Blackhawk Testing Application

BTech 450
End of First Semester Project Report

YuFeng Deng
Department of Computer Science
University of Auckland

Supervisors

Academic Mentor:
Dr S. Manoharan
University of Auckland
NewZealand

Industrial Mentor:
Andrew Radcliffe
Blackhawk Tracking System Ltd.
NewZealand

June 2008

Abstract

The main aim of this project is designing an application that can check whether the new produced Blackhawk units are working and update the databases accordingly. The application will also be able to generate a report of the testing result. It may be able to do some statistical analysis based on the report generated. This application will be developed using C# base on .net technology. It will be used by the factory tester.

Contents

Chapter 1

Introduction

This report is about the BTech project I am currently doing for Blackhawk tracking system Ltd Company. The project is about designing an application that can help the production staff to test the products efficiently.

1.1 The Company and products

Blackhawk Tracking Systems Ltd is a company providing tracking production for various types of vehicles. They currently have products for cars, trucks, motorbikes and so on. They are currently developing a new product for personal use. The Figure 1.1 shows a Blackhawk unit for car.

Figure 1.1: The Blackhawk Unit

The Blackhawk units are base on GPS technology. Each of them has a SIM car inside. Every Blackhawk unit is assigned a phone number, so that we can send text message to it. A Blackhawk unit can track any vehicles with a battery either by mobile phone or from the Blackhawk website. It can indicate the location of your vehicle at any time. Besides reporting geographic location of vehicles, the Blackhawk unit can do a lot of other cool things. For example, if your vehicle moves without the system being

deactivated, you will be notified by text message. You can even immobilise your vehicle by text message the next time it stops. These functionalities are very useful if the vehicle is stolen. Sounds amazing? Blackhawk units can also detect crash and over speed. If your children drive your car out, you can know where they are and what speed they are driving at.

Blackhawk unit uses internal aerials so thieves will not know there is a tracking system inside the vehicle.

1.2 How does Blackhawk work

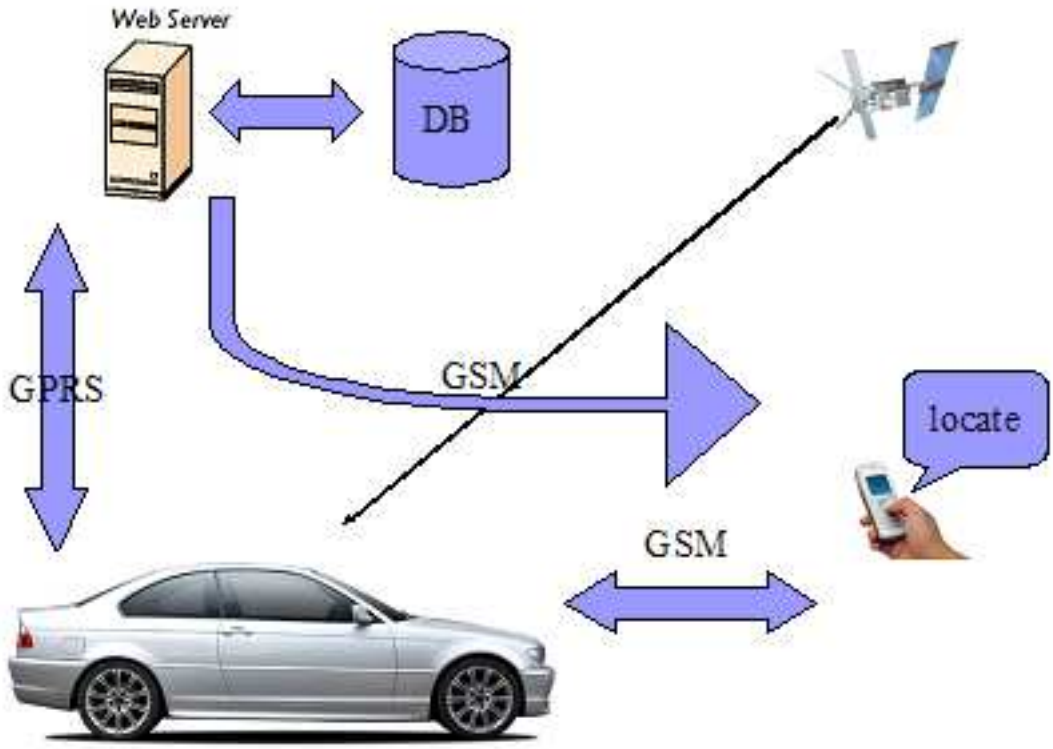

Figure 1.2: How does Blackhawk work

It is very important to understand how Blackhawk unit work before going to the project goal. I will not discuss how the Blackhawk's dual immobilisers

immobilise your vehicle or how it detects a crash internally. Instead of that, I will introduce how the Blackhawk unit is triggered, how it communicates with the web server and so on. Figure 1.2 shows a example of the Blackhawk working process.

We assume this car has a Blackhawk installed and there are several satellites, not only one. We need at least 4 satellites to get the accurate position. In this example, the car owner wants to know the location of his car. He sends a text message "locate" to the Blackhawk unit in his car. Once the Blackhawk unit gets the "locate" message, it will connect to the web server through GPRS. It will send an http request by HTTP get method. The query string will include the latitude and longitude that stands for the address of the Blackhawk unit. The query string can include other information such as vehicle speed and battery level. We call the information included in the query string as unit report. What information is included in the unit report will depend on what message you send to the Blackhawk unit or what other events trigger the Blackhawk unit. Once the server gets the request, it will search for the exact address of the Blackhawk unit in the database based on the latitude and longitude received. Then it will send back the information to the Blackhawk unit. The Blackhawk unit then can send the address to the owner. The server can send TXT message directly to the owner as well.

1.3 Motivation

After the new products are produced, we cannot take them out directly from the factory. Because it will bring troubles and cost money if a faulty Blackhawk unit enters the marketplace. To avoid selling faulty Blackhawk units, we need the production staff in the factory to check whether the new produce Blackhawk units are working properly. The checking process is based on the unit reports sent by the Blackhawk units. It's very inconvenient for production staff to check the reports manually. As we produce hundreds of Blackhawk units each time, it will take a lot of time for checking the products. The Blackhawk tracking system Ltd has to spend lots of money to the production staff for doing this. As the company growing, there will be more and more products needed in the future. At that time, it will be not possible for the production staff to check the products manually.

After checking, the faulty Blackhawk units will be returned to the factory for fixing. All the passed units will be sent to Blackhawk Company. The Blackhawk staff also needs to add these new products to the Black-

hawk database manually. Large amount of Blackhawk units will cost the Blackhawk staff a lot of time for doing this.

Due to these reasons, we need a tool to do the checking and update the Blackhawk database efficiently. So we will still be able to manage the production process in the future as the company grows.

1.4 Project Goal

The main goal of this project is to design an application that can assist the production staff to do Blackhawk unit checking more efficiently. Beyond this, we want a way to add the passed Blackhawk units to the Blackhawk database easily.

Chapter 2

The Blackhawk Testing Application

The Blackhawk testing application is designed to help the production staff to check the reports received from new produced Blackhawk units. It can tell whether the unit is good or bad. The checking will be based on the function 1 reports. The function 1 reports are sent by the Blackhawk units when they are powered on. The application will be able to add passed Blackhawk units to the Blackhawk database, generate reports of testing report and do some statistically analysis as well.

2.1 Introduction

The Blackhawk testing application is developed by C# using Visual Studio 2005. It will be used in the factory by the production staff. It will be able to access the factory database and the Blackhawk database outside the factory. Figure 2.1 shows the main structure of the application running environment.

2.1.1 Factory Database

The factory has a database for managing all new produced Blackhawk units. This database is factory's database and has nothing to do with Blackhawk Company. My application will connect to this database to get the basic information of the new Blackhawk units. Factory database is also inside the factory as the application, the application will treat it as local database. The Blackhawk unit information in the Factory database is different from the Blackhawk database.

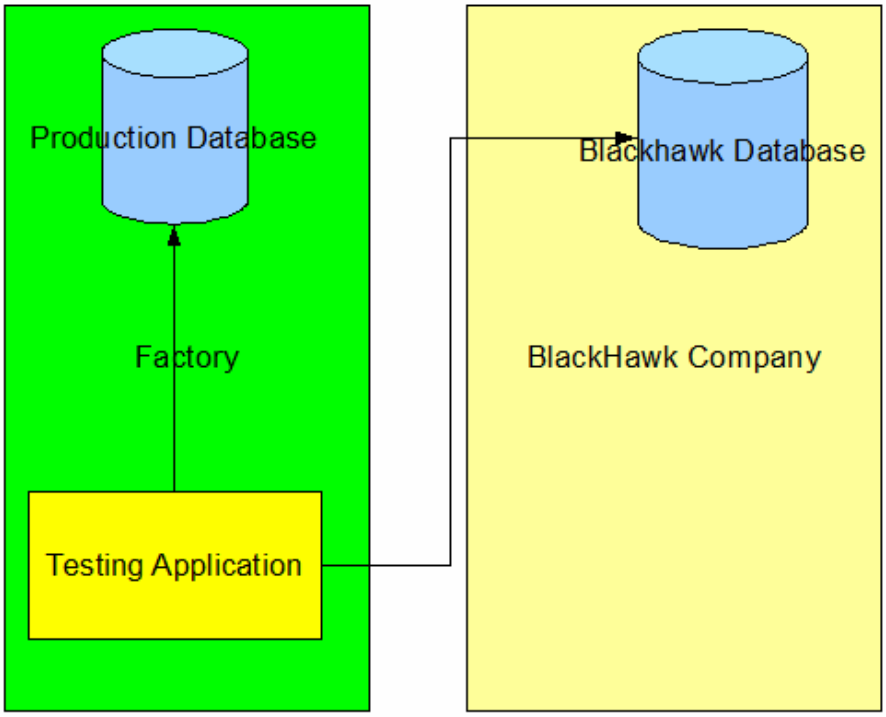

Figure 2.1: Main structure

2.1.2 Blackhawk Database

The Blackhawk database is the database from which the application can get the unit reports from. All the reports sent by the Blackhawk units will be recorded in this database. The Blackhawk database is outside the factory. The application will need internet connection to connect to the Blackhawk database.

2.2 How Can We Achieve It

The application can find out the faulty Blackhawk units base on the reports sent by the Blackhawk units. The application load the Blackhawk units from the database, then get the reports sent by these Blackhawk units using Blackhawk APIs and check the reports using the checking rules. If there are any problems with the reports, we can say the Blackhawk units that sent the reports are faulty. After checking, the application can add the passed

Blackhawk unit to the Blackhawk database using the Blackhawk APIs.

2.2.1 Blackhawk APIs

The Blackhawk Company has provided a bunch of APIs that I can use to operate the Blackhawk database. The Blackhawk APIs are based on web service. Here is a list of the APIs:

* BlackhawkAdminService
* BlackhawkCustomerService
* BlackhawkDeviceService
* BlackhawkEventService
* BlackhawkFactoryService
* BlackhawklnvoiceService
* BlackhawkLoginService
* BlackhawkReportService
* BlackhawkSMSService
* BlackhawkTrackService

BlackhawkLoginService

BlackhawkLoginServeice provides the Login, Logout and other methods. To use the Blackhawk APIs, the user have to login to the Blackhawk server within the Login method. Different users will have different access rights to the Blackhawk database. The security will be managed by the Blackhawk API Server, so that the application doesn't need to worry about it.

BlackhawkEventService

BlackhawkEventService contains the most important APIs for this application. Using these APIs, we can get all the Blackhawk events as we need. The word "Event" here is the same meaning as report. Each time the Blackhawk unit is triggered, we treat it as an event. Once an event happen, the

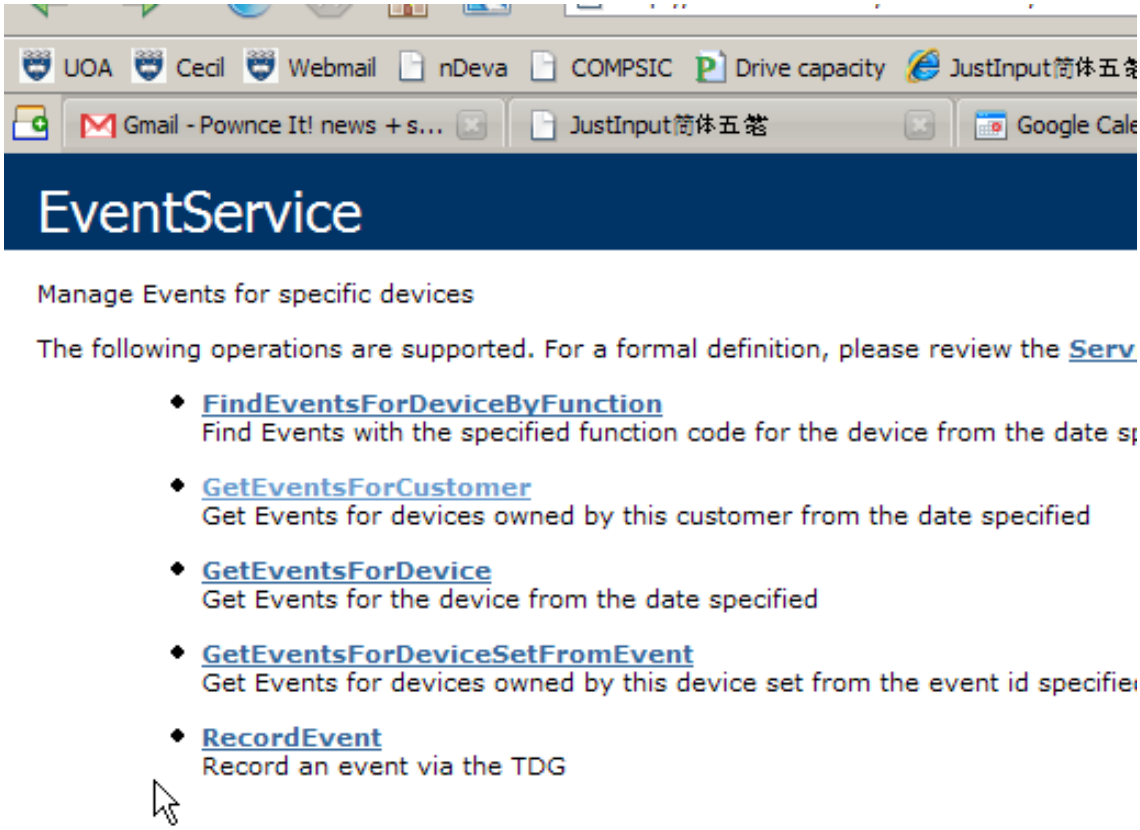

Figure 2.2: The BlackhawkEventService APIs

Blackhawk will send a report to the Blackhawk server. If we want to know what kind of event it is, we can have look at the event report.

Figure 2.2 shows all the web services I can use to get Blackhawk Events. The method FindEventsForDeviceByFunction() will be mainly used in this application. The powering on event will cause the Blackhawk unit to send a Function one report to the server. The application can use this method to get all the function one report for the devices. The function code of the report is made to indicate what kind of event is triggered. Once a Blackhawk unit is powered on, it will be automatically reset. This will cause the Blackhawk unit to set the function code to 1 and send the report including other information to the Blackhawk server. So we call this report as function one report.

Other Web services

Other web services are not used in the application at this stage. We may need to use them in the future. The company may add more requirements to the application to meet new conditions.

2.2.2 Unit Reports

Table 2.1 is a sample unit report I extracted from Blackhawk database. This is not a function one report. We can see that this is a "Blackhawk installed" report from the Table 2.1. I have no idea what the function code is, because they have mapped the function code to the event name.

FIX	3
ID	30243
Date Recorded	8/06/2008 19:18
SEN	120244
IMS	5.30011E+14
LAT	3656.8163S
LON	17450.2974E
ALT	32
SPD	0
COG	63
HRS	0
BAT	12.3
SSI	0
BAL	0
BER	0
SWV	703404016
FUN	Blackhawk installed
Latitude	-36.9469
Longitude	174.8383
Device	E10248
Address	11 Lippiatt Road, Otahuhu, Auckland City
Accuracy	0
Port	6672
Data	

Table 2.1: The Blackhawk Unit Report

The application doesn't need to check everything in the report, but most of them will be checked. Some fields of this report are easy to understand, for example, in this report the Address row gives the location of the Blackhawk unit which is 11 Lippiatt Road, Otahuhu, Auckland City. This is very obvious. But some fields are not that obvious, let us have a look. The FIX should be something new added recently which I don't really know. ID is the event ID. Date Recorded is the time at which this report is received. SEN is the serial number of the Blackhawk unit. IMS is the number belongs to the SIM card. LAT is the latitude of the Blackhawk location. LON is the Longitude of the Blackhawk location. If you look carefully, you will find another Latitude and Longitude in this report. They also stand for the location of the Blackhawk unit, but in different format. SPD is the speed of the vehicle. BAT is the battery voltage and so on.

The application will check this report using the checking rules.

2.2.3 Checking Rules

Checking rules are rules we made to check the unit reports. For example, The IMS number should be the same as the IMS number from the factory database, the address should be the factory address. If a unit sends a report which fails to pass any checking rule, we will treat it as a faulty unit. And it will be return to the factory for fixing.

2.3 The Application Achievement

2.3.1 The Software Requirement

To make the goal clearer, I have specified the requirements for the application.

* The application shall have a login panel that allows the user to login.
* The application shall have the ability to accept the input from the scanner.
* The application shall have the ability to read and update the factory database.
* The application shall have the ability to read the received units reports from the Blackhawk database.
* The application shall highlight the pass units with green color.

* The application shall highlight the faulty units with orange color.

* The application shall highlight the units whose reports cannot be detected with red color.

* The application shall be able to retest the units that are highlighted red.

* The application shall be able to add the new pass units to the Blackhawk database.

* The application shall be able to generate a report of the testing result.

* The Application should be able to do some statistical analysis of the results.

* The Application should be configurable so that can be used in different conditions.

2.3.2 The Checking Process

To make life easier, we want the application to do as much as it can. We don't want the production staff to waste any time on testing the Blackhawk units. Figure 2.3 shows how easy will be for the production staff with the help of the application.

Form the diagram we can see that the tester only needs to do two things. One: run the application. Two: Scan and power on all the units. All other things will be done by the application automatically. How does the tester do the job? The tester run the application and does some setting if needed. Then navigate to the testing panel. The tester then uses a scanner to scan the barcode of the Blackhawk unit to the application and put the Blackhawk unit one by one into a tray which can supply power. Then the last thing, simply click the "start" button in the application. Now the tester can sit down and have a cup of coffee waiting the application to give testing results. After 15 (can be configured to other value) minutes, the tester come back to the application, find out the faulty units if any and return them to the factory for fixing. All the passed units then will be packed by the tester and ready to be sent to the Blackhawk Company. When the Blackhawk Company receives these new products, all of them are ready for sell, because all the information of these new products has been updated to the Blackhawk database by the application.

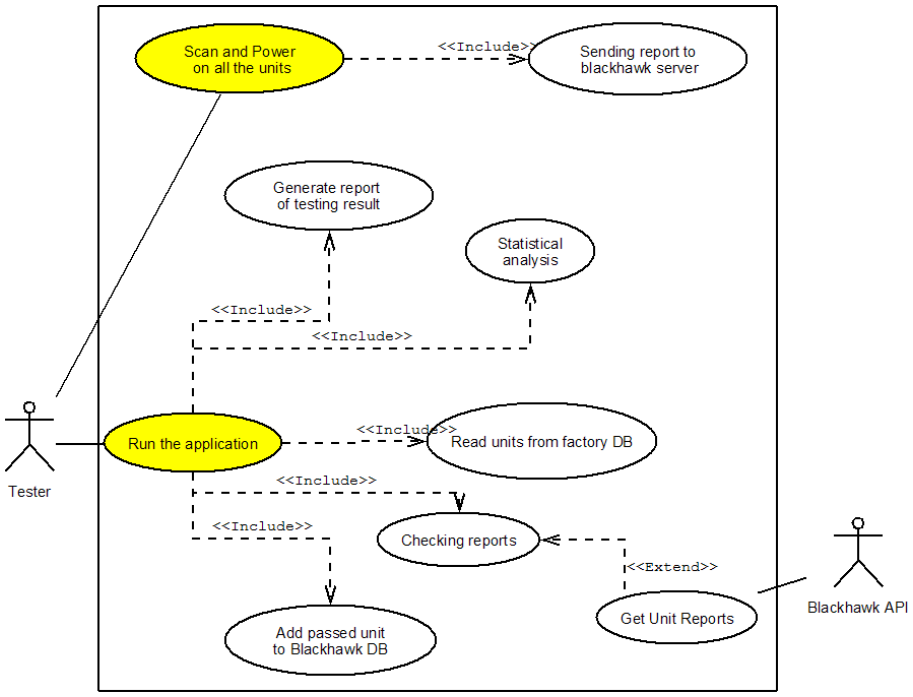

Figure 2.3: The Testing Process

You may be interested in what the application did in the background. Let me explain this diagram a little bit. When the tester scans the Blackhawk units, the application will load information of the Blackhawk units from the factory database. After clicking the "start" button, the application will get and check the units reports from the Blackhawk database every two minutes. The units that sent problem reports will be highlighted orange. The units that sent fine reports will be highlighted green. Other units whose reports cannot be received after time out will be highlighted red. Then, the application adds the passed units to the Blackhawk database. The application will need to generate report of testing result as well. We may upload the testing report to the Blackhawk database in the future. The application may be able to do some statistical analysis.

Why every two minutes

Once the tester put a Blackhawk unit in the tray and power on, the Blackhawk unit will be reset. Then it will send a function one report to the Blackhawk server. This process will take some time. Because the Black-

hawk unit has to warm up and gather all the information that needs to be sent, such as Latitude and Longitude. The latitude and longitude are got from the GPS satellites. If the GPS signal is too week, it take longer time for the Blackhawk unit to get the latitude and longitude. If there is not GPS signal, the testing cannot be done. The Blackhawk Company is planning to set up a GPS magnifier in the factory to avoid any GPS signal problem. Once all the information is gathered, the Blackhawk unit will connect to the Blackhawk server through GPRS. As GPRS is a kind of slow protocol, it also takes time to send the report to the Blackhawk server. Beyond that, we need to consider the GPRS signal as well. For this reason, the Blackhawk server will not get the units reports immediately after the units are powered on. Some reports may be received earlier; some reports may be received later. To make sure we can get all the sent reports, we set a polling interval which is two minutes. Also, we set a time out interval to avoid the application keeping polling Blackhawk APIs for getting reports forever. Both the polling interval and time out interval are configurable.

2.3.3 Confiuration

To make the application more flexible, we need to add the configurability to the application. For example, if we have a good GPS signal and GPRS signal, we may want to reduce the time out interval. The application does supply a setting panel for configuration. But we still need somewhere to save the change. I'm using a XML file to save the configuration.

```
<!file name: bta.exe.config-->
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<appSettings>
<add key="TimeOut" value="15" />
<add key="FactoryAddress" value="Auckland University" />
<add key= "Interval" value="2" />
<add key= "Lat_Min" value="0" />
<add key= "Lon_Min" value="0" />
..
</appSettings>
</configuration>
```

In this example, the timeout interval is set to 15 minutes. The factory address is set to Auckland University.

The advantage of using this format is that c# supports it very well. c# provides a ConfigReader class that can read this format easily. Although I have never tried to write the configuration programmatically, I believe there will be a way to do it. As this is a well format xml file, I can write my own ConfigWriter using XMLWriter if ConfigWriter is not provided.

It's possible to write the settings to a plain text ini file. But I may need to write a complicate ConfigReader and ConfigWriter by myself. Although the plain text file may be easier for human to read, I don't want to implement in this way.

The third way to save the settings is saving the setting into the windows registry. I didn't try to read and write windows registry before in a program. But I don't think that it's difficult. I know how to read and write windows registry using the windows command reg.exe. I believe that there is a c# library which can do similar thing as reg.exe. I chose not to use registry because I want to make the application portable. So it can be copied to any computer in the factory including the settings. An advantage of using the registry is that different user can use different settings if we save the settings in the CURRENT_USER node in the registry. But in our application, we don't really need personalised settings.

2.3.4 Generating Reports

For debugging purpose, the application should be able to generating reports for the checking result. Then the technician can use this report to debug the faulty Blackhawk units. Although the application can show the problem of the Blackhawk units, but the information shown on the application panel is not printable and not convenient to read directly from the monitor. Generated text format file will be easy to read and can be printed out easily.

It will be easy to write the result report in a plain text file. But the plain text file has no ability to highlighted text which we want to emphasis. For example, I want to highlight the error value in red, so the technician can found the problem more quickly. I chose to generate the report in XML format because I can easily convert the XML format to other human readable format such as html.

```
<!--report_1-1-2008.xml-->
<Testing Results>
<Unit>
<Number>1</Number>
<Serial No>123456</Serial No>
```

```
<IMS>123456789012345</IMS>
<Phone No>0211234567</Phone No>
<Status>Unknown</Status>
<Location>X</Location>
...
</Unit>
<Unit>
<Number>2</Number>
<Serial No>123456</Serial No>
<IMS>123456789012345</IMS>
<Phone No>0211234567</Phone No>
<Status>Faulty</Status>
<Faulty_Location>Sydney</Faulty_Location>
<Faulty_IMS>987654321012345</Faulty_IMS>
...
</Unit>
<Unit>
<Number>3</Number>
<Serial No>123456</Serial No>
<IMS>123456789012345</IMS>
<Phone No>0211234567</Phone No>
<Status>Pass</Status>
<Location>Auckland University</Location>
...
</Unit>
...
</Testing Results>
```

The faulty items will be recorded as an element with a name start with Faulty_ This XML file is not that straight forward for human. But it can be converted to other file format easily. Figure 2.4 is a example of HTML format.

In this html format, passed units are highlighted green, faulty units are highlighted yellow. Other units whose report cannot be received are highlighted red. The faulty values are highlighted pink and displayed below the yellow lines. For example, the third row has an IMS number which is 123456789012345, but the unit report has a different value which is 987654321012345. The faulty IMS is displayed below line 3 and has a pink color.

To convert XML format to HTML format, I can either use a XSLT

Serial No	IMS	Phone No	Status	Location
123456	123456789012345	0211234567	Pass	Auckland
123456	123456789012345	0211234567	Unkown	X
123456	123456789012345	0211234567	Faulty	Auckland
--Faulty Location: Sydney				
--Faulty IMS: 987654321012345				
123456	123456789012345	0211234567	Unkown	X
123456	123456789012345	0211234567	Faulty	Auckland
--Faulty IMS: 987654321012345				

Figure 2.4: The sample HTML format

file to do the transfer or use the XmlReader and StreamWriter to generate manually. I would like to write a XSLT file to do this, but the first thing is learning XSLT file first. I am still lack of knowledge about XSLT, I am keen to learn it in the future. Converting XML file to other format is also possible.

2.3.5 Update Factory Database

The Factory also needs to manage the Blackhawk units as well. They need to know what Blackhawk units are activated. The factory database is much simple comparing to Blackhawk database. It has only 9 columns. They are ID, QCID, BHID, SIMSERIALNO, WO, TIME, USER, ACTIVATION and GPS. After checking the report, the application should set the Activation and GPS columns of passed units to true.

2.3.6 Update Blackhawk Database

Before the Blackhawk units can be used, the information of the Blackhawk units should be added to the "Devices" table of the Blackhawk database. This table will manage all the functionality of the Blackhawk units. For example, this table controls whether the Alarm Alert is enable, whether the Lock Door Control is enable and so on. To add the new Blackhawk units into the "Devices" table, we need to use the UploadDevicesFromCSV() API under the FactoryService. This method will read a CSV file and add the Blackhawk units in this file to the Blackhawk database. Before the application can update the Blackhawk database, it needs to generate a CSV file first.

2.3.7 Statistic Support

The statistic support of the application is still in thinking. I don't really know how to do it.

2.3.8 The Testing Panel

To make sure the production staff can easily find out the faulty units, we designed a testing panel which can indicate the unit location in the tray. Figure 2.5 shows the old implement of the testing panel.

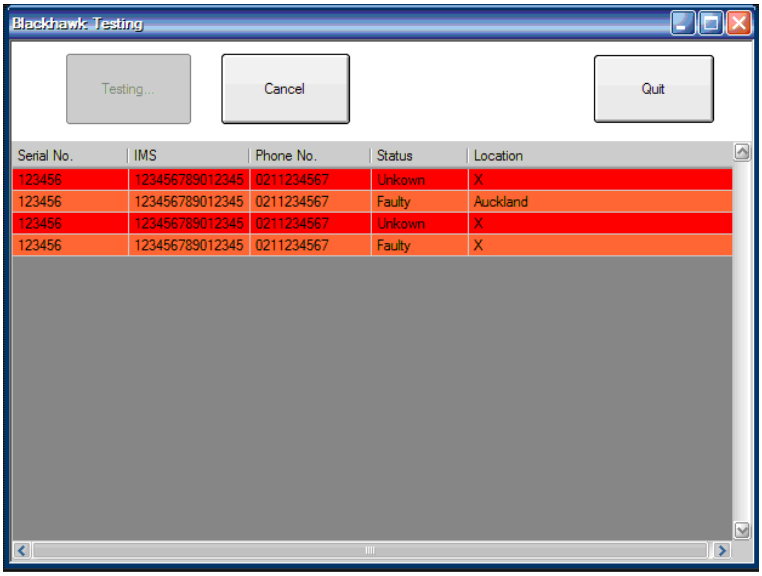

Figure 2.5: The old Design

In the old implementation, we remove all the passed Blackhawk units, so the tester can easily see the faulty reports. But the tester can only know the serial number of the faulty units other than the location. Because the application doesn't show the location of the faulty units in the tray, the tester have to check all the serial number printed on the Blackhawk units in order to find out the faulty ones. If the Blackhawk units are sorted by serial number, the tester may find out the faulty units easier. But it takes time to sort the Blackhawk units in the tray and sorted Blackhawk units still require the tester to check the serial number.

The solution is making a testing panel that can tell the location. Figure 2.6 shows the new designed testing panel that can do the task.

In this implementation, we use buttons to stand for Blackhawk units. Each button stands for a Blackhawk units. The location of buttons in the testing panel will be the same as the location of real Blackhawk units in the tray. How can we do this? The tester randomly picks up a Blackhawk unit, scans into the application, and puts it in the first holder of the tray. Then, the tester picks up the second Blackhawk unit, scans it into the application and puts it in the second holder of the tray and keep going. The first scan will activate the first button in the panel and moves the focus to the second button; the second scan will activate the second button in the panel and moves the focus to the third button and so on. If there is anything wrong with the scanning, the tester can click the button with problem and scan again. The focus then moves to next button. In this implementation, the tester doesn't need to care about the serial number at all. The location of button will tell the Blackhawk unit location in the tray. For example, if the second button in the second row goes to red, we can know that the second Blackhawk in the second row of the tray is faulty; the tester can go directly to pick up the faulty unit without thinking. If the tester wants to see any details about the faulty unit, he or she can just click that button on the testing panel. Then the unit information with faults highlighted will be popped up.

For the scanning part, we may add the setting so that we can choose which button to focus after rescanning. For example, after scanning ten units, the tester finds out that the fifth Blackhawk unit has been tested before. Then the tester clicks the fifth button and scans another Blackhawk unit then replaces the fifth Blackhawk unit by this unit. After this scanning, should the application move the focus to the sixth button or just move directly back to the first inactivated button? I would like to add a setting for this, so that the tester can choose which way to move the focus.

In current implementation, the application can only start checking after all the buttons are activated. This does make the application less flexible. If we have only one unit remaining for testing, the application will not able to do it. In the future implementation, I will remove this limitation, so that the application will be able to test any number (fewer than default) of Blackhawk units.

Another improper implementation so far is that the Blackhawk units can only be scanned and put into the tray in order. For example, the first scanned Blackhawk can be only put into the first holder of the tray. I would like to change this a little bit. I may enable the tester to put the Blackhawk unit to any holder of the tray. What the tester needs to do are clicking the corresponding button in the testing panel and scanning the unit

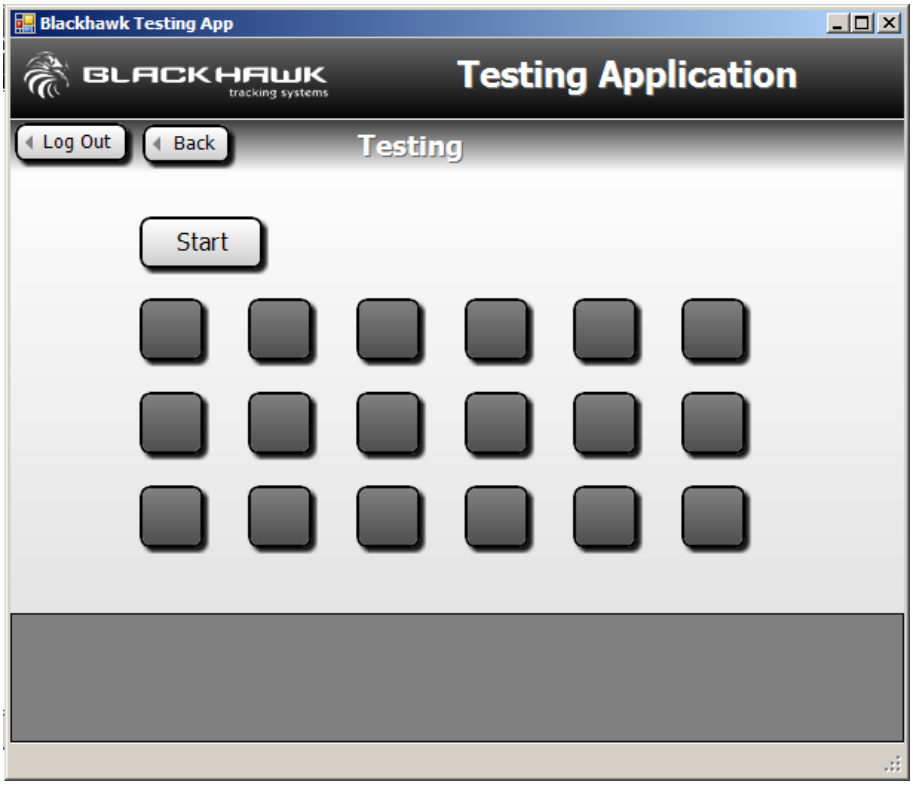

Figure 2.6: The testing panel

before putting the unit into the holder. For example, if we needs to test two Blackhawk units, and we want to put the first unit into the first holder and put the second unit into the last holder. How can we do this? Just click the first button to scan the first Blackhawk unit, and click the last button to scan the second unit. Then put the first unit into the first holder in the tray. Put the second unit into the last holder in the tray.

We may make the application more flexible if we can drag a button to another button to swap the location and drag the button out of the form to drop the scanned data. If I have enough time, I would like to implement this functionality.

2.4 Conclusion

During the first half part of the project, I have specified most of the application requirements and the way to implement this application in the specification. The spec also defined the formats of the configuration file and report of testing report. So far, the application can login to Blackhawk server to poll the Blackhawk APIs. It can get the data from the Factory database and Blackhawk database can do some checking.

The next step I need to implement these functionalities:

* Take input from scanner: So that the tester can scan the barcode to input the Blackhawk serial number.

* Add other two layers: Make the application can deal with three-layer tray.

* Finish the validation code: Finish the checking rules so that the application can give a correct feedback.

* Generate report: Generate report of testing results.

* Update Blackhawk database: Add the passed units into the Blackhawk database.

* The settings: Finish the settings panel, so that the tester can configure the application as needed.

Acknowledgements

I would like to say thanks to my Academic Mentor Dr S. Manoharan for providing lots information and advices on this project.

And I would like to thank my Industry Mentor Andrew Radcliffe for giving me many useful advices and a sample program I can work on.