Electronic Data Capture

Bachelor of Technology

Final Report

Siyu Zeng
1255777
szen010

This report summarizes the achievements of BTech last year project from both practical and academic perspectives. As a milestone, it also looks ahead for possible improvements and extensions.

Introduction

The appearance of Internet and rapid development of mobile applications have greatly affected people's everyday lives. The various roles of mobile devices have made them not only a platform for communication, but also a tool for entertainment and business. While traditional ways of information sharing are virtualized in many mobile applications, data processing has been a major task for them.

This report introduces the final year BTech project of electronic data processing. It firstly starts with an overview of the project. Afterwards, it discusses the requirements in detail. Through this several concepts related to the project are introduced. Then we describe our design principles. The design alternatives and functions are described as well. At last, we conclude the achievements and explore possible improvements for future.

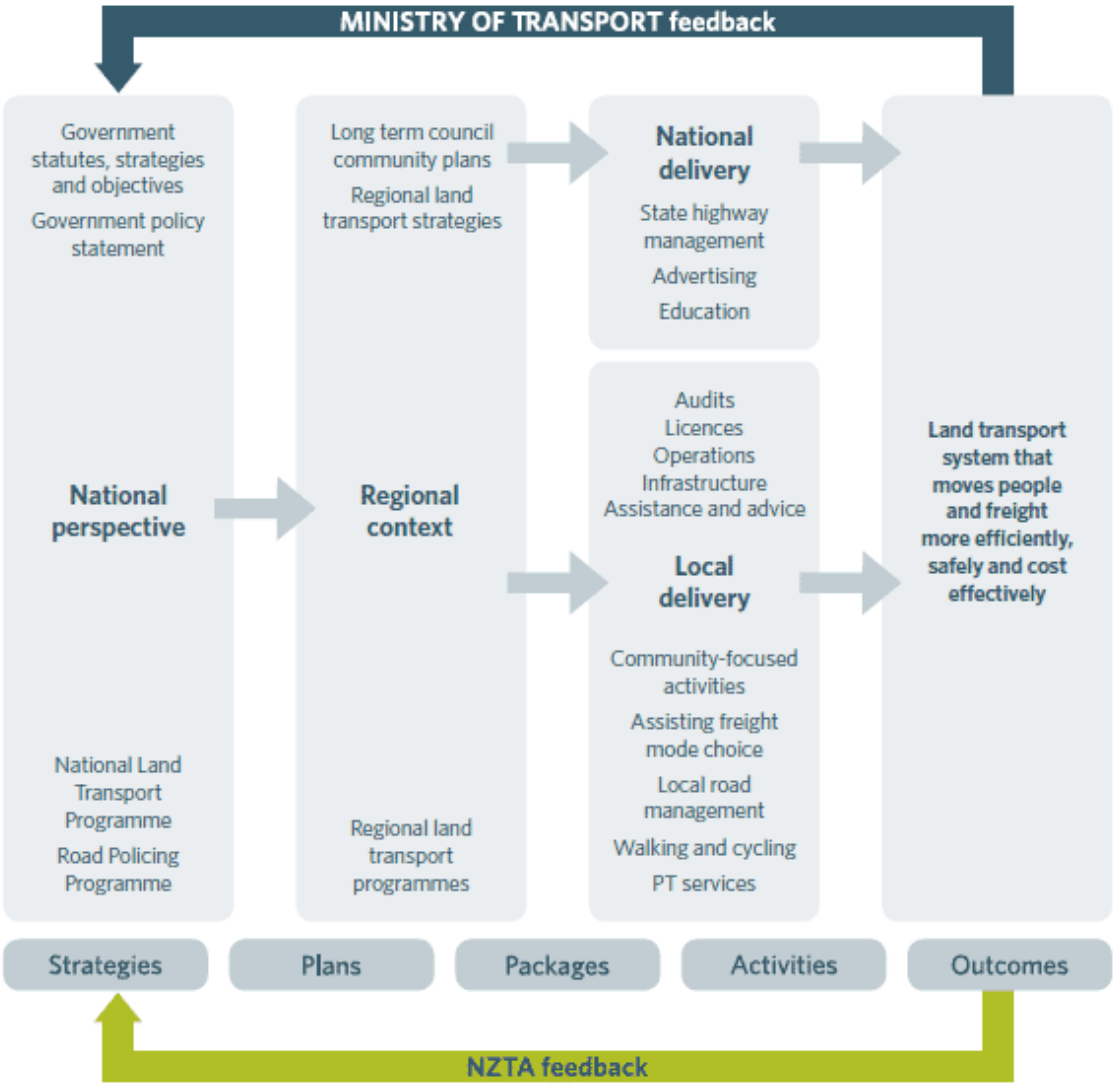
Agency

New Zealand Transport Agency (NZTA) was established on 1 August 2008 and has 12 offices around New Zealand. It is a Crown entity governed by NZ Transport Agency Board and is responsible for contributing to an affordable, integrated, safe, responsive and sustainable land transport system. This includes the planning and funding of land transport which involves the enforcement of laws, regulations and rules, and the collection of revenue. It also involves ensuring that New Zealanders have access to land transport, including through building, operating and maintaining land transport systems. This is the work it undertakes with a number of partners.

As just one part of the transport sector NZTA closely cooperates with sector agencies and others with an interest in land transport. Its role is to provide a vital link between government policy making and the operation of the transport sector.

It maintains close working relationships with:

- transport operators and the general public, who use and interact with transport
- transport committees, regional councils and territorial local authorities, which are responsible for implementing transport projects and other activities funded through the National Land Transport Programme
- suppliers, including contractors and consultants
- the NZ Police, which provides a range of road policing services
- the Ministry of Transport, which is responsible for leading the development of strategic transport policy and monitoring performance of the sector's Crown entities.



The NZTA's interaction with the land transport system

Background

Traffic crash reports (TCRs) are the forms completed by police officers at the scene of all road crashes, including non-injury cases. They record the details of when, how and why the crash happened. They provide essential information to many groups of professionals in the road safety and road construction fields: (a) police officers and police intelligence analysts (b) road controlling authority engineers and safety planners (c) traffic engineers

and consultants (d) researchers (e) central government, etc." The reports are used, among other things, to mitigate accidents of similar nature and to build a safer travel environment.

Currently, TCRs are paper forms. Manual entries on paper forms have several drawbacks including entry errors and illegibility. To analyze the data collected through paper forms, the data need to be re-entered into an electronic format. Data consistency checks are not possible in manual data entry: for instance, a single response to a single response question cannot be enforced.

The project turns up to solve these problems with the goal of developing a mobile application as a replacement to the paper form. Such electronic data collection, while reducing some of the issues seen with paper-based data collection, gives many other advantages: there is no need for dual data entry; some of the information can be auto-filled (e.g. time and location); some data can be auto-captured (e.g. driver information from the bar-code of the driving license); some media can be attached (e.g. photos, sketches, and audio). Thus the data collected can be transferred easily to the processing centre. This transfer can be in a manner that lends itself to easy processing (e.g. there may not be a need to re-code the data).

Requirement

Through the meetings with stakeholders, the requirement was explicitly confirmed. Some data fields which reveal to be unnecessary were discarded and additional ones such as airbag deployment were added. The key of the project is to maintain data consistency as well as to enable accurate capture of geo-location and crash diagram. Specifically, the requirement can be divided into four categories and should be fulfilled respectively:

Fast and Accurate Data Capture

As one of the main purposes of this project, the data entry process for police should be well accelerated. This relies on functionalities and user interface of the application. For example, to take advantage of such mobile applications, some additional features such as automatic data filling can be provided. This saves a great amount of time for police, although inconsistent data need to be changed manually by the police sometimes. Moreover, with the Global Positioning System (GPS) receivers on mobile devices, geo-location can be captured automatically. At last, the design of user interface should be carefully considered as different from paper-based TCR, mobile applications normally end up with considerable amount of pop-ups or dialogs which brings difficulties for users to manipulate. Think about a situation when a police has finished filling one section of TCR and wants to move on to next section. What he will do in paper-based TCR is to simply turn the page over. However, in the mobile application, he might found himself annoyed by a dialog asking “the data will be saved, are you sure to move on?”. Such behavior of the application will somehow slow down the process. Therefore, guaranteeing the easiness of using the application for fast data entering is a critical issue that also needs to be taken care of.

Data Validation and Consistency Check

With the current paper-based TCR, there is no way to ensure data validation while police are filling the form. This is embodied in the manual entry errors and the inconsistent information they are provided. It frequently happens in current TCR recording process as there are no indications when the police have made mistakes. Besides, the information that drivers provide can be inconsistent with the one stored in database. In this case, it is obvious that police have no way to check and correct it, which as a result, cause difficulties to identify the crash later on when the data are processed by NZTA. What is worse, it could also leads to the complete discard of a TCR.

Hence, in our application, constraints must be put on some data fields to reduce manual errors. For example, use a dropdown box for police to select the options from rather than simply give them a textbox to fill in. A reminder can also be provided so that whenever a police forgets to fill in some parts of the form, he will be informed of it. The most important is to allow automatic data acquiring for some sections such as driver information and vehicle information so that police can compare the data provided by the drivers with the one they get from database. As a result, they can decide which information to respect by discussing with the drivers whenever there is inconsistent information provided.

Multiple Media Support

Despite of the automatic processing capability of mobile application, multiple media support is another key factor that makes mobile devices to a certain extent engage people's everyday life. Clearly, although not the only one, an advantage of such functionality is that it represents events or facts in a more intuitive and interactive way than simple texts with an imaginary description that is more understandable to humans. Since user-friendly and user-centered designs have become primary concerns in most software, it is always important to consider a design from human's perspective at some points. Thus with the current mobile devices, some conventional text processing procedures can be enhanced or even replaced with media processing. Take driver interview as an example, normally what police do is to write down a whole paragraph whose length can vary from 30 to 150 words. This is definitely a time consuming and sometimes useless procedure as at the end of the day what NZTA needs is only a couple of key words describing the driver's status when the crash happens. With audio capture as a replacement, this problem can be solved because the only thing police need to do is to press the button for recording. Another example is the recording of environmental information such as road, light and vehicle factors. Such factors can be easily captured by taking photos as stronger evidences than words. Nevertheless, it is still the user's

preferences that to be respected as the design decision of whether to conform to current text processing.

Data Reform and Transfer

Another key concern is in which format the data should be stored and transmitted. Because the TCRs completed at the scene of crashes are later passed to NZTA for analyzing, a commonly used format for transmission must be used to produce documentation of the reports which is both human readable and machine readable. Extensible Markup Language (XML) is the first format that went in our sight as many application programming interfaces (APIs) have been developed for software developers to use to process XML data, and several schema systems exist to aid in the definition of XML-based languages. Furthermore, it is well supported by Wireless Application Protocol (WAP) and can be used to carry picture and audio file for our purpose. What is most important, a well defined XML schema ensures that the data represented must conform to it so that arbitrary data structure can be formatted to enable better management.

Goal

As mentioned before, the goal of the project is to build a prototype mobile application to explore possibilities of such application to take place of current paper-based TCR. While trying to stress on functionality and data consistency maintenance, the design should also consider likely problems that might be encountered during the implementation. This requires reasonable understanding of the project and research of related topics. Even though at the first glance it seems that the electronic TCRs are totally better than the old TCRs, there are still advantages that electronic ones cannot easily inherit. For example, due to inflexibility of mobile application lifecycle, the user might not feel that the application is completely dominated by him as normally he has

to follow the screen navigations, while on the other hand the paper-based TCRs are quite straight forward with everything kept handy. This implies another problem, that is, the inflexible filling order in mobile application doesn't respect users' manners or habits while processing such kind of form. For instance, with current TCR the order that a police fills in the forms may be different depends on various situations. A simple case would be that a police might talk to the driver of vehicle two first and therefore fill in the associated information of it before he actually starts processing information of vehicle one. However, within a design based on simple navigation, the user must follow the logical order defined by program in order to navigate among screens. This limitation can be solved by an innovative design approach as we will discuss later. Clearly there are other issues such as backing up TCRs for reuse and locking screen to reject careless touching events. Hence, the design should try to maximum the scope while following the basic principles of this project.

Related Work

Several works [15][13][31][22] have been proposed to compared the paper-based data collection methods with the electronic ones. Another recent one [29] introduces a mobile application for recording Applied Behavior Analysis (ABA) data. All of them come up with the conclusion that EDC would be more beneficial than traditional paper based methods.

In the first work [15], Weber et al. set up a research which involves recording data of 20 newspaper articles. The articles are selected from American newspaper reports with key word search in search engine. These articles report cases of older adults diagnosed with Alzheimer's disease who died as a result of becoming lost in the community. The goal is to record some variables provided by these articles such as age, sex and residence. Therefore, articles that cannot provide sufficient information of these variables will be discarded. Then the articles are divided into two subsets with ten articles in each set.

Data processing of the two sets is performed by research assistants (RAs) sequentially. For the first ten articles, RAs use traditional paper-based system and the statuses associated with the recording procedure such as processing time is recorded as well. Afterwards, RAs use the web-based system to collect data. Again, statuses are recorded. The result of the measurements of time, error and cost in this research show that using electronic data capture method reduces errors and time taken for completion. It also reveals that EDC has a higher fixed cost for setting up. This fixed cost includes purchasing hardware and software and staff training. However, as number of cases of data recording becomes large, EDC turns out to be much more cost efficient than paper-based methods.

Another study [13] is designed as a 5 by 5 Graeco Latin square to analyze the statistical result for 5 different data capture methods. The study firstly sets up 5 groups of interview between fieldworker and interviewees. Each group then uses a different method to record the data. These methods involves face-to-face interview with standard paper-based data capturing and processing, face-to-face interview with EDC using a netbook, face-to-face interview with EDC using tablet-PC, face-to-face interview with EDC using a Personal Digital Assistant (PDA) and telephone interview with EDC using a laptop. From the result, it is concluded that EDC is both time and cost effective in compared to standard paper-based data collection method. Furthermore, although accuracy enhancement property of EDC cannot be confirmed in this study as interviews with EDC using laptop and PDA cause more errors, the error rates of EDC with tablet and netbook approach those of paper-base method.

Bart [31] proposes a report to compare EDC with Paper Data Collection (PDC). It summarizes the evaluation result of EDC in previous publications and concludes that EDC is both time and cost saving. Yet, Bart argues that factors such as inertia of a conservative, heavily regulated market, service providers feeling uneasy of adopting a new method capable of transforming their entire business model and the extreme

frequency of change both in hardware and software products would slow down the adoption of EDC.

Fletcher et al. [22] conduct a study on alcohol sales in community events to measure the propensity of servers to illegally serve alcohol to underage or intoxicated patrons. The study involves using both PDC and EDC to collect data from a questionnaire. The EDC is carried out with the assistance of PDA. An analysis on rate of data agreement and disagreement between two methods is then performed. From the result, it is shown that data collected from both methods have high agreement at 95.5% while the disagreement is low at 1.3%. In addition, data are missing from PDA but available from paper 1.5% of the time and 1.7% of the time they are missing from paper but available from PDA. Another finding from the study is that data missing in PDA is mainly caused by absence of entire record, which indicates that a reminder technique is required for EDC.

Challenge

Although long-term adoption of EDC would give us potential benefits on both time and cost saving, there are quite a few limitations while implementing a mobile application. Fail to address or solve these problems when they are encountered would cause a huge impact on performance and usability of the application.

Power Consumption

Certainly with the mobile device, power consumption is a main concern. The limitation of power capacity is due the small size of battery which is restricted by the size of the mobile device. The benchmarks performed by Carroll et al. [10] shows that the power consumption is considerable for mobile devices even when they are in suspended state because the communication processor needs to be active in order to receive calls and

SMS messages. The number is quadruple when the device is active but without any application running. This implies that the efficiency of battery usage is of great importance especially when the device is to be fully relied on for performing some jobs. Therefore, an application should avoid using long-term running threads or services.

Connectivity

As wireless connections have limited bandwidth and unreliable connectivity, network usage is also a critical problem. As our application tends to be used by police for recording crash data, there will be many cases that it is used with poor of even no internet connectivity. This requires the application to provide some cache functions for rarely changed data sets and an optimized data format for transmission.

Processing Capability

The limited computing power and memory of mobile device lead to the absence of some applications. For example, three dimensional graphics applications that require real time rendering typically are difficult to implement because they demand a lot of memory and CPU power. This fact certainly causes the degradation of performance of some applications or even limits the functionality of them. Meanwhile, even if an application tends to exploit the full computing power of the mobile device, the power consumption is still a problem.

Data Entry Capability

Different mobile platforms have different styles of widgets for their views. These small widgets are sometimes used for collecting users' inputs. While achieving the basic functionality, these widgets are also trying to provide a better experience for users to enter inputs. For instance, the dropdown list in Android simply shows a scrollable list when user clicks on the dropdown box and in iPhone it behaves like a slide panel that lets users slide among items for selection. Even so, some small widgets such as buttons and labels still limit the efficiency of user while entering data [17].

Screen Size

The balance between mobility and display capability on mobile device has always made the later to suffer. Although previous researches suggest that the performance difference for users to browser text on various sizes of screen will not be obvious [26], many mobile applications still demand a large screen to achieve a high performance. Kim et al. [24] also argues that the user would experience more difficulty reading on-line text on small screen PDA. Nowadays, this performance difference caused by screen size has become more significant for graphical applications which mostly rely on gestures and touch events because small screen size tends to cause the result of the touch events inaccurate sometimes. This requires large screen to be available for some applications. However, as shown by Carroll et al. [10] that screen consumes large amount of power even when the device is idle, larger screen would contribute more to the power consumption. Moreover, the varied screen sizes of Android devices make it even more difficult to implement an application to fit all screen sizes and to predict the impact of screen size. Even with recent released iPhone 5, the change of screen size has caused modifications of many applications to fit in it. Therefore, the effects caused by screen size should never go out of our sight.

Storage

The limited size of storage on mobile devices requires the data to be either stored in binary format or compressed. Nevertheless, the former will cause the information unreadable while the later will result in more power consumption due to intensive CPU usage.

Despite of all these problems, a successful implementation should also consider issues such as user input, user acceptance, training cost and maintenance cost. It happens frequently when user inputs are obtained only from programmers or technology people during application testing phase. This could lead to a product failure later on as the perspectives between technology people and the real users may be different [19]. This

disparity would not be noticeable without proper user inputs. In addition, once installed the application could still be subject to both hardware and software failure. One common example is the incompatibility caused by frequent updated operating systems and new application programming interfaces (APIs). With all these challenges in mind, we need to consider in our design carefully how to avoid such situations.

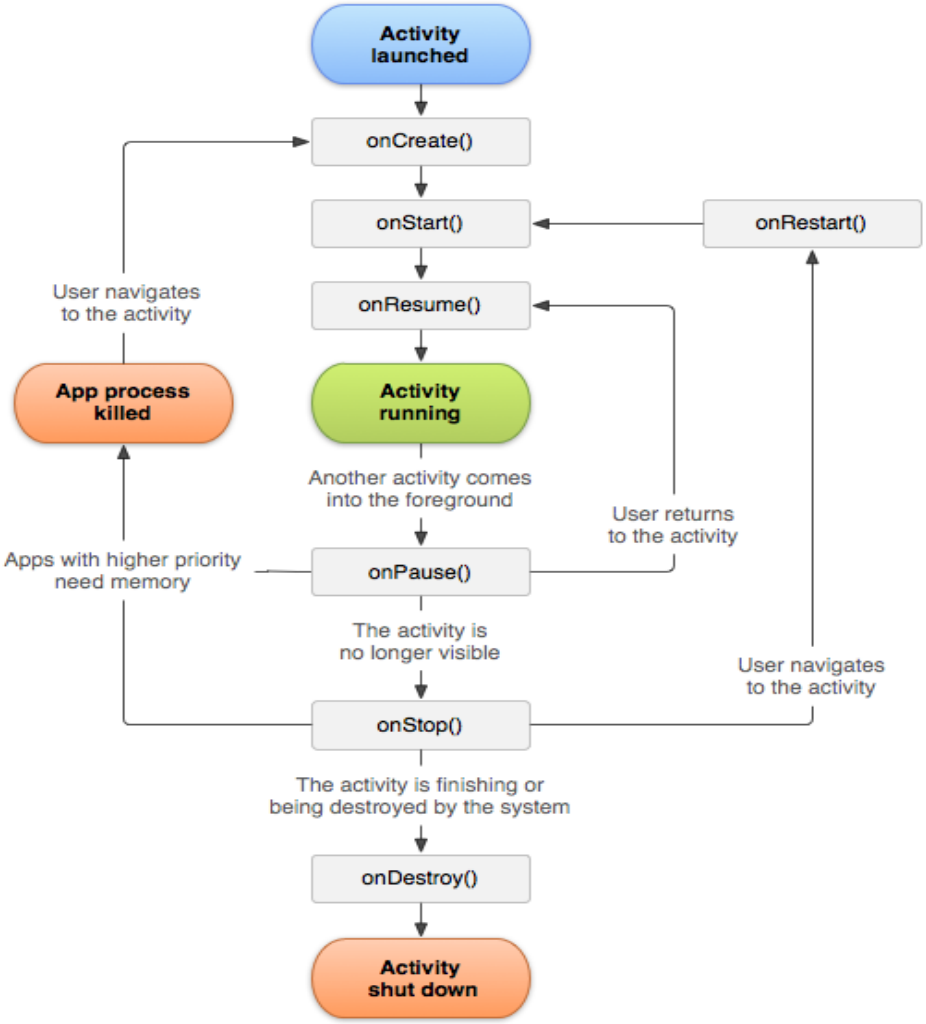
Design

The design is based on Android operating system which is developed by the Open Handset Alliance (OHA), led by Google, and other companies. The platform was created by Android Inc. which was bought by Google and released as the Android Open Source Project (AOSP) in 2007 with a group of 78 different companies forming the OHA that is dedicated to develop and distribute Android. As an open source software system, it benefits not only programmers but also device manufactures [27]. This openness has made Android one of the best platforms over the world.

The Android operating system is a multi-user Linux system in which each application is a different user. Each application is assigned a different Linux user ID by the system which is used to set permissions for ensuring unique access to its related resources. Also, every application runs in its own process with its own virtual machine (VM) isolated from others'. A process is only started when any part of the application's components is needed and shut down when the component is no longer used. These properties have provided a secured environment which prevents resource sharing among different applications without permissions. However, Android does allow applications to share same user ID, to run in same process, to share same VM as well as to access device data with permission. In this case, all permissions must be granted at install time.

Android Activity Lifecycle

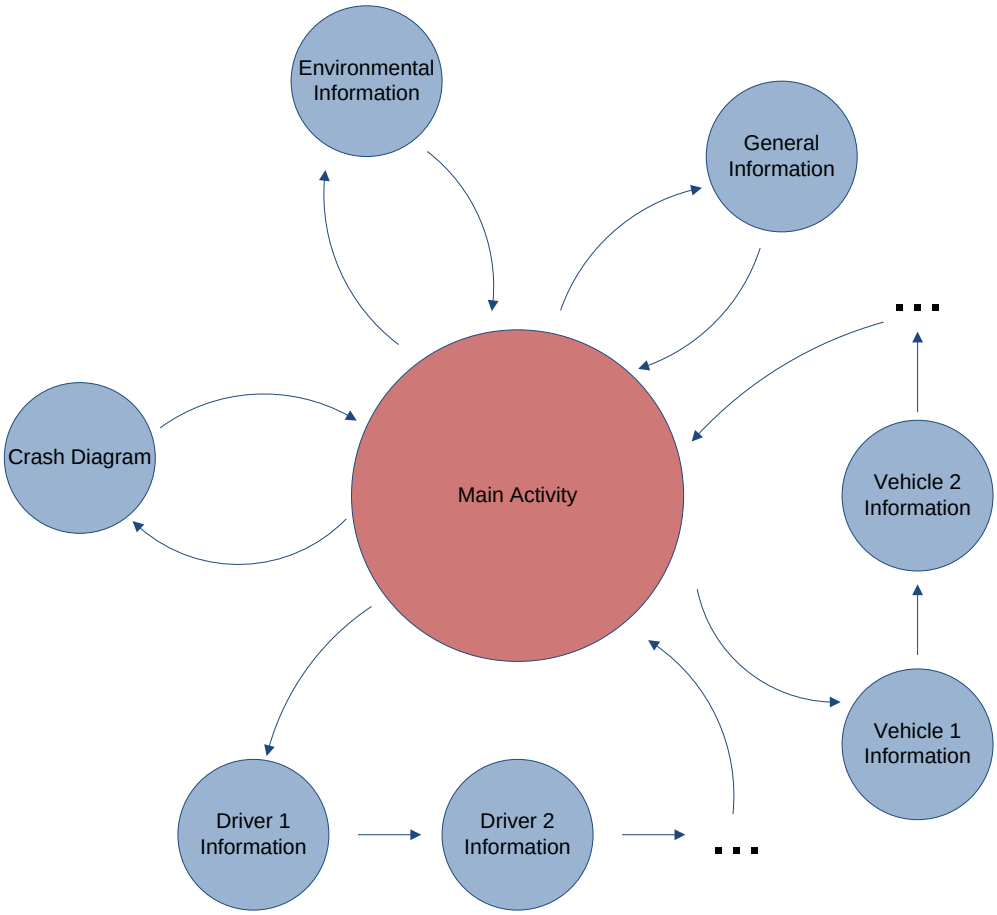
An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Therefore, screen navigations in an android application mainly rely on management of the activities' lifecycles. The figure below shows the entire lifecycle of an activity.



Typically, an Android application contains several activities and starts with the main activity. Different activities can have different views or actions. In order to switch to content contained in another activity, one must creates an intent specifying either the

exact activity it wants to start or the type of action it wants to perform. When a new activity is started, the previous one is stopped and preserved on the “last in, first out” (LIFO) “back stack” then the newly created activity is pushed on top of it. Thus whenever user presses the Back button, the current activity is popped from the stack and destroyed with the previous activity resuming.

Specifically, the screen navigations in our application will be controlled by different activities bound with each other with different views as their contents. Each time the user press a button defined in the view for navigation, the new activity is brought to focus and its view is displayed. This always causes some causal order of the screen. For example, pressing Next button in view one allows user continues to view two and later if he wants to switch to view three, he has to press the Next button in view two. Meanwhile, there is no way that he can go directly from view one to view three, despite of the reason why he would do so. The reason why this happens is because the activities are logically chained or cycled without a centralized activity having access to all of them. Therefore, our first step of designing the activity lifecycles is to split TCR in several sections with a main activity that can navigate to all of them. This normally can be achieved using tabs. The following figure shows the relationship among these screens.



Web Application vs Native Application

Although the objective is to implement EDC on mobile device, it doesn't mean that we are left with no choice of what kind of application we are developing. In fact, there are two main categories of applications on mobile application that we could choose from, that is, web application and native application.

Web Application

A web application is constructed in a browser-supported language such as HyperText Markup Language (HTML) with JavaScript. It is normally not installed on the device and downloaded through web requests in each run. There are several advantages of this approach. The biggest one would be the easiness of crossing platform. As web

applications are developed with browser-supported languages, any platform with browser that supports these languages could run the application. This also saves the developing cost as the developers don't have to develop specifically for different platforms. Also, because web applications don't require previous download or installation, users don't have to go through an intermediate such as Google Store to search for the application. Meanwhile, update is easy as the application is maintained on server side and the only thing users need to do is to keep their browser compatible with the application's language. The last advantage is the easiness of consistency maintenance of application version among users because the application is maintained on server side and all users will download the same version.

Undoubtedly, there are shortcomings for web applications. Firstly, web applications don't exploit as much device features as possible. It generally cannot access hardware and software on device. Thus some useful context-awareness functionalities such as GPS and accelerometer will not be available. Secondly, the application needs to be downloaded in each run even if its functions don't require any internet access. This makes web applications fully rely on internet connectivity to operate without caching mechanism. A recent solution is mentioned by Steven J. V. N. [30] that AppCache and some other features of HTML 5 allow caching of web applications' data and program. This allows a web application to operate off-line. Another big issue with web application is the application execution time as mentioned by Charland et al. [11]. As the programs of web applications are always interpreted by browser rather than compiled by the compiler, the larger the programs is, the longer it takes to execute it. Meanwhile, this is also a both time and power consuming procedure, as interpretation is generally a slower and CPU intensive work.

Native Application

As opposite to web application, native applications are downloaded from web stores and installed on devices. The applications are built with the programming language of

the specific platform. A big advantage of this approach is that it can utilize both hardware and software on the device and thus provides user a better compelling experience [23]. Furthermore, the application can run without internet meaning that all functionalities that don't require internet access will be functional. In addition to that, native application generally runs faster than web application as the code is compiled.

Clearly, there are also drawbacks of native applications. As the applications are developed in different programming languages, it is almost impossible for them to cross platform. In addition, this also leads to a high developing cost. Update and consistency maintenance of application version among users are complicated as users need to download the latest version, in which case they might decide to discard the download.

For our application, most of tasks are done by form filling which doesn't need internet access. Besides, the crash often happens in a circumstance without reliable wireless connection and requires fast data capture. Also, we want to make use of the device hardware for photo capture and voice recording. These all imply that our application would mainly run off-line and should use some of the device features. Considering all these factors, we decided to implement our application as native.

Data Serialization Formats

After we have made the decision of what sort of application should be implemented, we now consider how our data should be stored and transmitted. This brings us to the discussion of different data serialization formats.

XML

The Extensible Markup Language (XML) is part of the Standard Generalized Markup Language (SGML) that formulates a human readable format [9]. It is a textual and structural representation of a data object with the support of Unicode encoding. Through its achievement of providing simplicity, readability, usability and compatibility to the representation of the data objects, XML data processing has been adopted by

many APIs and leads to the appearance of many XML-based protocols and languages such as Simple Object Access Protocol (SOAP) and Extensible HyperText Markup Language (XHTML). The XML relies on the markups of entities to make up a well-formed document. It doesn't have pre-defined tags so that all tags are defined by user and the complete document needs to be checked against a schema for validation. The use of XML language is essential as it not only consolidates data consistency but also provides a human readable description of the captured data structure. However, despite of its increased size due to tagging, the textual representation property has predestined that data processing of XML will be much slower than that of binary format.

XML Schema

An XML schema typically describes XML documents in that type with a set of rules in addition to basic XML syntax to restrict the content of the document. This requires a schema itself to also be syntactically correct so that it can be used to validate against a given XML document. The main advantage of using schema is that detailed information is given for representing the content of a valid XML document and validators can be used to perform a more accurate job rather than manually checking the errors. Several schemas have been proposed during the past few years. These schemas include Document Type Definition (DTD), XML Schema Definition (XSD) and Regular Language for XML Next Generation (RELAX NG). In our project, we use XSD for defining our XML schema. The reason why we prefer XSD and RELAX NG than DTD is because they are widely used and proven to be more expressive [18][25]. The main difference between RELAX NG and XSD is that RELAX NG allows nondeterministic content models [25]. For a given schema, it is deterministic when the parser used by Schema Object Model (SOM) can always predict the next element when it encounters an element [4]. Therefore, it can always determine the sequence of occurrence of the elements. Oppositely, the schema is said to be nondeterministic. The indetermination is normally caused by elements with same name existing within different paths in a schema. For example, consider following schema.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="personName" type="personNameTypes" />
<xs:complexType name="complexName">
        <xs:choice>
            <xs:sequence>
                <xs:element name="lastName"/>
                <xs:element name="firstName "/>
            </xs:sequence>
            <xs:sequence>
                <xs:element name="lastName"/>
                <xs:element name="nickName"/>
            </xs:sequence>
        </xs:choice>
    </xs:complexType>
</xs:schema>
```

When the parser firstly encounters element lastName, it cannot predict the next element as two sequences both have an element with this name. This is the so called nondeterministic schema. In this case, if we change the name of second lastName element to fullName, the schema becomes deterministic as no elements with duplicated names exist in these two choices. Although this advantage makes RELAX NG preferred in some situations, XSD is sufficient to meet our requirement as our schema doesn't contain such nondeterministic situation.

JSON

JavaScript Object Notation (JSON) is another text-based lightweight format for data exchange [3]. It warps the data with special separators such as brackets, commas, quotes and colons into a textual and structural format. A JSON object is an unordered set of key/value pairs. The key and value are separated by colon and comma is used to separate pairs. The value sometimes can be a JSON array which is an ordered collection

of values separated by comma. All values and keys are double quoted. The following is an example of a JSON object.

```
{
	"firstName": "Albert",
	"lastName": "Einstein",
	"phoneNumbers": [
		{
		   "Location": "Home",
		   "Number": "1234567"
		},
		{
		   "Location": "Work",
		   "Number": "7654321"
		}
	]
}
```

The advantage of JSON representation is that its performance is much better than XML due to less markup overhead [12] [28].

Binary Format

While text-based data serialization formats are sometimes exposed to be insufficient to meet the performance requirement, binary formats are gradually involving in. These binary serialization formats include Protocol Buffers, Apache Thrift, Binary JSON (BSON), Structured Data Exchange Format (SDXF) and Wireless Application Protocol Binary XML (WBXML).

Protocol Buffers

Protocol Buffers [2] is a lightweight binary encoding format developed by Google. Users are required to define protocol buffer message types in .proto files in order to serialize structured data. The encoding mechanism is somehow similar to Wireless Session

Protocol (WSP)/Wireless Markup language (WML) encoding except that it uses Little Endian Base 128 (LEB128) varints for represent number. If the Most Significant Bit (MSB) of a byte in a varint is set, it indicates that further bytes will follow. When the message file is compiled, data access classes are generated to map each name to its value. Meanwhile, type is also mapped to an integer with a pre-defined type list. This is similar to code pages used for WSP header encoding. The encoded message is a byte stream. Below is an example of a simple message structure and the pre-defined type list.

```
message Test1 {
      [Field Rule] [Field Type] [Field Name] = [Field Key];
}
```

Type	Meaning	Used For
0	Varint	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	64-bit	fixed64, sfixed64, double
2	Length-delimited	string, bytes, embedded messages, packed repeated fields
3	Start group	groups (deprecated)
4	End group	groups (deprecated)
5	32-bit	fixed32, sfixed32, float

The whole line in the message is called a field. The “Field Rule” is used to limit number of occurrence of this field in a message. The “Field Type” and “Field Name” defines type and name of the field. The “Field Key” of each field must be unique for mapping during encoding. In this case, suppose we have,

```
message Test1 {
      required int32 a = 1;
}
```

In the example, the field has a type of “int32”, name of “a” and key of 1. Assume the value of “a” is set to 150 by the application, the encoded message will be 08 96 01 which is 0000 1000 1001 0110 0000 0001 in binary. The first byte 0000 1000 represents the key and type. Firstly, by dropping the MSB we get 000 1000. The last three bits are

used to indicate the type mapped from the type list. The type is used by parser for finding the length of the values so that it can skip unrecognized fields while decoding. In this case the type is 0 which indicates that bytes after it will be varints representing a number. If the type is 2, a length byte will follow to tell the parser the length of value. The remaining bits of first byte are used for defining field key. Therefore the binary 0001 indicates that the field name should be “a” as 1 is mapped to it. The remaining two bytes are used to represent the number 150. Again, by discarding MSB of each byte we get 001 0110 000 0001. As the format is little-endian, the second byte is actually most significant. Therefore, by reversing the two bytes we get 000 0001 001 0110 which is 150.

Protocol Buffers uses UTF-8 for encoding value of string type. Moreover, if the type of a field is another message object then the encoded result of that message is appended to the value. Suppose we now have another message definition,

```
message Test3 {
      required Test1 c = 3;
}
```

Assume again “a” in “Test1” is set to 150. The encoded message of “Test3” will be 1a 03 08 96 01. In this case, 1a represents key and type, 03 indicates the length of value and value 08 96 01 is just the encoded message of “Test1”.

Apache Thrift

Apache Thrift [5] is a Remote Procedure Call (RPC) framework developed at Facebook. It serves as an Interface Description Language (IDL) which allows the users to create services through several different languages. The encoding mechanism involves writing numeric tag for each piece of data. The tag/data pairs are then used to form up the byte stream. The encoding of integers is based on Variable-Length Quantity (VLQ) which uses last 7 bits of a byte to represent data and the MSB to indicate the continuation. Different from Protocol Buffers, it follows a big-endian rule.

BSON

BSON [1] is a binary-encoded serialization of JSON-like documents. It applies little-endian rules for its binary encoding. A BSON document is a list of elements composed of field name/value pair in ordered manner. Field name and text values are kept in string format while others are encoded into binary. Each element also contains a type and the length of its value. The array indices are included when encoding a JSON array. The first four bytes of BSON are used to indicate the length of BSON document and \x00 is used for termination.

SDXF

SDXF [6] is a self-describing data format that allows interchange structured data block with various types. The data are constructed in a unit called “chunk” with a header describing its ID, content type, length as so on. A chunk can also consist of multiple chunks which give user the ability to describe arbitrary structural data and to unpack without knowing the meaning of the individual data elements.

WBXML

WBXML [8] is a compact binary encoded version of XML. It intends to reduce the performance overhead of XML when adopted on devices with limited processing power such as mobile devices. It’s normally generated at Wireless Application Protocol (WAP) gate ways through the compilation of XML documents. The WBXML encoding follows a fixed structure and is mostly a tokenization process in which tags and elements are mapped to pre-defined tokens.

While XML and JSON both provide user readable documents with textual representation, the drawbacks are obvious, that is, the verbosity caused by markup. Although JSON has tried to improve this problem, some argue that the extensibility drawbacks and the lack of namespace and input validation make JSON not the best choice for data exchange

sometimes [28]. On the other hand, binary formats seem to be more efficient even if they don't provide any readability. However, there are also limitations for binary serialization formats. For example, the additional length prefixes, explicit array indices and textual representation of field names have made BSON require more space than JSON sometimes. For SDXF, there is a maximum chunk size of 16777215 bytes. Despite of all these, the binary formats applying little-endian might not be optimized solutions for transmission as the byte order in network is big-endian.

Although many works have been surrounding the performance comparison of these formats, the conclusion is not easy to make. As suggested by the study carried out by Nurseitov et al. [28], JSON is faster and uses fewer resources than XML. However this fact doesn’t apply when there are only a few objects. Moreover, the study doesn’t take compression into account which may have impact on their general performance. A more recent work [12] compares the XML, JSON, Protocol Buffers and Apache Thrift in terms of serialized data size, serialization speed and ease of use. The result shows that two binary formats followed by JSON manage to accomplish smaller data size and faster speed during serialization. In addition, binary formats are not suitable for web services as they are not human readable and cannot be deserialized by the receiver without according files. Again, compression is not included. Another work [14] evaluates the performance of XML, JSON and Protocol Buffers from various aspects both with and without Gzip compression. The result shows that JSON with compression achieves a best overall performance in terms of synchronization time, parsing on server side and battery management. Yet, the compression time for JSON is worse than that of XML and XML is the biggest beneficiary from compression. Meanwhile, binary formats outperformed XML when there is no compression and compression on binary format generally has little effect.

Based on above results, binary formats seem to be best choice for storing data locally as it has smaller serialized size and faster serialization speed. However, binary formats are not suitable when it comes to network transmission as they are not human readable

and requires extra software support. So we are left two choices for data transmission, XML and JSON. Although previous benchmarks show that JSON has a better general performance, things can get tricky when applied to special situations. For our project, as the data recorded may contain audio files and pictures, compression might be needed. In this case, a great improvement of performance when using XML could be achieved. Furthermore, the lack of namespace makes JSON not adequate for referring individual files. On the other hand, this can be done with Uniform Resource Identifiers (URIs) in XML. What is most important, as we mentioned before, the data validation is a key point of this project. Therefore, the data received must be validated to fulfill our purpose. Considering these factors, we decide to use XML for final data transmission with XSD for its validation.

Web Service

After the data is captured, we need to send them to the server. This brings another issue which is the adoption of web service (WS). Current WS design generally falls into two categories which are using Simple Object Access Protocol (SOAP) and following the Representational State Transfer (REST) architecture.

SOAP

SOAP [7] is a lightweight protocol for arbitrary structured data exchange over a distributed environment. It uses XML for embedding its message and is used together with Web Services Description Language (WSDL) for implementation of WS. Although not necessary, it normally operates upon Application Layer protocol such as HyperText Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP) for information transmission. Messages are delivered in SOAP envelopes which encapsulate both SOAP header and body. While implementing a WS, WSDL is usually used for describing the functionality. It includes the information of the WS in a XML file and by accessing this file a user is able to understand and call the operations of the WS. Since SOAP relies on

XML for its message passing, WS using SOAP has two main drawbacks. As mentioned by Tekli et al. [21], SOAP utilize considerable amount of network usage and is computationally expensive due to conversion between in-memory data and the ASCII-based XML. Both of these bottlenecks appear to be critical when SOAP-based WS is implemented for mobile devices. Several solutions have also been mentioned by Tekli et al. for reducing network traffic and improving service execution time. For example, compression on XML or multicasting can be applied. The former tends to reduce the overall size of the message while the later saves local bandwidth by multicasting responses with same content as the messages are only duplicated at node with split links. On the other hand, reduce of execution time can be achieved by using special-purpose parsers and applying differential serialization or deserialization mechanisms which store previous processed objects and are only applied to unprocessed objects.

REST

The REST architecture involves making use of the existing well-defined HTTP for implementing WS. The messages passed between clients and servers are simply basic HTTP requests and responses. The contents of these requests and responses don't have to be XML based. In other word, the HTTP Accept header might be required to indicate the type of content. RESTful WS has some primary characteristics. First, it's resource-oriented which means every resource is denoted by URI. Moreover, the communication is stateless meaning that the server is completely unaware of the client's state. This stateless principle has enabled it to further extend its functionality to allow intermediates such as gateways and proxy servers which also brings capabilities for different levels of caching. Meanwhile, this statelessness property also leads to a better suitability when adopting RESTful WS in an environment with unreliable internet connectivity. The communication processes are simple HTTP message passing procedures whereas requests are continuously fulfilled by server with responses of various types of contents. Although the network usage and processing time for RESTful WS are on average less than those of SOAP-based WS, several common issues related to

distributed technology should still be concerned. One of these issues as mentioned by Christensen [20] is the balance between request frequency and average user waiting time. As larger data payload in request tends to cause longer execution time on server side, this will result in a longer user waiting time. A solution might be to break a single request into several small ones in order to keep the user responded. However, this could leads to a degrading of the applications performance.

For implementing our WS, we will adopt REST architecture. There are several reasons for this. Firstly, the transmission and processing cost of SOAP message is not affordable in internet environment for mobile devices as these bring massive drawbacks such as high internet usage, high requirement for processing power and high power consumption. On the other hand, RESTful WS allows content with different types such as JSON. This leads to a greater improvement of the performance. Secondly, since RESTful WS follows a stateless principle, it's more adoptable in internet environment with unreliable connectivity. At last, our WS may contain only two operations of which one is to query the server database with driver license number for driver information to accomplish automatic filling functionality. The second operation will be to upload the crash report to the server. In this case, simple HTTP Get and Post methods with optimized data format can fulfill our requirement. Therefore, RESTful WS has been made our choice.

Crash Diagram

At the early stages of our design, quite a number of discussions are surrounding the topic of crash diagram. Different from human's habit of using a pen for drawing, free sketching on a tablet doesn't give us a relatively good appearance. Meanwhile, this kind of free sketching doesn't actually reduce the process time and workload for users. Moreover, distinct drawing styles of different users leads to inconsistent appearance even with the same object. The practical results in past TCRs have reflected this fact. Therefore, we are left only two choices, to make use of existing map application for capturing our road picture and draw our own template objects on it or to provide the

road template ourselves as well. After a few attempts of the former approach, we conclude that it is difficult to embed a third party map application in our own especially when it's not open source. Meanwhile, although Android supports map view that embeds GoogleMap, the zoom level is not enough for our purpose. This leads to the adoption of second consideration in which we make our own templates for different types of roads as well as objects. This is a significantly challenging task because we have to implement our own methods of handling touch events such as selecting, dragging and rotating. Also, a lot of geometric computations need to be implemented to perform these operations.

Data Field Priority

After we have obtained the initial blueprint for our project, the next step is to define priorities of data fields. Clearly, there are a couple of fields which are usually not filled or used in current TCR and a few fields that are extremely important. The statistical results from NZTA give us the clue which fields can be discarded and which must be enforced. Also, during the meetings with stakeholders, the decision of abandoning some fields as well as adding extra fields is made. Enforcements on accuracy of several fields are also stressed. By then, the amounts of data to be captured as well as their priorities in TCR are well defined and the requirement is mostly confirmed.

Two Design Alternatives

The previous discussions imply that we have not only one design option of how to implement TCR in our mobile application. Here we introduce two design alternatives that we found feasible during research. Although both approaches involve form filling, the way how application interacts with users significantly differs.

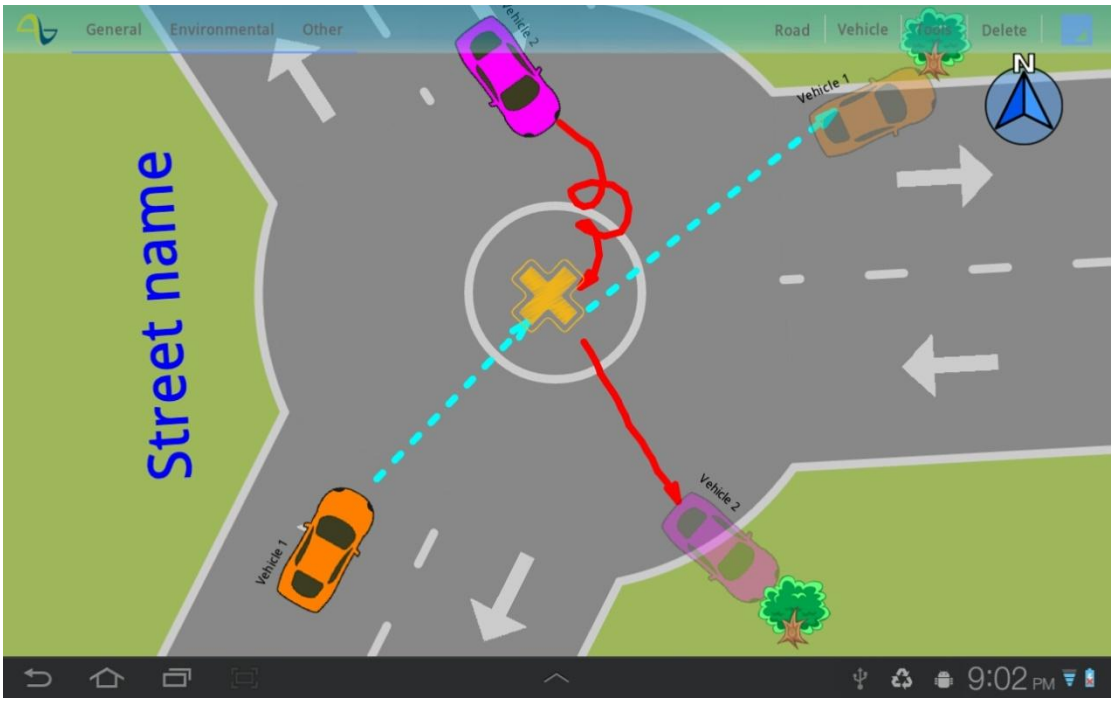
Screen Navigation Based

A direct way of interpreting current TCR in our mobile application is to let the users navigate among screens with each screen containing corresponding view that matches the original form. This is done by using simple logic on buttons in each view to switch among activities. Also, as an optimization mentioned before, a main activity can be provided as the central navigator so that the structure is no longer a ring topology but a star topology. Even better, an Android menu can be implemented together to make it a mesh topology. In this case, the user can have several routines to choose from and can switch to any other routine at any time rather than to strictly follow a single routine. This approach might be preferred by most police as it preserves the basic manner of users while filling such forms, that is, to go from section to section. However, it appears to be unattractive and still has some problems. Firstly, although the logical order of completing different sections can vary, the information of each section must be filled in sequence, that is, the user still has to follow a specific routine within a section containing several screens even if an Android menu is provided, because each section must have a fixed start screen and we cannot use menu for all these sub-screens. For instance, although not exactly the case, while filling information for vehicles and drivers, the user always has to go from first to last. The factor that causes this behavior is the way we implement our model for storing vehicle and driver objects. These objects are stored in collections such as ArrayLists so that whenever user wants to add one, a new instance is initialized and given the last index of collection as its ID as well as its position in the collection. This process predefines the ID for these objects which thus follows a numerical order. Even though this behavior can be solved by letting users enter ID themselves, it doesn't conform to the basic principle of this project, which is, to make the application as much responsible as possible for maintaining data consistency. Also, storing vehicle and driver information in separated collections doesn't result in solid relationship between them as the only way we can associate a driver with a vehicle is to compare their respective index in the collection. This can cause serious problem if the

user wants to perform deletion when the operation is not correctly implemented. At last, it is difficult for users to identify the corresponding information of a vehicle later when they start to draw it on crash diagram. It sometimes causes mismatch between vehicle information and crash diagram which brings a lot of trouble for NZTA. With all these disadvantages, we look for a solution that can mostly solve these problems. This leads us to the second approach, which is graphic based.

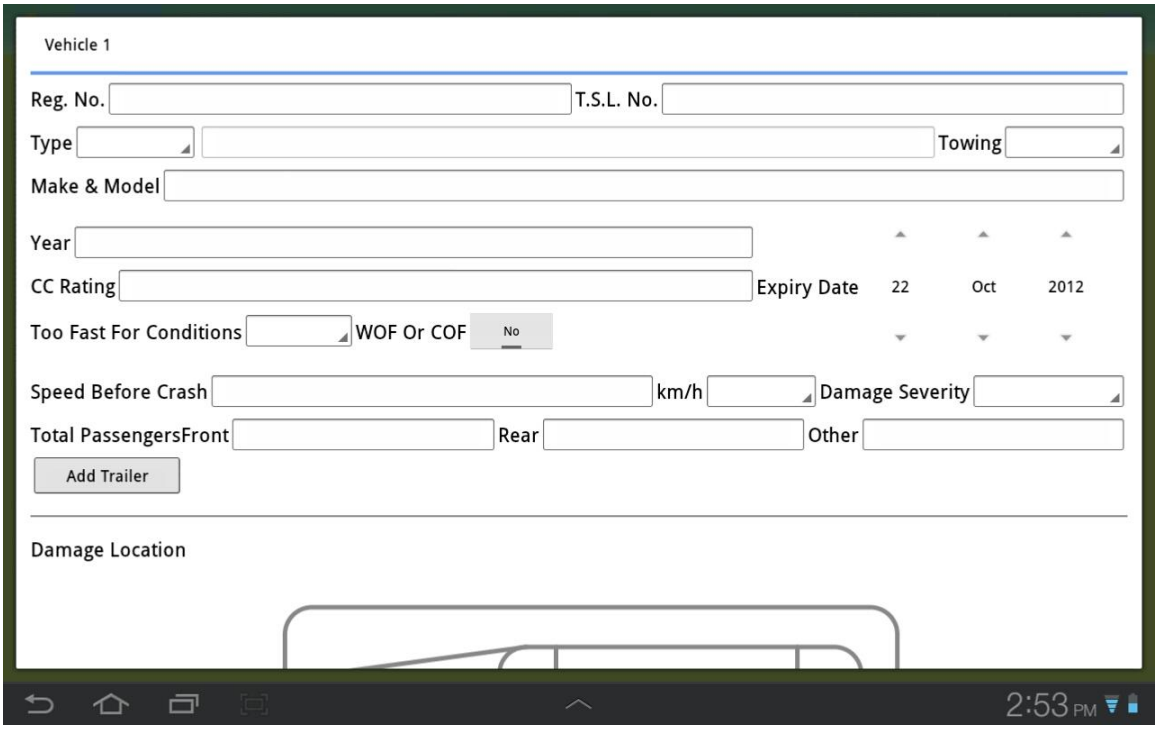
Graphical User Interface (GUI) Based

Regardless of how we build our application, the users always need to complete forms at some stage of recording process. Yet, we can change the way how users interact with the application by importing a GUI as the basis for entering all information. The concept will be explained step by step. First, the application starts with a GUI which requires the drawing of crash diagram that imaginarily describe the crash. After the user has finished drawing, he can start filling in the according information. The following figure shows a complete diagram at this point.



Now since everything has been lain down, the user can start entering information. The way how we achieve recording of vehicle information is by popping up a dialog when

use clicks on a vehicle. The dialog is basically a form that contains all relative data fields. To solve the driver and vehicle mismatch problem, we decided to put vehicle and driver information altogether in a same scrollable dialog. Therefore, the user will know exactly which driver and vehicle he's dealing with and never be able to get it wrong. However changing view without modifying our model cannot solve the problem as vehicle and driver objects are still different objects. Therefore, we decided to merge our driver object into vehicle object so that a vehicle will be a composition of driver object. By compositing the driver object in its corresponding vehicle object, each vehicle will have its own driver which results in indivisible relationship between them. The screenshot of the pop-up dialog is shown below.



With this approach, user still can enter information for other sections using the menu we provided while the order of entering information for vehicles is completely flexible. Each section will also be displayed as a pop-up dialog that located upon the crash diagram. This intuitive design provides user more control over the application which at the same time could result in unawareness of what has been done and what needs to be done. As a result, the freedom of manipulating the application without specified routine

can make user forget to enter part of the information. Suppose a crash involving five vehicles, the user normally will click on vehicles in a random order to process. Since we combine vehicle and driver information altogether, there will be quite an amount of information for each vehicle. Therefore, after filling information for several vehicles, the user could end up with forgetting to record information for the remaining ones. To avoid this, we need to provide a reminder telling the user whenever there is information that has not been filled. User preference is another issue that we currently cannot locate. As we haven't take surveys of users' preferences for processing TCR, we cannot conclude that this new approach will be well adopted.

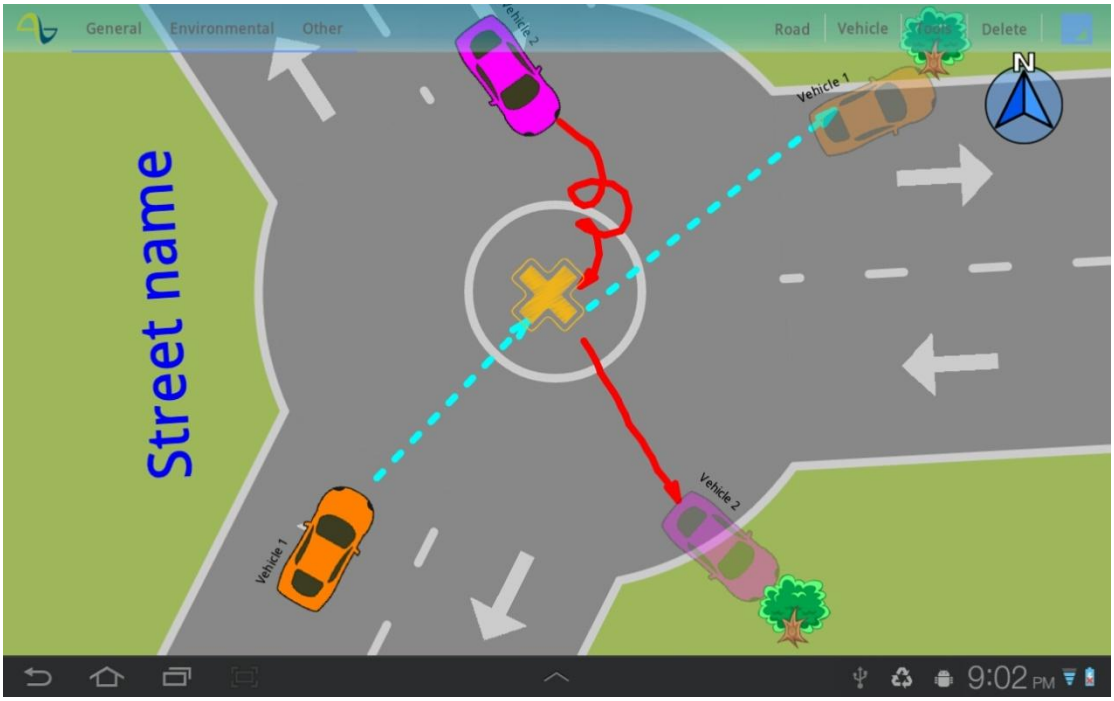
Functions

Several functions have been implemented to assist the intuitive design. While trying to utilize both existing device software and hardware as much as possible, these functions intend to enable a better user experience as well as to consolidate the accuracy of application output.

Crash Diagram

The crash diagram is one of the main features in our project just as it is for the original paper-base TCR. The conventional crash diagram in paper-base TCR is simply a white drawing area which allows polices to sketch freely on it. It has happened sometimes in the past that after the TCR is delivered to NZTA for coding, the drawing is too rough or informal for coders to identify what actually has happened at the crash scene. Also, as the style of drawing varies from person to person, there is generally no consistency. Trying to improve this problem, we decide to implement our crash diagram as object-based rather than a free sketch pad. This means that we need to previously define objects such as streets, vehicles, lines, crash location, street name and objects hit during the crash. Besides, we also use icons to represent these objects as Wilding [16] suggests that Icons can make an application more intuitive and interactive. Therefore our crash

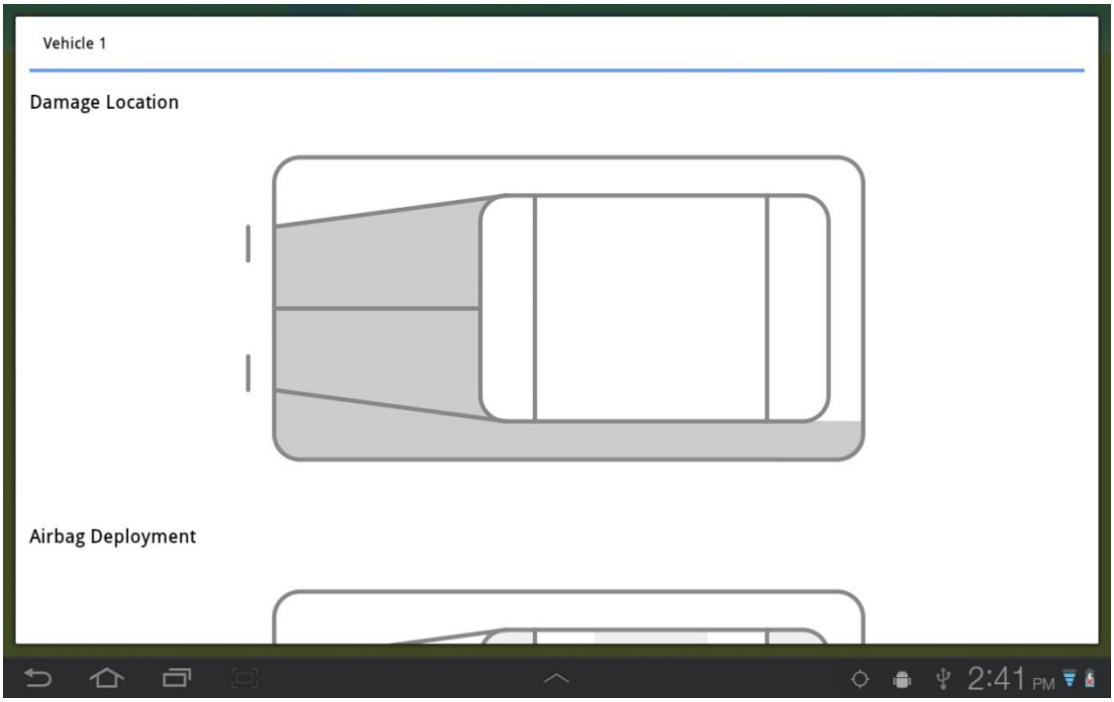
diagram is more like a drawing surface with a toolkit where user can continuously add objects from the menu. This solves both readability and consistency issues as every crash diagram can only contain the objects we defined. A screenshot of a complete crash diagram is given below.

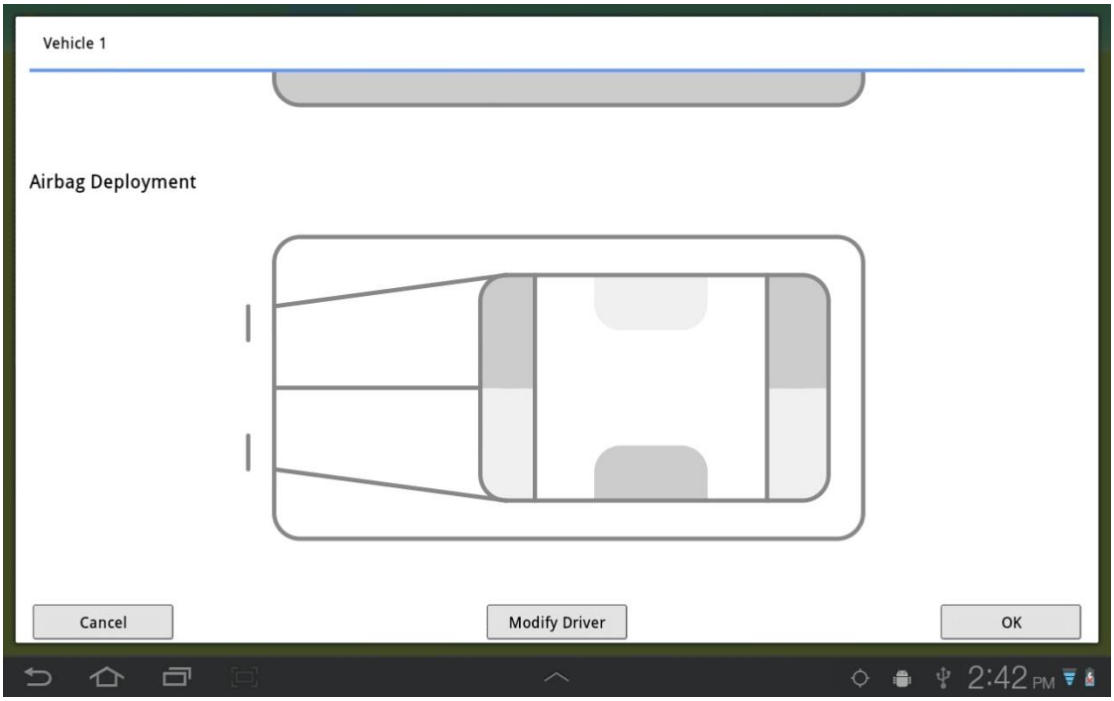


The layout is implemented by extending AbsoluteLayout class with our own override onDraw and onMeasure methods. The AbsoluteLayout is a layout that places its children based on x/y coordinates. This matches exactly to our purpose that each object on the layout is located by either a center point or a series of points indicating the path. When the application's main activity is started, this layout is set as its content view. In this activity, we have an override onTouchEvent method which handles both simple and multi touch events on this layout. This allows the user to move, drag, rotate and delete objects after they are added. Each time the layout is updated, a invalidate call is invoked to refresh the drawing surface. This allows the user to concurrently view the changes when they are manipulating the application.

Damage Location and Airbag Deployment

For vehicle information, it is also required that its damage location and airbag deployment should be indicated by image. Again, this is achieved by defining our own layout which treats each damage location and airbag as object. When user clicks on these areas, a highlight effect will appear to indicate the selection. Screenshots of the two custom layouts are shown below.

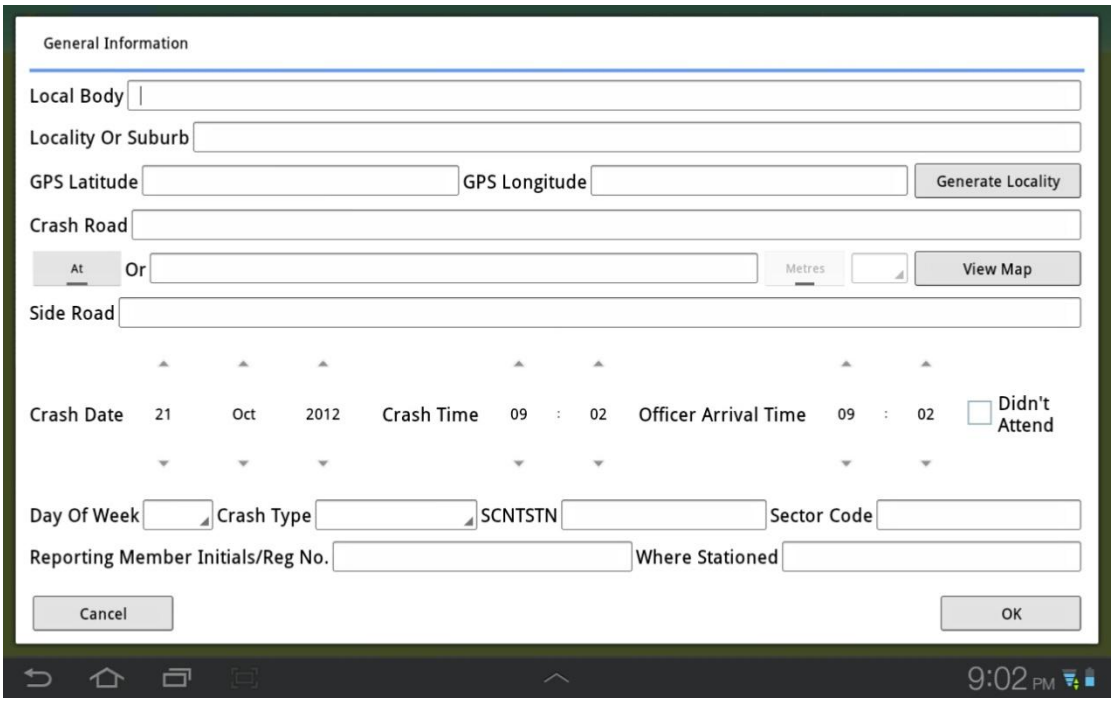




GPS

The form for general information contains a Generate Locality button which will update the associated locality information when pressed. This is achieved by using

LocationManager, Geocoder and our own LocationListener. A screenshot of general information form are given below.

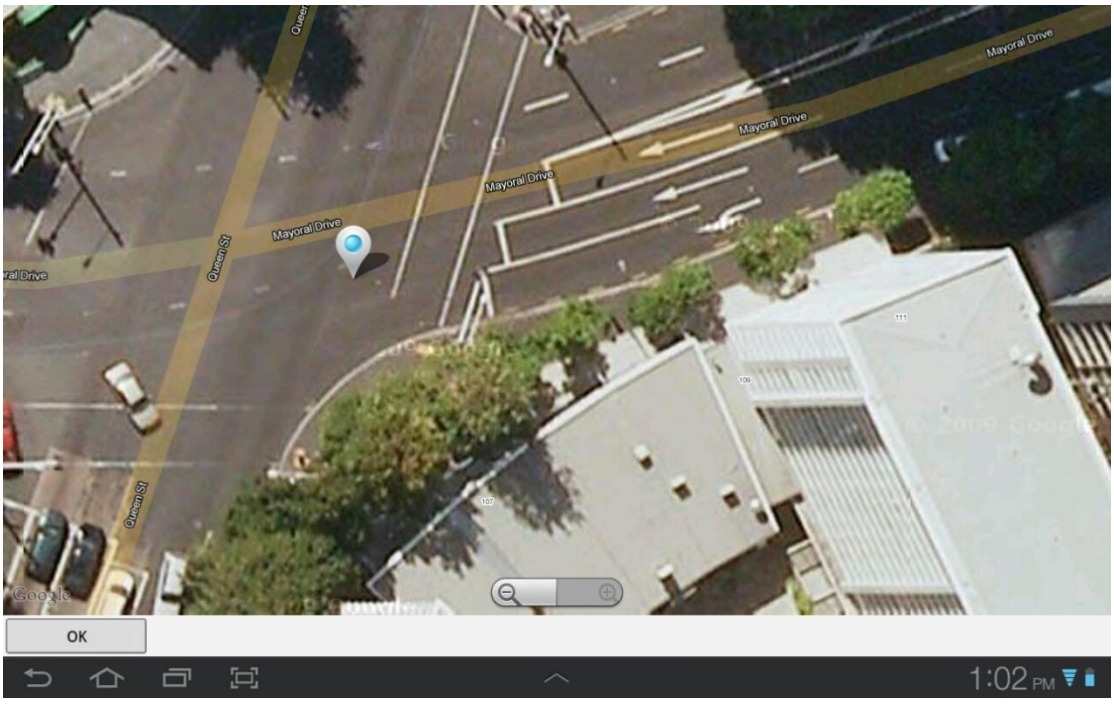


When user pressed the Generate Locality button, a location update will be required and a Location object will be returned. Using this Location object, we are able to access current GPS latitude and longitude. The Geocoder is then used to translate GPS coordinates into addresses which we use to fill in other fields such as Local Body, Locality of Suburb and Crash Road. It is worth to mention here that for location query using GPS provider, the time to first fix (TTFF) can be long. A possible naive solution would be having a background worker consistently requesting for update based on a fixed time interval. Each time the update is completed, the GPS coordinates are saved in associated variables. Therefore, when user pressed the Generate Locality button, the application actually will not send out the request but only retrieves from the variables. Although this approach could save waiting time for user, it has several drawbacks including high power consumption and inaccuracy. The power consumption is caused by sending too many GPS requests. The inaccurate result is mainly generated when user

moves a certain distance within the fixed time interval. In this case when he pressed the button again, what he will get is still the last GPS location.

Google Map

In the form shown above, there is another View Map button which allows user to obtain a visualized image of his current location from Google map. To accomplish this, our MyMapView class extends MapActivity. In addition we also provide our own OverlayItem by extending ItemizedOverlay<OverlayItem>. In order for the user to move the OverlayItem, we also override OnTouchEvent method in our OverlayItem class. A screenshot is given below.

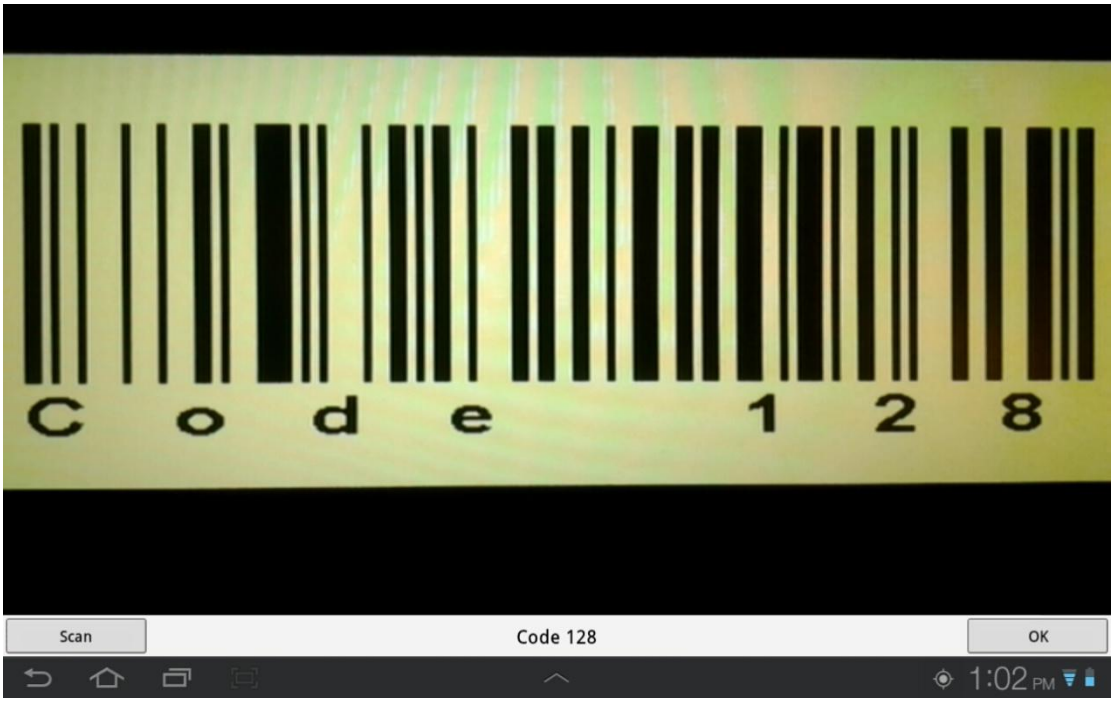


When user clicks on the View Map button, an intent of our map view class will be started for result. This will navigate the user to Google map view. Also, if there is already GPS coordinates filled in the form, they will be put as extra data while starting the intent. These coordinates will be used to create the OverlayItem when the map view is shown. Otherwise, if no data is put in the intent, the map view will start with no OverlayItem and send a request for location update first. Then it will create an OverlayItem according

to the location information it gets. When the intent returns, it will send back the GPS coordinates associated with the OverlayItem if there is one on the map and the according longitude and latitude will be updated. The main reason we implement it this way is because the GPS coordinates obtained using Generate Locality button can mean nothing to user without visualized view of the actual location. Thus by putting this GPS location on Google map, the user is aware of the correctness of these coordinates and if they feel that the coordinates are not accurate, they can manually move the OverlayItem on Google map to change actual GPS location.

Barcode Scanning and Automatic Form filling

In order to allow automatic data processing, we implement a barcode scanning function that allows the user to scan driver license instead of manually entering the information. Currently NZ driver license uses Code 128 and therefore we use a third party library called zbar which supports this code. In addition, we need to start an intent with our own camera activity in order to use camera hardware. This requires creating a layout that contains our camera preview which displays live data from camera. The barcode scanning process involves performing a camera autofocus every 1 second and having a zbar scanner continuously operating while camera preview is active. When the barcode is scanned, camera preview is stopped and user can click Scan button to start it again. A screenshot of this function is shown below.



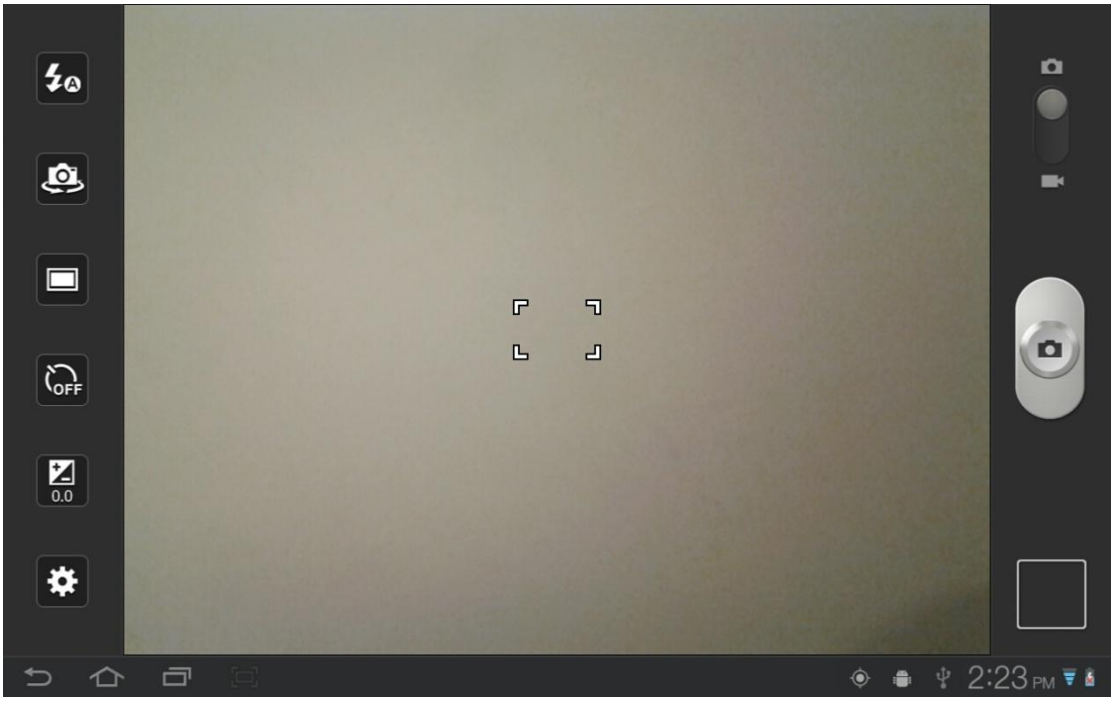
When user clicks Ok buton, the intent will return with the scanned driver license number and a HTTP request will be sent to server for retrieving driver information associated with this number. This brings us to the introduction of next function which is automatic form filling.

Previous discussions have led us to the adoption of RESTful WS which doesn't rely on XML for its data transmission. This flexibility allows us to use JSON format for data querying and XML for uploading our final report. When the driver license number field has been filled either manually or through barcode scanning, an AsyncTask is executed to send an HTTP request to the server for driver information. The server then will respond with the data in JSON format. Afterwards, the information is extracted and associative fields are updated.

Photo Capture and Audio Recording

In addition to textual data in the crash report, it is also suggested that media typed data could be used as a strong evidence. According to this requirement, we provide functions for environment photo capturing and driver interview recording. The photo capturing is

implemented using existing Android camera application and audio recording and replay is done by using MediaRecorder and MediaPlayer. A screenshot of photo capturing function is given below.



The user can use this intent to take as many photos as he wants and the photos are maintained in a Gallery as an ArrayList of bitmap in our own ImageAdapter which extends BaseAdapter.

XML Generation

As we discussed previously, the final submission involves serializing all data into an XML file. This is done by using a third party library called simple-xml. This library provides user convenience for generating XML file without specifically using XML writer. The user only needs to add annotations above variable and method names to indicate which ones should be serialized. Besides, the annotation also allows user to specify the type and tag name for a variable to appear in XML. The ordering of elements is accomplished by using an Order annotation above the class name. A code snippet is provided to help the explanation.

```
@Root(name="report")
@Order(elements={"generalInformation", "vehicleInformation",
"crashDiagram", "environmentalInformation","otherInformation"})
public class Report implements Serializable{
	private static final long serialVersionUID = -4657524837455311101L;

	@Element(name="generalInformation")
	private General general;

	@ElementList(entry="vehicleInformation",inline=true)
	private ArrayList<VehicleData> vehicles;

	@Element(name="crashDiagram")
	private String crashDiagramURI;

	@Element(name="environmentalInformation")
	private Environmental environmental;

	@Element(name="otherInformation")
	private Other other;
	...
```

TCR Workflow

After introducing all components of our application, we now go through the entire workflow of generating a TCR using our application. The crash diagram is shown when the application firstly runs. The user can either starts drawing the crash diagram or filling one of the general information, environmental information or other information. However, he won't be able to fill vehicle information or driver information form without adding the vehicle object on crash diagram. The crash diagram has centralized control over all forms which allows user to fill them in any order. After completing TCR, the user can either save the report locally for further modification or submit it to the server. All

files including XML, driver interview recordings and photos will be saved locally and prepared for submission.

Evaluation

Several meetings with stakeholders and a crash simulation have been proposed to evaluate the usability of our application. During the meetings, stakeholders are asked to manipulate the application without any hints or suggestions from programmers. The result is positive as all of them manage to handle the application in around 10 minutes. This reveals the easiness of use of our application. Also, the intuitive designed and some individual functions such as photo capturing and driver interview recording are proven to be both interesting and useful. The stakeholders are indeed satisfied with the application's performance. Another simulation involves mocking a crash event between two cars and two police simultaneously recording the crash event using our application. Both of them finish in about 15 minutes of time and the result TCRs can hardly contain any inconsistent data. The time taken in this case is much less as it normally takes 30 minutes to 1 hour for police to finish a crash report of two cars using paper-base forms. However, as both police mention that the road templates of our application might not be able to cover all kinds of road shapes, it can be a further work for us.

Conclusion and Future Work

We have introduced an Android application developed for TCR which intends to solve the consistency issues in existing paper-base TCR and to achieve a high efficiency for users. Despite of basic form filling, the application also provides functions which utilize both device hardware and software. RESTful WS has been chosen to allow the transmission of various formats of data over internet. For storing report data locally, we

use binary format by implementing Java Serializable interface. For driver information querying, the response is in JSON format. For uploading our final TCR, XML is used to encapsulate the data. The evaluation of the application shows that the performance is acceptable and therefore it is indeed feasible.

Meanwhile, there are also great possibilities that we can extend the application with further work. Firstly, our application currently doesn't perform enough validation check when user submits a report because it only checks for existence of locality information and vehicle. As in previous discussion of data field priority, we have mentioned that some of the data fields are compulsory. These fields must not be absent in a valid TCR. Therefore, more complex validation and human readable reminder could be implemented to fulfill this job.

Another issue is the possibility of crossing platform. Currently it seems impossible as different platforms use different programming languages. However in Android, since static interface are defined in XML documents which generally is understandable on any mobile platform, one could build an interpreter to translate the interface layout into the layout on another platform based on the XML. This is a hard task as different platforms have different user interface style and dynamic interface update caused by program basically cannot be dealt this way.

Probably one of the most important improvements is to make the application completely work off-line. This requires a design off-line map and map data caching as current easily-embedded Google map doesn't support these features. With such off-line map, the application can work completely without internet access for map viewing. Also for obtaining GPS location, Google map has a Projection API which translates between x/y coordinates on screen and GPS coordinates. This API is used in our Google map for getting the GPS coordinates of the OverlayItem. Similarly, such function can be used in an off-line map so that user doesn't need to send request for GPS location over internet. Therefore, the application can operate completely off-line.

A last challenge is to enable automatic coding for NZTA. Current coding is a manual process which maps the recorded data to a coding sheet. This process can be easily replaced by automatic coding as the process is simply a fixed mapping between data and code. However, for coding crash diagram, the coders need to examine what happened in the crash scene by looking at the graph. For example, a Lost Control Turning Right situation will be represented by code DA. This is a difficult task in our application as it relies on machine learning to make the conclusion. Meanwhile, this kind of machine learning also requires complicated algorithm to achieve a high accuracy. In our application, this approach is feasible as our crash diagram is object-base rather than a simple bitmap drawing surface. This means each object has its own identifier and properties which can be used for machine to make the conclusion. For instance, the lines and arrows are represented by both start and end points which can indicate the direction. Also, we could further attach the arrow to a vehicle to tell which vehicle is going to that direction.

Reference

[1] bsonspec.org. BSON. Retrieved from http://bsonspec.org.

[2] Google Developers. Protocol Buffers. Retrieved from https://developers.google.com/protocol-buffers.

[3] json.org. JSON. Retrieved from http://www.json.org.

[4] MSDN. Deterministic and non-deterministic schemas. Retrieved from http://msdn.microsoft.com/en-us/library/9bf3997x(v=vs.71).aspx.

[5] thrift.apache.org. Apache Thrift. Retrieved from http://thrift.apache.org.

[6] tools.ietf.org. Structured Data Exchange Format (SDXF). Retrieved from http://tools.ietf.org/html/rfc3072.

[7] W3C. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Retrieved from http://www.w3.org/TR/soap12-part1.

[8] W3C. WAP Binary XML Content Format. Retrieved from http://www.w3.org/TR/wbxml.

[9] W3C. XML 1.0 (Fifth Edition). Retrieved from http://www.w3.org/TR/REC-xml.

[10] Aaron Carroll and Gernot Heiser. 2010. An analysis of power consumption in a smartphone. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIXATC'10). USENIX Association, Berkeley, CA, USA, 21-21.

[11] Andre Charland and Brian Leroux. 2011. Mobile application development: web vs. native. Commun. ACM 54, 5 (May 2011), 49-53.

[12] Audie Sumaray and S. Kami Makki. 2012. A comparison of data serialization formats for optimal efficiency on a mobile platform. In Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12). ACM, New York, NY, USA, Article 48, 6 pages.

[13] Brigitte Walther, Safayet Hossin, John Townend, Neil Abernethy, David Parker and David Jeffries. 2011. Comparison of electronic data capture (EDC) with the standard data capture method for clinical trial data. PLoS ONE 6(9): e25348. doi:10.1371/journal.pone.0025348

[14] Bruno Gil and Paulo Trezentos. 2011. Impacts of data interchange formats on energy consumption and performance in smartphones. In Proceedings of the 2011 Workshop on Open Source and Design of Communication (OSDOC '11). ACM, New York, NY, USA, 1-6.

[15] Bryan A. Weber, Hossein Yarandi, Meredeth A, Rowe and Justus P. Weber. 2005. A comparison study paper-based versus web-based data collection and management. Applied Nursing Research vol. 18 issue 3 August, 2005. p. 182-185. Elsevier USA.

[16] Cliff Wilding. 1998. Practical GUI screen design: making it usable. In CHI 98 conference summary on Human factors in computing systems (CHI '98). ACM, New York, NY, USA, 125-126.

[17] Dongsong Zhang and Boonlit Adipat. 2005. Challenges, methodologies, and issues in the usability testing of mobile applications. International Journal of Human-Computer Interaction, 18:3, 293-308.

[18] Dongwon Lee and Wesley W. Chu. 2000. Comparative analysis of six XML schema languages. SIGMOD Rec. 29, 3 (September 2000), 76-87.

[19] James A. Welker. 2007. Implementation of electronic data capture systems: Barriers and solutions. Contemporary Clinical Trials, 28, 329-336

[20] Jason H. Christensen. 2009. Using RESTful web-services and cloud computing to create next generation mobile applications. In Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA '09). ACM, New York, NY, USA, 627-634.

[21] Joe M. Tekli, Ernesto Damiani, Richard Chbeir and Gabriele Gianini. 2012. SOAP processing performance and enhancement. Services Computing, IEEE Transactions on , vol.5, no.3, pp.387-403.

[22] Linda A. Fletcher, Drain J. Erickson, Traci L. Toomey and Alexander C. Wagenaar. Handheld computers. A feasible alternative to paper forms for field data collection. Eval Rev 2003, 27:165-178.

[23] Lionbridge. Mobile web apps vs. mobile native apps: How to Make the Right Choice [White paper]. Retrieved from http://en-us.lionbridge.com/kc/mobile-web-apps-vs-mobile-native-apps.htm

[24] Loel Kim and Michael J. Albers. 2001. Web design issues when searching for information in a small screen display. In Proceedings of the 19th annual international conference on Computer documentation (SIGDOC '01). ACM, New York, NY, USA, 193-200.

[25] Makoto Murata, Dongwon Lee, Murali Mani, and Kohsuke Kawaguchi. 2005. Taxonomy of XML schema languages using formal language theory. ACM Trans. Internet Technol. 5, 4 (November 2005), 660-704.

[26] Matt Jones, Gary Marsden, Norliza Mohd-Nasir, Kevin Boone, and George Buchanan. 1999. Improving web interaction on small displays. Comput. Netw. 31, 11-16 (May 1999), 1129-1137.

[27] Mohsen Anvaari and Slinger Jansen. 2010. Evaluating architectural openness in mobile software platforms. In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (ECSA '10), Carlos E. Cuesta (Ed.). ACM, New York, NY, USA, 85-92

[28] Nurseitov N., Paulson M., Reynolds R, XIzurieta C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science Montana State University -- Bozeman, Montana, 59715, USA

[29] Silvia Artoni, Maria Claudia Buzzi, Marina Buzzi, Claudia Fenili, Barbara Leporini, Simona Mencarini, and Caterina Senette. 2012. Designing a mobile application to record ABA data. In Proceedings of the 13th international conference on Computers Helping People with Special Needs - Volume Part II (ICCHP'12), Klaus Miesenberger, Arthur Karshmer, Petr Penaz, and Wolfgang Zagler (Eds.), Vol. Part II. Springer-Verlag, Berlin, Heidelberg, 137-144.

[30] Steven J. Vaughan-Nichols. 2010. Will html 5 restandardize the web? Computer, vol.43, no.4, pp.13-15.

[31] Thomas Bart. 2003. Comparison of electronic data capture with paper data collection —is there really an advantage? Bus Brief Pharmatech 2003:1–4.