT Description Language, Language processing and code generation	<p>Completeness :- 10</p> <p>For implementing, we created two files called led.xml and tmp.xml for the led and temperature sensors respectively. led.xsl and tmp.xsl files contain the code for auto generation of the device driver files which use the xml files. These files names led.c and tmp.c are generated at compile time in the folder "system" in Xinu code repository.</p>
Implementation and testing of overall on-board driver code (upper- and lower-level drivers, including generated code)	<p>Completeness :- 10</p> <p>We implemented 3 methods namely readDigitalValue(pin) , setDigitalPin(val,pin), readAnalogVal(pin), setAnalogVal(val,pin). These methods are generic and can be used to read/write values from/to any type of pin. The pin value is abstracted from the DDL while parsing. The generated c files have functions abstracted based on DDL will be using the above methods to perform I/O operations.</p> <p>For testing, we used the LED and temperature sensors connected to the BBB, running on Xinu. We also used multiple BBBs (upto 5) with the above mentioned sensors connected to each of them. We used Xinu shell as a testing interface. We created various Xinu shell commands for testing different aspects of our driver code. For example, turning on a LED and reading temperature from temperature sensor by creating commands "led on","tmp" respectively for them.</p>
Did you use the same existing device driver structure and mechanisms in Xinu?	<p>Completeness :- 10</p> <p>No, the device drivers were implemented from scratch.</p>
Approximate % driver code generated with respect to overall	<p>Completeness :- 10</p> <p>100%.</p>

on-board driver code	
<p>Which device externalization abstraction have you chosen (which existing technology or any new ideas)? You may, or may not explain the reason for your choice.</p>	<p>Completeness :- 10</p> <p>We have used Service Oriented Device Architecture (SODA) to expose all the sensors that are attached to a device as a service to the outer world. All the sensors expose themselves as websocket endpoint via edge server.</p> <p>We're using existing WebSocket design implemented in Java by Spring Boot APIs. We are using WebSockets because they are Full-Duplex over HTTP and are very light weight. Spring has a Message Broker System implemented totally on WebSockets which made our implementation really time and energy efficient.</p>  <pre> sequenceDiagram participant User participant WP as Web Page participant Cloud participant Edge participant BB as BeagleBone User->>WP: 1: Click Method Button activate WP WP->>Cloud: 2: Send method name (turnOn) deactivate WP activate Cloud Cloud->>Edge: 2.2: Invoke method (id/turnon) deactivate Cloud activate Edge Edge->>BB: 3.1.1: UDP/Request deactivate Edge activate BB BB->>Edge: 3.1.1.1: UDP/Response deactivate BB Edge->>Cloud: 4: processed response deactivate Edge activate Cloud Cloud->>WP: 4.1: Display response deactivate Cloud activate WP WP->>User: 5: Read Message deactivate WP </pre>
<p>How, where, and when do you specify the edge and cloud addresses of the device? Explain how device configuration and initialization are done including device externalization.</p>	<p>Completeness :- 10</p> <p>Whenever the BBB wants to register to the cloud application, we run the “register” command with IP address of the Edge server passed as a parameter to this command. We also enter information of all the attached sensors as a second parameter. This will send a UDP request to the edge, which will be further, forwarded to the cloud server using a restful request. For example,</p> <pre>xsh\$ register 192.168.0.14 led,tmp</pre> <p>The device is abstracted in such a way that it has to know the edge ip only and nothing else. So device has no idea about Cloud server, only Edge server knows Cloud address.</p>
<p>Give the details of the externalization abstractions design.</p>	<p>Completeness :- 10</p>

	<p>As explained earlier our externalization is based on Spring Message Broker System, we have externalized all the sensors as Message endpoints so anybody can interact with them.</p> <p>For example all the messages sent on the endpoint <code>/edge/devices/{deviceId}/{sensorId}/call</code> are relayed to the appropriate sensor with the help of Edge Server and BeagleBone <code>startlistening.c</code> code. The endpoint <code>/edge/devices/{deviceId}/{sensorId}/response</code> is used to relay the sensor response back to whoever called the function, which must have subscribed to the same endpoint.</p>
Describe the implementation of the abstractions (how they connect to the actual device), and discuss any IoT-specific concern (including energy) that may have been addressed by your implementation.	<p>Completeness :- 10</p> <p>We have three levels of abstraction which are independent of each other:</p> <ol style="list-style-type: none"> 1. Device Abstraction: Device exposes a UDP based Interface from where the outer world connects to it. It is not limited to connection from Edge, but any system capable of sending/receiving UDP packets can communicate with it. 2. Edge Abstraction: Edge basically works as a relay server which relays the incoming WebSocket packets to UDP packets. This layer doesn't worry about origin or destination of those packets. 3. Cloud Abstraction: Cloud exposes a layer that is accessible from the web. Cloud just knows the websocket endpoints and publishes to those endpoints. Cloud is responsible for publishing and receiving the data and nothing else. <p>All these abstractions create a modular ecosystem where each module can be coded, updated and released independently without breaking the other, given they confine to same service call interface.</p>

	
<p>Describe your on-board IoT devices Demo App.</p>	<p>Completeness :- 10</p> <p>Devices: We have used basic devices only. However, our system is extensible to support virtual devices as well.</p> <p>App: Our application can be used to control the sensor attached to the BBB running on Xinu. The application is pretty simple and straightforward to use. It is dynamic and can accommodate any new device (as well as new sensors) on the fly.</p> <p>All the device needs to do is register itself to the edge server with all the sensors attached to it and it will be visible on the website, without reloading the page, with the help of Server Sent Event (SSE).</p>
<p>Describe your web-based IoT devices Demo App.</p>	<p>Completeness :- 10</p> <p>We created a HTML5 based Bootstrap application. We have handled dynamic interactions using jQuery, ajax, Server Sent Events(SSE). This way we can control all of our devices and sensors such as turning on an LED, getting the temperature of the environment (in both centigrade and Fahrenheit).</p> <p>The device must first register on the website (through the Edge). Once it's registered, all of its functionalities will be available on the website. The devices can also be unregistered after which they are no longer visible on the website.</p> <p>WebSite URL: http://xinu-mukki.rhcloud.com/</p>

	<p>On page load it displays the list of BBBs connected. The home page shows description about each BBB with parameters such as its id, name, list of sensors attached to it, each sensor type and id.</p> <p>The displaying of each BBB is dynamic</p> <ul style="list-style-type: none"> - On registering a BBB the details about it are displayed automatically in web page without refreshing the page. This is done by listening to Server Side Events (SSE). - On unregistering a single/all device(s) will remove respective BBB description panels from the UI. <p>Clicking on room description will pop open a modal which shows list of sensors in the room with methods that can be called on each sensor. For example: LED: turnOn, turnoff, isOn.</p> <p>Clicking on method button will make a call to cloud using STOMP Websocket Protocol with respective sensorID and type as data and subscribes for a response url on which it waits for response. We are using SockJS javascript library for WebSockets on front end.</p> <p>Upon getting the response from the subscribe url, it parses the response and displays the output accordingly on the UI.</p> <p>For Example, Calling <code>getTemperatureinFahrenheit()</code> will send call to <code>'/ws/cloud/'+deviceId+'/'+sensorID+'/call'</code> with object <code>{'methodName': 'getTemperatureinFahrenheit'}</code> and subscribes to <code>'/ws/cloud/'+deviceId+'/'+sensorID+'/response'</code>.</p> <p>On getting the response <pre>{ "methodName": "getTempratureInFahrenheit", "result": "83.1565" }</pre> It is parsed and displayed the temperature value on UI.</p> <p>The UI has additional debug mode.</p> <ul style="list-style-type: none"> - In normal mode the UI shows animations as response like a bulb icon glowing when LED is turned on, a semi-circular progress bar showing the value of temperature. - When we call the page with http://xinu-mukki.rhcloud.com?debug=true, the debug mode is turned on and instead of animations the respective results of each call are shown on the web page.
--	--

2. Challenges

Challenges your group faced. What was the most time consuming parts of the project? what piece(s) would you have really liked to have us provide to you so the total effort is more manageable (again, if any)?

The first challenge we faced was that the router we were using was not able to run the Xinu code. The DHCP was breaking on the router so we updated the DHCP handshake in Xinu.

Secondly, we wanted to print float values coming from our sensors. But we found that Xinu printf does not support float values. So, we implemented a function in Xinu that is able to print float values.

Thirdly, we were trying to use REST for two way communication but it was resulting in a lot of boilerplate code. Then we digged about various technologies such MQTT, COAP etc. but settled on Spring Message Broker based on WebSockets since it made our application full duplex and efficient via using standard Spring annotations only.

Another challenge which we faced was making an platform independent mobile application using phonegap. But due to shortage of time we were able to build the application only for android devices.

3. Overall Experience

The overall experience was good. We got to learn about various new technologies including xsltproc. Websockets and REST API. We also learnt a great deal from working on hardware and it's programming by writing device drivers including upper and lower level drivers.

4. Effort Distribution

Report only if effort was considered by any member of the group to not be even. In this case a table showing the names, ID's, and percentage of effort should be provided.