Report:
Algorithm Evaluation Platform–SimPOS

Yongcheng Huang

2016 年 6 月 26 日

1 Introduction

Our lab has developed a located system in Android with the technology of combination of WiFi, bluetooth and gyroscope. But with the demand of precision, we need the algorithm have better performance. Therefore, we need an algorithm evaluation platform to help us improve our algorithm. First, we apply this platform in the bluetooth algorithm. We know that,Since GPS does not work indoors, Bluetooth is a good alternative for indoor positioning and indoor navigation. Bluetooth beacons are able to send out signals, but they can' t receive them. They are relatively cheap, can run on button cells up to two years and have a maximum range of 30 meters indoors. Accuracy is up to one meter. On the one hand they are used in client based solutions, that is to say, positioning via app on the smartphone itself. In this case, Bluetooth must be activated on the device. On the other hand, server based tracking solutions using beacons are possible as well. For positioning in client based applications, several beacons are required. They send out unique signals with which the app determines the position by means of fingerprinting. Based on beacons, it is possible to trigger an action, for example displaying a coupon or information on the smartphone. Then we can use it to do location.

For our platform, we edit it through Java. There are 3 layers in it, which are database class, simulation class and algorithm class. This is the structure called MVP structure.Model - view - presenter (MVP) is a derivation of the model - view - controller (MVC) architectural pattern, and is used mostly for building user interfaces. In MVP the presenter assumes the functionality of the "middleman". In MVP, all presentation logic is pushed to the presenter.MVP is a user interface architectural pattern engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic:

The **model** is an interface defining the data to be displayed or otherwise acted upon in the user interface.

The **presenter** acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.

The **view** is a passive interface that displays data (the model) and routes user commands (events) to the presenter to act upon that data.

2 The Architecture Design Chart

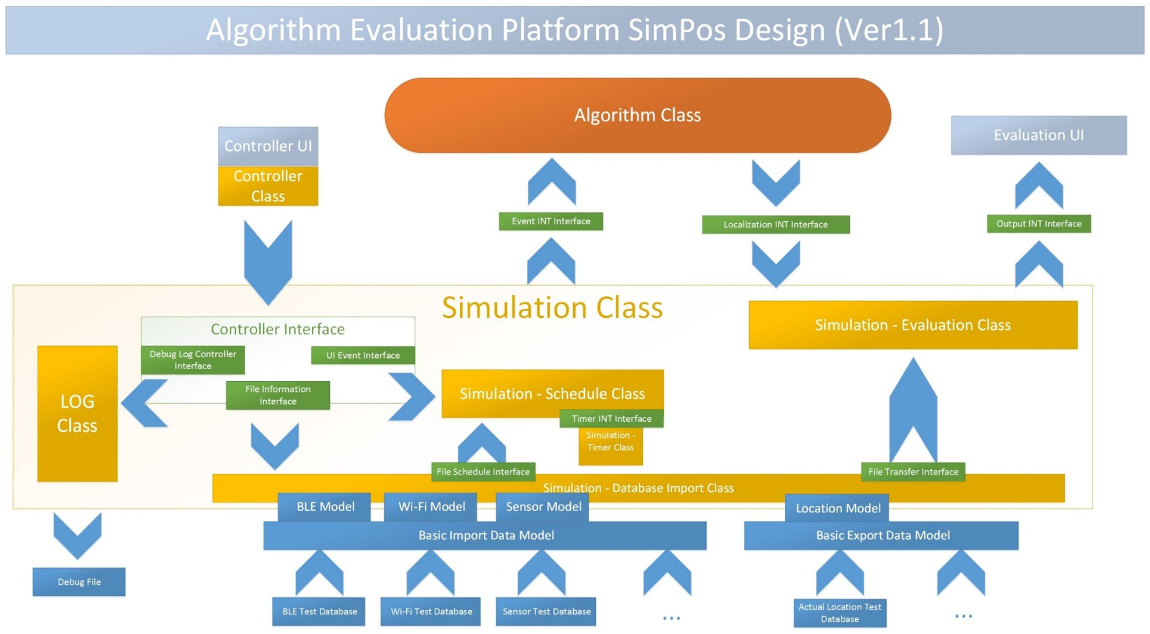


Figure 1: The Architecture Design Chart

2.1 Database class

When we set up this platform, we first store many models in it, like bluetooth model, wifi model and any other models. Then if we want to use one of them, we just need to call them. Furthermore, we leave many port for other models. It means that our platform have expansibility. What's more, in the database we also have space for the debug file and test file. All the models have to inherit the base class Model. But we can expand it by some abstract class. For example, in the bluetooth model, it has some properties that other models does not have like the bluetooth address and the RSSI. Then we will use the abstract to expand these properties. So that, in other class we can use the same control to use different models. Once we choose the model we use, then we can turn to the Simulation class.

2.2 Simulation class

When we choose the model we use, we will turn to the simulation class. We will turn the data we input into some type that our platform can accept. Moreover, the platform will convert it into the model we choose. And then insert it in the schedule. Then we will get the parameter from the Controller interface like the total simulation time and the interval of each simulation. Meanwhile, when we do such work, the platform will create a Log file for us to debug and test. Then we will send the schedule which has been become complete to the algorithm class and finish the last work, test the algorithm.

2.3 Algorithm class

The class is a java file that have to edit in the interior of platform. For example, if we want to use an algorithm of bluetooth location. We should edit it first in the platform, and then use the port to connect to the platform. Then it will work. When we send the schedule to the algorithm class. It will automatically calculate the result of location and show it in the debug windows.

3 Some mainly class

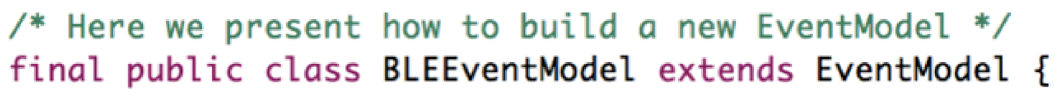
3.1 Schedule class

This class is to insert the event to the schedule in time. The insert mode is that, based on the type of events, it will create a subclass which is suitable for the type of the event. And then it will use the ParseEvent function of the subclass to analyse the information of the input file and then get the corresponding attribute.

3.2 Input data superclass–Model class

There is some rules for this platform.

First rule, every new input event have to inherit the base model. Like the graph 2, this is the inheriting of the Bluetooth low energy input event.

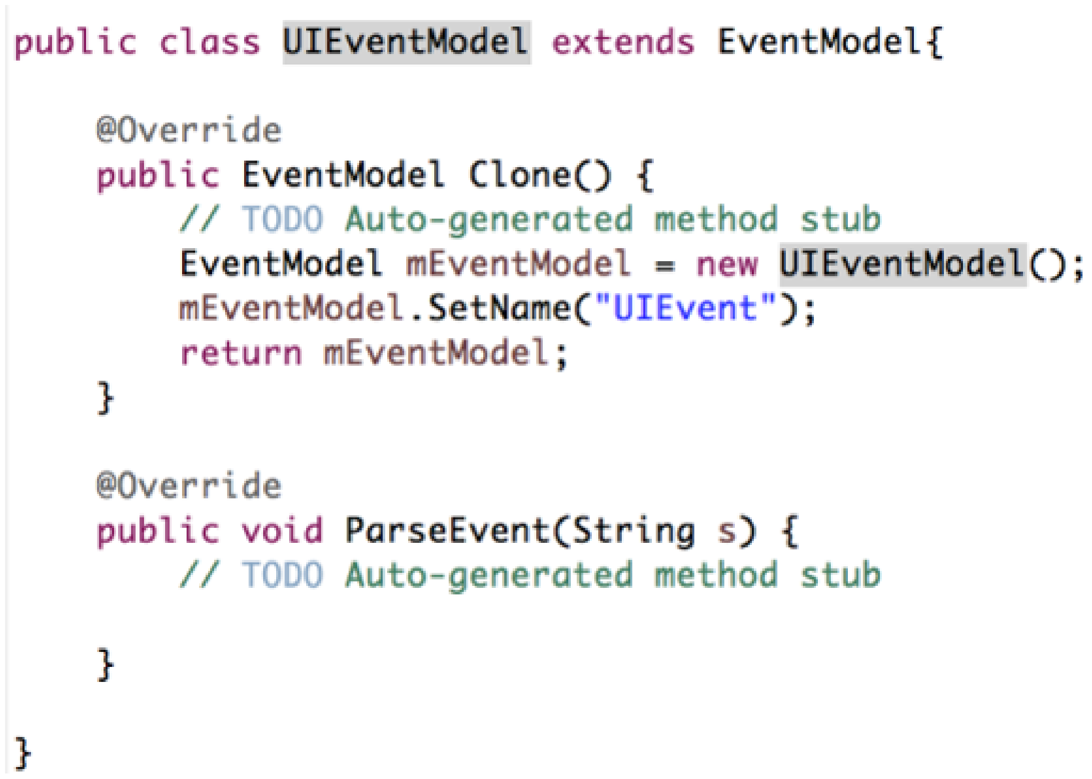


```
/* Here we present how to build a new EventModel */
final public class BLEEventModel extends EventModel {
```

Figure 2: Bluetooth low energy inheriting

Second rule, we need to rewrite the abstract class after inheriting because there are abstract classes in Eventmodel class. Like the graph 3, this the the function we should rewrite after

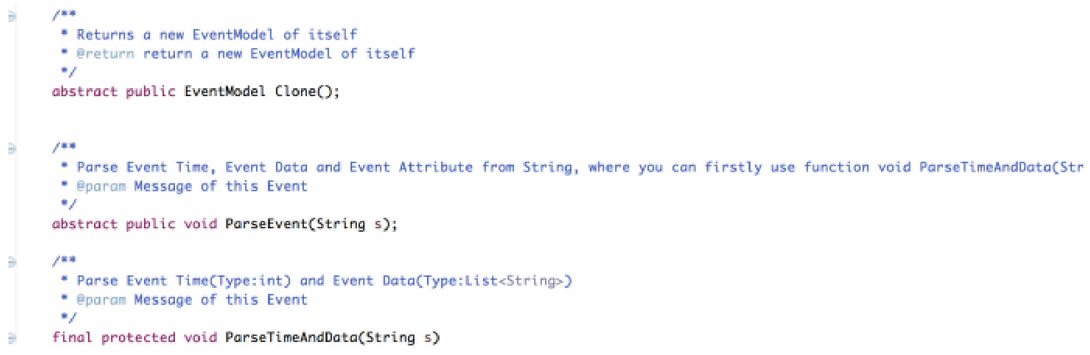
UIEventModel inheriting.



```
public class UIEventModel extends EventModel{

    @Override
    public EventModel Clone() {
        // TODO Auto-generated method stub
        EventModel mEventModel = new UIEventModel();
        mEventModel.SetName("UIEvent");
        return mEventModel;
    }

    @Override
    public void ParseEvent(String s) {
        // TODO Auto-generated method stub

    }

}
```

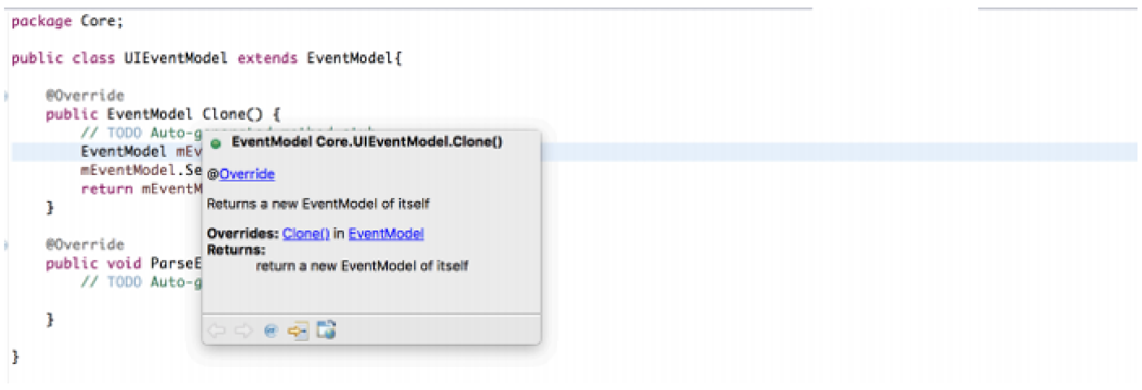
Figure 3: The rewrite of EventModel class

Third rule, we keep use a stable annotation pattern in favour of the readability of the code. graph 3 show us the basic annotation of EventModel and then we can create the tips of usage of function like graph 4.



```
/**
 * Returns a new EventModel of itself
 * @return return a new EventModel of itself
 */
abstract public EventModel Clone();


/**
 * Parse Event Time, Event Data and Event Attribute from String, where you can firstly use function void ParseTimeAndData(Str
 * @param Message of this Event
 */
abstract public void ParseEvent(String s);

/**
 * Parse Event Time(Type:int) and Event Data(Type:List<String>)
 * @param Message of this Event
 */
final protected void ParseTimeAndData(String s)
```

Figure 4: The annotation pattern of JAVA function



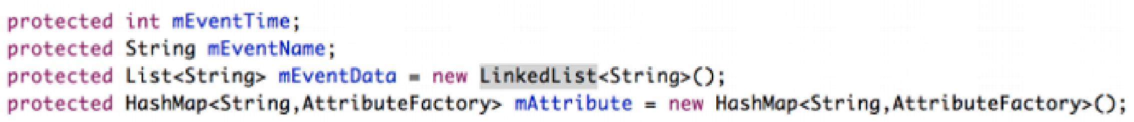
```
package Core;

public class UIEventModel extends EventModel{

    @Override
    public EventModel Clone() {
        // TODO Auto-g
        EventModel mEv
        mEventModel.Se
        return mEventM
    }

    @Override
    public void ParseE
        // TODO Auto-g

    }
}
```

```
EventModel Core.UIEventModel.Clone()

@Override

Returns a new EventModel of itself

Overrides: Clone() in EventModel
Returns:
        return a new EventModel of itself
```

Figure 5: The usage tips

3.2.1 The variable of basic model

For the basic event model, here are some basic model like graph 6.



```
protected int mEventTime;
protected String mEventName;
protected List<String> mEventData = new LinkedList<String>();
protected HashMap<String,AttributeFactory> mAttribute = new HashMap<String,AttributeFactory>();
```

Figure 6: The variables of eventmodel

The mEventTime is the virtual simulation time, mEventName is the event trigger name, mEventData is the data of event and HashMap type variable mAttribute will stroe the relevant variables based on the variable name and type.

If we send some input data:(mEventName + , + mEventTime + , + mEventData): BLEScan, 13657, 84:EB:18:58:A9:30,-87

After analysis, we will get the data table like table 1. mEventName, mEventTime, mEventData,

number	supervariable	subvariable	value
1	mEventName	/	BLEScan
2	mEventTime	/	13657
3	mEventData	/	84:EB:18:58:A9:30, -87
4	mAttribute	MAC Address	84:EB:18:58:A9:30
5		RSSI	-87

Table 1: variable data talbe

mAttribute is the basic variable of eventmodel. Because all event are inherited from the superclass, all subclass will have these four variable. However, the actual data will have BLE information, WiFi information and sensor information. Every data will have its own information. Like the BLEEventmodel, MAC address and RSSI is the subvariable of BLEEventmodel.

The subvariable will get the value through analysing the data of mEventData. And the key of subvariable will be got by the put function of hashmap.

3.3 Input event model manager class–EventManager class

This manager class is used for all the subclass which register and inherit from the superclass. Through instantiation of the eventmodel subclass, and use EventManager class to register, it can create the input time and send to the schedule based on the mEventName in EventModle. The graph 7 show the relevant process.

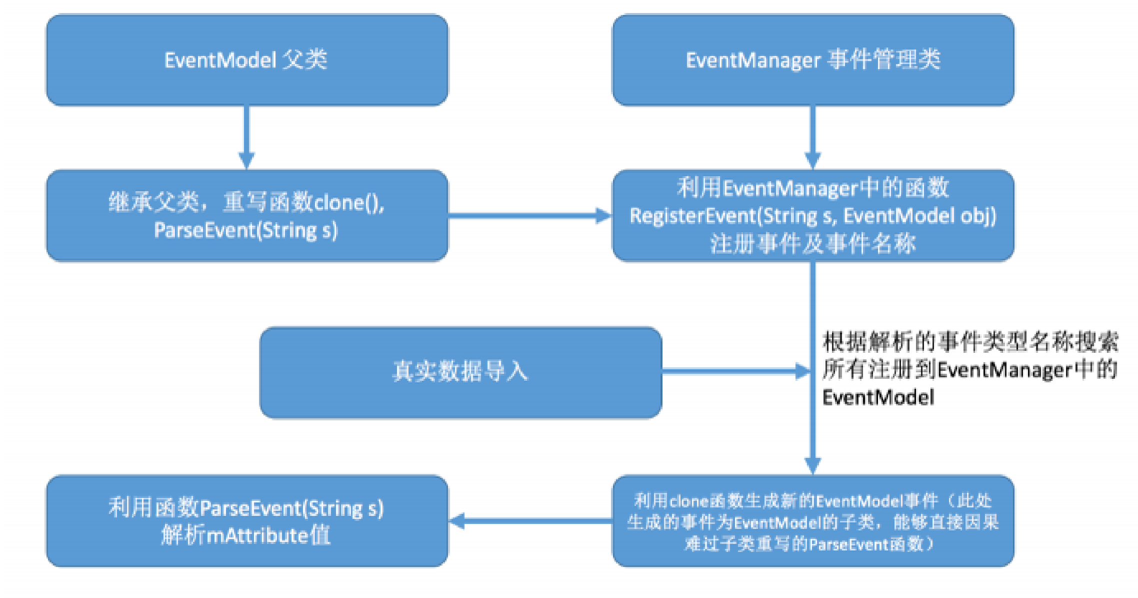
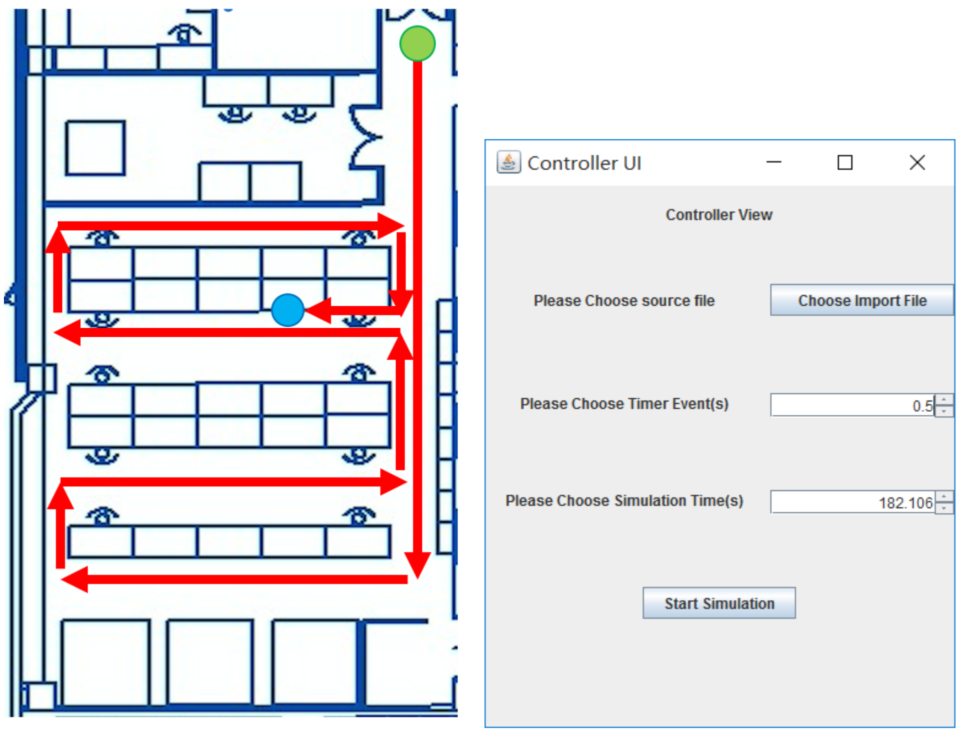

EventModel 父类
EventManager 事件管理类
继承父类，重写函数clone(), ParseEvent(String s)
利用EventManager中的函数 RegisterEvent(String s, EventModel obj) 注册事件及事件名称
真实数据导入
根据解析的事件类型名称搜索所有注册到EventManager中的EventModel
利用函数ParseEvent(String s) 解析mAttribute值
利用clone函数生成新的EventModel事件（此处生成的事件为EventModel的子类，能够直接因果难过子类重写的ParseEvent函数）

Figure 7: the process of EventManager

4 Simulation

We will input some data from the foxxcom. The data is from 10 bluetooth access point and the route is like graph 8.



Controller View

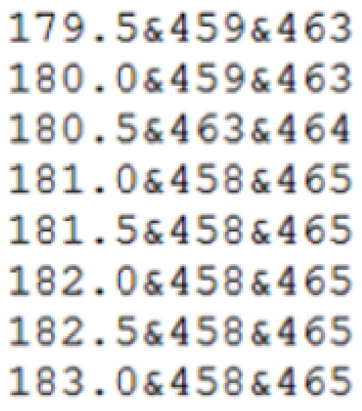
Please Choose source file

Please Choose Timer Event(s) 0.5

Please Choose Simulation Time(s) 182.106

Figure 8: test route

After simulation, we will get the result.



```
179.5&459&463
180.0&459&463
180.5&463&464
181.0&458&465
181.5&458&465
182.0&458&465
182.5&458&465
183.0&458&465
```

Figure 9: result

If we input the result into matlab and plot it. We will get the graph like graph 10.

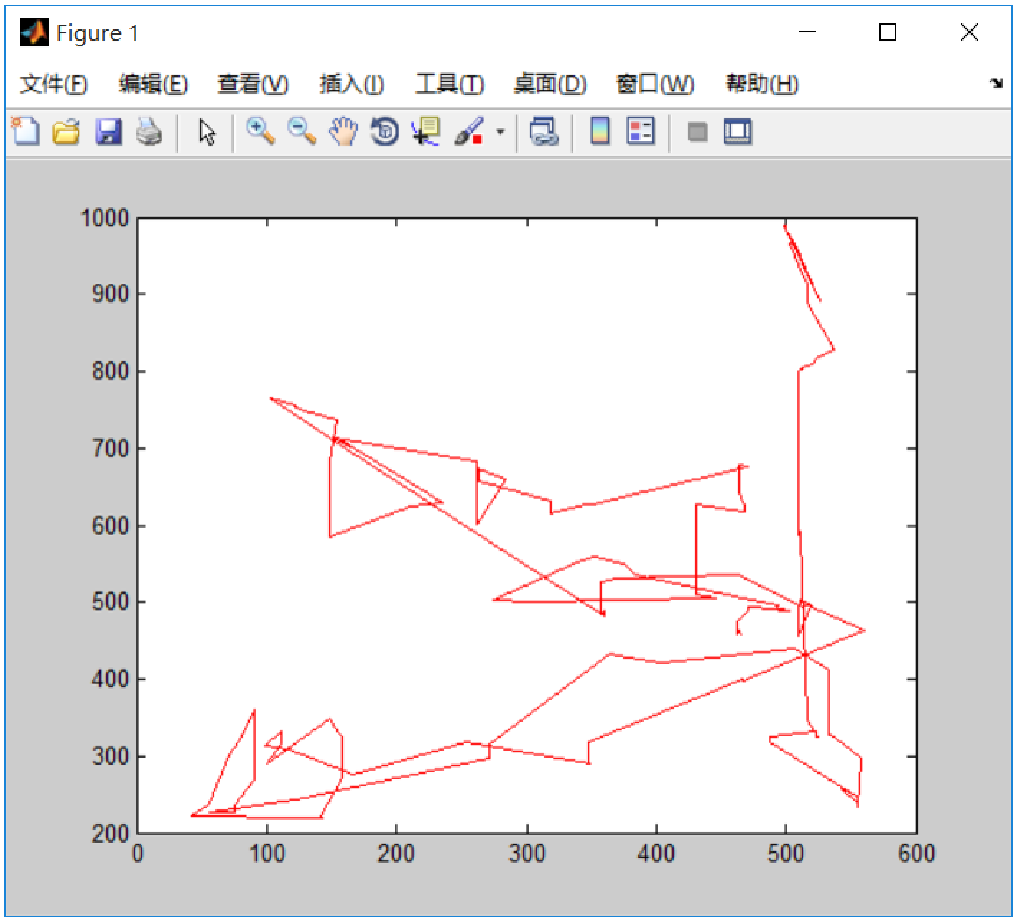


Figure 10: matlab plot

We can see that it's roughly correct but with errors.