

Bachelor's thesis



**CZECH TECHNICAL
UNIVERSITY
IN PRAGUE**

F3

**Faculty of Electrical Engineering
Department of Measurement**

Car-to-Robot communication using a mobile phone

Lukáš Maruniak

Supervisor: Ing. Michal Sojka, Ph.D.

Field of study: Open Informatics

Subfield: Internet of Things

May 2022

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Maruniak** Jméno: **Lukáš** Osobní číslo: **491941**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra měření**
Studijní program: **Otevřená informatika**
Specializace: **Internet věci**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Car-to-Robot komunikace pomocí mobilního telefonu

Název bakalářské práce anglicky:

Car-to-Robot communication using a mobile phone

Pokyny pro vypracování:

- Cílem práce je vytvořit prototyp integrace „chytrého mobilního robotického asistenta“ s automobilem Škoda.
1. Seznamte se s existujícími komunikačními protokoly pro komunikaci mezi mobilním telefonem a autem (Android Auto, MirrorLink, OBD) a pro tzv. Car-to-X komunikaci (např. ITS-G5, 5G, CAM, DENM, ...). Rovněž se seznamte s komunikačním middlewarem DDS.
 2. Na základě komunikace s firmou Škoda Auto vyberte vhodnou technologii pro propojení mobilního telefonu a auta Škoda. Cílem bude zobrazování informací pro řidiče na palubní desce.
 3. Implementujte na mobilním telefonu komunikaci s mobilním robotem (roverem), který bude monitorovat prostor na ulici a vysílat vozidlům informace. Ke komunikaci bude pravděpodobně použita komunikace DDS (implementace Fast DDS nebo Cyclone DDS) přes 4G/5G mobilní síť.
 4. Ve spolupráci se zaměstnanci Škoda Auto implementujte GUI, které bude informace z roveru zobrazovat na palubní desce auta. Otestujte funkčnost ve vozidlech Škoda Auto.
 5. Výsledky pečlivě zdokumentujte.

Seznam doporučené literatury:

- Intelligent Pedestrian Assistant to Everyone, EIT UM project proposal, 2021
- eProsima Fast DDS Documentation: <https://fast-dds.docs.eprosima.com/en/latest/>
- ETSI EN 302 571 Intelligent Transport Systems (ITS); Radiocommunications equipment operating in the 5 855 MHz to 5 925 MHz frequency band; Harmonized EN covering the essential requirements of article 3.2 of the R&TTE Directive

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Michal Sojka, Ph.D. vestavěné systémy CIIRC

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **31.01.2022**

Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce:

do konce letního semestru 2022/2023

Ing. Michal Sojka, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta

Acknowledgements

I would like to thank Ing. Michal Sojka, Ph.D., for his supervision and help with the problems I encountered. I would also like to thank David Košťál for his help in processing the graphic side of this work and Adrien Michaut for helping with testing. Last but not least, I must thank my family and friends for their support.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Abstract

Road accidents caused by distracted drivers are a serious problem in our modern society. The aim of this work is to develop an Android application that will warn drivers when pedestrians cross the street guarded by a robotic rover. This application uses the Fast DDS library that runs on the phone as native code and uses JNI to communicate with the user interface. The application communicates with the rover guarding the crossing and also with the vehicle whose infotainment unit displays a warning in case of imminent danger. The result is a working prototype for the possible future integration of similar technology directly into vehicle systems. We not only succeeded in fulfilling our task, but we also enriched the community by fixing a bug in the main development branch of the Fast DDS library.

Keywords: Android, road safety, v2x communications, DDS, Android Auto

Supervisor: Ing. Michal Sojka, Ph.D.
místnost: A-517a,
Jugoslávských partyzánů 1580,
Praha 6

Abstrakt

Dopravní nehody způsobené nepozornými řidiči jsou vážným problémem dnešní doby. Cílem této práce je vyvinout Android aplikaci, které bude řidiče varovat před chodcem na přechodu střeženém robotickým roverem. Tato aplikace využívá knihovny Fast DDS, která běží na telefonu jako nativní kód a používá JNI ke komunikaci s uživatelským rozhraním. Aplikace komunikuje s roverem hlídajícím přechod a také s vozidlem, na jehož infotainment jednotce se zobrazí varování v případě hrozícího nebezpečí. Výsledkem je fungující prototyp pro budoucí možné začlenění podobné technologie přímo do systémů vozidla. Podařilo se nám nejen uspět v plnění našeho úkolu, ale také jsme obohatili komunitu o opravu chyby v hlavní vývojové větvi knihovny Fast DDS.

Klíčová slova: Android, bezpečnost v silniční dopravě, v2x komunikace, DDS, Android Auto

Překlad názvu: Car-to-Robot
komunikace pomocí mobilního telefonu

Contents

1 Introduction	1	4.3.3 JNI	26
1.1 Goals	2	4.4 Native Publisher and Subscriber	27
1.2 Thesis structure	2	4.4.1 CarInfo publisher	27
2 Background	3	4.4.2 CrossingInfo subscriber	30
2.1 Android	3	4.5 Networking	33
2.1.1 Applications	3	4.5.1 Local network	33
2.2 Java Native Interface	4	4.5.2 Discovery server over the Internet	33
2.3 Kotlin	5	4.5.3 VPN	33
2.3.1 Kotlin samples	6	4.5.4 Solution	33
2.4 MirrorLink	6	4.6 Supplemental files	33
2.5 Android Auto	8	4.6.1 IDLs	34
2.5.1 Testing	9	4.6.2 Generated files	35
2.6 AA Mirror	10	4.7 Installation guide	35
2.7 OBD-II	10	4.7.1 Compilation from sources	36
2.8 CAN	12	4.7.2 Launch from Android Studio	38
2.9 ELM327	12	4.8 Shapes	39
2.9.1 Reading data from a vehicle	13	4.9 Supporting software	40
2.9.2 Testing	14	4.9.1 Publishers	41
2.10 5G cellular network	14	4.9.2 Subscribers	41
2.10.1 Testing	16	5 Evaluation	43
2.11 DDS	16	5.1 Performance	43
2.11.1 Discovery	17	6 Conclusion	47
2.11.2 Fast DDS	18	6.1 Future work	47
3 Design & Analysis	19	Bibliography	49
3.1 Requirements	19	A Latency Measurements	57
3.2 Selected technology	20		
3.2.1 Backwards compatibility	20		
3.2.2 Programming language	20		
3.2.3 Connection to the vehicle	20		
3.2.4 DDS	20		
3.3 Architecture	21		
3.3.1 Model	21		
3.3.2 Communication	22		
3.3.3 Support components	22		
4 Implementation	23		
4.1 User interface	23		
4.2 Main Activity	23		
4.2.1 onCreate	23		
4.2.2 onDestroy	24		
4.2.3 onConfigurationChanged	24		
4.2.4 onLocationChanged	24		
4.2.5 drawDanger	25		
4.3 Handlers	26		
4.3.1 InfoHandler	26		
4.3.2 CrossingHandler	26		

Figures

1.1 Demonstration of autonomous rover protecting pedestrian crossing [1]	1	5.1 Orange warning screenshot from a mobile phone	44
2.1 JNI native method	4	5.2 Orange warning displayed on vehicle infotainment unit	45
2.2 JNI Interface pointer [2]	5	5.3 Red warning displayed on vehicle infotainment unit	45
2.3 Calling Java method from JNI	5		
2.4 Simple “Hello world” example	6		
2.5 Function with multiple parameters and return value <code>Int</code>	6		
2.6 Variables in Kotlin	6		
2.7 For cycles in Kotlin	7		
2.8 Original Android Auto interface [3]	8		
2.9 Android Auto interface after redesign	9		
2.10 OBD connector pinout [4]	11		
2.11 Measurement of RPM and speed via Car Scanner using OBD-II	15		
3.1 Communication schema	19		
3.2 Android application structure	21		
3.3 Typical Model–View–Controller scheme [5]	22		
4.1 Landscape mode layout	24		
4.2 Portrait mode layout	25		
4.3 Kotlin-side JNI functions in <code>CarInfoPublisher</code>	27		
4.4 Publisher variables	27		
4.5 Init function	29		
4.6 JNI Wrapper for <code>initInfoPublisher</code> function	30		
4.7 Thread scheme	31		
4.8 Attaching current thread	32		
4.9 Global reference creation	32		
4.10 Thread destructor we passed to the <code>pthread_key_create</code> function	32		
4.11 Coordinates structure	34		
4.12 <code>CarInfo</code> message structure	34		
4.13 <code>CrossingInfo</code> message structure	35		
4.14 Generated files	36		
4.15 CMake with added check for API versions	39		
4.16 Shapes user interface	40		
4.17 <code>CrossingInfo</code> publisher and subscriber	42		
4.18 <code>CarInfo</code> publisher and subscriber	42		

Tables

2.1 Descriptions of message fields . .	13
2.2 Descriptions of communication modes	13
2.3 Request message structure	14
2.4 Response message structure	14
2.5 Measurements using Samsung Galaxy A52s	17
4.1 Descriptions of used datatypes [6]	28
5.1 Latency measurement results . . .	44
A.1 Latency measurement with hotspot inside the vehicle	57
A.2 Latency measurement with hotspot outside the vehicle	58

Chapter 1

Introduction

Today, the intensity and demands of transport are still growing. This has negative effects on traffic safety. One important factor is the lack of attention as a significant factor in the causes of traffic accidents. Often trivial and easily avoidable situations can be fatal. Various modern technical approaches are offered as a possible solution to eliminate transport hazards. This thesis is a part of a bigger project of the IPA2X consortium's – to protect pedestrians as the most vulnerable traffic participants. The IPA2X consortium wants to demonstrate that adopting intelligent solution on an autonomous rover will make it possible to improve safety, reduce noise and pollution and promote active mobility. IPA2X consortium promotes walking and allows rethinking the urban space focusing on user-centric instead of vehicle-centric use [1].



Figure 1.1: Demonstration of autonomous rover protecting pedestrian crossing [1]

■ 1.1 Goals

The goal of our work is to create a prototype for vehicle communication with a rover. Instead of the vehicle itself, we will use a mobile phone that will communicate with the rover and also with the car. This allows us to display a warning to the driver without having to intervene directly in the systems running inside the vehicle.

The partial goals are:

- Getting familiar with the existing technologies applicable to this project.
- On the basis of communication with partners select suitable technologies.
- Implement these technologies and create a prototype for communication using a mobile phone.
- Test the functionality in a Škoda Auto vehicle.

■ 1.2 Thesis structure

This bachelor thesis is divided into four main chapters. Each chapter represents a part of our work over the last year.

- Chapter 2 (Background) deals with a search for applicable technologies for this project.
- Chapter 3 (Design & Analysis) deals with selected technologies and application design.
- Chapter 4 (Implementation) describes the parts of the application itself and other auxiliary software that was created during the work.
- Chapter 5 (Evaluation) reports the results of testing the developed solution with a vehicle.

Chapter 2

Background

The first part of this work consists of technology research for this project. We consulted the project partners with the research results and selected the most suitable options (see section 3.2). In the following sections, we present the results of the research and introduce individual technologies.

2.1 Android

Android is an open-source, Unix based mobile operating system developed by Google. It targets touchscreen devices such as mobile phones and tablets primarily. Its core (AOSP) is free and open-source.

Android was originally developed by Andy Rubin, Rich Miner, Nick Sears and Chris White. In October 2003, they founded Android Inc. in Palo Alto, California, USA [7]. The first intentions for Android were that it should be an operating system for digital cameras. However, later they changed their target and directed Android to mobile phones. Later in 2005, Android Inc. was acquired by Google [8].

The first Android mobile phone was HTC Dream [9]. It was released in June 2009 with Android 1.0. New devices and new versions soon followed. Notably, Android 2.3 Gingerbread, which powered such classics as Samsung Galaxy S [10], HTC Desire S [11] or the entry-level Samsung Galaxy Y [12]. New versions were released at least yearly. Notable versions are 4.x (Ice Cream Sandwich, Jelly Bean and KitKat) from 2011–2014, Android 7 (Nougat) from 2016, Android 8 (Oreo) from 2017, and Android 11 (from 2020). The latest version is Android 13, which will be released in 2022 [13].

2.1.1 Applications

Android applications are developed using the Android SDK¹, which contains the Android API. It is a set of instructions and standards. Google addresses API compatibility using versions. Each version of Android has its respective API revision according to the capabilities of that version of Android [14].

¹Software development kit - collection of software development tools in one installable package

Initially, Android applications were written in Java, but since 2019, Kotlin (see fig. 2.3) has been the primary language. In addition to these languages, it is also possible to use native C/C++ code using JNI (see fig. 2.2). Individual applications consist of so-called activities. We can describe them as individual “screens” of applications the user communicates. Usually, one activity is for single functionality [15].

Applications have traditionally been distributed in the form of `apk` packages, but today the industry is moving to the more economical `aab` [16]. However, the user rarely encounters the package itself. The search and installation of applications is usually done using the app store. Dominant is the Google Play store. An app store directly from Google. There are already over 3.48 million [17] apps on Google Play.

2.2 Java Native Interface

Java Native Interface (JNI) is a technology that enables the connection of Java and native code (namely C, C++ and even assembly). It can be helpful when a programmer needs to implement a portion of time-critical code, when Java does not support required features, or when existing code needs to be integrated into the Java project [18]. Another typical usage is mobile gaming. Graphics engines such as Unity [19] or OpenGL [20] use JNI.

Java code can call native function, and native function can call back Java code. Native code can create and work with Java objects and even modify existing ones. A disadvantage of using JNI is losing Java’s multiplatform abilities [21]. Significant overhead hides in resources marshalling and callbacks from native code to Java. It is both slow and error-prone. Recommended practice is to reduce such interaction to a minimum [22].

```
extern "C"
JNIEXPORT jboolean JNICALL
Java_cz_cvut_fel_marunluk_ipa2xwarning_CrossingHandler
↪ _killCrossingSubscriber(JNIEnv *env, jobject thiz, jlong
↪ pointer) {
    // do some stuff
    return true;
}
```

Figure 2.1: JNI native method

Native functions called from Java must be in a particular form (see fig. 2.1). Such form is called a wrapper, and it provides 1:1 mapping of native functions. These functions always take one special parameter. It is called `JNIEnv`, the “JNI Interface Pointer” – a pointer to pointers in a table, pointing to JNI interface functions. (see fig. 2.2) This `JNIEnv` pointer is valid only for the current thread [23]. When interacting with Java from a different

thread, that thread must be registered with JVM. This is done by calling `AttachCurrentThread()`. It is possible to call `AttachCurrentThread()` from the same thread multiple times. Subsequent calls perform no operation. However, before exiting, all manually attached threads must be detached. This can be done either by calling `DetachCurrentThread()` manually or by setting thread “destructor” using `pthread_key_create()` [24].

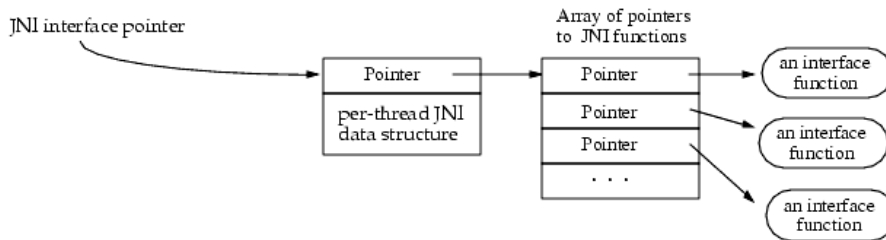


Figure 2.2: JNI Interface pointer [2]

Calling native methods from Java is not difficult, but the opposite direction (Java methods from native code) might be required. Calling in this direction is possible but not as straightforward. It is done through JNI Interface functions provided through `JNIEnv`. To call a Java method, reference to the method itself and to the class it belongs to is needed. Java Class reference is in the form of `jclass`, and it is reachable through the function `FindClass()`. Method reference, `jmethodID`, is provided by `GetMethodID()`. In the case of a non-static method, its associated object reference is also necessary (represented as a `jobject`). [2]. Non-void method call is demonstrated in figure 2.3.

```
jclass lClass = env->FindClass("cz/cvut/fel/marunluk/
↳ ipa2xwarning/CrossingHandler");
jmethodID warningMethod = env->GetMethodID(lClass,
↳ "parseCrossing", "(ZZDD)V");
env->CallVoidMethod(object, warningInfoMethod,
↳ <params>)
```

Figure 2.3: Calling Java method from JNI

2.3 Kotlin

In 2011, JetBrains announced the development of a new programming language, which was released on February 15, 2015 under the name Kotlin. Kotlin is an open-source cross-platform statically typed programming language distributed under the Apache 2.0 license. Kotlin can run both natively (Kotlin/Native), compiled into JVM bytecode (Kotlin/JVM) or to Javascript (Kotlin/JS). The design took into account 100% interoperability with the

Java language. Therefore, it is possible to use libraries written in Java in Kotlin and vice versa [25].

At the I/O 2019 conference, Google declared Kotlin the official language for developing mobile applications on the Android platform. Thanks to interoperability with Java, the transition is simple and allows developers to combine components written in Java and Kotlin [26]. Although Kotlin's syntax is similar to other languages, it is different enough to be confusing for those switching from traditional programming languages such as C. The following snippets demonstrate Kotlin syntax (see figs. 2.4, 2.5, 2.6 and 2.7).

2.3.1 Kotlin samples

```
fun main() {
    println("Hello world!")
}
```

Figure 2.4: Simple “Hello world” example

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Figure 2.5: Function with multiple parameters and return value Int

```
val a: Int = 1 // immediate assignment
val b = 2      // `Int` type is inferred
val c: Int    // Type required without initializer
c = 3         // deferred assignment
c += 4        // invalid -> val cannot be reassigned

var x: Int = 5 // `Int` type is inferred
x += 1         // valid -> var can be reassigned
```

Figure 2.6: Variables in Kotlin

2.4 MirrorLink

MirrorLink is one of the technologies that allow mobile phones and cars to communicate. It is an open standard connecting allowing access to applications running on a mobile phone through vehicle infotainment. Created by the Nokia development centre, it is now maintained by CCC (Car Connectivity Consortium). Today, it is a declining technology. South Korean

```

if (i in 1..4) { // equivalent of 1 <= i && i <= 4
    print(i)
}

repeat(2) { index ->
    println("Cycle no. ${index + 1}")
}

for (i in 1..4) {
    print(i)
}

for (item in items) {
    println(item)
}

```

Figure 2.7: For cycles in Kotlin

Samsung traditionally supported this technology in its devices, but even this manufacturer ended support in 2020 (Galaxy S8) [27][28]. On 8th September 2021, CCC announced that they are shutting down MirrorLink operations by 30th September 2023 [29].

MirrorLink must be supported both on the mobile phone and infotainment unit of the vehicle. When connected to the onboard system via a USB cable, the phone automatically pairs. The system automatically verifies the certification required to run the application in the MirrorLink system when the application starts [30].

The CCC sets out the requirements for applications to follow in order to obtain a certificate. It is a matter of traffic safety – the driver must not be distracted. CCC distinguishes between two levels of certification. One allows the application to be used in park mode (when the vehicle is not moving), and the other while driving (drive mode). The requirements [30] are:

- **Display Minimal Text** – do not display a lot of text or complex messages
- **Do not use Text Input** – Though allowed in some regions it is a bad practice in drive mode. Use park mode for text entry or voice input if possible.
- **Use Park Mode for Additional Functionality** – Since a MirrorLink app is aware if it is in drive or in park mode, you can offer additional features or information to the driver while stopped. When the car goes into drive mode you switch to the simpler UI.
- **Use High Contrast** – The UI must be legible in a variety of lighting conditions, even in direct sunlight. To achieve this, all foreground/background text combinations must have a minimum contrast level.

2. Background

- **Use Big Fonts** – Large font sizes are required in order to be easily readable on all displays. Use a standard Android stock font for good legibility. Note that the minimum size will be measured on the display of the in-vehicle screen.
- **Use Giant Buttons** – buttons must be large enough to be usable during driving. The minimum edge length for a button is 10 mm with an area of 200 sq mm.

2.5 Android Auto

Android Auto is an application that allows running mobile phone applications on the infotainment unit in the car. Google presents this service as an easy way to use applications on the vehicle display. However, all applications are subject to strict safety regulations and can only enter the Google Play Store (as the only official software source on Android Auto) when certified by Google. A mobile phone with the Android operating system (version 6.0 and higher) and the Android Auto mobile application are required to connect to the vehicle. Phones with Android 10 and later already have the Android Auto app integrated. Furthermore, this technology must be supported by the other party, i.e. the vehicle's infotainment system or multimedia unit (there are car radios that support Android Auto and can be retrofitted to older vehicles). Most existing devices require a USB cable connection, but some vehicle/phone combinations can also support wireless communication over a WiFi connection [31].

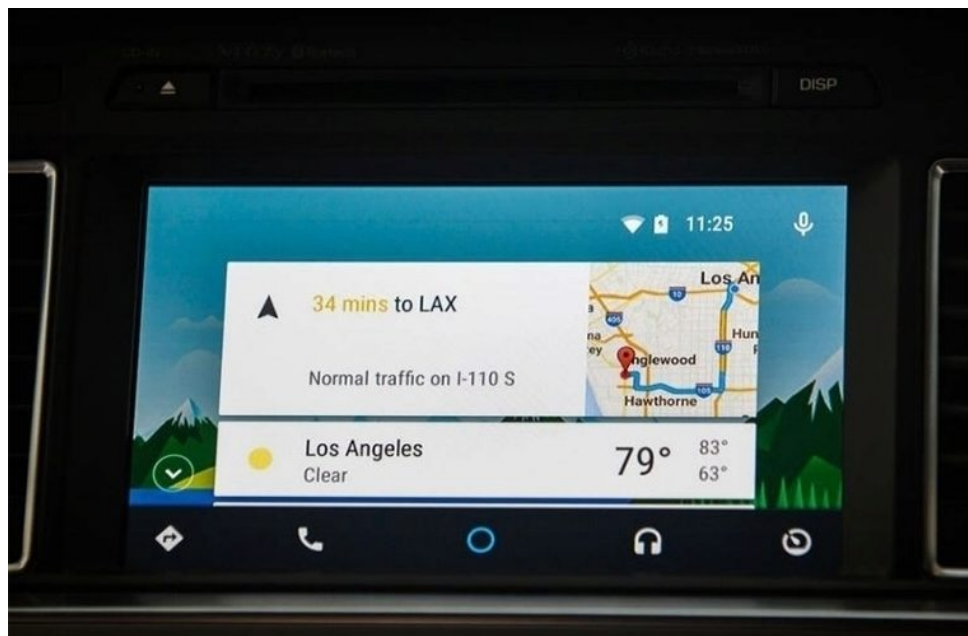


Figure 2.8: Original Android Auto interface [3]

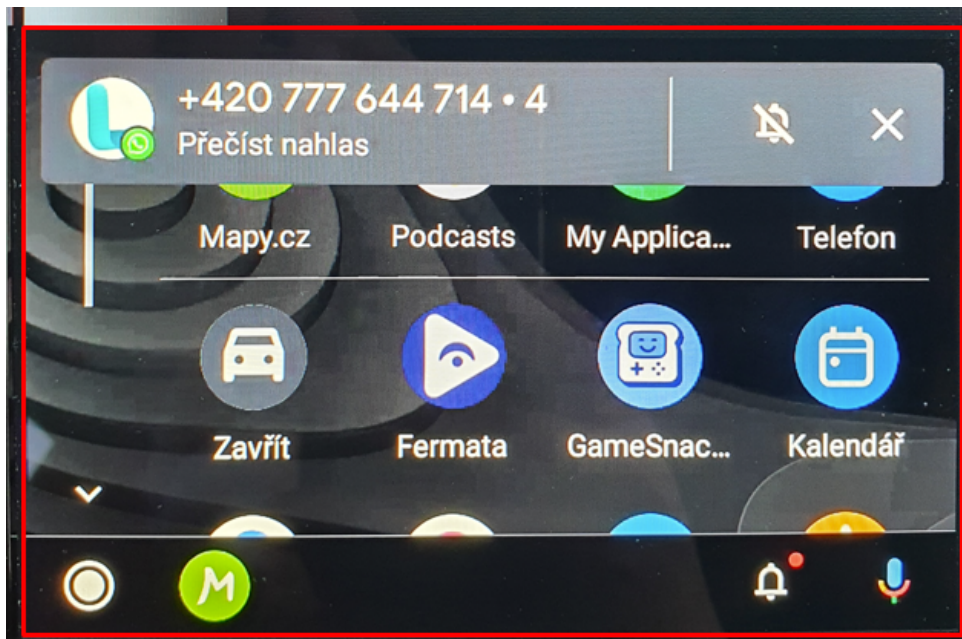


Figure 2.9: Android Auto interface after redesign

In 2019, Google came up with a significant user interface redesign. Instead of Google Now-style categories and tabs (see Fig. 2.8), Android Auto now resembles a standard interface on a mobile phone or a competing Apple CarPlay (see Fig. 2.9). User interaction with the system is severely limited for safety reasons [32]. Calls or music playback work normally (in most modern cars, this is already handled by the onboard infotainment system), but notifications or other information are only available through Google Assistant [33]. Although the system displays the notification and concentrates it in the “notification centre”, the only option is to have it read through the voice assistant. In the same way, answering is possible only using voice [34].

■ 2.5.1 Testing

The goal of testing was to test the applications and functionalities of the Android Auto system. Testing took place on Samsung Galaxy A52s and Samsung Galaxy S10e mobile phones. The tested applications were Google Maps, Mapy.cz, Phone, SMS, WhatsApp and Facebook Messenger. Testing took place on two Volvo XC90 cars (MY2019² and MY2020).

It turned out to be essential during testing to have enabled Google Assistant, to which Android Auto is linked. Android Auto will not work as expected if information settings are further restricted in the name of privacy. However, even if the user approves all requests, the system response is insufficient. The system announces notifications with a voice assistant. However, its interaction causes problems and reacts poorly to user input. We tried to set up and speak with the assistant in Czech and English. The relatively popular Facebook

²Model year 2019

Messenger application was a surprising failure, which stopped working for more users around the turn of 2020 and 2021 [35][36].

The Android Auto user experience is poor, and despite Google's commitment to traffic safety, I was overly distracted by the system's problems, even though I drove about 10 km/h on an empty road. In addition to this, the integration of this system closer to the voice assistant, the inability to run user interface applications without significant restrictions, and the need for certification from Google make the Android Auto system relatively unsuitable for the planned use in this project.

■ 2.6 AA Mirror

AA Mirror is an Android application that enables mirroring of Android phone screen to the infotainment unit of a car that runs Android Auto. But, applications providing full screen mirroring are unacceptable service for Google and its safety policies [32]. This application lets users watch YouTube on the infotainment screen while driving. This results in constant efforts from Google to stop such applications. Developers try to fight back, but the outcome is a product with intermitted functionality and an uncertain future. As of May 2022, the only working way to get AA Mirror working is through AAAD – an installer for Android Auto application. Neither AA Mirror nor AAAD is in the Google Play store, and they are not even in any reputable alternative app sources such as F-Droid. This casts a bad light on these applications and their legitimacy. Furthermore, the possibility of this application stopping working is likely. Unfortunately, we do not yet know of a better screen mirroring solution.

■ 2.7 OBD-II

OBD-II (On-Board Diagnostics) is a vehicle diagnostic protocol and interface. It is an improvement over its predecessor – OBD-I. OBD-II allows service technicians worldwide to access vehicle condition data and quickly diagnose faults for which a mere engine fault indicator light is not enough. The system was standardized in 1994 and has been a mandatory part of all passenger cars sold in the United States since 1996. In 1998, the obligation to equip cars with this system was extended to Canada. In 2001 EU required this protocol in petrol cars and since 2004 in diesel cars [37]. The standard defines five communication protocols, of which the vehicle must support at least one. Those protocols are [4]:

■ CAN (ISO 15765-4)

- It is used in most cars today
- Since 2008, its support has been mandatory in cars sold in the USA

- **KWP2000 (ISO 14230-4)**
 - To be found usually in cars manufactured after 2003 in the USA and Asia
- **VPW (SAE J1850) & PWM (SAE J1850)**
 - Today used rarely
 - Used in older Ford and General Motors vehicles
- **K-Line (ISO 9141-2)**
 - Used mainly in cars from EU countries from 2000-2005
 - It is the European equivalent of the KWP2000 protocol

Standard SAE J1962 also defines a standard connector for such diagnostic devices (see Fig. 2.10). The connector exists in two variants, “A” and “B”. They differ from each other in communication speed (baud rate) and especially in power voltage, where variant A (designed for passenger cars) is powered by +12V and variant B (designed for heavier vehicles) has a power supply of +24V. There are 16 pins, of which pins 16 (power +), 4 (ground), and 5 (power -) will undoubtedly be used. Other pins can be used depending on the supported protocol. Most likely, pins 6 and 14 will also be active on which communication via the CAN bus occurs. Unspecified pins might be used at the manufacturer’s discretion for their own communication solutions [38].

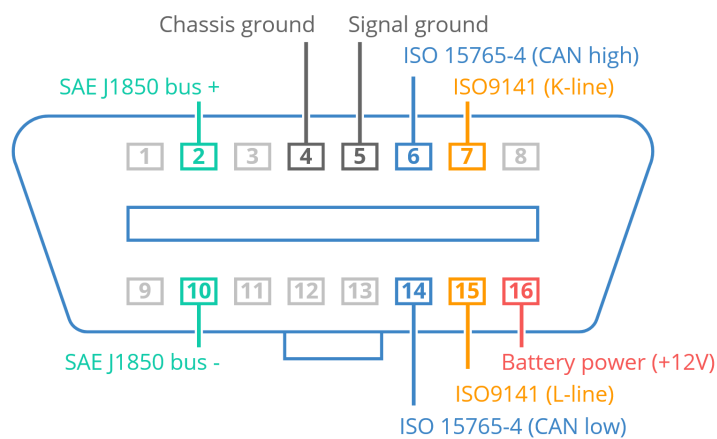


Figure 2.10: OBD connector pinout [4]

Although the OBD-II was designed primarily to assist technicians in diagnosing vehicle faults, the ability to read up-to-date driving information has helped expand all sorts of devices for ride monitoring or monitoring technical vehicle parameters. Unfortunately, this does not appeal to automakers, which are increasingly inclined to restrict and close their vehicles. According to Christoph Grote (BMW Vice President of Electronics), the German carmaker BMW plans to disable OBD-II communication while driving [39].

2.8 CAN

The CAN (Controller Area Network) bus was developed by Bosch and introduced in 1986. First implemented was in the Mercedes-Benz W140 (1991) [40]. The primary motivation was to get rid of a complicated model with many wires connecting directly all units that needed to communicate with each other. Thanks to low price, reliability and simplicity, the CAN bus has spread from cars to other means of transport, including trucks, trains and airliners. Apart from transport, the CAN bus is also popular in industrial automation and can be found in some operating rooms [41].

Via the CAN bus, nodes (individual components) can communicate serially at speeds of up to 1 Mbit/s using broadcast packets [42]. Each node evaluates whether the message is addressed to it and whether it is not corrupted (CRC check). Each message also has its priority, which ensures that a large number of less critical messages won't block the delivery of critical information. For example, fuel injection errors must be delivered as soon as possible, while outdoor temperature change information can wait [38].

Today, CAN FD is primarily used in vehicles. It is an extension of the classic CAN bus. The main benefits are larger data frames, higher speeds (up to 5–15 Mbit/s) and better reliability. In addition, backward compatibility allows cars to move to a new standard gradually [43].

2.9 ELM327

ELM327 is a chip developed by ELM Electronics, and it is a converter between OBD-II and RS232 communication. It supports 9 OBD-II protocols, including all five mandatory ones [38]. Due to this chip's incredible popularity and broad applicability, many cheap clones arose (especially from Chinese manufacturers). We can find them in many devices at a lower price than the chip itself from ELM Electronics. When buying more than 1000 pieces, one original ELM327 chip costs ~12US\$ [44], but one finished Bluetooth dongle is priced at ~4US\$).

I bought such a small ELM327 adapter with Bluetooth connectivity. These devices are popular among car enthusiasts and do-it-yourselfers. They use those devices when servicing their vehicles at home or use them to collect driving data. Thanks to the broad adoption, many applications for OBD communication using such devices are on the market. However, according to internet discussions, the quality and compatibility of these low-cost devices vary from piece to piece.

2.9.1 Reading data from a vehicle

The adapter communicates with the vehicle using OBD2 messages with a precisely defined structure. Each message begins with a CAN ID (an identifier that determines whether it is a request or response). It is an 11-bit field containing the value 7DF for requests and 7E8–7EF for responses (see Table 2.1) [4][45]. For example see Tables 2.3 and 2.4.

Identifier	Size	Mode	PID	A	B	C	D
------------	------	------	-----	---	---	---	---

Field	Description
Identifier	message identifier (7DF = request, 7E8–7EF = response)
Size	message size (in bytes)
Mode	communication mode (01–0A for requests, 41–4A for responses)
PID	standardised parameter ID ³
A–D	values

Table 2.1: Descriptions of message fields

Mode	Description
01	Show current data
02	Show freeze frame data
03	Show stored Diagnostic Trouble Codes
04	Clear Diagnostic Trouble Codes and stored values
05	Test results, oxygen sensor monitoring (non CAN only)
06	Test results, other component/system monitoring (oxygen for CAN only)
07	Show pending Diagnostic Trouble Codes (detected during recent driving cycle)
08	Control operation of on-board component/system
09	Request vehicle information
0A	Permanent Diagnostic Trouble Codes (DTCs) (Cleared DTCs)

Table 2.2: Descriptions of communication modes

Example

7DF	02	01	0D	XX	XX	XX	XX
-----	----	----	----	----	----	----	----

7DF	outgoing request
02	message size (bytes)
01	show current data
0D	current speed
XX	unused

Table 2.3: Request message structure

7E8	03	41	0D	40	XX	XX	XX
-----	----	----	----	----	----	----	----

7E8	incoming response
03	message size (bytes)
41	current data response
0D	current speed
40	0x40 = 64 km/h
XX	unused

Table 2.4: Response message structure

2.9.2 Testing

The purpose of testing the ELM327 dongle was to verify the reading of values from the vehicle. We tested it using cheap ELM327 clone with the Samsung Galaxy A52s phone and on the Volvo V70 (MY2001) and Volvo XC70 (MY2015) cars. Among the applications used were AndrOBD, Tourque Lite, Car Scanner and Obd Army. We encountered difficulties connecting to the dongle, and application compatibility was not 100%. Otherwise, readings were as expected. See fig. 2.11.

2.10 5G cellular network

5G networks represent the 5th generation of mobile data networks. 5G networks were first deployed in South Korea by SK Telecom [46]. Compared to the previous generation, 5G brings a significant increase in speed and a decrease in latency. Theoretically, speeds can reach 20 Gbit/s, and the latency can be in the range of milliseconds. It is essential in the industrial

use of this technology. For example, in autonomous transport or industrial automation (local networks covering the production plant or logistics centres). 5G networks also provide more significant potential to serve a large number of clients connected to a single station—for example, a sports match or several IoT devices in a building [47].

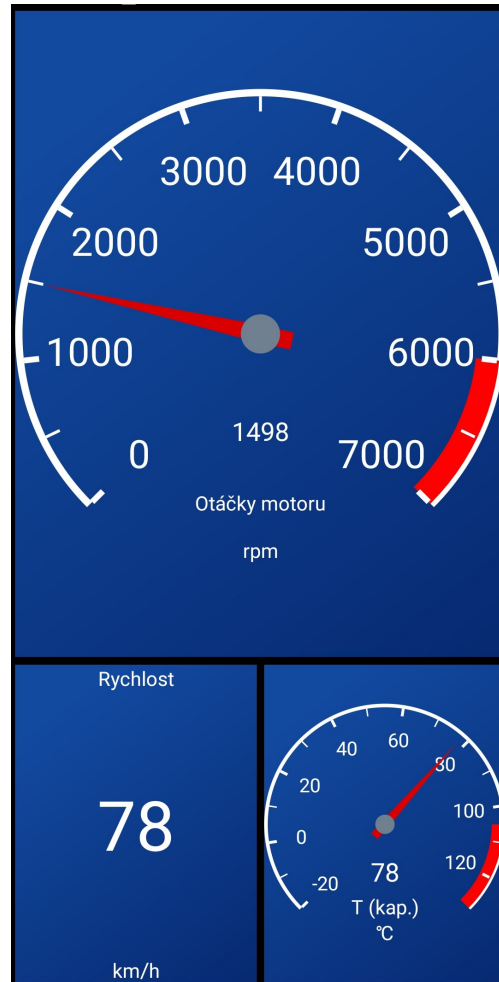


Figure 2.11: Measurement of RPM and speed via Car Scanner using OBD-II

We divide 5G networks into three categories depending on the frequency used (and the resulting speed) [48][49].

- **Low-band** (<1 GHz) It is similar to the existing 4G/LTE and is designed to cover larger areas with lower speeds (relative to other bands – it still beats older generations)
- **Mid-band** (2.3–4.7 GHz) frequency is in a similar range as Wi-Fi technology. Both in the 2.4GHz and 5GHz bands. It is designed for small and medium-sized cities, providing a balanced coverage:speed ratio. 5G in these bands can reach speeds between 100 Mbit/s–1 Gbit/s. Most 5G networks deployed today are of this category

- **High-band** (24–47 GHz) Uses frequencies, which are already in the lower range of millimetre waves. This category offers very high speeds and low latency. They are used mainly in large cities and places where it is necessary to serve a large number of clients.

5G networks also present specific problems. One of the obstacles is the lower distance that one transmitter can cover. Therefore, it is necessary to build a much denser network of transmitters, which results in higher infrastructure prices [49]. It is also possible to monitor the movement of devices (and thus their users) much more accurately than in the case of networks of previous generations. Together with the fact that critical infrastructure (Integrated Rescue System) will soon likely rely on 5G networks, it forces NÚKIB (National cybersecurity office of the Czech Republic) to consider 5G networks as critical infrastructure⁴ [50]. The US government reached the same conclusion and therefore decided to make it impossible for Chinese company Huawei to supply any 5G-related technology to the US [51].

Concerns about the health risks of 5G networks are also a significant problem. Although the health problems associated with 5G networks are at least heavily disputed, supporters of these conspiracy theories, convinced of the harmful nature of 5G technology, sometimes go very far in their fight against 5G. There are several recorded cases of arson directed against 5G transmitters [52][53][54].

■ 2.10.1 Testing

We first tested the 5G network in the CIIRC lab on Samsung Galaxy S21 5G. The phone has connected to the local network without any issues. However, it was impossible to connect the Samsung Galaxy A52s 5G phone to this closed network due to incompatible firmware. So we got a 5G Vodafone SIM card that could be used outside the school lab.

The coverage of 5G internet in Prague is not dazzling. All transmitters use the 2100 MHz band, i.e. mid-band. Vodafone promises speeds of up to 330 Mbit/s, but speeds of around 100 Mbit/s are most common. However, sometimes happen that 4G is a better choice in signal strength and even internet speed (see Table 2.5). In these cases, the phone automatically selects the 4G network connection. When we try to force the phone to switch to 5G, the speed on the 5G network is lower than on 4G.

■ 2.11 DDS

DDS is a datacentric, event-driven middleware protocol developed by OMG (Object Management Group) standards consortium. It provides low-latency, reliable data sharing between many endpoints [55]. Thanks to the absence of a centralised controller, DDS is easily scalable and lacks the single point

⁴Critical infrastructure is a term used by governments to describe assets that are essential for the functioning of a society and economy

Measurement	Download Mb/s	Upload Mb/s	Ping ms	Jitter ms
Dejvice 5G	27.9	18.7	25	105
Dejvice 4G	42.3	17.9	62	39
D. Břežany 5G	97.0	8.45	17	7
D. Břežany 4G	77.5	9.4	21	8

Table 2.5: Measurements using Samsung Galaxy A52s

of failure – a central controller. DDS eliminates the need for complicated communication development and allows the developer to focus directly on the application and takes care of communication through many supported means of transport [56]. Communication is possible via UDP, TCP or even shared memory.

Its event-driven publisher/subscriber structure reduces application and integration complexity and ensures the performance required for real-time applications. Furthermore, its “global data space” makes it seem like you are accessing local memory via an API. On the back-end, DDS manages to transport data to its destination. Communication happens on domains. One registers into the domain as a publisher, and the application that wants such data becomes a subscriber. Then when the publisher sends a message (updates data), it is delivered to all subscribers. Thanks to isolated domains, communication can be separated from other traffic, avoiding overcrowding or even collision. Endpoints can communicate only when being in one domain [57].

DDS enables custom settings for liveness, reliability and security. It can be configured in QoS (Quality of Service) based on the current application requirements. Not all endpoints need all messages. DDS can dynamically filter data and modify transport reliability as the situation requires [57].

■ 2.11.1 Discovery

Dynamic Discovery greatly simplifies the issue of finding other counterparts. It ensures automatic discovery of endpoints, whether they are subscribers, publishers, and what communication parameters they provide. This means that DDS supports “Plug’n’Play” for connecting new endpoints [57]. Discovery consists of two parts. Participant discovery and Endpoint discovery. In Participant discovery, participants recognize other participants [58].

By default (defined by DDS), each participant sends “announcement messages” in which it announces its IP address and the port on which it listens. The two participants match when they catch their message and are in a common domain. These messages are (like most of this protocol) configurable via QoS [58]. Fast DDS, one of DDS implementations (see 2.11.2), also offers another type of discovery. The so-called Discovery server. It is a discovery mechanism with a central element [59]. Individual participants can be:

- **SERVER** – Concentrates and redistributes DDS Discovery information
- **BACKUP** – Backs up server information to the database
- **CLIENT** – The participant receives only the necessary information from the server
- **SUPER_CLIENT** – The subscriber who receives all the information from the server

Endpoint discovery is the part where already paired participants share their topics and decide whether to start communication.

■ 2.11.2 Fast DDS

Fast DDS is an implementation of DDS. It comes bundled with Fast DDS-Gen. Java tool for generating interface code between application and DDS middleware. It generates code based on data types defined in .idl file. IDL, being developed by OMG, is a descriptive language used to define data structures that will be communicated via DDS [60].

Chapter 3

Design & Analysis

In this chapter, we analyze the assignment and present overall design of the final mobile application as well as helper tools.

The mobile application communicates with the rover, which guards the pedestrian crossing (see Fig. 3.1). The moment the rover detects that the crossing state has changed, i.e., pedestrian entered the crossing, it sends a message to the surrounding cars. The rover distinguishes three states of the crossing: free, crossing, and danger. The mobile phone will display an appropriate warning on the infotainment unit based on crossing status. The phone also sends back information about the speed and position of the vehicle.

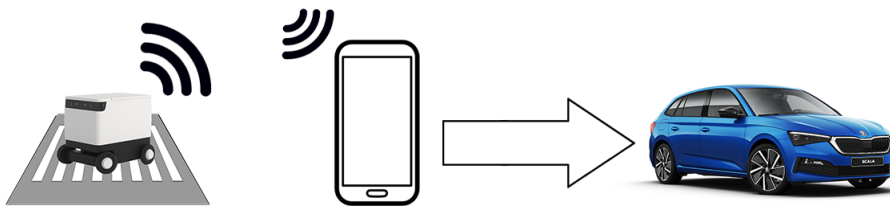


Figure 3.1: Communication schema

3.1 Requirements

Here are listed the application requirements:

- Receive information about crossing status
- Notify the driver visually and audibly about impending danger
- Send current information (position and speed) about the vehicle
- Use Android OS
- Use DDS communication protocol
- Display warnings on the onboard infotainment system

3.3 Architecture

The overall architecture of the project is depicted in Fig 3.2. The rover uses DDS to send pedestrian crossing status information to the application. This message is received by the native part of the application via Fast DDS and forwarded to the user interface using JNI. A warning animation appears on display, and an audible alert sounds. This image is further transmitted using the AA Mirror application to the display of the vehicle's on-board unit. The application also informs the rover about the current position and speed of the vehicle. Currently, the source of information about the location and speed is the cell phone. Assuming that the user will place it in the vehicle, the information it provides is sufficient. It is planned to obtain speed information directly from the car in the future.

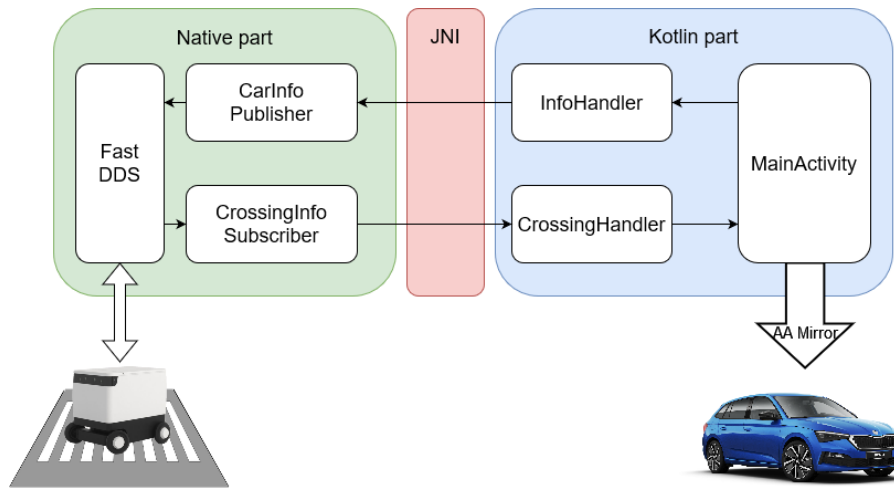


Figure 3.2: Android application structure

3.3.1 Model

The application consists of three main parts that follow the Model-View-Controller layout (see fig. 3.3). This layout is natural to Android applications and is a common approach.

The most important part is the **Model** part. In this case, it consists of two layers. The Kotlin layer manages and communicates with the native layer, and together they cater to the application's external communication using the Fast DDS library, which runs as a native C++ code under JNI. This JNI interconnection is specific for this application. When sending vehicle information, the Kotlin layer calls the native layer. On the contrary, the Kotlin layer receives a callback from the native layer when receiving information.

The last part, the **Controller**, besides connecting the previous two parts, also handles the retrieval of the vehicle's position and velocity. The **View** (user interface) defines the GUI elements and layout.

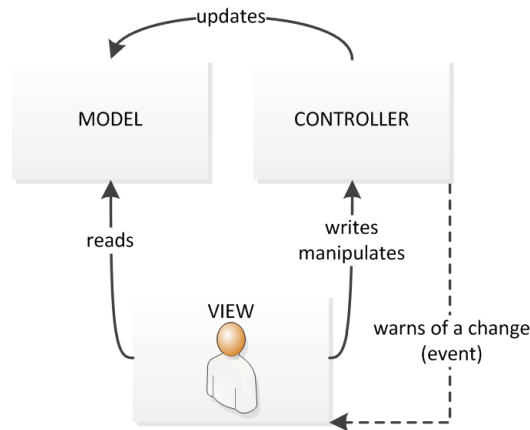


Figure 3.3: Typical Model–View–Controller scheme [5]

3.3.2 Communication

The discovery mechanism of Fast DDS (see section 2.11.1) ensures that the Android application can communicate with other applications on the same network. As mentioned previously (2.11.1), there are several possible configurations of discovery mechanism. This application uses the default setting, Simple discovery, for connecting inside the local network. Future plans are to move to a Discovery server to allow more appropriate device interconnection.

3.3.3 Support components

Several test versions and experiments were developed apart from this application (IPA2X Warning). Worth mentioning is the FastDDS Shapes application. Most implementations of the DDS protocol (FastDDS, rti DDS, openDDS) use the “Shapes” application as a demonstration. It consists of moving shapes on a canvas. This app is a sort of port of such app for Android. Even though it was a way to learn how to work with JNI and the FastDDS library, it turned out to be a working app.

Chapter 4

Implementation

This section describes the individual parts of the IPA2X Warning application implementation. Each part of the application or accompanying software that we have also developed will be covered in a subchapter.

4.1 User interface

The application's user interface consists of two parts: portrait mode (see fig 4.2) and landscape mode (see fig 4.1). Landscape mode is designed for in-vehicle operation, while additional information in portrait mode is mainly for debugging. In landscape mode, the elements intended for portrait mode are hidden. Only the canvas for warnings is present. The individual components are:

- **imageView** – displays the graphical part of the warning
- **labels** – display speed, GPS coordinates of the vehicle and the crossing, as well as the number of connected CrossingInfo publishers
- **button** – closes the application

4.2 Main Activity

Android applications can define system callbacks. Some of them (specifically onCreate) even have to define. The following sections will present callbacks implemented in the IPA2X Warning application.

4.2.1 onCreate

onCreate function is one of the system callbacks. Android calls it when the application is starting [61]. It initializes the View, and further calls the configure function, which sets up the user interface elements. It also checks if location access rights are allowed (those must be declared in the `AndroidManifest.xml` file as well) and asks the system for periodic location information. It furthermore creates media player objects for playing audio

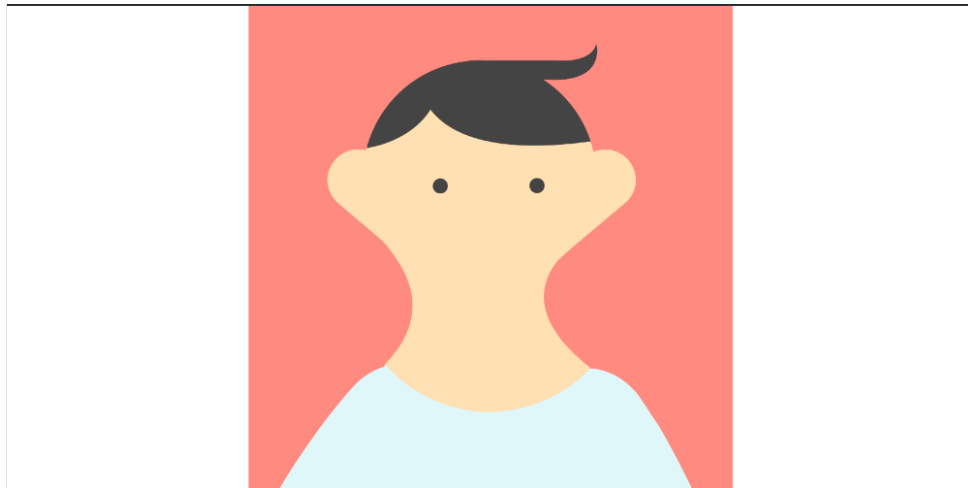


Figure 4.1: Landscape mode layout

alerts. Two separate players are chosen because switching an audio track is not a trivial matter. Finally, it starts threads with Fast DDS communication handlers.

■ 4.2.2 `onDestroy`

`onDestroy` is in charge of the final cleanup when closing the application. It is designed specifically for releasing system resources such as threads [62]. In this case, it sends termination information to both handler threads and waits for them to complete their termination. In the meantime, it turns off and releases media players.

■ 4.2.3 `onConfigurationChanged`

`onConfigurationChanged` is a function that the system calls when the configuration change has occurred. In our case, it is a layout change – the user rotated the phone from portrait mode to landscape mode or vice versa. This feature is called only if the app in Android Manifest announces that it will process the changes itself. Android's standard behaviour is to destroy existing activity and restart it with the new configuration [63]. In this application, the `onConfigurationChanged` function performs the same initial UI configuration as the `onCreate` [61] function. Because `onConfigurationChanged` is called when the application is running, we need to assume that warning was already displayed. Therefore, the warning graphics will be restored according to the stored status in an internal variable.

■ 4.2.4 `onLocationChanged`

`onLocationChanged` is a callback that the system calls after registering the Location Listener. In our case, the registration takes place in the `onCreate` function. In this function, we request updates in the interval of 500 ms.

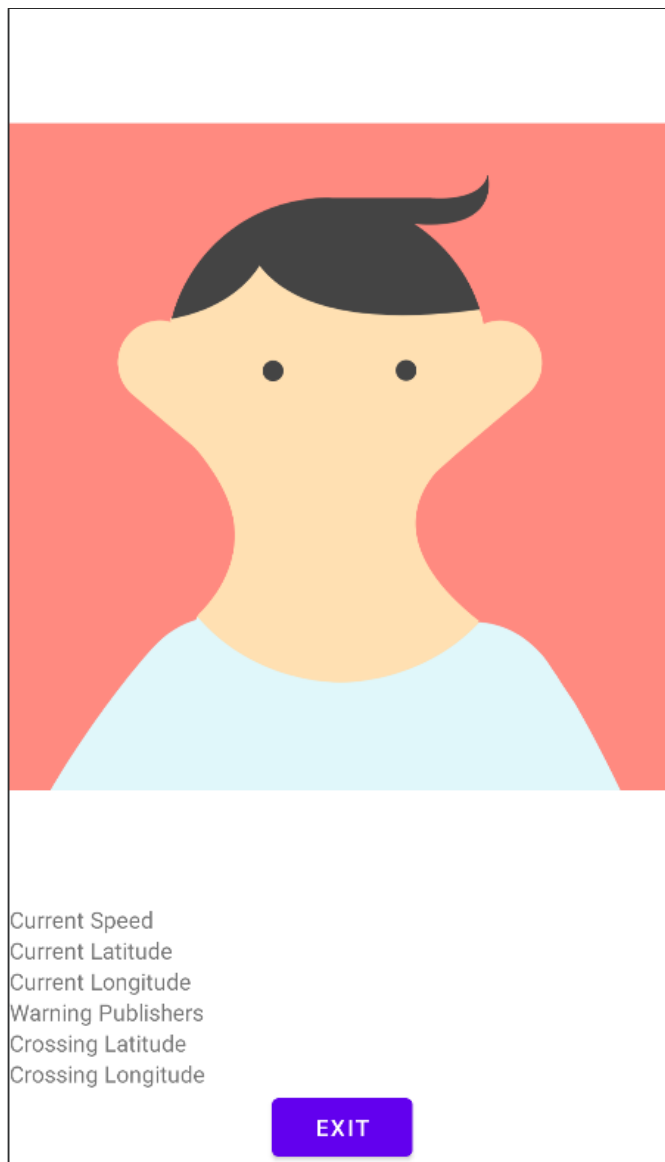


Figure 4.2: Portrait mode layout

This function is therefore called in this same interval, but only if there is a change in location [64]. When calling this function, our application saves the delivered location and displays it on the screen (only in portrait mode). At the same time, it displays and saves the current speed, if available.

■ 4.2.5 drawDanger

Task of `drawDanger` function is to render and display warnings to the user. It plays the appropriate animation and sound (or lack of such) based on incoming information from the rover (the Crossing Handler primarily calls this function). The animation is an SVG animation converted to `AnimatedVectorDrawable`

using `shapeshifter.design`¹ online tool. This function uses the supplemental functions `stopPlayer` (which stops and rewinds the warning sound) and `drawDangerImage` (which is responsible for drawing graphics on the screen).

4.3 Handlers

Handlers start, terminate and communicate with the native part of the application. They implement the `Runnable` interface because they run on separate threads. They, therefore, contain a `run` method – a method that the system starts when the thread starts.

4.3.1 InfoHandler

The `InfoHandler` initializes the `CarInfo` publisher in the `run` method and, in a loop, periodically calls the `publish` function into native code every 500 ms. The loop runs conditioned by the `boolean` function `running`, which can be flipped to `false` by calling the `terminate()` method. The publisher destroys when the loop ends, and the thread ends.

4.3.2 CrossingHandler

`CrossingHandler` has a rudimentary structure identical to `InfoHandler`. The only difference in the `run` loop is that it does not call the `publish()` function. It just waits. Receiving information takes place in the form of callbacks. `CrossingHandler` has two functions for these callbacks.

- `parseCrossing` – called when `CrossingInfo` is received
- `updateCrossingPublisherInfo` – called when the number of connected publishers changes

4.3.3 JNI

For JNI, there are function declarations whose implementations are available through JNI from native code. These declarations are marked with the keyword `external` in Kotlin (see fig 4.3). Next, the system must load the JNI library, which is done using `System.loadLibrary("library")`. Loading the JNI library into the class must be `static`². But Kotlin does not have a `static` keyword. It solves this functionality using `companion object` (see fig 4.3). If JNI functions should be static, the programmer must also declare them in this object.

¹<https://shapeshifter.design/>

²`static` means that this method is now a class method. It is not connected to object, but its class.


```

external fun initInfoPublisher(): Long

external fun killInfoPublisher(pointer: Long): Long

external fun sendInfoPublisher(pointer: Long, longitude:
    ↪ Double, latitude: Double, speed: Int): Boolean

companion object {
    init {
        System.loadLibrary("ipa2xwarning")
    }
}

```

Figure 4.3: Kotlin-side JNI functions in CarInfoPublisher

4.4 Native Publisher and Subscriber

Publisher and subscriber are written in C++ and contain JNI wrappers – functions called by Kotlin (Java) and have a particular format for this purpose. IPA2X Warning application includes publisher for the `CarInfo` messages and a subscriber for `CrossingInfo`. Both are based on the HelloWorld example given in the Fast DDS library documentation [6].

4.4.1 CarInfo publisher

`CarInfo` publisher consists of the `CarInfoPublisher` class. It contains the variables necessary for running the FastDDS library. All variables and their types can be seen in Fig. 4.4 and they are described in Tab. 4.1.

```

CarInfoType message_;
DomainParticipant* participant_;
Publisher* publisher_;
Topic* topic_;
DataWriter* writer_;
TypeSupport type_;

```

Figure 4.4: Publisher variables

Init function

The `CarInfoPublisher` class contains an `init()` function (displayed in Fig. 4.5) that ensures the initialization of the whole publisher. The workflow follows the recommendation in [6], i.e.:

4. Implementation

1. Assigns a name to the participant through the QoS of the DomainParticipant.
2. Uses the DomainParticipantFactory to create the participant.
3. Registers the data type defined in the IDL.
4. Creates the topic for the publications.
5. Creates the publisher.
6. Creates the DataWriter with the listener previously created.

CarInfoType	Type of current message
DomainParticipant	Acts as a container for all other Entity objects and as a factory for the, subscriber, and Topic objects.
Publisher	It is the object responsible for the creation of DataWriters.
Topic	Represents the fact that both publications and subscriptions are tied to a single data-type [65].
DataWriter	Allows the application to set the value of the data to be published under a given Topic.
TypeSupport	Provides the participant with the functions to serialize, deserialize and get the key of a specific data type.

Table 4.1: Descriptions of used datatypes [6]

A check is made between each step where an element is created or initialized to see if the task was completed.

In this function, modification occurs in the case of switching to another type of discovery protocol or another QoS parameter. This is the part of the program where these parameters are set. In the code above, we can see the init participant receives a custom QoS as a parameter, but others (e.g. Topic) receives the default QoS.

■ Publish function

```
bool publish(int speed, double latitude, double longitude) {
    message_.speed(speed);
    message_.coords().latitude(latitude);
    message_.coords().longitude(longitude);
    writer_->write(&message_);
    return true;
}
```

The publish function takes GPS coordinates and speed as parameters. It writes them to the message and then submits the message itself to the DataWriter. The approach of the DDS library can be seen here, that it appears to the programmer (user of the library) as a write to local memory.

```
bool init() {

    DomainParticipantQos participantQos;
    participantQos.name("ANDROID PUBLISHER");

    participant_ = DomainParticipantFactory::get_instance()->
        ↪ create_participant(0, participantQos);
    if (participant_ == nullptr) { return false; }

    type_.register_type(participant_);

    topic_ = participant_->create_topic("CarInfoTopic",
        ↪ "CarInfoType", TOPIC_QOS_DEFAULT);
    if (topic_ == nullptr) { return false; }

    publisher_ =
        ↪ participant_->create_publisher(PUBLISHER_QOS_DEFAULT,
        ↪ nullptr);
    if (publisher_ == nullptr) { return false; }

    writer_ = publisher_->create_datawriter(topic_,
        ↪ DATAWRITER_QOS_DEFAULT);
    if (writer_ == nullptr) { return false; }

    return true;
}
```

Figure 4.5: Init function

■ JNI Functions

At the end of the file are three JNI functions that can be called from the InfoHandler. From the name of these functions, we can read the complete path to the function in Java. See Fig. 4.6. We can see there the full name of the Java package. In our case `cz.cvut.fel.marunluk.ipa2xwarning`, which translates to `Java_cz_cvut_fel_marunluk_ipa2xwarning` in the JNI function name. Then the name of the class the function belongs to `InfoHandler` and finally the name of the Kotlin counterpart of this function `initInfoPublisher`. In this application, no complicated conversion between data types is necessary. Here, we only see a trivial cast between pointer and `jlong` (Java Long).

```
extern "C"
JNIEXPORT jlong JNICALL
Java_cz_cvut_fel_marunluk_ipa2xwarning_InfoHandler
↳ _initInfoPublisher(JNIEnv *env, jobject this) {
    CarInfoPublisher* publisher = new CarInfoPublisher();
    if (publisher->init()) {
        return (jlong) publisher;
    }
    delete publisher;
    return 0;
}
```

Figure 4.6: JNI Wrapper for `initInfoPublisher` function

initInfoPublisher. function (displayed above) creates a new instance of `CarInfoPublisher` and runs the `init()` function. It returns a pointer to the created instance of the `CarInfoPublisher` object, or 0 if initialisation is not successful.

killInfoPublisher. function is a wrapper for the JNI object destructor. When called, it releases an instance of the object, if it exists.

sendInfoPublisher. function serves as a wrapper for the `Publish` function belonging to the `CarInfoPublisher` class.

■ 4.4.2 CrossingInfo subscriber

`CrossingInfo` subscriber is based on the same basis as `CarInfoPublisher`. the fundamental part of the code is very similar, with the only difference that some components have unique variants for subscribers and publishers. However, `CrossingInfo` subscriber is more complex, both from the point of view of the Fast DDS library and JNI.

■ Listener

The `CrossingInfoSubscriber` class contains an instance of the `SubListener` class that contains the `on_subscription_matched` and `on_data_available` functions. These are functions that the Fast DDS library calls. As the name suggests, `on_subscription_matched` responds to a change in the number of connected publishers, and `on_data_available` is called when data arrives. This listener is registered when creating the reader in the `init` function.

■ JNI Callbacks

Unlike in `CarInfo` publisher, in `CrossingInfo` subscriber, we use the JNI interface in the opposite direction. We call the Kotlin function from C++.

Communication in this direction is more complex and requires extra code. First, we need to get the class to which the called function belongs. The `JNIEnv` pointer is used for this, thanks to which we get a `jclass` – reference to the class.

```
jclass lClass = env->FindClass("cz/cvut/fel/marunluk/
↳ ipa2xwarning/CrossingHandler");
```

In our case, when we call a function that is not static, we also need an instance of that class. However, we already got this as a parameter during JNI Call. Lastly, we need a reference to the function itself. We get this reference again via `JNIEnv` by calling the `GetMethodID` function with the already found class and name parameters.

```
jmethodID warningMethod = env->GetMethodID(lClass,
↳ "parseCrossing", "(ZZDD)V");
```

Now, all that is left to do is to call the function:

```
env->CallVoidMethod(object, warningInfoMethod, <params>);
```

■ Threading

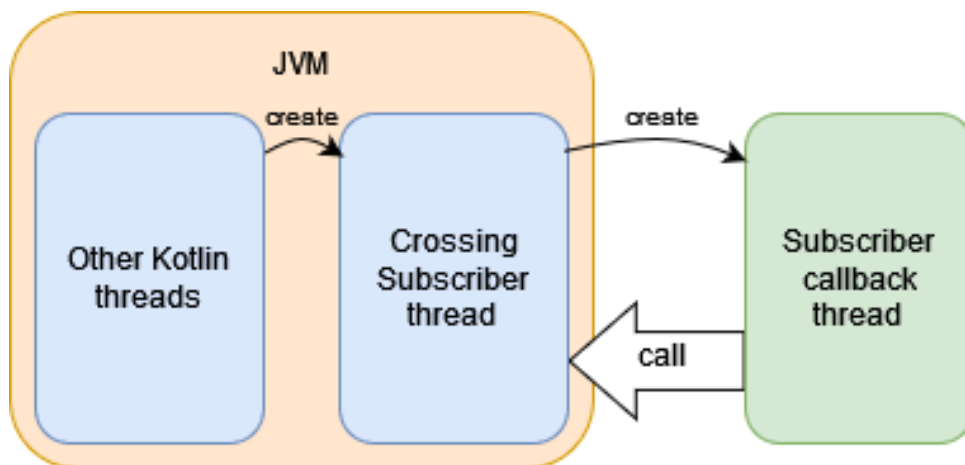


Figure 4.7: Thread scheme

During callbacks, we encounter the problem that `on_data_available` and `on_subscription_matched` are called from a different thread than we initiated the subscriber. It poses a problem because it is not a thread created by the Java Virtual Machine (JVM) but by the native code itself, and therefore the JVM does not allow the thread to interact with itself (see Fig. 4.7). Consequently, it is necessary to register this thread in the JVM using function `AttachCurrentThread()`. Also, a new `JNIEnv` pointer is needed because each thread needs a distinct value. See Fig. 4.8

```
java_vm->GetEnv(reinterpret_cast<void **>(&env),
↳ JNI_VERSION_1_6);
java_vm->AttachCurrentThread(&env, NULL);
```

Figure 4.8: Attaching current thread

```
gObject = env->NewGlobalRef(thiz);
```

Figure 4.9: Global reference creation

Also, the reference to the called object is thread-specific. We solve this by creating a new, global reference (see Fig. 4.9).

The other required variables are already valid for all threads, and one only needs to share them with the new thread. We do this here in the form of a global variable.

Our code already works as expected, but as found on the Android developer website [24], we must also detach each thread after manually attaching it. We do this by calling the `DetachCurrentThread()` function. However, this function is called often, and we would create too significant an overhead by repeatedly attaching and detaching threads. Therefore, we will create a “thread destructor”, with which we will call the disconnect only once, at the end of the thread. Even though we call `AttachCurrentThread()` several times in the body of the function, it does not pose a problem because repeated calls of the `AttachCurrentThread()` function are NOP³ [24]. We create the destructor using thread-specific variables. We can store some data here, but we will use the fact that we can define a destructor (see Fig. 4.10). for this data. This destructor will be called when the thread exits.

We will store some thread specific data there. And in their destruction, we will also disconnect the current thread from the JVM.

```
pthread_key_create(&key, thread_destructor);
```

```
static void thread_destructor(void* ptr) {
    JNIEnv* env = static_cast<JNIEnv *>(ptr);
    java_vm->DetachCurrentThread();
    __android_log_print(ANDROID_LOG_VERBOSE, "Sub
↳ attached", "Detached!");
}
```

Figure 4.10: Thread destructor we passed to the `pthread_key_create` function

³no-operation

We store some value in the variable in the code. Here it is a pointer to JNIEnv. So we know that this thread will eventually call the destructor we created.

■ 4.5 Networking

In this section, we will describe the network solution of the project.

Fast DDS Discovery needs to somehow connect with its counterparts for its operation. It is primarily designed for local communication. Namely same-host delivery, e.g. shared memory (see sect. 2.11). However, Fast DDS can also communicate over IP networks using UDP or TCP.

■ 4.5.1 Local network

The local network is the simplest solution for Fast DDS discovery communication. We need the device to reach others and the broadcast domain to work. Internet is not needed in this case. A typical local network allows all this.

■ 4.5.2 Discovery server over the Internet

The Discovery server should eliminate the need for a broadcast domain, as all devices query a specific point (see sect. 2.11.1). Unfortunately, the Discovery server is riddled with issues. We encountered problems compiling with the new compiler versions. Moreover, even when we managed to compile it, most of the features did not work. The only thing that did work was the communication within the localhost. We failed to make further progress on local network attempts.

■ 4.5.3 VPN

Another option is to emulate the local network using a VPN. VPN will allow the devices to see each other, but we need to set up a broadcast domain for the discovery protocol. Then we came across a WireGuard VPN test that does not support broadcasts.

■ 4.5.4 Solution

Based on these facts, we have decided to use a local WiFi network for the time being. To connect a mobile phone from a moving vehicle using a mobile 4G/5G network, it is necessary to configure a VPN with a functional broadcast domain.

■ 4.6 Supplemental files

These are files required by the Fast DDS library. Namely, the IDL files and the source code generated from them. They represent a link between the

internals of the library and the application (and its developer).

■ 4.6.1 IDLs

IDL files declare the data structure of forwarded messages via the DDS protocol. IDL supports most data types. In this application, we use two types of messages.

■ GPS

```
struct gps {
    double longitude;
    double latitude;
};
```

Figure 4.11: Coordinates structure

This file (see Fig. 4.11) contains GPS coordinates in the form of two figures – latitude and longitude. Both are saved in floating-point format. Double is used as the data type because the float data type is insufficient for its accuracy, and any errors could be too significant for our use.

■ CarInfo

```
#include "Gps.idl"

struct CarInfoType {
    short speed;
    string dummy;
    gps coords;
};
```

Figure 4.12: CarInfo message structure

It contains the GPS coordinates of the vehicle (struct imported from `gps.idl`) and the car's current speed (see Fig. 4.12). This file defines the messages sent by the application (running on a phone inside a vehicle) back to the rover.

■ CrossingInfo

This message (see Fig. 4.13) is sent by the rover and received by the application. Like `CarInfo`, we find GPS coordinates here. This time these are the coordinates of the crossing that the rover guards. Furthermore, two booleans carrying information about the state of the pedestrian crossing.


```
#include "Gps.idl"

struct CrossingInfoType {
    boolean danger;
    boolean crossing;
    string dummy;
    gps coords;
};
```

Figure 4.13: CrossingInfo message structure

- crossing – in case of a pedestrian crossing the street
- danger – for another dangerous situation (e.g. when a child runs into the road)

■ Dummy

Both messages (CarInfo and CrossingInfo) contain a “dummy” string. The application does not use this string in any way, but the library contains a bug where it is not possible to initiate an Android application without a more complex C++ datatype (e.g. string or hashmap). This error does not occur on a regular computer (Linux on x86-64 architecture). The library error could not be located yet. Therefore, this is a temporary workaround for this issue.

■ 4.6.2 Generated files

From IDL files, the Fast DDS Gen tool will generate auxiliary files for the required platform and programming language. Our generated files can be seen in Figure 4.14.

Files can be generated with the command [66]:

```
/path/to/Fast-DDS Gen/scripts/fastddsgen yourFile.idl
```

■ 4.7 Installation guide

The reader can install this application directly from the .apk package or compile the project. In the case of custom compilation, the reader will need to compile the application’s libraries. These are specifically Fast DDS and its pre-requisites (Foonathan Memory and Fast CDR). The compilation instructions are a modified version of the Fast DDS library installation instructions [67].

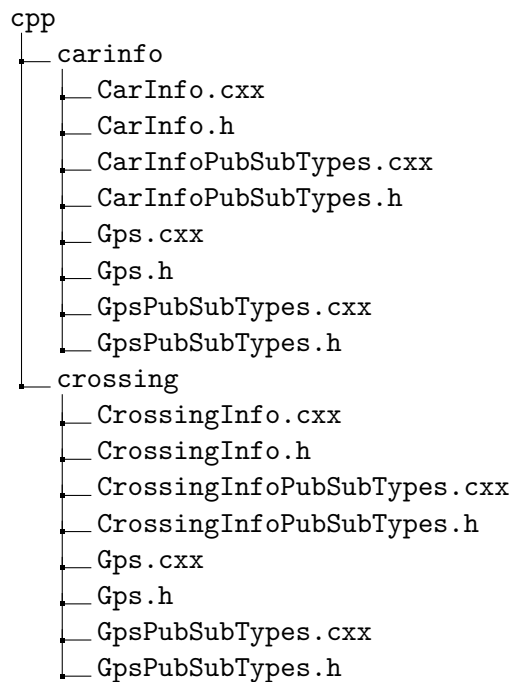


Figure 4.14: Generated files

4.7.1 Compilation from sources

Prerequisites

The project has been compiled several times during development on Ubuntu 20.04.3 and 21.04. Since we are cross-compiling for Android on the `aarch64` architecture, we use the Android NDK cross-compiler, which can be downloaded from the Google website⁴. Should the reader decide to use Windows Subsystem for Linux (WSL) for compilation, we recommend to extract the .zip file directly from the command line. We encountered errors when unpacking the compiler in Windows. Several packages, mainly libraries, are required for compilation. These are:

- `cmake`, `g++`, `pip3`, `wget` and `Git`
- `Asio` library, `TinyXML2` library, `OpenSSL`, `Libp11`

When using Debian, or Debian based Linux distros (i.e. Ubuntu) you can use:

```

sudo apt install cmake g++ python3-pip wget git
sudo apt install libasio-dev libtinyxml2-dev libssl-dev
↪ libp11-dev libengine-pkcs11-openssl
  
```

⁴<https://developer.android.com/ndk/downloads>

■ Compilation

- Create a directory for this library and its dependencies. Recommended is `~/Fast-DDS` (please note, that this directory is used in this guide)

```
mkdir ~/Fast-DDS
```

- Install and cross-compile Foonathan Memory with following commands:

```
cd ~/Fast-DDS
git clone
→ https://github.com/eProsima/foonathan_memory_vendor.git
mkdir foonathan_memory_vendor/build
cd foonathan_memory_vendor/build
cmake .. -DCMAKE_INSTALL_PREFIX=~/Fast-DDS/install
→ -DBUILD_SHARED_LIBS=ON
→ -DCMAKE_TOOLCHAIN_FILE=path/to/android-ndk-r23b/
→ build/cmake/android.toolchain.cmake
→ -DANDROID_ABI=arm64-v8a -DANDROID_NATIVE_API_LEVEL=24
cmake --build . --target install
```

- Install and cross-compile Fast-CDR library

```
cd ~/Fast-DDS
git clone https://github.com/eProsima/Fast-CDR.git
mkdir Fast-CDR/build
cd Fast-CDR/build
cmake .. -DCMAKE_INSTALL_PREFIX=~/Fast-DDS/install
→ -DCMAKE_TOOLCHAIN_FILE=path/to/android-ndk-r23b/
→ build/cmake/android.toolchain.cmake
→ -DANDROID_ABI=arm64-v8a -DANDROID_NATIVE_API_LEVEL=24
cmake --build . --target install
```

- Clone and cross-compile Fast DDS itself

```
cd ~/Fast-DDS
git clone https://github.com/eProsima/Fast-DDS.git
mkdir Fast-DDS/build
cd Fast-DDS/build
cmake .. -DCMAKE_INSTALL_PREFIX=~/Fast-DDS/install
→ -DCMAKE_TOOLCHAIN_FILE=path/to/android-ndk-r23b/
→ build/cmake/android.toolchain.cmake
→ -DANDROID_ABI=arm64-v8a -DANDROID_NATIVE_API_LEVEL=24
→ -DCMAKE_BUILD_TYPE=Release -DTHIRDPARTY=FORCE
→ -DCMAKE_FIND_ROOT_PATH=~/Fast-DDS/install
cmake --build . --target install
```

- Compiled library can be found in `~/Fast-DDS/install`.

■ CMake arguments

Some critical parameters for cross-compilation that we pass to CMake are:

- **CMAKE_TOOLCHAIN_FILE** tells CMake which compiler to use. In our case, we do not want to compile using the standard Linux compiler but the cross compiler for Android.
- **ANDROID_ABI** determines for which platform we will compile. Android supports four architectures: `armeabi-v7a`, `arm64-v8a`, `x86`, and `x86_64` [68]. We are interested in `arm64-v8a`, so we chose the `aarch64` instruction set that `arm64-v8a` uses.
- **THIRDPARTY** allows us to choose whether we want CMake to download the libraries or provide them ourselves. In `OFF` mode, it searches libraries only locally. In `ON` mode, it downloads them, but only if it does not find them locally. `FORCE` mode ensures that it downloads and compiles libraries itself [69]. Therefore, there is no collision where CMake would try to link a library intended for another instruction set.
- **ANDROID_NATIVE_API_LEVEL** (or **ANDROID_PLATFORM**) Specifies the minimum API level supported by the application or library. This value corresponds to the application's `minSdkVersion` [70].

■ ifaddrs

During development, we encountered a problem with the `android-ifaddrs` library. Android previously did not support the features from `ifaddrs.h`, it was necessary to develop an alternative solution. That alternative became the `android-ifaddrs` library.

In the Android API24 (Android 7.0), Google has added official support for `ifaddrs`. So the `android-ifaddrs` library has become redundant. However, in API30 (Android 11), Google removed the functionality on which `android-ifaddrs` relied. Then this library became undesirable because it broke functionality when targeting newer versions [71][72]. Therefore, we have updated the CMake file not to include this library when compiling for specific versions (see Fig. 4.15). We also sent this change as a pull request⁵ to the Fast DDS repository, and this request was accepted.

■ 4.7.2 Launch from Android Studio

To work with the application in the Android Studio development environment, open the application source codes in Android Studio (**File** -> **Open**). Select the `IPA2XWarning` folder. The project folder should have a green Android icon. It indicates that Android Studio has recognized it as a project. If it was not recognized or the folder is missing, restart Android Studio. Sometimes it takes several attempts until the folder appears.

⁵<https://github.com/eProsima/Fast-DDS/pull/2677>

```

if(ANDROID)
  if((ANDROID_PLATFORM LESS_EQUAL 23) OR
    ↪ (ANDROID_NATIVE_API_LEVEL LESS_EQUAL 23))
    eprosimafindthirdparty(android-ifaddrs
    ↪ android-ifaddrs)
  endif()
endif()

```

Figure 4.15: CMake with added check for API versions

Next, open the `CMakeLists.txt` file located in the `app/cpp` folder and on line 5, edit the path to the directory with the library we compiled in the last step. It can be `~/Fast-DDS/install` or another location where the library was compiled or copied to.

The application can be launched with a green arrow in the upper right corner of Android Studio. The launch is possible only on a mobile phone with the Android operating system version 7.0 and higher. The cell phone must be of the `aarch64` architecture. Reader can find out if their phone is compliant using the CPU-Z⁶ application. Please note that Android Studio's default emulated devices run on x86 architecture. Therefore launch on these is not possible

4.8 Shapes

Shapes application was developed as a prototype to test Fast DDS library and JNI. The IPA2X Warning application is based directly on the Shapes application.

The concept of the Shapes Demo consists of shapes moving on the screen, which are also shared via Fast DDS to other devices. This is a common demo that uses all major DDS implementations (Fast DDS, openDDS, rti DDS).

This Android Shapes implementation connects via UDP over a local network to another instance of the same application, or with Fast DDS Shapes on a PC.

The user interface of Shapes Demo application (see Fig. 4.16) is straightforward. The individual elements are

- Canvas on which the individual shapes move
- Subscriber switches for individual shapes
- Menu for creating a new shape for the publisher

⁶https://play.google.com/store/apps/details?id=com.cpuid.cpu_z

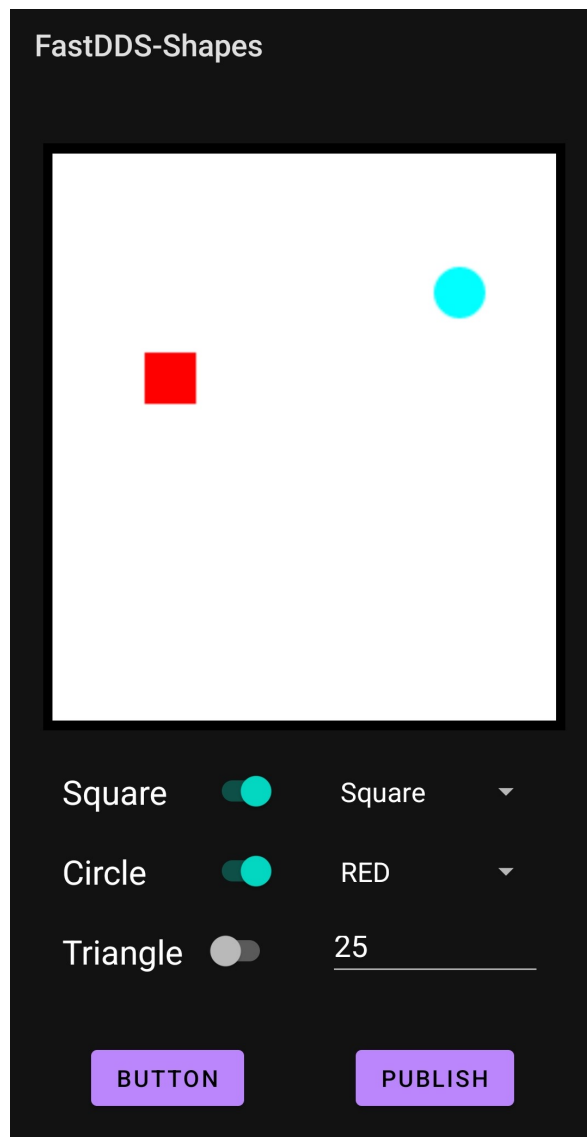


Figure 4.16: Shapes user interface

4.9 Supporting software

To test the Android application, we developed simple console applications for testing applications, which replace the function of a rover. The code is based on the same base as the native part of the Warning application. There are a total of 4 programs. It is a pair of publishers and subscribers for both CarInfo and CrossingInfo.

Due to the fact that these are applications for testing during development and are not intended for the end-user, the sanitisation of inputs or the graphical aspect of the user interface was not taken into account.

■ 4.9.1 Publishers

CrossingInfo publisher (see Fig. 4.17) waits for keyboard input, where it sends a message based on the key pressed. It also sends coordinates, which are hardcoded. The keys that control the program are:

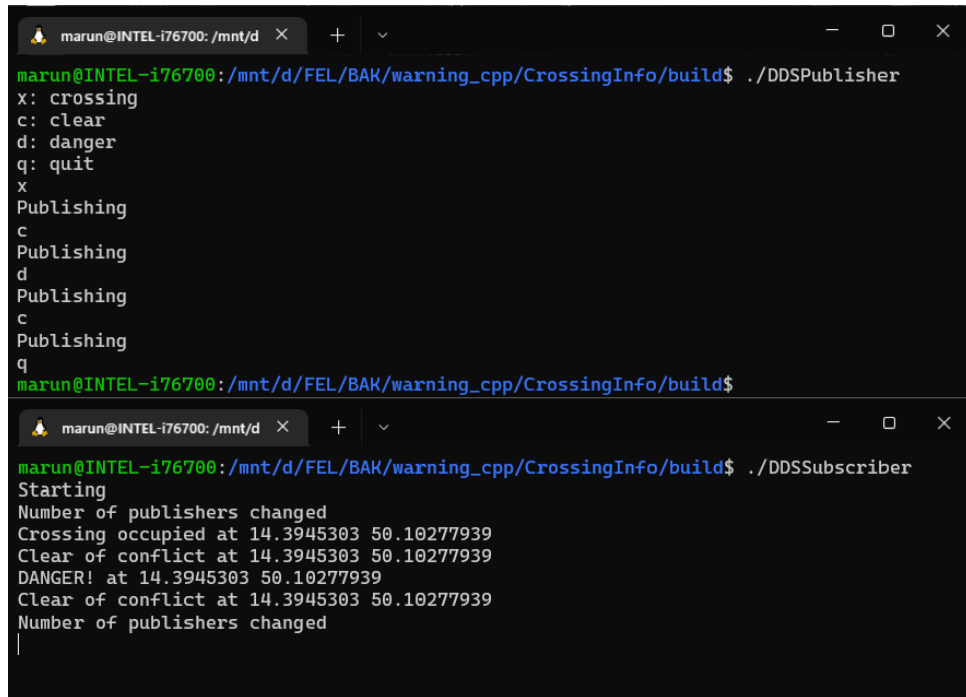
- **x** – crossing
- **d** – danger
- **c** – clear
- **q** – quit

CarInfo publisher (see Fig. 4.18) is also waiting for keyboard input. It awaits integer input to be sent as the speed value. Geolocation will also be adjusted based on this value (+5/speed).

■ 4.9.2 Subscribers

CarInfo subscriber (see Fig. 4.18) always listens for a certain number (100) of messages that it writes to the console. It will end after the writing of the 100th message. CrossingInfo subscriber (see Fig. 4.17) works on the same principle.

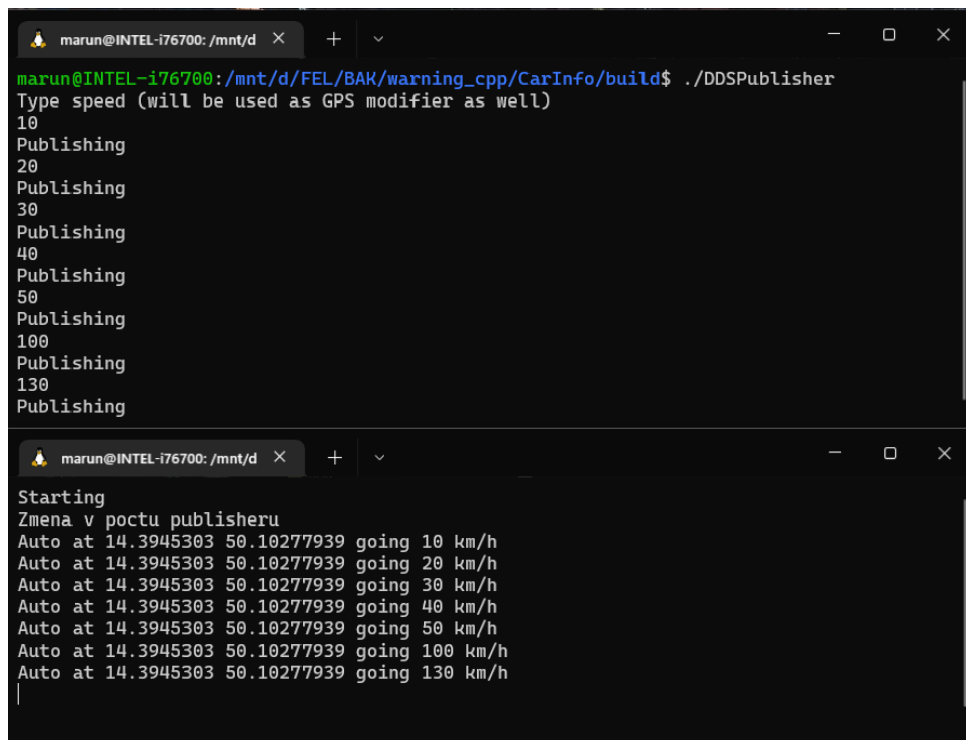
4. Implementation



```
marun@INTEL-i76700: /mnt/d
marun@INTEL-i76700:/mnt/d/FEL/BAK/warning_cpp/CrossingInfo/build$ ./DDSPublisher
x: Crossing
c: Clear
d: danger
q: quit
x
Publishing
c
Publishing
d
Publishing
c
Publishing
q
marun@INTEL-i76700:/mnt/d/FEL/BAK/warning_cpp/CrossingInfo/build$

marun@INTEL-i76700:/mnt/d/FEL/BAK/warning_cpp/CrossingInfo/build$ ./DDSSubscriber
Starting
Number of publishers changed
Crossing occupied at 14.3945303 50.10277939
Clear of conflict at 14.3945303 50.10277939
DANGER! at 14.3945303 50.10277939
Clear of conflict at 14.3945303 50.10277939
Number of publishers changed
```

Figure 4.17: CrossingInfo publisher and subscriber



```
marun@INTEL-i76700: /mnt/d
marun@INTEL-i76700:/mnt/d/FEL/BAK/warning_cpp/CarInfo/build$ ./DDSPublisher
Type speed (will be used as GPS modifier as well)
10
Publishing
20
Publishing
30
Publishing
40
Publishing
50
Publishing
100
Publishing
130
Publishing

marun@INTEL-i76700:/mnt/d
Starting
Zmena v poctu publisheru
Auto at 14.3945303 50.10277939 going 10 km/h
Auto at 14.3945303 50.10277939 going 20 km/h
Auto at 14.3945303 50.10277939 going 30 km/h
Auto at 14.3945303 50.10277939 going 40 km/h
Auto at 14.3945303 50.10277939 going 50 km/h
Auto at 14.3945303 50.10277939 going 100 km/h
Auto at 14.3945303 50.10277939 going 130 km/h
```

Figure 4.18: CarInfo publisher and subscriber

Chapter 5

Evaluation

This chapter discusses the evaluation and testing of the IPA2X Warning application together with other components of the project (such as in-vehicle use and rover communication).

To evaluate the functionality of the developed solution, we integrated all the developed components and tested them together with a Škoda vehicle, as shown in Fig. 3.1. However, we simulated the rover with the supportive CrossingInfo publisher application (see Section 4.9).

The user interface was, after consultation, accepted by the project partner (Škoda Auto). A comparison between the user interface appearance on a mobile phone and the vehicle's infotainment unit can be seen in Figures 5.1, 5.2 5.3.

The overall functionality is demonstrated in the video available in the attached data files. To evaluate the performance, we measure the latency between sending the crossing state change and displaying the warning on the car screen.

5.1 Performance

We measured latency by recording the screens of a phone, infotainment and a laptop representing the rover. All these elements are part of the shot. We obtain the times by subtracting the difference between the timestamp of pressing the key (sending a message) and the display of warnings on the mobile screen and the vehicle's infotainment unit. We get three results. Total latency (between sending and display in the vehicle), latency between rover (substitute laptop) and display on the mobile phone, and latency between phone and vehicle displays. Our testing setup consists of:

- Samsung Galaxy S10e mobile phone for recording (1080p 60fps)
- Samsung Galaxy A52s mobile phone connected to the infotainment unit via AA Mirror
- Škoda vehicle (Škoda Superb)
- Samsung Galaxy S10e and Xiaomi Redmi Note 7 served as hotspots

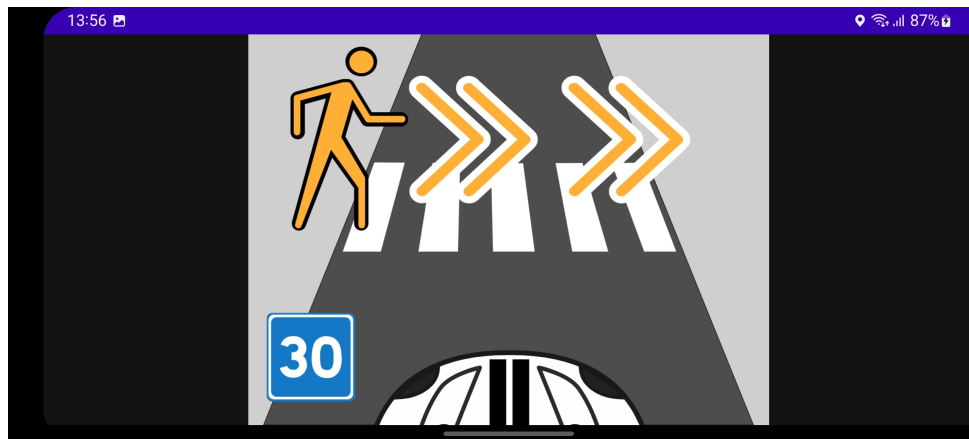


Figure 5.1: Orange warning screenshot from a mobile phone

- HP EliteBook 8470p running rover substitute application

Testing took place in two instances. In one case, the router (hotspot on the mobile phone) was placed in the vehicle. In the second case, it was outside the vehicle behind a wall. In both cases, comparable values were measured (see Table 5.1). A complete table of measured values is in the appendix A (see Tables A.1 and A.2).

Attempt	Overall [s]	Network [s]	Android Auto [s]
1	0.293	0.197	0.096
2	0.289	0.193	0.096

Table 5.1: Latency measurement results



Figure 5.2: Orange warning displayed on vehicle infotainment unit



Figure 5.3: Red warning displayed on vehicle infotainment unit



Chapter 6

Conclusion

In this thesis, we designed and developed the IPA2X Warning application, which warns drivers audibly and visually via Android Auto (AA Mirror). We used the chosen technologies. We imported the Fast DDS library into the Android operating system using the Java Native Interface. We tested the IPA2X Warning application in a car from Škoda Auto company, which approved our GUI design.

A particular success is the Fast DDS Shapes Demo for Android application, which was created as a prototype to test Fast DDS on Android. We presented this application to the developers of the Fast DDS library (eProxima), and we received positive feedback.

Another success is finding and fixing a bug in the Fast DDS library. After review, our patch has been approved and incorporated into the main development branch of the Fast DDS library.



6.1 Future work

There are still opportunities to develop this project. Despite our efforts, we were unable to establish communication over the open Internet using mobile 4G/5G networks.

We are still waiting for the possibility of integrating this application directly with the rover.

During development, we encountered several bugs, such as the need for dummy strings in IDL files. There is an opportunity to thoroughly debug this error and develop further patches for the Fast DDS library.



Bibliography

- [1] “IPA2X Project — rtsl.cps.mw.tum.de.” <https://rtsl.cps.mw.tum.de/ipa2x>. [Accessed 20-May-2022].
- [2] “Java Native Interface Specification: 2 - Design Overview – docs.oracle.com.” <https://docs.oracle.com/en/java/javase/17/docs/specs/jni/design.html>. [Accessed 18-May-2022].
- [3] <https://www.facebook.com/d4nny.cz>, “Do hry vstupuje Volkswagen, Android Auto bude ve většině modelů 2016 | Svět Androida — svetandroida.cz.” <https://www.svetandroida.cz/volkswagen-android-auto/>. [Accessed 20-May-2022].
- [4] “OBD2 Explained - A Simple Intro [2022 | The #1 Tutorial] – csselectronics.com.” <https://www.csselectronics.com/pages/obd2-explained-simple-intro>. [Accessed 18-May-2022].
- [5] “File:MVC Diagram (Model-View-Controller).svg - Wikimedia Commons — commons.wikimedia.org.” [https://commons.wikimedia.org/wiki/File:MVC_Diagram_\(Model-View-Controller\).svg](https://commons.wikimedia.org/wiki/File:MVC_Diagram_(Model-View-Controller).svg). [Accessed 20-May-2022].
- [6] “1.3. Writing a simple C++ publisher and subscriber application ; Fast DDS 2.6.0 documentation – fast-dds.docs.eprosima.com.” https://fast-dds.docs.eprosima.com/en/latest/fastdds/getting_started/simple_app/simple_app.html. [Accessed 18-May-2022].
- [7] “Cell Phone News - PhoneArena – phonearena.com.” https://www.phonearena.com/news/Googles-Android-OS-Past-Present-and-Future_id21273. [Accessed 18-May-2022].
- [8] “Android founder: We aimed to make a camera OS – pcworld.com.” <https://www.pcworld.com/article/451350/android-founder-we-aimed-to-make-a-camera-os.html>. [Accessed 18-May-2022].
- [9] “The first Android phone was announced 13 years ago today – android-police.com.” <https://www.androidpolice.com/2021/09/23/the-first-android-phone-was-announced-13-years-ago-today/>. [Accessed 18-May-2022].

6. Conclusion

- [10] “Samsung I9000 Galaxy S - Full phone specifications – gsmarena.com.” https://www.gsmarena.com/samsung_i9000_galaxy_s-3115.php. [Accessed 18-May-2022].
- [11] “HTC Desire S - Full phone specifications – gsmarena.com.” https://www.gsmarena.com/htc_desire_s-3776.php. [Accessed 18-May-2022].
- [12] “Samsung Galaxy Y S5360 - Full phone specifications – gsmarena.com.” https://www.gsmarena.com/samsung_galaxy_y_s5360-4117.php. [Accessed 18-May-2022].
- [13] E. Belinski, “Android API Levels – apilevels.com.” <https://apilevels.com/>. [Accessed 18-May-2022].
- [14] “Android API Levels | Android Developers – dre.vanderbilt.edu.” <http://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/guide/appendix/api-levels.html>. [Accessed 18-May-2022].
- [15] “Activity | Android Developers – developer.android.com.” <https://developer.android.com/reference/android/app/Activity>. [Accessed 18-May-2022].
- [16] “About Android App Bundles | Android Developers – developer.android.com.” <https://developer.android.com/guide/app-bundle>, 2022-03-17. [Accessed 18-May-2022].
- [17] <https://www.facebook.com/avi.it.128>, “Key Google Play Store Statistics in 2022 You Must Know – appinventiv.com.” <https://appinventiv.com/blog/google-play-store-statistics/>. [Accessed 18-May-2022].
- [18] “Java Native Interface Specification: 1 - Introduction – docs.oracle.com.” <https://docs.oracle.com/en/java/javase/17/docs/specs/jni/intro.html>. [Accessed 18-May-2022].
- [19] U. Technologies, “Unity - Manual: Building Plugins for Android – docs.huihoo.com.” <https://docs.huihoo.com/unity/5.5/Documentation/Manual/PluginsForAndroid.html>. [Accessed 18-May-2022].
- [20] “OpenGL ES | Android Developers – developer.android.com.” <https://developer.android.com/guide/topics/graphics/opengl>. [Accessed 18-May-2022].
- [21] “JNI tips | Android NDK | Android Developers – developer.android.com.” <https://developer.android.com/training/articles/perf-jni#jclass,-jmethodid,-and-jfieldid>. [Accessed 18-May-2022].
- [22] “What makes JNI calls slow? – stackoverflow.com.” <https://stackoverflow.com/questions/7699020/what-makes-jni-calls-slow/7809300#7809300>. [Accessed 18-May-2022].

- [23] “JNI tips | Android NDK | Android Developers – developer.android.com.” <https://developer.android.com/training/articles/perf-jni#javavm-and-jnienv>. [Accessed 18-May-2022].
- [24] “JNI tips | Android NDK | Android Developers – developer.android.com.” <https://developer.android.com/training/articles/perf-jni#threads>. [Accessed 18-May-2022].
- [25] “Kotlin on Android FAQ | Android Developers – developer.android.com.” <https://developer.android.com/kotlin/faq.html#android>. [Accessed 18-May-2022].
- [26] “Android’s Kotlin-first approach | Android Developers – developer.android.com.” <https://developer.android.com/kotlin/first>. [Accessed 18-May-2022].
- [27] A. a.s., “Co je MirrorLink? | Alza.cz – alza.cz.” <https://www.alza.cz/slovník/mirrorlink>. [Accessed 18-May-2022].
- [28] “Seznam compatibility – compatibilitylist.skoda-auto.com.” <https://compatibilitylist.skoda-auto.com/>. [Accessed 18-May-2022].
- [29] “MirrorLink® Operations Sunsetting by September 30, 2023. - Car Connectivity Consortium – carconnectivity.org.” <https://carconnectivity.org/press-release/mirrorlink-operations-sunsetting-by-september-30-2023/>. [Accessed 18-May-2022].
- [30] “MirrorLink – mirrorlink.com.” <https://mirrorlink.com/Developers>. [Accessed 18-May-2022].
- [31] “Android Auto | Android – android.com.” <https://www.android.com/auto/>. [Accessed 18-May-2022].
- [32] “Interaction principles | Design for Driving | Google Developers – developers.google.com.” https://developers.google.com/cars/design/design-foundations/interaction-principles#avoid_hazardous_or_distracting_activities. [Accessed 18-May-2022].
- [33] 24net s.r.o., “Test Jak se posunulo Android Auto? – V něčem lepší, v něčem horší | fDrive.cz – fdrive.cz.” <https://fdrive.cz/clanky/test-jak-se-posunulo-android-auto-v-necem-lepsi-v-necem-horsi-5530>. [Accessed 18-May-2022].
- [34] N. Garun, “Google has made Android Auto work more like your phone — for better or worse – theverge.com.” <https://www.theverge.com/2019/7/30/20746885/google-android-auto-2019-review-features-app-phone>. [Accessed 18-May-2022].
- [35] “r/AndroidAuto - Facebook messenger completely disappeared from Android Auto – reddit.com.” https://www.reddit.com/r/AndroidAuto/comments/kxygj7/facebook_messenger_completely_disappeared_from/. [Accessed 18-May-2022].

6. Conclusion

- [36] “r/AndroidAuto - Facebook Messenger not working correctly? – reddit.com.” https://www.reddit.com/r/AndroidAuto/comments/j3hzd3/facebook_messenger_not_working_correctly/. [Accessed 18-May-2022].
- [37] “How Do I Know Whether My Car is OBD-II Compliant? – scantool.net.” <https://www.scantool.net/blog/how-do-i-know-whether-my-car-is-obd-ii-compliant>. [Accessed 18-May-2022].
- [38] “The Car Hackers Handbook – opengarages.org.” <http://opengarages.org/handbook/ebook/>. [Accessed 18-May-2022].
- [39] “German car industry plans to close OBD interface eeNews Automotive – eeneautomotive.com.” <https://www.eeneautomotive.com/en/german-car-industry-plans-to-close-obd-interface/>. [Accessed 18-May-2022].
- [40] “can-newsletter.org - Applications – can-newsletter.org.” https://can-newsletter.org/engineering/applications/160322_25th-anniversary-mercedes-w140-first-car-with-can/. [Accessed 18-May-2022].
- [41] “Controller Area Network (CAN) Overview – ni.com.” <https://www.ni.com/cs-cz/innovations/white-papers/06/controller-area-network--can--overview.html>. [Accessed 18-May-2022].
- [42] “Sbernice CAN – elektrorevue.cz.” <http://www.elektrorevue.cz/clanky/03021/index.html>. [Accessed 18-May-2022].
- [43] “CAN FD Explained - A Simple Intro [2022 | The #1 Tutorial] – csselectronics.com.” <https://www.csselectronics.com/pages/can-fd-flexible-data-rate-intro>. [Accessed 18-May-2022].
- [44] “ELM327 v2.3 #x2013; Elm Electronics – elmelectronics.com.” <https://www.elmelectronics.com/ic/elm327/>. [Accessed 18-May-2022].
- [45] Wikipedia contributors, “Obd-ii pids – Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=OBD-II_PIDs&oldid=1086105625, 2022. [Online; accessed 14-May-2022].
- [46] D. Sližek, “Jižní Korea spustí první komerční 5G síť, operátoři už oznámili ceny balíčků - Lupa.cz – lupa.cz.” <https://www.lupa.cz/aktuality/sk-telecom-spusti-prvni-komercni-5g-sit-oznamil-uz-take-ceny-balicku/>. [Accessed 18-May-2022].
- [47] “From 1G to 5G: A Brief History of the Evolution of Mobile Standards – brainbridge.be.” <https://www.brainbridge.be/en/blog/1g-5g-brief-history-evolution-mobile-standards>. [Accessed 18-May-2022].
- [48] “5G Spectrum Guide - Everything You Need to Know – gsma.com.” <https://www.gsma.com/spectrum/5g-spectrum-guide/>. [Accessed 18-May-2022].

- [49] “5G spectrum bands explained— low, mid and high band | Nokia – nokia.com.” <https://www.nokia.com/networks/insights/spectrum-bands-5g-world/>. [Accessed 18-May-2022].
- [50] J. Sedlák, “O Huawei bez Huawei. V Praze se řešila bezpečnost 5G sítí - Lupa.cz – lupa.cz.” <https://www.lupa.cz/clanky/o-huawei-bez-huawei-v-praze-se-resila-bezpecnost-5g-siti/>. [Accessed 18-May-2022].
- [51] K. Wolf, “Spojené státy varují před nasazováním Huawei i do nekritických součástí 5G sítí - Lupa.cz – lupa.cz.” <https://www.lupa.cz/aktuality/spojene-staty-varuji-pred-nasazovanim-huawei-i-do-nekriticky-ch-soucasti-5g-siti/>. [Accessed 18-May-2022].
- [52] C. Reichert, “5G coronavirus conspiracy theory leads to 77 mobile towers burned in UK, report says – cnet.com.” <https://www.cnet.com/health/5g-coronavirus-conspiracy-theory-sees-77-mobile-towers-burned-report-says/>, 2020. [Accessed 18-May-2022].
- [53] “Burning Cell Towers, Out of Baseless Fear They Spread the Virus (Published 2020) – nytimes.com.” <https://www.nytimes.com/2020/04/10/technology/coronavirus-5g-uk.html>. [Accessed 18-May-2022].
- [54] A. News, “Conspiracy theorists burn 5G towers claiming link to virus – abcnews.go.com.” <https://abcnews.go.com/Health/wireStory/conspiracy-theorists-burn-5g-towers-claiming-link-virus-70258811>. [Accessed 18-May-2022].
- [55] “How Does DDS Work? – dds-foundation.org.” <https://www.dds-foundation.org/how-dds-works/>. [Accessed 18-May-2022].
- [56] R.-T. Innovations, “Data Distribution Service (DDS) for Complex Systems | RTI – rti.com.” <https://www.rti.com/products/dds-standard>. [Accessed 18-May-2022].
- [57] “What is DDS? – dds-foundation.org.” <https://www.dds-foundation.org/what-is-dds-3/>. [Accessed 18-May-2022].
- [58] “5. Discovery; Fast DDS 2.6.0 documentation – fast-dds.docs.eprosima.com.” <https://fast-dds.docs.eprosima.com/en/latest/fastdds/discovery/discovery.html>. [Accessed 18-May-2022].
- [59] “5.3.4. Discovery Server Settings; Fast DDS 2.6.0 documentation – fast-dds.docs.eprosima.com.” https://fast-dds.docs.eprosima.com/en/latest/fastdds/discovery/discovery_server.html#discovery-server. [Accessed 18-May-2022].
- [60] “DDS API; Fast DDS 2.6.0 documentation – fast-dds.docs.eprosima.com.” <https://fast-dds.docs.eprosima.com/en/latest/>. [Accessed 18-May-2022].

6. Conclusion

- [61] “Activity | Android Developers – developer.android.com.” [https://developer.android.com/reference/android/app/Activity#onCreate\(android.os.Bundle\)](https://developer.android.com/reference/android/app/Activity#onCreate(android.os.Bundle)). [Accessed 18-May-2022].
- [62] “Activity | Android Developers – developer.android.com.” [https://developer.android.com/reference/android/app/Activity#onDestroy\(\)](https://developer.android.com/reference/android/app/Activity#onDestroy()). [Accessed 18-May-2022].
- [63] “Activity | Android Developers – developer.android.com.” [https://developer.android.com/reference/android/app/Activity#onConfigurationChanged\(android.content.res.Configuration\)](https://developer.android.com/reference/android/app/Activity#onConfigurationChanged(android.content.res.Configuration)). [Accessed 18-May-2022].
- [64] “LocationListener | Android Developers – developer.android.com.” <https://developer.android.com/reference/android/location/LocationListener>. [Accessed 18-May-2022].
- [65] “17.1.5.4. Topic ; Fast DDS 2.6.0 documentation – fast-dds.docs.eprosima.com.” https://fast-dds.docs.eprosima.com/en/latest/fastdds/api_reference/dds_pim/topic/topic_class.html. [Accessed 18-May-2022].
- [66] “1. Introduction ; Fast DDS 2.6.0 documentation – fast-dds.docs.eprosima.com.” <https://fast-dds.docs.eprosima.com/en/latest/fastddsgen/introduction/introduction.html>. [Accessed 18-May-2022].
- [67] “3. Linux installation from sources; Fast DDS 2.6.0 documentation – fast-dds.docs.eprosima.com.” https://fast-dds.docs.eprosima.com/en/latest/installation/sources/sources_linux.html. [Accessed 18-May-2022].
- [68] “Android ABIs | Android NDK | Android Developers – developer.android.com.” <https://developer.android.com/ndk/guides/abis>. [Accessed 18-May-2022].
- [69] “6. CMake options; Fast DDS 2.6.0 documentation – fast-dds.docs.eprosima.com.” https://fast-dds.docs.eprosima.com/en/latest/installation/configuration/cmake_options.html. [Accessed 18-May-2022].
- [70] “CMake | Android NDK | Android Developers – developer.android.com.” https://developer.android.com/ndk/guides/cmake#android_platform. [Accessed 18-May-2022].
- [71] “ifaddrs.h header not found when compiling SDL for android – stackoverflow.com.” <https://stackoverflow.com/a/57112520/18940278>. [Accessed 18-May-2022].

- [72] “NETLINK_ROUTE socket binding not available on Android 11+, can we live without it for Android? · Issue #6251 · arvidn/libtorrent – github.com.” <https://github.com/arvidn/libtorrent/issues/6251>. [Accessed 18-May-2022].

Appendix A

Latency Measurements

Overall [s]	Network [s]	Android Auto [s]
0,233	0,133	0,100
0,233	0,133	0,100
0,500	0,367	0,133
0,267	0,167	0,100
0,300	0,167	0,133
0,200	0,100	0,100
0,250	0,167	0,083
0,233	0,167	0,067
0,233	0,100	0,133
0,183	0,117	0,067
0,200	0,100	0,100
0,400	0,300	0,100
0,200	0,133	0,067
0,267	0,133	0,133
0,633	0,567	0,067
0,233	0,133	0,100
0,367	0,300	0,067
0,317	0,217	0,100
0,317	0,250	0,067

Table A.1: Latency measurement with hotspot inside the vehicle

A. Latency Measurements

Overall [s]	Network [s]	Android Auto [s]
0,250	0,083	0,333
0,150	0,108	0,258
0,433	0,083	0,517
0,100	0,117	0,217
0,400	0,100	0,500
0,117	0,083	0,200
0,100	0,083	0,183
0,300	0,083	0,383
0,150	0,100	0,250
0,183	0,100	0,283
0,133	0,067	0,200
0,083	0,117	0,200
0,083	0,100	0,183
0,150	0,100	0,250
0,133	0,100	0,233
0,317	0,117	0,433

Table A.2: Latency measurement with hotspot outside the vehicle