CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
ENGENHARIA DE COMPUTAÇÃO
Sistemas Distribuídos
Prof. Anolan Barrientos
2017-1

Trabalho Prático 1: Comunicação

Pedro Felipe Froes
Saulo Antunes

Trabalho prático realizado para a disciplina de Sistemas Distribuídos do curso de Engenharia de Computação do CEFET-MG, lecionada pela Prof. Anolan Barrientos.

1 Introdução

Um sistema distribuído será construído através de um jogo no sistema operacional Android, utilizando a IDE Android Studio para o desenvolvimento da aplicação.

Através de uma integração entre os dispositivos (*machine to machine*, M2M), variáveis como localização e luz ambiente podem ser compartilhadas. O protocolo *Message Queuing Telemetry Transport* (MQTT) pode ser utilizado para implementar um sistema de *publishers* e *subscribers*, com os dispositivos móveis Android publicando essas variáveis, que podem ser acessadas por usuários através de um cliente conectado a um servidor HTTP.

Para a obtenção das variáveis atreladas a cada dispositivo, pode-se utilizar a API do Google Play atrelada à localização e à luz ambiente, por exemplo. Essa API é responsável por gerenciar os serviços relacionados à variável de localização, disponibilizando métodos para o programador escolher quais mais se adequam à sua aplicação.

2 Desenvolvimento da aplicação

2.1 *Setup* inicial do projeto

Um novo projeto pode ser criado no Android Studio selecionando a opção *Empty project* ao iniciá-lo. O projeto criado possui as pastas `manifest` e `java`, que contem arquivos de metadados e de código do projeto, respectivamente, e uma seção de *scripts* do Gradle que auxiliam na compilação e criação do projeto.

Para acessar variáveis do dispositivo como localização e luz ambiente, é necessário importar a API do Google Play para o projeto, o que pode ser feito através do Android Studio na seção de `Tools/Android/SDK Manager`, incluindo a API no Gradle posteriormente. Foram importadas especificamente a API relacionada à localização e luz ambiente. Posteriormente, é necessário atualizar as permissões da aplicação para acessar a localização do dispositivo.

2.2 Coleta de variáveis do dispositivo

A localização do dispositivo pode ser obtida implementando os métodos da interface `LocationListener` na classe principal. É necessário inicializar um objeto do tipo `LocationManager`, que será responsável por chamar os métodos `getSystemService` e `requestLocationUpdates`. Esse último método possui quatro parâmetros principais:

- `provider`, que recebe o provedor da localização do dispositivo – nesse caso, o GPS do mesmo;
- `minTime`, o intervalo mínimo entre a atualização da localização;
- `minDistance`, a distância mínima entre a atualização da localização;
- `listener`, o método que implementa a interface `LocationListener` – nesse caso, a classe `MainActivity` da aplicação.

Já a luz ambiente pode ser obtida por meio dos objetos `Sensor` e `SensorManager`, sendo especificada no método `getDefaultSensor` com o parâmetro `TYPE_TEMPERATURE` sendo passado. Finalmente, tanto a localização e quanto a luz ambiente são atualizadas em tempo real por meio do métodos `onLocationChanged` e `onSensorChanged`.

2.3 Sistema *publish-subscribe*

Um sistema *publish-subscribe* é um padrão de troca de mensagens onde componentes enviam mensagens (os *publishers*) de um determinado tópico, enquanto outros componentes (os *subscribers*) recebem mensagens de um ou mais tópicos. Os *publishers* não precisam necessiariamente saber quais *subscribers* assinam suas mensagens; analogamente, os *subscribers* não precisam saber quais *publishers* publicam em certo tópico.

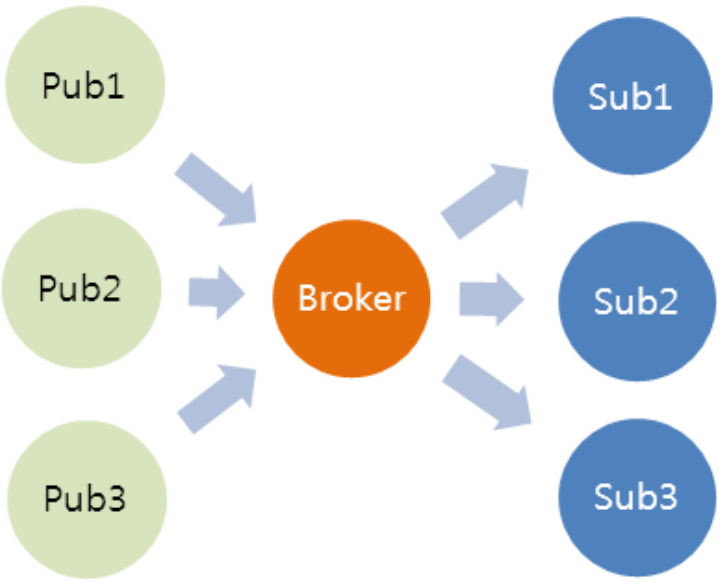

Figura 1 – Esquema de funcionamento de um sistema *publish-subscribe*.

Nesse trabalho, os *publishers* correspondem aos dispositivos móveis, que atualizam um cliente HTML com variáveis como localização e luz ambiente, por exemplo, em tempo real por meio de um Broker MQTT. O Broker é responsável por distribuir as variáveis para clientes interessados de acordo com o tópico da mensagem.

O Broker MQTT será implementado através de uma biblioteca *open-source* em JavaScript, a MQTT.js, cuja instalação dá-se por meio do Node Package Manager (NPM). A MQTT.js fará o papel do Broker, distribuindo as informações das variáveis para quem tiver assinado para receber.

2.4 Cliente HTML e envio das variáveis

O cliente HTML foi construído por meio de HTML, CSS e da biblioteca em JavaScript Mows, que é implementada em conjunto com a MQTT.js, criando um canal com o Broker através do protocolo Websocket na porta `8080`. Portanto, o cliente web utiliza o Websocket para se conectar no Broker, enquanto os dispositivos Android utilizam o protocolo MQTT. Esses dispositivos podem mandar suas variáveis na forma de mensagem através da URL `ws://iot.eclipse.org:80/ws` para a porta `1883`, enviando uma mensagem e o tópico a qual ela pertence, como mostrado no código abaixo:

```
public void sendMessage(String topic, String message) {
    try {
        MQTT mqtt = new MQTT();
        mqtt.setHost("iot.eclipse.org", 1883);
        BlockingConnection connection = mqtt.blockingConnection();
        connection.connect();
        connection.publish(topic, message.getBytes(), QoS.AT_LEAST_ONCE, false);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

No cliente, primeiramente o usuário deve-se conectar ao endereço do Broker MQTT no websocket. Após isso, ele pode escolher qual tópico deseja assinar, e as mensagens desse tópico chegarão em tempo real para o usuário. Também é possível que o usuário deixe de assinar um tópico se ele quiser.

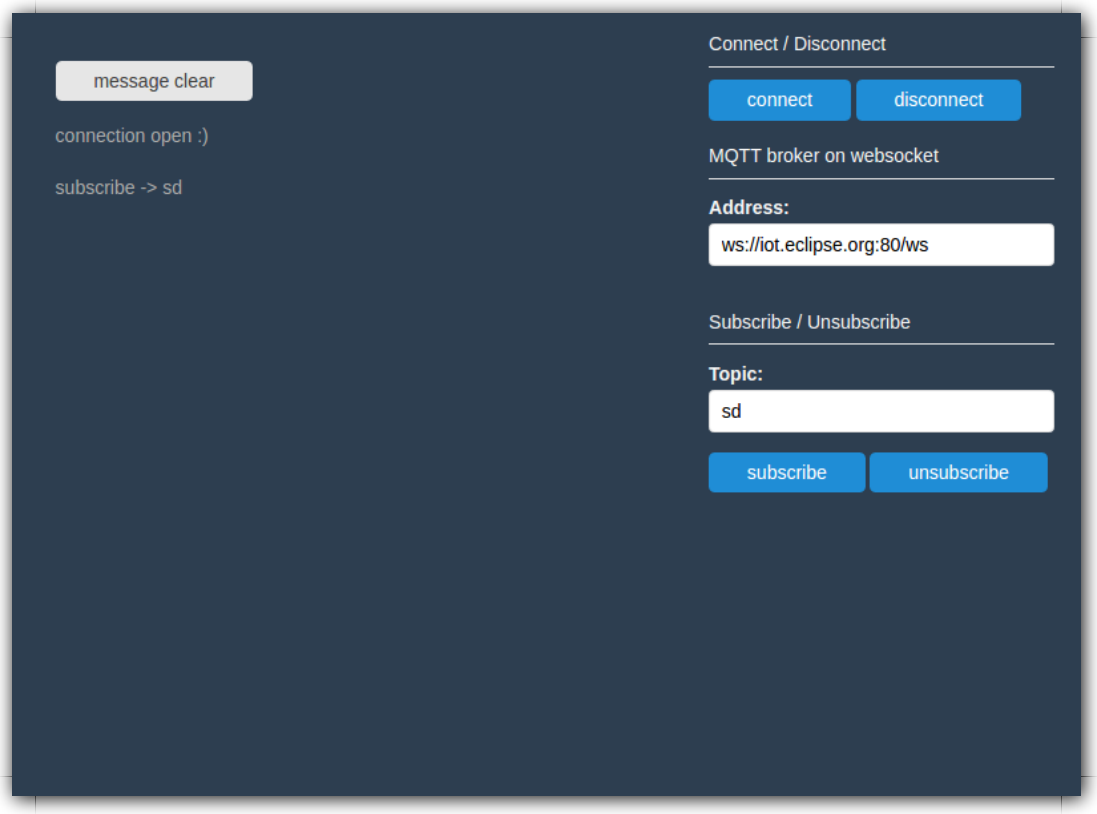

Figura 2 – Cliente HTML conectado ao Broker MQTT, porém sem nenhuma mensagem.

3 Resultados

Pode-se utilizar a aplicação através de um emulador de Android ou no próprio dispositivo. Utilizando em um emulador, deve-se utilizar um simulador de sensores para captar alguma leitura. Para utilizar em um dispositivo, basta abrir a aplicação que tanto a localização quanto a luz ambiente serão atualizadas em tempo real, enviando seus valores para o cliente HTML.

No cliente HTML, o tópico escolhido tem suas mensagens exibidas em tempo real. A medida que a localização é atualizada, o cliente é atualizado com a localização de um *publisher* conforme mostrado na Figura 3:

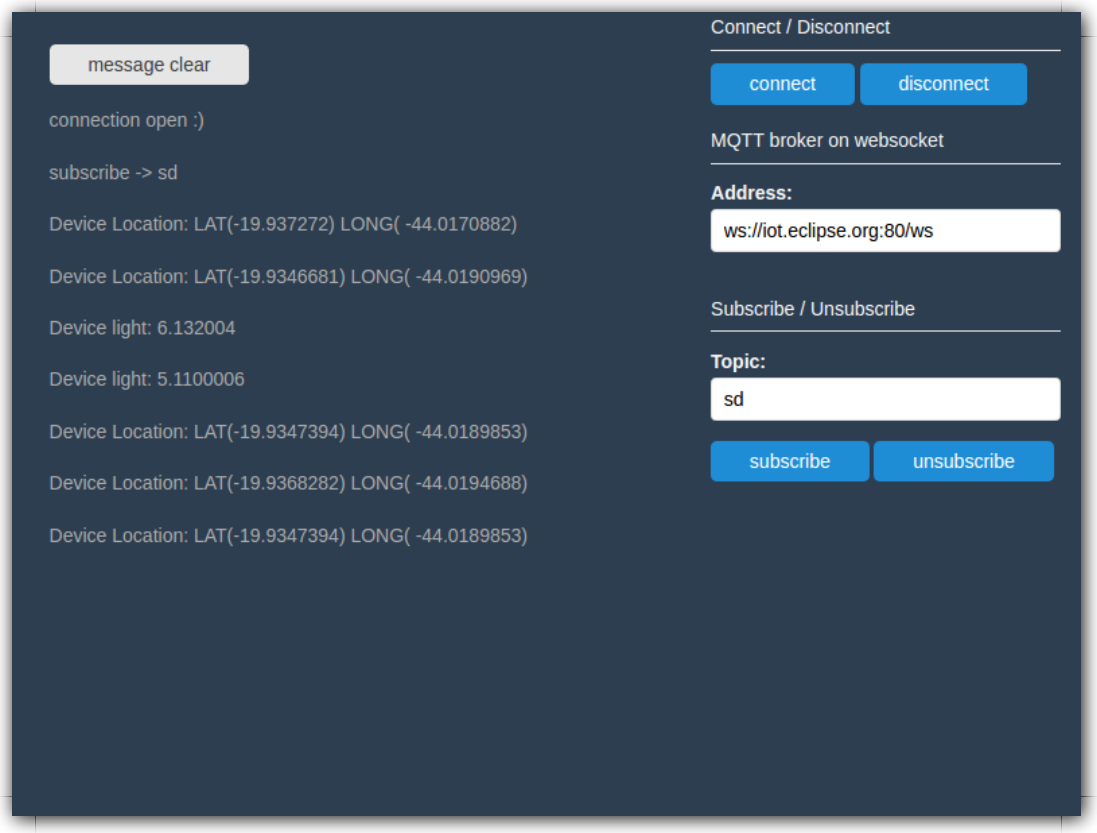

Figura 3 – Cliente conectado ao Broker recebendo atualizações no tópico `sd`.

Entre algumas dificuldades encontradas, está a disponibilidade de sensores no hardware de dispositivos Android. Por exemplo, alguns dispositivos não possuíam sensores de temperatura, pressão, etc. Não existe uma lista de quais hardwares possuem quais sensores, de modo que essa dificuldade foi superada através de experimentação (verificou-se se o sensor estava presente testando sua leitura na aplicação).

Referências

***Android Developers**: Getting the Last Known Location*. Disponível em: https://developer.android.com/training/location/retrieve-current.html. Acesso em: 20, Mar, 2017.

***Google API for Android**: Sensors Overview*. Disponível em: https://developer.android.com/guide/topics/sensors/sensors_overview.html. Acesso em: 15, Abr, 2017.

***Google API for Android**: Set Up Google Play Services*. Disponível em: https://developers.google.com/android/guides/setup. Acesso em: 20, Mar, 2017.

***TechLoveJump**: Android Tutorials. Android GPS – Location Manager Tutorial*. Disponível em: http://techlovejump.com/android-gps-location-manager-tutorial/. Acesso em: 20, Mar, 2017.