VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁVRH A IMPLEMENTACE UŽIVATELSKÉHO ROZHRANÍ FRAMEWORKU ROS V PROSTŘEDÍ ANDROID

DESIGN AND IMPLEMENTATION OF USER INTERFACE FOR ROS FRAMEWORK USING ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Ondřej Podolinský
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Jiří Krejsa, Ph.D.
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Ondřej Podolinský**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**doc. Ing. Jiří Krejsa, Ph.D.**
Akademický rok:	2019/20

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Návrh a implementace uživatelského rozhraní frameworku ROS v prostředí Android

Stručná charakteristika problematiky úkolu:

Většina autonomních mobilních robotů používá interně framework ROS (Robot Operating System), který je složitý z hlediska uživatele. Předmětem práce je návrh a implementace přívětivého uživatelského rozhraní pro tento framework, které poběží na mobilním zařízení s operačním systémem Android.

Cíle bakalářské práce:

1. Seznamte se s frameworkem ROS a možnostmi jeho konektivity.
2. Vytvořte aplikaci v prostředí Android zahrnující základní funkcionalitu modulu RViz.
3. Vytvořte terminál SSH, který bude schopen ovládat robota nezávisle na ROSu.

Seznam doporučené literatury:

O'KEANE J.: A Gentle Introduction to ROS, 2013

LENTIN J.: Mastering ROS for Robotics Programming, Packt Publishing Ltd, 2015

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2019/20.

V Brně, dne 24. 10. 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ
DĚKANÁT

L. S.

prof. Ing. Jindřich Petruška, CSc.
ředitel ústavu

doc. Ing. Jaroslav Katolický, Ph.D.
děkan fakulty

Abstrakt

Práce se zabývá vývojem mobilní aplikace určené k vizualizaci dat z frameworku ROS a k jeho jednoduché správě. Dále pojednává o vývoji uživatelských aplikací a popisuje strukturu systému ROS, včetně jeho komunikačního schématu. Součástí práce je přehled podobných dostupných mobilních aplikací.

Summary

This bachelor's thesis deals with developing new mobile application, which enables ROS data visualization and administration. Application development, structure of ROS system, communication in ROS are described too. Thesis contains summary of available applications.

Klíčová slova

Android, mobilní aplikace, ROS, Java

Keywords

Android, mobile application, ROS, Java

PODOLINSKÝ, O. *Návrh a implementace uživatelského rozhraní frameworku ROS v prostředí Android*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2020. 53 s. Vedoucí doc. Ing. Jiří Krejsa, Ph.D.

Prohlašuji, že jsem tuto práci zpracoval samostatně s využitím zdrojů uvedených v seznamu na konci této práce.

Ondřej Podolinský

Děkuji vedoucímu práce doc. Ing. Jiřímu Krejsovi Ph.D. za podporu při tvorbě této práce. Děkuji též Ing. Miroslavu Čeplovi a Bc. Martině Podolinské za cenné podněty a rady při vývoji aplikace. V neposlední řadě děkuji své rodině i přátelům za možnost se touto prací i studiem zabývat.

Ondřej Podolinský

Obsah

Úvod

Pokud lze definovat mechatroniku, jako inženýrské odvětví konstruující *inteligentní stroje*, pak lze považovat robot za její ztělesnění. Robotika prodělává bouřlivý vývoj. Každým rokem se posouvají možnosti tohoto druhu techniky kupředu.

Pod pojmem *inteligentní* se skrývá celá řada konkrétních vlastností. Jednou z nich je bezesporu i schopnost stroje komunikovat se svojí obsluhou. Na tu se v poslední době klade stále silnější důraz, který často souvisí s tím, že se tento druh techniky dostává stále blíže reálné aplikaci, tj. k lidem, kteří často mají jinou úlohu, nežli studium konstrukce robotů. Taková obsluha musí být tudíž víceméně intuitivní, jednoduchá, zároveň však přesná a předvídatelná.

Tato práce určitě neaspiruje na průlom nové myšlenky v této oblasti. Její výsledek, mobilní aplikace, by měl však zaplnit určitou mezeru na tomto poli. Vytkla si za cíl rozšířit možnosti široce využívaného a oblíbeného frameworku ROS, nástroje pro jednodušší vývoj robotických systémů, v oblasti mobilních zařízení.

Samotný požadavek na tuto aplikaci vzešel z praktických zkušeností při využívání systému ROS. Byly jimi zejména těžkopádnost spouštění systému a povinnost míti v provozu kromě vyvíjeného fyzického zařízení (robotu) i počítač (popř. další zařízení), který slouží ke sledování dat nebo konfiguraci systému v běhu.

Tyto 2 negativní faktory však nejsou jedinými příčinami vzniku požadavku na vývoj nové aplikace. K nim je nutno přidat i faktory pozitivní. Mezi ně patří touha vývojářů posunout možnosti ROSu blíže reálné aplikaci a rozšířit tímto směrem jeho možnosti.

Kap. 2

Přehled dostupných aplikací

Úvodní kapitola by měla odpovědět na otázku, zdali již v dnešní době není dostupná mobilní aplikace, která by řešila výše zmíněné problémy. Dále poskytne přehled dostupných mobilních aplikaci určených ke správě a kontrole robotických systémů ROS nebo jiných, komerčních. V neposlední řadě též ukáže strukturu jednotlivých aplikací.

Dostupné aplikace lze rozdělit dle více či méně objektivních kritérií na různé kategorie. Pro posouzení aplikací jsem zvolil ty, které jsou pro jejich funkci rozhodující.

Kriteria pro posouzení vhodnosti a kvality aplikace	
vizualizace dat	podpora aplikace
náročnost ovládání	grafické provedení
možnosti aplikace	báze aplikace

2.1 Aplikace na bázi frameworku ROS

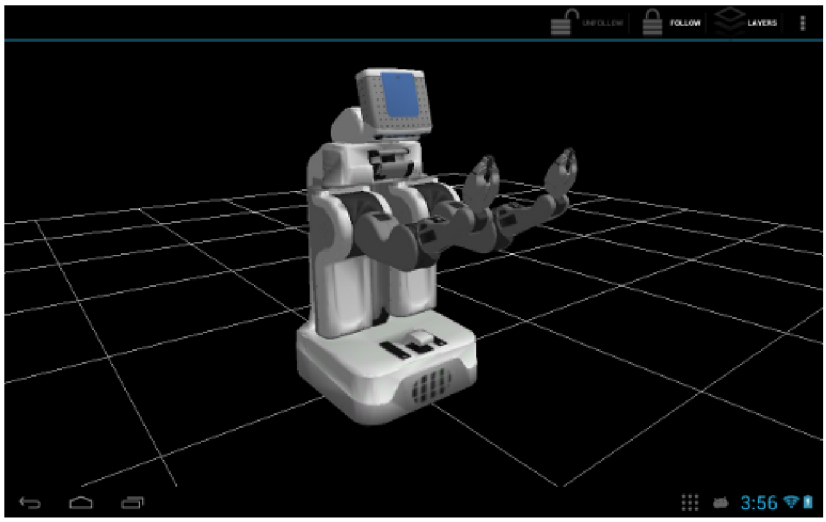
Prakticky všechny volně dostupné aplikace vytvořené na bázi ROSu jsou dnes ve fázi buď utlumeného nebo úplně ukončeného vývoje. Jejich zdrojové kódy jsou však ve většině případů dostupné.

Jejich společným rysem je využití knihovny *rosjava*.

2.1.1 Android RViz

Tuto aplikaci vytvořil Adam Zimmermann z Univerzity Illinois v USA během stáže ve Willow Garage. Poslední aktualizací prošla v roce 2012. Je dostupná pouze ve formě zdrojového kódu, který je volně ke stažení z [28].

Aplikace je schopna vizualizovat data na tabletu ze systému ROS v prakticky stejné podobě jako v modulu RViz na PC. Je uzpůsobena pro verzi Androidu 3.2 nebo vyšší. Pro vykreslování 3D grafiky je využíváno API OpenGL ES. Pro správnou funkci je nutné, aby vedle samotného ROSu běžel i implementovaný Python server, který zajišťuje přeposílání textur ze souboru URDF a jiných. Díky němu je též schopna vytvářet nové uzly [28].



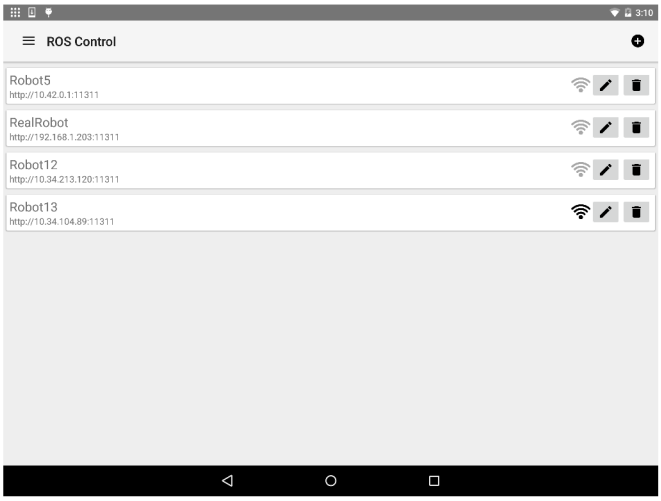
Obrázek 2.1: Aplikace Android RViz [29, převzato]

2.1.2 ROS Control

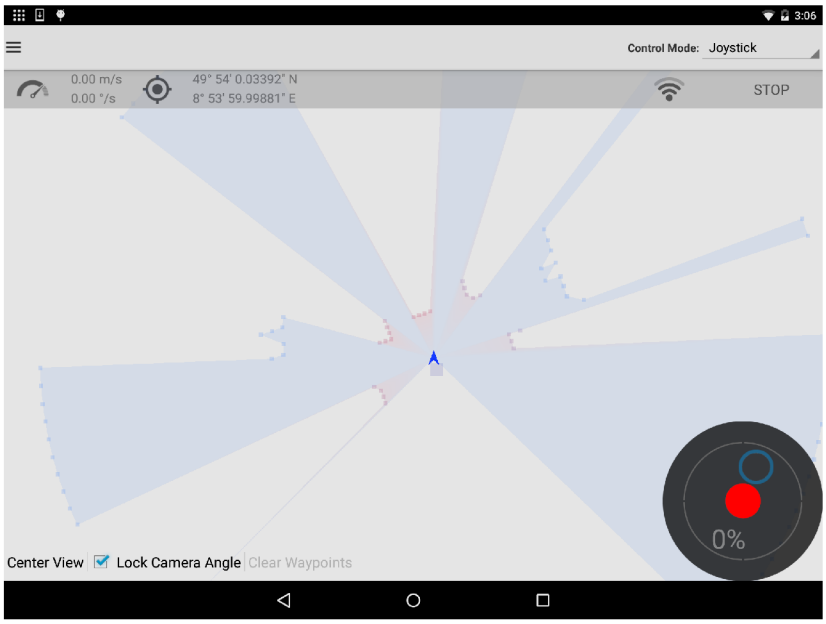
Je aplikace vyvinutá týmem akademiků na Univerzitě v Jižní Karolíně v USA. Poslední aktualizací prošla v roce 2017. Je dostupná jak ve formě zdrojového kódu (z [31]), tak i přímo v aplikaci Play systému Android [30].

Umožňuje vizualizovat data ze senzorů jednoho robotu ve 2D. Má implementováno celkem 5 typů zpráv a to: *laser scan, odometrie, joy-teleop, pose a navsat.* Je schopen zobrazit pozici na mapě na základě souřadnic ze zprávy *navsat*. Aplikace si též pamatuje už připojená zařízení.

Je kompatibilní se systémem Android verze 4.0 nebo vyšším. Založena je též na knihovně *rosjava* [31].



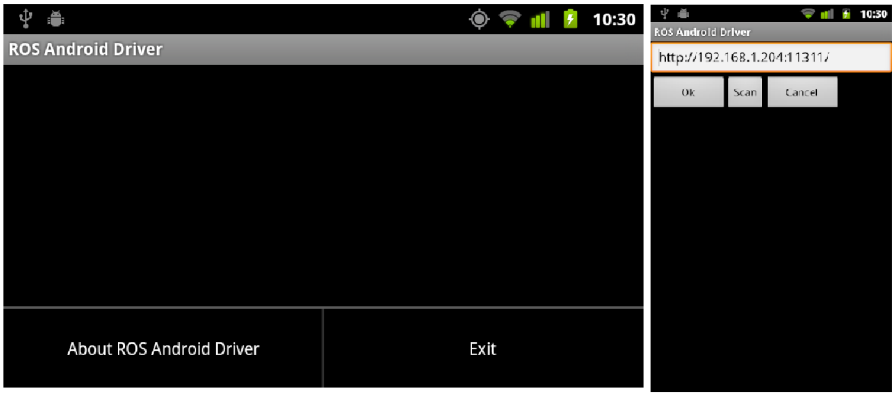
Obrázek 2.2: Aplikace ROS Control - úvodní obrazovka [30, převzato]



Obrázek 2.3: Aplikace ROS Control - vizualizace dat [30, převzato]

2.1.3 ROS Android Sensor Driver

Je aplikace Chada Rockeyho a Axela Furlana. Naposledy byla aktualizována v roce 2014 [32, 33]. Je dostupná, jak ve formě zdrojového kódu (dostupný z [33]), tak i přímo v aplikaci Play systému Android.



Obrázek 2.4: Aplikace ROS Android Sensor Driver [34, převzato]

Tato aplikace se odlišuje od předchozích tím, že umožňuje pouze publikování dat ze senzorů přenosného zařízení (mobil, tablet). Má implementované pouze čtyři typy zpráv, které přenášejí data z akcelerometru/gyroskopu, GPS a fotoaparátu. Jejich názvy jsou uvedeny v tabulce. Nemá žádnou vizualizaci a obsluha robotu je možná jen po navázání těchto dat na nějaký řídící algoritmus [33].

Název	Sensor
/android/imu	gyroskop, akcelerometr
/android/fix	GPS
/camera/camera_info	Fotoaparát
/camera/image/compressed	Fotoaparát

Tabulka 2.1: *Topics* publikované aplikací ROS ASD [33, převzato]

2.1.4 ROS android_core tutorial

Pro úplnost je nutné do této sekce přidat i balíček již hotových příkladů využití knihovny *android_core*, která úzce souvisí s knihovnou *rosjava*. Tato knihovna je k dispozici ve formě zdrojového kódu.

Dostupných je po jednom celkem 5 příkladů hotových aktivit jednoduchého *publishera/subscribera*, přenos obrazu z kamery atd. [35]. Příklady jsou zcela funkční, jsou však určeny spíše pro otestování funkčnosti komunikace. K ovládání robotu se příliš nehodí.

2.2 Aplikace na bázi komerčních systémů

Každý výrobce si svou aplikaci vždy uzpůsobil na míru svého výrobku. Jakékoliv změny nebo vylepšení uživatelem jsou pak prakticky vyloučeny.

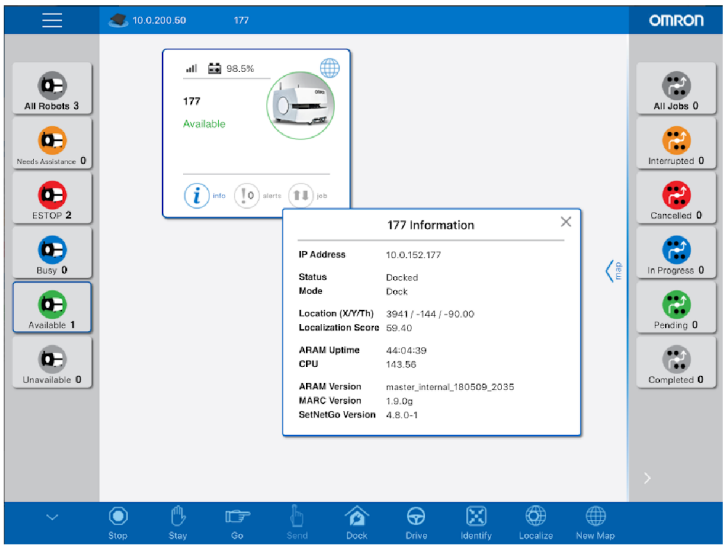
V dnešní době je už hojně využívána tato forma komunikace s robotem v oblasti domácích robotických vysavačů, venkovních sekaček a v oblasti skladovacích/transportních systémů, pokud tyto činnosti nejsou řízeny zcela automaticky.

2.2.1 Omron Mobile Planner

Omron Mobile Planner je aplikace vyvinutá japonskou firmou Omron zejména pro jejich mobilní roboty řady LD. Tyto roboty se využívají jako základ různých transportních/skladovacích systémů v průmyslu nebo ve zdravotnictví. Její odlehčená verze *Tablet edition* je podporována systémy iOS a Android. Lze ji zdarma stáhnout na stránkách výrobce ve formě hotové aplikace, nebo v obchodu Play [36], či v iOS App Store.

Aplikace umožňuje bezdrátové spojení a ovládání většího počty robotů najednou. Je určena zejména pro vnitřní prostory. Dovoluje sledovat operátorovi aktuální zaneprázdněnost zařízení, jejich pozici na mapě, aktuální frontu úkolů, nabití baterie a momentální rychlost. Je též schopen pomocí ní je ovládat, předávat nové úkoly a poslat jej neprodleně na dokovací místo [38].

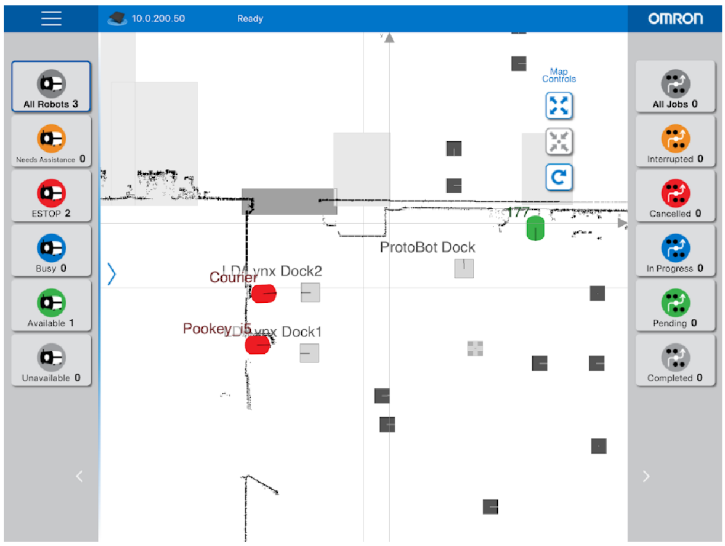
Aplikace je uzpůsobena pro systémy Android verze 6.0 a vyšší a pro iOS verze 10 a vyšší. Jediným dalším omezením je minimální velikost úhlopříčky obrazovky 5,5 *palce* [37].



Obrázek 2.5: Omron Mobile Planner - zobrazení informace o robotu [38, převzato]

2.2.2 iRobot HOME

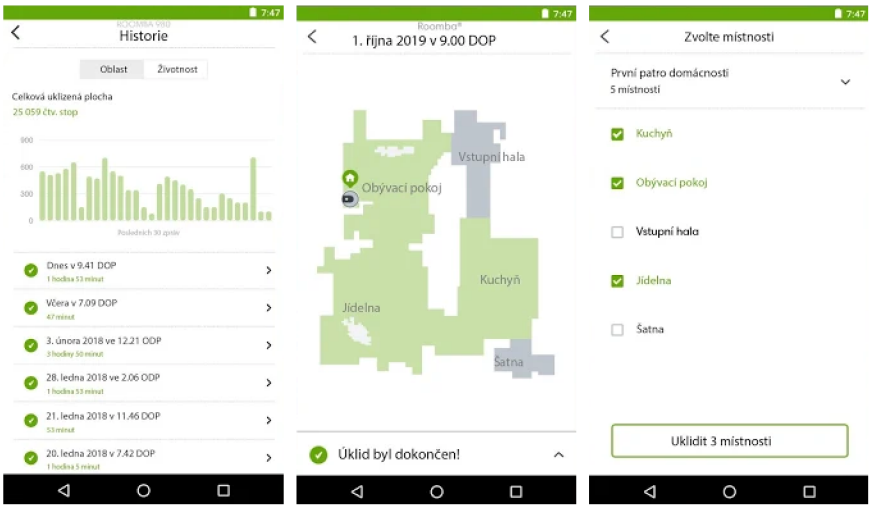
Tato aplikace byla vyvinutá americkou firmou iRobot pro jejich robotické vysavače řady Roomba. Aplikace je uzpůsobena pro systémy Android a iOS. Je volně k dis-



Obrázek 2.6: Omron Mobile Planner - vizualizace dat [38, převzato]

pozici na iOS App i v obchodu Play [39].

Jak už samotný produkt, pro který je určena, naznačuje, jedná se o uživatelsky přívětivou aplikaci, která má za úkol především zjednodušit základní nastavování robotu a případnou aktualizaci jeho systému. Přitom umožňuje i plánování čištění, sledování činnosti (např. už vyčištěnou plochu) a informuje uživatele o svém aktuálním stavu.



Obrázek 2.7: Aplikace iRobot HOME - podoba aplikace, vizualizace dat [39, převzato, upraveno]

2.3 Shrnutí

Z kriterií uvedených na začátku kapitoly je patrné, že žádná aplikace nevyhovuje úplně všem. Co se týče funkce aplikací, žádná z nich neřeší plně problémy uvedené v úvodu práce.

Báze aplikace - je důležitá pro její víceúčelovost. Je filosofickou otázkou, proč se už hotové dostupné mobilní aplikace na bázi systému ROS neuchytily v široké komunitě nebo nějakém komerčním systému, oproti jednoúčelovým komerčním aplikacím.

Vizualizace dat - srovnání vychází z porovnání s programem RViz. Z rešerše je patrné, že se mu nejvíce blíží aplikace Android RViz a ROS Control.

Aplikace ROS Control přebírá funkce RVizu pouze částečně. Má nedostatečný počet implementovaných typů zpráv.

I přes skoro úplnou podobu první uvedené aplikace RVizu je otázkou, zdali je potřebná 3D vizualizace na mobilních zařízeních, které mají sloužit nejen k zobrazování dat, nýbrž i k určitému zjednodušenému ovládání tohoto systému. 3D vizualizace též přináší určité nároky na hardware, ať už ve formě potřebného výpočetního výkonu k zajištění zobrazování v reálném čase, ale též i v samotné velikosti zařízení.

Proto je zajímavé zkoumat i aplikace na bázi komerčních systémů, kdy je jasně patrná snaha o maximálně přesnost uzpůsobení aplikace pro daný cíl. Naštěstí jsou uvedené komerční aplikace určené pro podobné využití, jako robotické systémy vyvíjené v ROSu (jedná se zejména o vizualizaci pohybu v prostoru, zadávání cílů a úkolů). Oba systémy jsou 2D vizualizace, kdy uživatel vidí pouze plochu s jednotlivými zařízeními.

Náročnost ovládání - cílem každého vývojáře aplikací by měla být bezesporu intuitivnost při jejich ovládání. Tomu napomáhá i grafická podoba aplikace. Časem se vkus člověka i aktuální podoba užitého designu mění. Tyto změny by však na funkčnost neměly mít vliv.

U většiny výše uvedených aplikací lze hodnotit tuto stránku jen z pohledu designu, neboť při tvorbě této práce nebylo k dispozici zařízení, na kterém by je šlo v praxi vyzkoušet.

Grafická podoba - hodnocení z pohledu designu je však poněkud ošidné. Lze tedy jen konstatovat, že většina výše uvedených aplikací netrpí od pohledu *informačním smogem* a svou podobou zřejmě plní svůj účel.

Z grafických náhledů je patrná jasná obecná struktura aplikace (posloupnost aktivit) - úvodní obrazovka konfigurace jednotlivých zařízení, která přechází k jednotlivým funkčním blokům.

Možnosti aplikace - je nutno definovat, co člověk od aplikace na bázi systému ROS vyžaduje. Pokud očekává jen funkcionality RVizu, pak ho lze překvapit i novými možnostmi, které poskytuje mobilní zařízení.

Proto nelze nechat bez povšimnutí i ty aplikace, ve kterých k vizualizaci vůbec nedochází. Je jím aplikace Android Sensor Driver. Ukazuje, že vestavěné sensory mobilního zařízení lze elegantně zapojit do fungování systému.

Je zajímavé si všimnout i možnosti zadávání úkolů v komerčních aplikacích. Obdobná funkcionalita by mohla výrazně pomoci širokému rozšíření nové aplikace.

V poslední době je na výrazném vzestupu tzv.virtuální realita a rozšířená realita. Oba směry mají velký potenciál, který je dosud ve spojení s ROSem nevyužit.

Podpora aplikace - je největším kamenem úrazu dnešních mobilních aplikací ROS. Dosud všechny pracují s knihovnou *rosjava*, která však podporuje pouze verzi ROS Kinetic. Otázkou zůstává, zdali dojde k její oficiální aktualizaci na nové verze, nebo bude vytvořena úplně nová (i v jiném programovacím jazyce), využitelná při vývoji na platformě Android nebo Apple.

Kap. 3

Framework ROS

Před formulací požadavků na podobu aplikace je nutné analyzovat systém ROS z pohledu jeho možností a komunikačního schématu.

Framework ROS (*Robot Operating System*) je sada nástrojů a knihoven, která usnadňuje a zrychluje vývoj komplexních robotických systémů. Tento systém umožňuje softwarové spojení podsystémů robotu založených na různých platformách.

Celý koncept jeho vývoje je založen na myšlence svobodného softwaru (konkrétně licence BSD [7]), kdy je k dispozice již hotové jádro systému a kdokoliv může pracovat na jeho vylepšení. Vzhledem k celosvětovému úspěšnému rozšíření tohoto frameworku, je v dnešní době k dispozici celá řada knihoven a balíčků, které zásadně posunují jeho možnosti.

Jeho počátky můžeme datovat do roku 2007, kdy na Stanfordově univerzitě v USA započali Eric Berger a Keenan Wyrobek práci na systému, který by byl schopen usnadnit, zrychlit vývoj a umožnit všem, kteří nemají potřebné znalosti ve všech oborech spojených s robotikou pracovat i v této oblasti [8]. Od této doby prošel bouřlivým vývojem. Do dnešních dnů bylo vydáno přes 10 distribucí. Starší distribuce už jsou nepodporované, aktuálními podporovanými verzemi jsou Melodic Morenia (vydáno 2018) a Kinetic Kame (vydáno 2016)[8].

Oficiálně je podporován pouze operačním systémem Linux.

3.1 Koncepce systému

Ve tvorbě jakýchkoliv děl v různých oborech lze spatřovat jeden společný rys. Tento rys se dá nazvat různými termíny - dělbou práce, dekompozicí ... - tyto názvy však popisují jedno a totéž. Při tvorbě je nutno vždy dané dílo (problém) rozložit na jednotlivé funkční celky. Tyto celky se pak v průběhu tvorby spojují a vytvářejí výsledné dílo.

Tento přístup lze pozorovat i v systému ROS. V zásadě pracuje na principu dekompozice problému na jednotlivé funkční celky, které pak vzájemně komunikují. V této síti lze rozpoznat následující prvky [13, 12].

3.1.1 Nodes

Nodes jsou základními stavebními kameny systému. Jsou to bloky, které přímo provádí nějakou činnost (lokalizace, plánování, ovládaní motorů atd.) [9]. Existuje mnoho balíčku/knihoven (*client libraries*), které umožňují vytváření těchto bloků v různých programovacích jazycích (nějčastěji balíček *rospy* pro Python, knihovna *roscpp* pro C++ nebo v mé práci často využívaná experimentální knihovna *rosjava* pro jazyk Java).

Tyto uzly jsou po spuštění schopny pracovat samostatně. Jako součást nějakého systému by však jejich samostatná činnost neměla smysl. Je nutné, aby byly spojeny mezi sebou a mohly komunikovat. Toho je docíleno tzv. *topics*, které jsou *nodem* odebírány nebo publikovány.

3.1.2 Topics

Topic je v podstatě označení informace, která probíhá systémem. Je jasné, že je jich nebřeberné množství a záleží jen na konkrétním robotikovi, jak svůj systém dekomponuje. Je nutno poznamenat, že k označení informace je nutno připojit i informaci samotnou. To se provádí pomocí tzv. zpráv - *messages*. Při každé inicializaci *topic* je nezbytné zadefinovat i druh přijímané *message* [10].

3.1.3 Messages

Message je informace samotná. Každá zpráva musí splňovat jasně daný formát a musí obsahovat jen předem známé datové typy [11]. Při komunikaci je doporučováno využívat již definované typy zpráv, kterých je celá řada v různých aplikacích (např. základní sada *std_msgs*). Pomocí těchto zpráv je tedy možno předávat kontinuálně informace mezi *nodes* v programu.

3.1.4 Services

To však u některých akcí není výhodné. Občas je potřeba provést jednorázový úkon typu „požadavek - odpověď“ (*Request / reply*). To je v systému ROS umožněno implementací tzv. *Services* [12].

3.1.5 Master

Představuje server, který zastřešuje celou komunikaci v systému ROS. Musí být vždy definován pomocí proměnné prostředí *ROS_MASTER_URI*. Pokud není nadefinováno jinak, jako *Master* je bráno to zařízení, na kterém je spuštěn systém (*roscore*) [18].

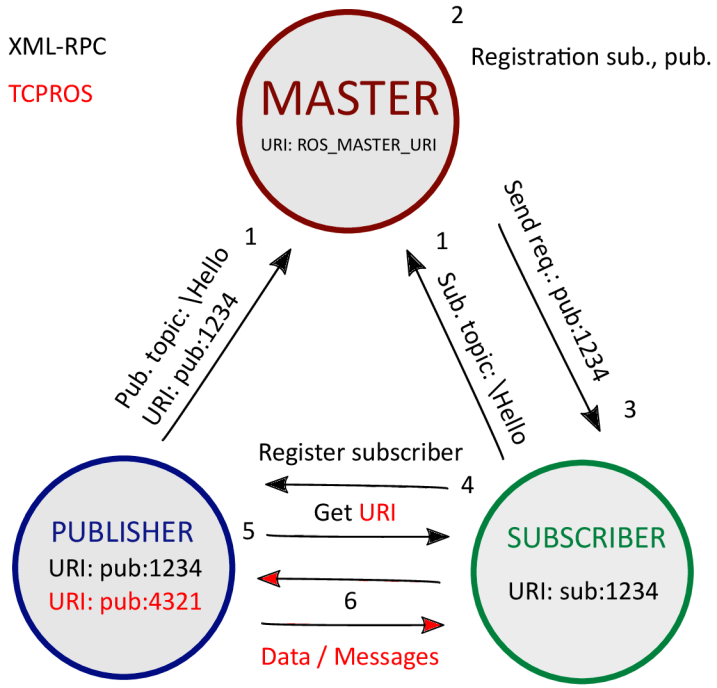
Je nutno zde zmínit, že *Master* též spravuje tzv. **Parameter Server**, což je slovníková databáze, ve které mohou být uložena ta data, která jsou důležitá pro činnost celého systému [15].

3.2 Komunikace v systému ROS

Vzhledem k povaze této práce je nutné se detailně zaměřit na komunikaci uvnitř frameworku.

ROS je nejen multiplatformní, nýbrž disponuje možností propojení více nezávislých zařízení přes lokální síť. Celý již běžící systém se velice podobá počítačové síti *peer-to-peer*, je zde však určitá odlišnost. Při spouštění frameworku je vždy nutné definovat tzv. *mastera*, který celou komunikaci inicializuje a umožňuje spojení jednotlivých funkčních bloků. To vede k myšlence, že ostatní zařízení v tom okamžiku je nutné brát jako tzv. *slaves* [13, 12].

Celé komunikační schéma je znázorněno na diagramu. Je z něj jasná posloupnost dějů při navazování kontaktu mezi uzly. V první fázi, po spuštění všech zařízení v systému, dojde k registraci všech *subcriberů* a *publisherů*. Na základě shody odebíraných/publikovaných *topics*, *master* rozešle jednotlivým *subcriberům* adresu *publisherů* jimi požadovaných *topics*. Poté jim *subscribteři* odešlou požadavek na zasílání požadovaného *topicu*. *Publisher* odpoví formou adresy, ze které budou zprávy odesílány. Poté mezi sebou navážou komunikaci přímo, bez účasti *mastera* [13].



Obrázek 3.1: Schéma propojení *Subscribera* a *Publishera* [13, převzato, upraveno]

Parameter Server je spuštěn na *Masteru* a pomocí dotazování přes síťové API, má možnost každý člen systému z něj získat potřebné data (pokud zná klíčové slovo). Tato komunikace probíhá opět přes XML-RPC [13].

Tato komunikace probíhá s využitím několika protokolů. Pro komunikaci s *masterem* je využíván protokol XML-RPC, pro posílání dat jednotlivých *topics* je pak

využíváno protokolů TCPROS nebo UDPROS.

3.2.1 XML-RPC

Před samotným popisem tohoto protokolu je třeba zmínit, co to znamená tzv. vzdálené volání procedur (*remote procedure call - RPC*). Je to metoda, kdy můžeme z jednoho zařízení, vykonat nějakou proceduru na zařízení jiném. Toto zařízení však musí mít danou proceduru implementovanou a musí být připravené na případné volání [16]. Tohoto systému je s oblibou využíváno při tvorbě distribuovaných systémů (tedy i při tvorbě systému ROS).

Jedním z implementací výše zmíněného principu je právě komunikační protokol XML-RPC, jehož počátek je datován zhruba do roku 1998, kdy byl vytvořen Davem Winerem. XML-RPC je v zásadě založen na kombinaci protokolu HTTP a značkovacího jazyka XML [17].

Protokol HTTP slouží jako základní komunikační rámec, ze kterého je převzata celá struktura a tvar hlavičky. Celý proces pak probíhá jako klasické dotazování a odezva dvojice počítač-server.

Jméno procedury a parametry pro zařízení, na kterém chceme danou proceduru vykonat, je nutné naformátovat do podoby značkovacího jazyka XML. Tato část je potom připojena k části dotazu HTTP.

Hlavička (HTTP) [17, převzato] musí vždy obsahovat položku *User-Agent* (specifikátor prohlížeče), *Host* (adresa/doména serveru), *Content-Type* (druh obsahu souboru, u XML-RPC je to vždy *text/xml*) a *Content-Lenght* (délka souboru v bytech).

Soubor (XML) [17, převzato] má jasně definovaný kořen se jménem *<methodCall>* (při dotazu) a *<methodResponse>* (při odpovědi). Při dotazu musí být uvedeno jméno volané procedury. Pokud má volaná procedura parametry, musejí být uvedeny jako *<params>*. Tyto parametry mohou být jen některé datové typy (*string, boolean* atd.). Všechny povolené typy jsou uvedeny ve specifikaci.

3.2.2 TCPROS

TCPROS je používaný komunikační protokol se zárukou doručení k zasílání dat jednotlivých *topics* a *servisů* využívající standardní síťový socket [20].

Rámec zasílaný přes tento protokol má vždy hlavičku a připojený soubor s daty.

Hlavička může obsahovat různé údaje v závislosti na tom, zdali posílá daný paket *subscriber*, *publisher* nebo je třeba k provedení tzv. *service*.

Subscriber [20, převzato] musí uvést vždy následující položky *message_definition*, *callerid*, *topic*, *md5sum* a *type*. Dále může připojit i položku *tcp_nodelay*.

Položka	Popis
callerid:	*jméno nodu*
topic:	*jméno topicu*
service:	*jméno servisu*
md5sum:	*kontrola MD5*
type:	*typ zprávy*
message_definition:	*detailní popis zprávy (výstup z gendep –cat)*
error:	*flag - indikuje, že komunikace nebyla navázána*
persistent:	*flag - indikuje, zdali je service volatelný*
tcp_nodelay:	*flag - indikuje možnou změnu TCP_NODELAY*
latching:	*flag - indikuje, zdali je publisher v režim latching*

Tabulka 3.1: Možné položky údajů v hlavičce paketu TCPROS [20]

Publisher [20, převzato] musí uvést vždy *md5sum* a *type*. Nepovinně může přidat i *callerid* a *latching*. Režim *latching* je specifický tím, že *publisher* při něm posílá poslední publikovanou zprávu nově registrovaným *subscriberům*.

Service [20, převzato] pro úspěšné provedení musí obdržet v hlavičce následující údaje: *callerid*, *service*, *md5sum* a *type*. Nepovinně se může uvést i položka *persistent*. Na úspěšné spojení vždy odpovídá paketem obsahující jeho *callerid*.

Ke každému paketu je pak možné uvést položku *error*, která slouží k detekci chyby spojení uživatelem.

3.2.3 UDPROS

Je dalším možným protokolem při zasílání dat. Je však bez záruky doručení (je založen na bázi protokolu UDP). Často je využíván tam, kde nezáleží na tom, zdali dostaneme všechny posílaná data, ale na tom, abychom tyto data dostávali co nejrychleji. V tomto protokolu neprobíhá kontrola o ztrátě dat, nebo o pořadí přicházejících datagramů [19].

Struktura datagramu je obdobná jako u paketu. Obsahuje hlavičku a uživatelská data.

Hlavička obsahuje celkem 4 položky.

Položka	Popis
Connection ID	*cílová adresa datagramu*
Opcode	*typ datagramu*
Message ID	*pořadí datagramu*
Block #	*mění se v závislosti na Opcodu*

Tabulka 3.2: Položky údajů v hlavičce datagramu UDPROS [19]

Block # obsahuje v závislosti na *Opcodu* různá čísla, pokud je DATA0, pak obsahuje předpokládaný počet datagramů k vytvoření zprávy, pokud není DATA0, pak je nulový, pokud je DATAN, pak obsahuje aktuální číslo datagramu [19, převzato].

Opcode	Popis
DATA0 (0)	*je posílán v prvním datagramu*
DATAN (1)	*je posílán ve všech následujících datagramech*
PING (2)	*posílán periodicky ke zjištění funkčnosti spojení*
ERR (3)	*poslán při přerušení komunikace*

Tabulka 3.3: Typy *Opcodu* [19]

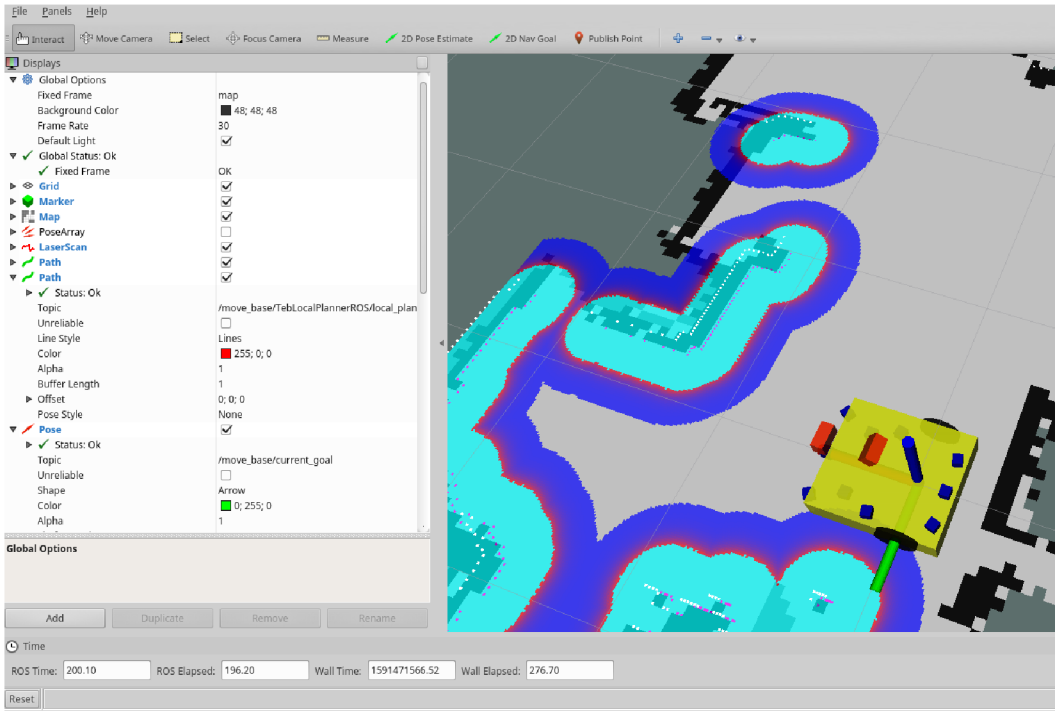
3.3 Součásti frameworku ROS

Z pohledu uživatele a vývojáře uživatelského rozhraní je nutné se zde zmínit o již implementovaných a ověřených uživatelských rozhraních systému ROS.

Systém v základu obsahuje řadu nástrojů usnadňujících práci při jeho obsluze, eventuálně umožňujících například i vizualizaci některých dat, které jsou v něm přítomny.

3.3.1 RViz

Je nástroj umožňující 3D vizualizaci dat pohybujících se v systému. Bývá součástí instalačního balíčku ROSu. Uživatel má možnost nastavit si, jaké data si přeje zobrazit a také to, jak se mu budou zobrazovat [25].



Obrázek 3.2: Prostředí RViz

Je zde implementována celá řada typů zpráv, které mají nejčastěji geometrický význam a jsou vztaženy k nějakému souřadnicovému systému. RViz též vypisuje informace obsažené v hlavičce zprávy. Umožňuje též zobrazit model robotu, nadefinovaný v souboru typu *.urdf*.

Je schopen některé zprávy nejen odebírat, nýbrž i publikovat (jedná se o zprávu typu *PoseStamped* a *PointStamped*) [25]. Na tom je jasně vidět, že RViz není pouze vizualizačním nástroj, ale je schopný uživateli umožnit jeho prostřednictvím i jednoduchou interakci se systémem.

Název	**Popis**	**Balíček - typ zprávy**
Axes	*Zobrazuje souřadnicové osy*	
Effort	*Zob. otočení v rotačních vazbách*	sensor_msgs/JointStates
Camera	*Zobrazí obraz z kamery*	sensor_msgs/Image sensor_msgs/CameraInfo
Grid	*Zobrazuje 2D/3D mřížku*	
Grid Cells	*Zobrazí mřížku*	nav_msgs/GridCells
Image	*Zobrazí obrázek*	sensor_msgs/Image
InteractiveMarker	*Zobrazí interaktivní marker*	visualization_msgs/InteractiveMarker
Laser Scan	*Zobrazuje data z lidaru*	sensor_msgs/LaserScan
Map	*Zobrazuje mapu*	nav_msgs/OccupancyGrid
Markers	*Zobrazí interaktivní marker*	visualization_msgs/Marker visualization_msgs/MarkerArray
Path	*Zobrazuje cestu*	nav_msgs/Path
Point	*Zobrazuje bod*	geometry_msgs/PointStamped
Pose	*Zobrazuje pozici a natočení*	geometry_msgs/PoseStamped
Pose Array	*Zobrazuje pozice a jejich natočení*	geometry_msgs/PoseArray
Point Cloud(2)	*Zobrazuje mrak bodů*	sensor_msgs/PointCloud sensor_msgs/PointCloud2
Polygon	*Zobrazuje mnohoúhelník*	geometry_msgs/Polygon
Odometry	*Zob. data z odometrie*	nav_msgs/Odometry
Range	*Zob. data ze snímačů vzdálenosti*	sensor_msgs/Range
RobotModel	*Zob. model robotu*	
TF	*Zob. data o transformacích*	
Wrench	*Zob. sílu nebo moment*	geometry_msgs/WrenchStamped
Oculus	*Zob. RViz pro virtuální realitu*	

Tabulka 3.4: RViz - implementované typy zpráv [25, převzato, upraveno]

3.3.2 Rosbash

Je klíčové uživatelské rozhraní systému ROS fungující v příkazovém řádku systému Linux. Umožňuje spouštění systému, nastavování systémových proměnných, výpis ze systémových proměnných, konfiguraci a sledování jednotlivých jeho částí v běhu v reálném čase [23].

I přes terminálový styl práce je velice užitečný. Uživatel má v rukou celou sadu příkazů, pomocí nichž je schopen svůj systém analyzovat a konfigurovat. Jejich výpis je uveden v tabulce.

Příkaz	Využití
rosbag	*práce s Bag soubory (analýza systému)*
roscd	*změna aktuálního balíčku/složky*
rosclean	*čištění log souborů generovaných ROSem*
roscore	*spuštění systému*
rosdep	*instalace balíčků*
rosed	*otevře editor s požadovaným souborem*
roscreate-pkg	*vytváření nových balíčků*
roscreate-stack	*vytváření nových stacků*
rosrun	*spouštění scriptů v balíčku*
roslaunch	*spouštění systému a scriptů (pomocí XML)*
rqt_bag	*graf. výpis z Bag souborů*
rqt_deps	*generuje PDF s graf. strukturou systému*
rosnode	*práce s uzlem*
rospack	*práce s balíčkem*
rosparam	*práce s Parameter Serverem*
rossrv	*práce se Service*
rosservice	*práce se Service*
rosstack	*práce se stacky*
rostopic	*práce s tzv.Topics*
rosversion	*zjištění verze ROSu*
rosmake	*práce s balíčky*
rosmsg	*práce se zprávami*
rqt_graph	*zobr. graf. strukturu systému*
rqt_plot	*vypisuje data ze systému*

Tabulka 3.5: Rosbash - implementované příkazy [24, převzato, upraveno]

Kap. 4

Návrh podoby aplikace

Následující kapitola by měla detailně definovat funkce nové mobilní aplikace, ale též i směr a cíl jejího vývoje.

4.1 Požadavek

Z rešerše je patrné, že žádná aplikace neřeší všechny problémy uvedené v úvodu práce a ty, které řeší, tak pouze částečně. Na základě této skutečnosti a analýzy systému ROS byly formulovány 3 typy požadavků na aplikaci.

Nutné funkcionality aplikace - jsou takové, bez jejichž implementace by vývoj neměl smysl (jsou uvedeny jako cíl práce).

Je nutno vytvořit jednoduchý terminál, který by umožňoval start systému, jeho konfiguraci, výpis ze systémových proměnných a jiné s ním spojené činnosti. V podstatě je nutno umožnit uživateli na mobilním zařízení využívat *Rosbash*.

Druhou nutnou funkcionalitou je implementace vizualizace dat, které jsou přítomny v systému ROS. Bylo rozhodnuto, že by kvůli uživatelskému komfortu a přehlednosti měla být pouze dvourozměrná a podobná systému RViz.

Funkcionality přidané hodnoty - jsou takové, které posunují hranice využití této aplikace od pouhé vizualizace dat a ovládání systému k větší interakci s uživatelem. Jedná se zejména o možnost toho, aby uživatel mohl v reálném čase určovat nějakým způsobem chování ovládaného zařízení (robotu), zadávat mu úkoly apod..

Obecné požadavky - jsou více méně intuitivní, neboť k nim neexistuje jasná definice. Jsou jimi přehlednost aplikace, jednoduché ovládání, ohromující grafická podoba a celková přívětivost. Je jasné, že tyto požadavky budou splněny jen v rámci vkusu každého robotika ji využívajícího.

4.2 Návrh

Z předešlé sekce plyne závěr nutnosti vývoje mobilní síťové aplikace interagující se systémem ROS. Po důkladné analýze požadavků bylo rozhodnuto, že budou spl-

něny následujícím způsobem.

Aplikace bude vyvíjena pro systém Android v jazyce Java, z důvodu velké rozšířenosti a jednoduchosti vývoje. Z důvodu nedostupnosti novějších zařízení při vývoji bude podporována již verzí operačního systému Android 4.4.

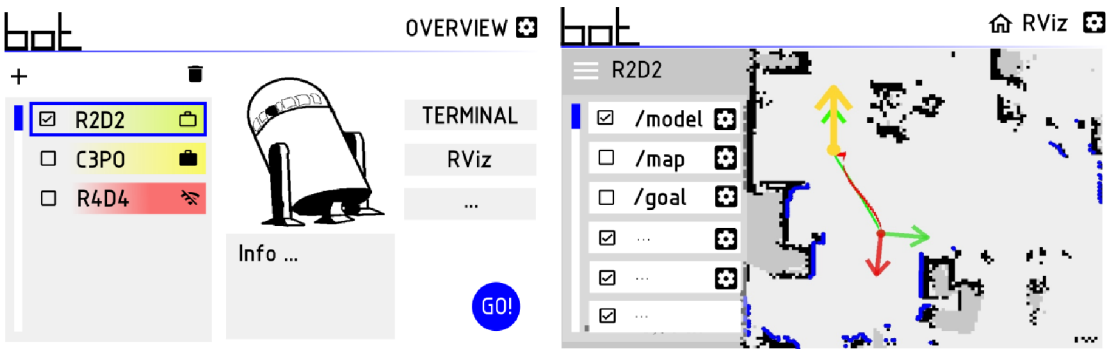
Jazyk Java byl vybrán i z důvodu dostupnosti jediné funkční aktuální knihovny ROSu *rosjava* pro systém Android. Knihovna podporuje verzi Kinetic. Tím byla vybrána i distribuce systému ROS.

Nutná funkcionalita Rosbashe bude řešena implementací shellu, který bude připojen přes síť k ovládanému zařízení. Pro tento účel byla zvolena volně dostupná knihovna s implementací protokolu SSH-2 *JSch* (*Java security channel*).

4.2.1 Rozvržení aplikace

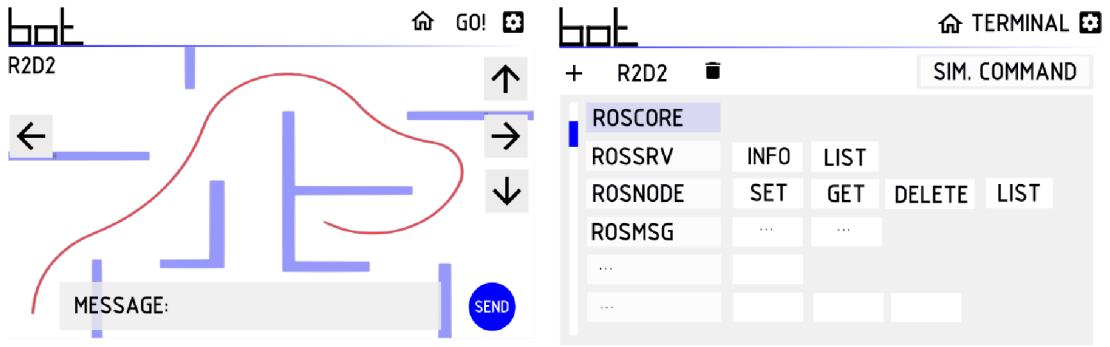
Při návrhu tzv.*layoutu* aplikace bude dodrženo obvyklé a již osvědčené schéma aplikace. Bude rozdělena do následujících částí - tzv. *aktivit*.

Přehled - úvodní aktivita bude sloužit k připojení zařízení a jeho základnímu nastavení. Aplikace by si měla pamatovat jednotlivá zařízení.



Obrázek 4.1: Návrh - Přehled a Vizualizace

Terminál - aktivita spojená s implementací Rosbashe. Mělo by se jednat o v podstatě přívětivější příkazový řádek umožňující rychlé zadání požadovaného příkazu.



Obrázek 4.2: Návrh - Jeď! a Terminál

Vizualizace - aktivita spojená s vizualizací dat. Půjde v podstatě o zjednodušenou implementaci RVizu. Je nutno se vyvarovat přílišnému množství informací pro uživatele.

Jeď! - tato aktivita by měla umožňovat zjednodušené přímé ovládání zařízení (robotu). Měla by obsahovat jen předem definované typy bloků, které by měly jasně danou funkci (ovládací šipky, mapa atd.).

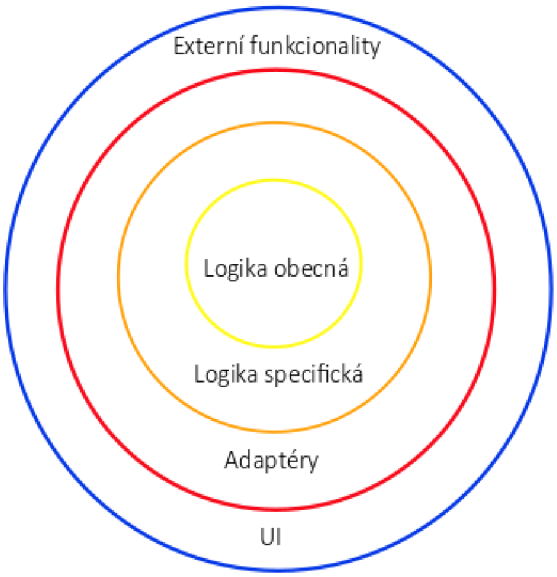
Kap. 5

Struktura uživatelských aplikací

Před popisem vlastní aplikace je nezbytné se krátce zmínit o uživatelských aplikacích z hlediska jejich obecné ideální vnitřní struktury.

Uživatelské aplikace jsou dnes nedílnou součástí každého mobilního zařízení. Pod tímto termínem si lze představit celou řadu různě provedených programů. Jejich základním účelem je interakce s uživatelem. Každá taková aplikace tedy musí obsahovat nějaké rozhraní mezi ní a uživatelem. Nejčastěji se jedná o textové, nebo grafické (*GUI - graphic user interface*).

Obecnou ideou je strukturovat aplikaci tak, že je tvořena jednotlivými vrstvami, které komunikují jen v rámci svých sousedů. Dle [1] to lze jednoduše naznačit diagramem.



Obrázek 5.1: Ideální struktura aplikace [1, převzato, upraveno]

Obecná logika je předem dána platformou, pro kterou je daná aplikace vyvíjena. Přebírá od ní celou strukturu dat a vazeb mezi nimi.

Specifická logika zajišťuje konkrétní úlohu aplikace užitím struktur z logiky obecné. Pokud je aplikace vyvíjena samostatně, přebírá úlohu logiky obecné.

Adaptéry slouží jako mezivrstva, která data z logiky převádí do požadovaných struktur vrstev na ni navazujících.

Externí funkcionality a UI je vrstva, která je tvořena těmi strukturami a funkcionalitami, které neprovádí samotná aplikace. Často se jedná o konkrétní implementaci práce s různými periferními zařízeními a knihovnami.

5.1 Návrhové vzory

Při vývoji aplikací se v dnešní době používá mnoho tzv. návrhových vzorů, což jsou obecné postupy a struktury, které by měli být dodrženy, aby výsledkem byla aplikace s přehledným členěním kódu. Je samozřejmé, že k návrhovým vzorům se přidávají obecně platné poučky ke vytváření tzv. čistého kódu.

Obecným pravidlem návrhových vzorů je dekompozice struktury aplikace. Návrhový vzor neříká nic o implementaci jednotlivých struktur. Hovoří vždy jen o ideálním provedení dané obecně používané funkcionality.

V dnešní době se rozšířilo v oblasti GUI několik návrhových vzorů.

5.1.1 Tree View

Celé grafické rozhraní lze rozdělit do jednotlivých ovládacích prvků. Ty lze poté sdružovat do celků, které vytvářejí stromovité struktury. Každý jednotlivý prvek pak reaguje na podněty, které má implementované a tím ovlivňuje prvky, které se nacházejí v hierarchii pod ním [5].

5.1.2 Pozorovatel

Schéma pozorovatele je využíváno při vyvolání změn jakýkoliv změn vzájemně propojených celcích. Vždy je nutné nadefinovat tzv.*observer* a *object*, kdy jednotlivý pozorovatelé mohou být navázáni na jeden objekt [5].

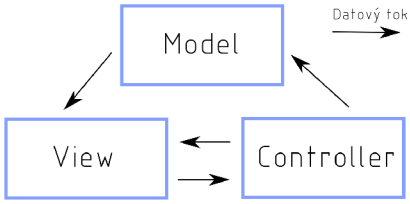
5.1.3 Model-View-Controller

Je schéma kombinující více přístupů. Jeho hlavní myšlenkou je dělení aplikace na 3 vzájemně propojené části - tzv. *model*, *view* a *controller*. Každá část zajišťuje jinou funkci [6].

View se stará o obsluhu grafického rozhraní. Měly by v něm být nadefinovány všechny jeho prvky. Zobrazuje předem nadefinovaná data.

Model zajišťuje obsluhu dat aplikace. O případných jejich změnách informuje *view*, které je zobrazí.

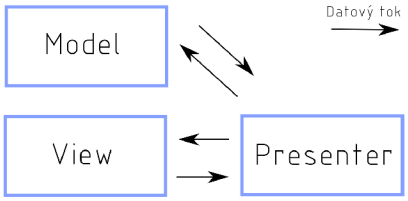
Controller je část, která zajišťuje obsluhu vstupů od uživatele, které obvykle souvisejí se změnou dat nebo rozhranní. Ty pak dále propaguje do *view* a *modelu*.



Obrázek 5.2: Schéma MVC [4, převzato, upraveno]

5.1.4 Model-View-Presenter

Se liší od předchozího existencí tzv. *presenteru*, který přebírá funkci *controlleru* a zároveň obsluhu celé aplikační logiky. Tím se stává prostředníkem mezi *view* a *modelem* [6].

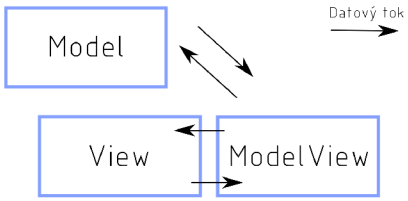


Obrázek 5.3: Schéma MVP [3, převzato, upraveno]

5.1.5 Model-View-Binder

Vychází s předchozího schématu, liší se však použitím tzv. *View modelu*, který obsahuje tzv. *binder*.

View model na rozdíl od *presenteru* umožňuje přístup k jednotlivým ovládacím prvkům celého *view*.



Obrázek 5.4: Schéma MVB [2, převzato, upraveno]

Kap. 6

Využité technologie

Při vývoji této aplikace je nutné využít nebo se alespoň seznámit s několika různými platformami a způsoby jejich možného propojení.

6.1 Operační systémy

Operační systém je nedílnou součástí softwarového vybavení počítače. Zajišťuje spojení mezi periferiemi počítače a vysokoúrovňovými aplikacemi.

Jeho hlavní úlohou je správa jednotlivých běžících procesů. Pod tímto abstraktním názvem si lze představit prakticky jakoukoliv operaci spuštěných aplikací. Ty často provádějí rychlé jednorázové asynchronní/synchronní činnosti. Provádí jejich řízení a umožňuje jejich současný běh.

Též přehledně strukturuje a spravuje uživatelova data a zároveň provádí rozhraní mezi ním, jeho aplikaci a hardwarem, na kterém celý tento systém běží.

Framework ROS je podporován několika verzemi operačnímu systému Linux. Cílovou platformou budoucí aplikace je systém Android. Ten sdílí s verzemi Linux společné jádro.

6.1.1 Jádro Linux

Vznik prvního jádra operačního systému Linux lze datovat do 1991, kdy ho vytvořil Fin Linus Torvalds pro architekturu procesorů i386. Od počátku byl zamýšlen jako svobodný software (licence GNU), založený na myšlenkách operačního systému Unix. Díky své propracovanosti a dostupnosti se stal celosvětově populárním.

Linuxové jádro se vyznačuje obecnými rysy systému Unix. Při jeho využívání je nutné si nadefinovat jednotlivé uživatele a jejich přístupová práva.

Uživatelský program pak může běžet obvykle ve dvou režimech, v uživatelském nemá přístup k datům jádra, v tzv *kernel* módu ano. Procesy jsou přepínány do jednotlivých módů na základě systémových volání nebo přerušení (při asynchronních procesech, zejména periferních zařízení). Při každém jejich přerušení nebo zastavení je systémem uložen jejich aktuální stav, ke kterému je třeba se posléze vrátit (číslo instrukce, potřebná data atd.)[40, str. 19].

Pro každý proces je vyčleněn vlastní adresový prostor v paměti. Při požadavku procesu nebo jádra může sdílen. Mezi jednotlivými procesy je též zajištěna syn-

chronizace několika způsoby, aby nevznikla chyba při přístupu a přepisu sdílené systémové proměnné [40, str. 23].

Pro zajištění synchronních a asynchronních událostí jsou implementovány tzv.*signály*, které jsou buď ignorovány nebo obslouženy nějakým předem definovaným procesem.

Obsluha paměti jádrem patří k nejkomplexnějším partiím tohoto systému. Systém vytváří tzv. *virtuální paměť*, která tvoří mezičlánek mezi fyzickou pamětí a jednotlivými uživatelskými daty. Ta je tvořena jednotlivými tzv.*virtuálními adresovými prostory*, které jsou odlišné od těch fyzických [40, str. 30]. .

Jádro též upravuje používání paměti RAM, kdy ji dle aktuálního využití dělí mezi virtuální paměť a místo určené pro dat jádra samotného. Pro správný běh je v systému implementován tzv. *kernelovský paměťový alokátor*, který se snaží uspokojit paměťové požadavky jednotlivých běžících procesů [40, str. 31]. .

V neposlední řadě je nutno zmínit i obsluhu periferních zařízení. Kdy jsou v paměti uloženy jednotlivé ovladače těchto zařízení a ty pak interagují dle potřeb s jednotlivými procesy.

Kvůli nutnosti ruční instalace některých balíčků umožňujících klasický uživatelský komfort práce (grafika, správce souborů, textové editory), volí většina uživatelů raději instalaci některé z již uživatelsky přívětivějších distribucí operačního systému Linux.

6.1.2 Operační systém Linux

Pod pojmem operační systém Linux se často rozumí už jádro samotné. Obvykle si však pod tímto termínem člověk představí spíše nějakou z distribucí implementující přívětivější uživatelské prostředí (např. Ubuntu, Raspbian atd.).

Se samotným operačním systémem Linux je aplikace v přímém kontaktu při činnostech spojených se zapínáním, konfigurací a kontrole systému ROS. Tyto činnosti se obvykle provádí pomocí jednoduchého uživatelského rozhraní typu příkazového řádku.

Shell v Unixu podobných systémech se toto rozhraní často nazývá tzv.*shellem*. Umožňuje uživateli komunikaci s jádrem systému. V něm implementována nejen řada příkazů souvisejících od správy souborů, přes obsluhu překladačů až ke správě sítě, nýbrž i řídící struktury. Je tedy s nimi možné i scriptovat. Shellů je celá řada, mezi nejznámější patří např. *Bourne shell* (*sh*), *C shell* (*csh*) nebo *Korn shell* (*ksh*).

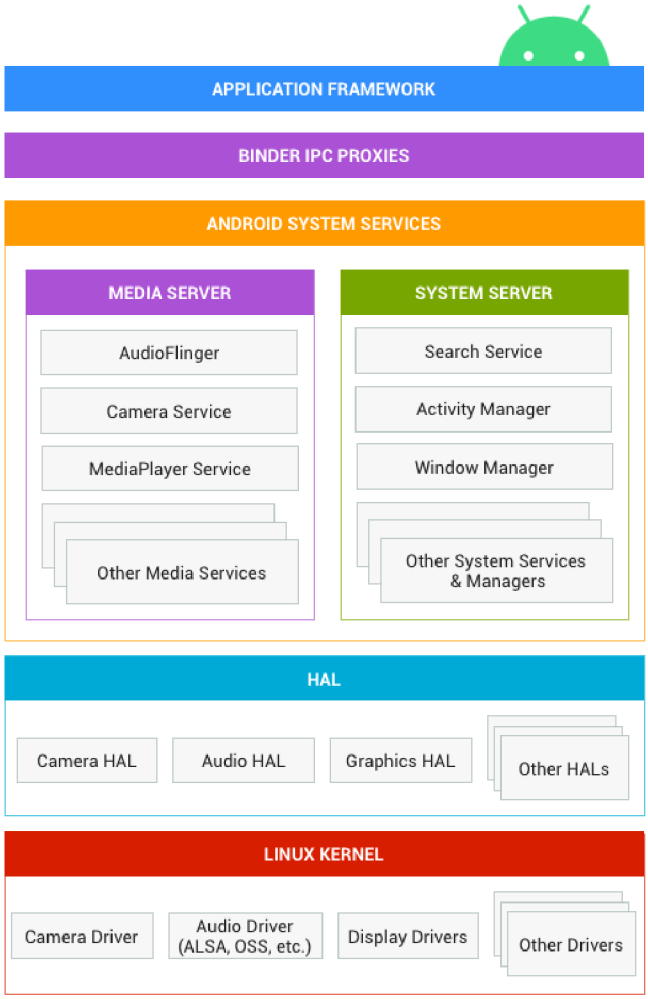
Implementací *shellovských* příkazů je i dříve zmíněný Rosbash. Pro budoucí aplikaci je tedy nutné, aby uměla pracovat s tímto rozhraním.

6.1.3 Operační systém Android

Vznik jeho první verze lze datovat zhruba do roku 2007. Jeho vývoj je úzce spjat s firmou Google, která v roce 2007 iniciovala jeho vznik založením uskupení několika technologických firem, které mělo za cíl definovat požadavky na společný systém pro mobilní zařízení. Od počátku je vyvíjen jako open source. V dnešní době se jedná, díky množství mobilních zařízení, o nejrozšířenější operační systém na světě [41].

Od jeho uvedení na trh bylo vytvořeno mnoho distribucí, lišící se implementovanými funkcemi, knihovnami, grafickým provedením apod.. Z pohledu budoucí aplikace je důležité zmínit, že bude podporována už verzí Android 4.3.

Android je založen na linuxovém jádře. Je rozdělen do několika vrstev, jejichž struktura je jasně patrná v diagramu. Systém byl vyvíjen s důrazem na zajištění jeho kompatibility s různými zařízeními.



Obrázek 6.1: Struktura systému Android [42, převzato]

Linuxové jádro systému zajišťuje přímou komunikaci s hardwarem. Verze vhodné pro systémy Android se liší v několika aspektech. Často souvisejí s vlastnostmi mobilních zařízení.

Prvním z nich je nutnost uzpůsobení jádra pro častější přepínání systému do režimu šetření energií. To je řešeno tzv. *wakelocks*, které systému musejí říci, kdy je to možno provést a kdy nikoliv (čekání na vstup od uživatele, běžící proces na pozadí atd.) [43, str. 36].

Dalším uzpůsobením je implementace dalšího uvolňovače paměti (*Low memory killer*), který ukončuje procesy ještě před základním linuxovým, který je tak vyu-

žíván až v krajních případech. Procesy vybírá dle důležitosti pro systém samotný, takže, mezi prvními ukončovanými jsou uživatelské aplikace [43, str. 37].

Je nutno zde zmínit i implementaci RPC/IPC mechanismu tzv.*Binderu*, který umožňuje mezi procesovou komunikaci a volání procedur uvnitř systému Android [43, str. 39]. Jiným implementovaným mechanismem IPC je ovladač sdílené paměti (*ashmem*), což je vhodnější obdoba linuxové *Posix* pracující s virtuální pamětí. Její obdobou je ovladač (*pmem*), který ovšem pracuje s pamětí fyzickou.

Pro účely systému Android bylo nutné přepracovat i systémový časovač (*Alarm*), kdy je nutné při vzniku překročení časovače zavolat zadefinované metody, i když není zařízení aktivní [43, str. 41].

Pro vývojáře je nutno poznamenat, že došlo k určitým změnám i při zápisu systémových chyb a procesů (*logging*). Pro zajištění menší hardwarové náročnosti byl vytvořen nový zapisovač, který ukládá všechny vzniklé logy do bufferů jádra ve formě binárního kódu. Logy mohou být 4 druhů (*main* - hlavní, *event* - informace o systémových událostech, *radio* - souvisí se radiovým spojením a *system* - zejména nízkoúrovňové), kdy každý druh se ukládá do svého bufferu. Dále se rozlišuje dle použití aplikační vrstvou logy systémové a od uživatele [45].

Dále je pro úplnost nutné říci, že jádro systému Android se liší i vyšší mírou bezpečnosti. Často je zmiňovaná přítomnost tzv. *Paranoid Networking*, kdy systém vyžaduje zvýšenou míru udělování pověření a ověřování při navazování a provozování komunikace v různých typech sítích [45].

HAL na jádro navazuje tzv. *Hardware Abstraction Layer*, což je mezivrstva, která umožňuje vrstvám nad ní být úplně nezávislými na typu zařízení. Obsahuje mnoho knihoven v jazyce C a C++, které mají za cíl komunikovat s periferními zařízeními nebo jinými službami typu kamera, bluetooth, zabudované senzory atd.. Systém pak využívá tyto knihovny prostřednictvím různých typů volání [43, 46, str. 48].

Bázové knihovny systém Android pracuje v základu s více než 100 knihovnami, které zajišťují jeho základní funkčnost. Jsou to například knihovny spojené s implementací grafického enginu OpenGL (grafika), knihovny umožňující práci s kódy C a C++, knihovny spojené s virtuálními stroji atd. [43, str. 54-56].

Android Runtime *ART* /Dalvik jsou virtuální stroje umožňující běh aplikací založených na jazyku Java. Ve verzi Android 5.0 *ART* plně nahradil Dalvik. Jedná se v podstatě o obdobu virtuálního stroje Java Virtual Machine *JVM*. Tento stroj provádí kompilaci souborů *.java* aplikací na tzv. *bytekód .dex* (nezávislý na typu zařízení), který je posléze přeložen jeho interpretem [43, str. 60-61].

Systémové služby umožňují meziprocesovou komunikaci uvnitř systému. Jsou zajišťované dříve zmíněným *Binderem*. Při běhu systému je spuštěn tzv. *Service Server*, který registruje a spravuje všechny definované služby systému [43, str. 63].

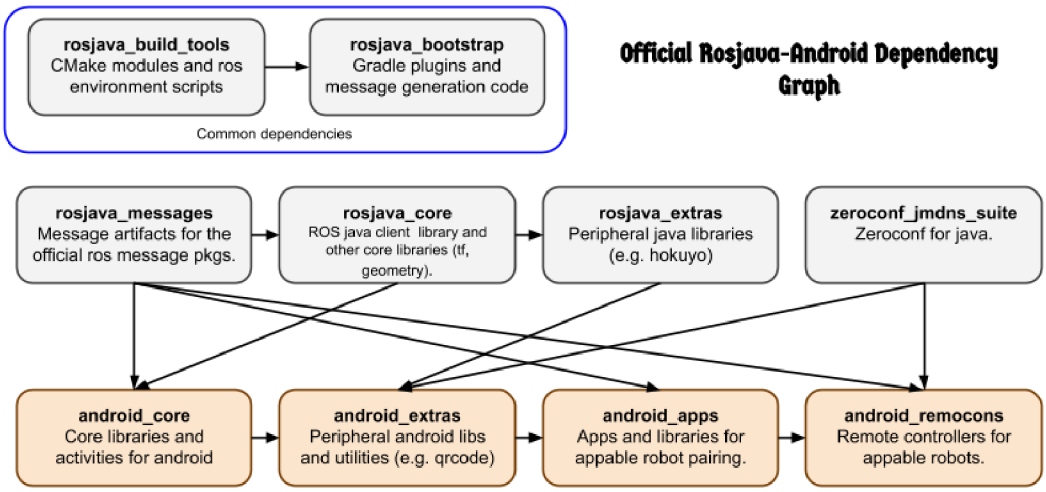
Aplikační framework je část systému, se kterým je v přímém kontaktu uživatel i vývojář.

6.2 Jazyk JAVA

Programovací jazyk Java poznal světlo světa v roce 1995, kdy ho vyvinula firma Sun Microsystems pod vedením Jamese Goslinga. Je to objektově-orientovaný multiplatformní jazyk. Od uvedení na trh se dočkal širokého rozšíření a získal si velikou oblibu. Pod pojmem Java se často rozumí celá platforma obsahující kromě API (v různých edicích, dle potřeby vývojářů) i virtuální stroj *Java Virtual Machine*.

6.2.1 rosjava

Je volně dostupnou knihovnou implementující základní funkcionality spojené se systémem ROS. Aktuálně je uzpůsobena pro distribuci Kinetic. Skládá se z několika balíčků, které hrají odlišnou roli.



Obrázek 6.2: Knihovna *rosjava* [49, převzato]

6.3 Gradle

Při vývoji aplikací je možno se setkat se systémem Gradle. Je obdobou javovských sestavovačů Maven a Ant. Jedná o volně dostupný automatizovaný sestavovač kódu, který narozdíl od výše zmiňovaných používá místo zápisu sestavovacích scriptů v XML jazyk Groovy. Jeho použití zaručuje neměnnost a opakovatelnost sestavovacího procesu zdrojového kódu na kterémkoliv zařízení. Umožňuje jednoduché používání sdílených knihoven, tím usnadňuje vývoj komplexních projektů [47].

Jeho aktivita se skládá z mnoha činností. Uživatel v něm jasně definuje jednotlivé děje (kompilace kódu, mazání souborů, kopírování atd.), jejich posloupnost a návaznost. Řídí použití externích knihoven. Na základě knihovnou definovaných požadavků je schopen navázat i uživatelem nespecifikované knihovny.

6.4 Síťová komunikace

Pod termínem síťová komunikace je v dnešní době skryto ohromné množství pojmů. Vždy je nutné míti na paměti, že prvotním cílem tohoto procesu je spojení dvou nebo více zařízení, které pak mezi sebou mohou komunikovat a posílat data. Aby tento systém byl pružný a hlavně jednotný, byla přijata celá řada tzv. *protokolů* či jiných úmluv, které definují jejich standarty.

Samotnou síť je možno rozdělit dle různých modelů do několika vrstev.

ISO-OSI je obecně přijatá norma, která dělí síť do sedmi vrstev.

Prvotní vrstvou je tzv. *aplikační*, jejímž prostřednictvím je umožněno aplikacím připojit se k síti. Na ni navazuje vrstva tzv. *prezentační*, která zajišťuje zpracování posílaných dat. *Relační* vrstva zajišťuje správu navázaných spojení. *Transportní* vrstva zajišťuje dopravu dat z jednoho zařízení do druhého. *Síťová* vrstva se stará o směrování a adresování v síti na ni navazují vrstva *linková* - přenos a oprava mezi jednotlivými síťovými prvky a *fyzická*, která, jak už název naznačuje, se stará o skutečný fyzický přenos informace [48, str.59-62].

Je nutno poznamenat, že některé komunikační protokoly úplně nesplňují toto rozdělení z důvodu sloučení některých vrstev. Vždy však dodržují pravidlo, že jednotlivé vrstvy na sebe vzájemně navazují.

TCP-IP je nejrozšířenější model (protokol). Tento protokol zahrnuje celou řadu samostatných protokolů mající různou funkci.

Jeho model slučuje některé vrstvy z výše uvedeného modelu *ISO/OSI*. Obsahuje vrstvu *aplikační*, *transportní*, *síťovou* a tzv. *vrtstvu síťového rozhraní*.

Jednotlivé služby v *aplikační* vrstvě (např. SSH, HTTP atd.) mohou využívat 2 protokoly vrstvy transportní - TCP (*Transmission Control Protocol*] a UDP (*User Datagram Protocol*). Ty se postarají o zakódování dat do jednotlivých paketů/datagramů. Ty nesou informaci i službě, která si o tento přenos zažádala (číslo portu (socketu) atd.) [48, str. 61].

TCP je protokol, který zaručuje dopravu každého paketu ve správném pořadí do cílového zařízení. To je prováděno zanesením různých údajů (parita, kontrolní bity atd.), které jsou kontrolovány při každém přenosu v síti. Při nalezení chyby je vyslán požadavek na opětovné zaslání.

Oproti tomu UDP nezaručuje doručení žádného datagramu. Jedná se tedy o nespolehlivý, ale rychlý, systém posílání dat.

K paketům a datagramům z *transportní* vrstvy jsou přidány v *síťové* vrstvě další informace, které se týkají adresy vysílacího a cílového zařízení. To je často provedeno známým protokolem IP. Ten přiřazuje každému zařízení v síti tzv. *IP adresu*, podle níž je možno každé zařízení identifikovat. Ta se skládá obvykle ze 4 čísel, které (dle zvolené masky) určují adresu sítě a adresu zařízení v dané síti.

Vrstva síťového rozhraní se pak stará o fyzický přenos dat.

Je zřejmé, že aplikace bude pracovat s protokoly TCP/IP. Již dříve bylo zmíněno několik služeb z aplikační/transportní vrstvy využívaných v systému ROS, jsou jimi XML-RPC, TCPROS a UDPROS. Pro úplnost je nutno popsat protokol SSH, který bude využit v terminálové části aplikace.

6.4.1 SSH

Komunikační protokol SSH-2 (*Security shell*) je hojně využívaný nástroj pro šifrovanou komunikaci typu klient - server. Vytváří kanál, skrze který je možno připojit se k zařízení v síti, na kterém běží *SSH Server*. Pod pojmem SSH se často rozumí už hotové komerční nebo volně dostupné programy, který tento protokol implementují.

Jeho historie sahá do roku 1995, kdy jeho první verzi SSH-1 vyvinul Fin Tatu Ylönen. V roce 1996 byla vydána vylepšená verze SSH-2 (ovšem s SSH-1 nekompatibilní) [51, str. 10].

Klíčovými vlastnostmi protokolu jsou autentizace - kontrola identity zařízení, šifrování - přenášená data jsou chráněna proti odposlechu a integrita - zaručuje, že data nejsou při přenosu změněna. Umožňuje též tzv. *tunelling*, což je zakomponování jiného protokolu do SSH, kdy je zaručená bezpečnost jeho šifrováním [51, str. 42-43].

Knihovna JSch - pro potřeby aplikace byla vybrána druhá verze protokolu SSH implementovaná pro jazyk Java americkou softwarovou firmou JScraft ve formě volně dostupné (licence *BSD*) knihovny *JSch*. Ta umožňuje síťové připojení k *SSH Serveru* (*sshd*).

Kap. 7

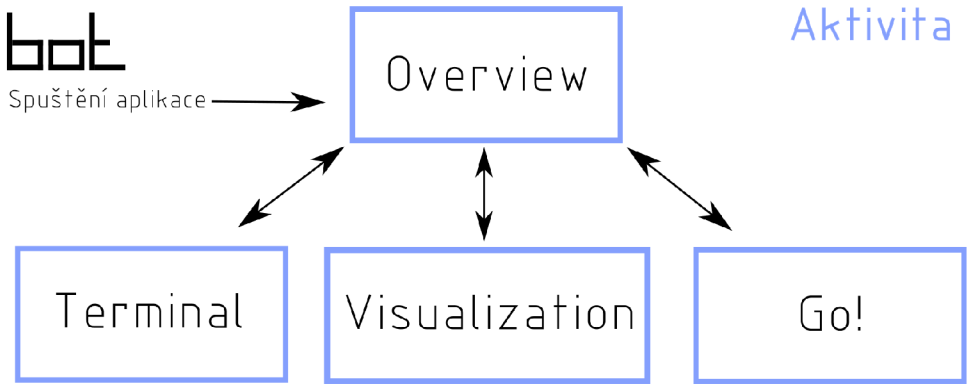
Aplikace bot

Poslední kapitola je věnována výsledku práce - nové mobilní aplikaci. Nese pojmenování *bot* (zkrácenina slova ro*bot*) a je syntézou všech dříve popsaných technologií. Nechť tato kapitola slouží nejen jako její popis, ale též i její přívětivá uživatelská příručka.

Program bot je mobilní aplikací určenou pro systém Android (min. verze 4.4), umožňující výpis a 2D vizualizaci vybraných dat v systému ROS a též jednoduchou obsluhu tohoto systému pomocí terminálu. V neposlední řadě je možné pomocí ní i jednoduché řízení robotu. Je uzpůsobena pro různé velikosti mobilních zařízení.

7.1 Struktura aplikace

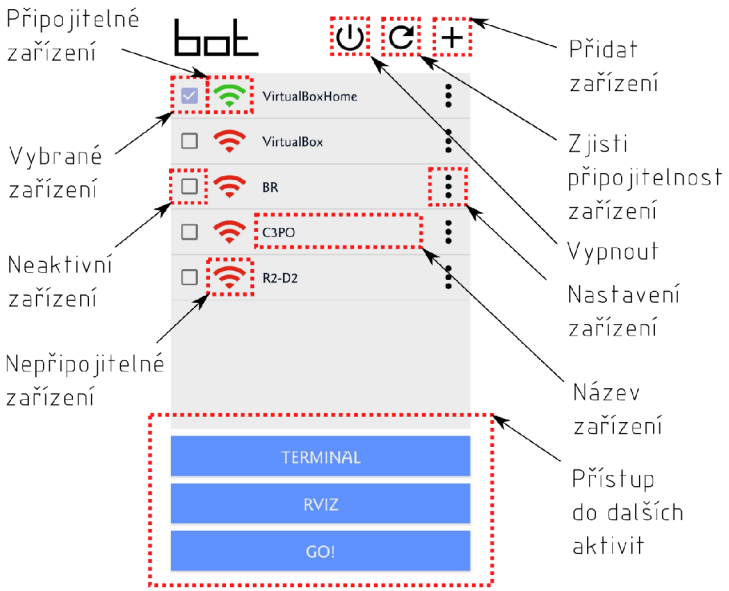
Aplikace je členěna do čtyř aktivit, které mají jasně danou úlohu. Struktura aplikace z pohledu uživatele je názorně zobrazena v diagramu.



Obrázek 7.1: Struktura aplikace z pohledu jejího uživatele

7.1.1 Přehledová aktivita

Aktivita je určena k zobrazení a správě uživatelem nadefinovaných zařízení. Pokud je uživatelem vybráno jedno připojitelné zařízení, slouží též jako přístupový uzel do dalších aktivit. Zobrazuje aktuální možnost jejich připojení. Pro opětovné zjištění stavu jednotlivých zařízení je zde obnovovací tlačítko. Uživatel má možnost v této aktivitě aplikaci ukončit.



Obrázek 7.2: Přehledová aktivita

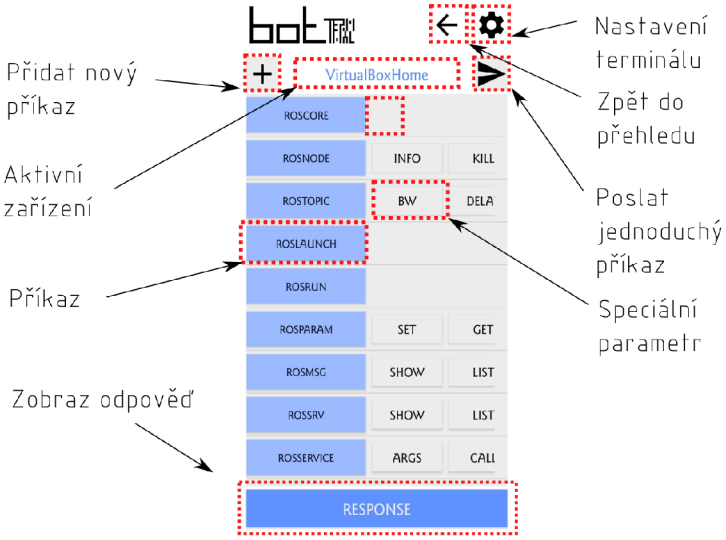
Správa zařízení - uživatel při definování nového zařízení musí nastavit v dialogovém okně několik parametrů. Jsou jimi jeho název, uživatelské jméno, heslo a adresa IP. Lze je poté jednoduše přenastavit.

Parametr	**Popis**	**Formát**
Name	Název zařízení	*C3PO*
Username	Uživatelské jméno	*MyProfile*
Password	Heslo k zařízení	*MyPassword*
IP Address	IP adresa	*192.0.0.0*

Tabulka 7.1: Základní parametry jednotlivých zařízení

7.1.2 Terminálová aktivita

Tato aktivita je určena k jednoduchému ovládání systému ROS. Je zde nutno zdůraznit, že se nejedná o klasický terminál v pravém slova smyslu. Je to spíše prostředek k zasílání jednotlivých příkazů prostřednictvím SSH, který je schopen zaznamenat i odpověď.



Obrázek 7.3: Terminálová aktivita

Uživatel má k dispozici rychlý přístup k přednastaveným příkazům, u kterých lze změnit před odesláním jejich parametry.

Pro potřebu opakovaně odesílaných uživatelem definovaných příkazů je zde implementována možnost vytvořit si pro dané zařízení specifický příkaz a spravovat ho. Bylo myšleno i na odesílání jednorázových příkazů. Uživatel má k dispozici i tuto rychlou volbu.

Přednastavení terminálu - před samotným odesíláním je nutno nastavit některé proměnné, které jsou potřeba ke správné funkci jednotlivých příkazů, popř. ke správné konfiguraci systému ROS. Jedná se zejména o systémové proměnné ROSu *ROS_IP*, *ROS_HOSTAME* a *ROS_MASTER_URI* a adresu souboru *setup.bash Rosbashe* a adresu souboru *setup.sh* balíčku *catkin_ws*. Uživatel je definuje prostřednictvím dialogového okna pod tlačítkem nastavení.

Parametr	**Popis**	**Formát**
Rosbash	Adr. *setup.bash Rosbashe*	*/opt/ros/kinetic/setup.bash*
Ctk_ws setup	Adr. *setup.sh* balíčku	*.../catkin_ws/devel/setup.sh*
ROS_IP	ROS_IP	*192.0.0.0*
ROS_HOSTNAME	ROS_HOSTNAME	*192.0.0.0*
PORT	PORT	*11311*
ROS_MASTER_URI	ROS_MASTER_URI	*http://192.0.0.0:11311*

Tabulka 7.2: Parametry Terminálové aktivity každého zařízení

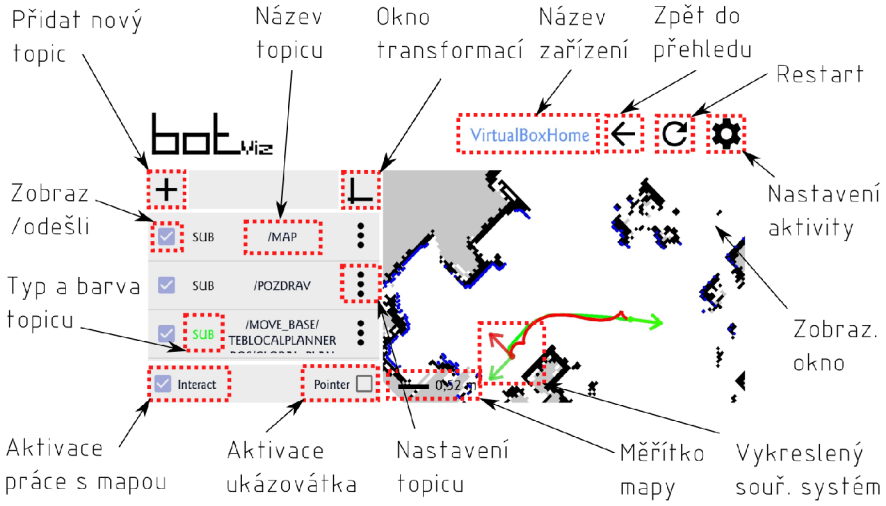
Odesílaný příkaz - má vždy podobu několika příkazů spojených do série.

Příkaz
source *Rosbash*;
source *Ctk_ws setup*;
export *ROS_IP*;
export *ROS_HOSTNAME*;
export *ROS_MASTER_URI*;
Příkaz;
Parametr;

Tabulka 7.3: Podoba každého odeslaného příkazu

7.1.3 Vizualizace

Vizualizace je hlavní aktivitou této aplikace. Umožňuje zobrazení vybraných dat systému ROS pomocí 2D grafiky nebo jednoduchého a přehledného textového výpisu. Kromě toho umožňuje i odesílání některých typů zpráv.



Obrázek 7.4: Vizualizace

Přednastavení aktivity - pro správnou funkci aktivity je nutné nastavit pouze IP adresu zařízení, na kterém běží aplikace (*ROS_IP*).

Parametr	**Popis**	**Formát**
ROS_IP	ROS_IP	*192.0.0.0*

Tabulka 7.4: Nastavení Vizualizace

Přidání a správa *topiců* - přidávání nových *topiců* je umožněno samostatným tlačítkem. Při vytváření je nutno definovat jeho jméno (lze poté změnit), balíček zprávy, typ zprávy a to, jestli je daná zprávy tzv.*subscriber* nebo *publisher* (tyto parametry změnit nelze). Nadefinovaný *topic* lze pak přenastavit pomocí tlačítka nastavení. Pokud je vykreslitelný, lze upravit jeho barvy ARGB. Číslo u každé barvy představuje pravděpodobnost (tj. některé typy zpráv zahrnují i tuto informaci). Pro typy zpráv, které tuto informaci nezahrnují je možno použít jen *Colour90* (pozn. toto nastavení se ukáže až po prvním vykreslení této zprávy aplikací). Je možno též nastavit i vrstvu, ve které se daný *topic* vykreslí.

Balíček	**Zpráva**	**Typ**	**Vykreslitelné**
std_msgs	/String	SUB	Ne
	/Bool	SUB	Ne
	/Int16	SUB	Ne
	/Int32	SUB	Ne
	/Int64	SUB	Ne
geometry_msgs	/Pose	SUB	Ne
	/PoseStamped	SUB/PUB	Ano
	/PoseWithCovariance	SUB	Ne
	/PoseWithCovarianceStamped	SUB	Ano
	/Point	SUB	Ne
	/PointStamped	SUB/PUB	Ano
	/Quaternion	SUB	Ne
	/QuaternionStamped	SUB	Ne
	/Twist	SUB	Ne
	/TwistStamped	SUB	Ne
	/TwistWithCovariance	SUB	Ne
	/TwistWithCovarianceStamped	SUB	Ne
	/Vector3	SUB	Ne
	/Vector3Stamped	SUB	Ano
nav_msgs	/OccupancyGrid	SUB	Ano
	/Path	SUB	Ano
	/Odometry	SUB	Ne
tf2_msgs	/TFMessage	SUB	Ne
sensor_msgs	/LaserScan	SUB	Ano
	/BatteryState	SUB	Ne
	/FluidPressure	SUB	Ne
	/Illuminance	SUB	Ne
	/Imu	SUB	Ne
	/NavSatFix	SUB	Ne
	/NavSatStatus	SUB	Ne
	/Range	SUB	Ne
	/RelativeHumidity	SUB	Ne
	/Temperature	SUB	Ne

Tabulka 7.5: Implementované zprávy

Zobrazení dat z *subscribera* - pokud je daný *topic* aktivní a definovaný v aplikaci, po kliknutí na jeho jméno se z něj zobrazí výpis. Pokud je navíc i vykreslitelný ukáže se v zobrazovacím okně.

Posílání dat z *publishera* - pokud je daný *topic* definovaný v aplikaci jako tzv.*publisher*, lze odeslat s použitím tzv.*pointeru* (ukazovátka). Ukazovátko je nutné nejdříve aktivovat. Po aktivaci jej lze dvojitým ťuknutím umístit do zvoleného místa (a natočit). Pro odeslání daného *topicu* je posléze nutné dvojité kliknutí na tlačítko *Odešli*.

Správa transformací - transformace jsou nedělitelnou součástí systému ROS. Pro správnou funkci aplikace je nutné definovat *subscribera* transformačních typů zpráv *TFMessages/tf2_msgs*. Může jich být i více. Všechny se slučují do společného prostoru.

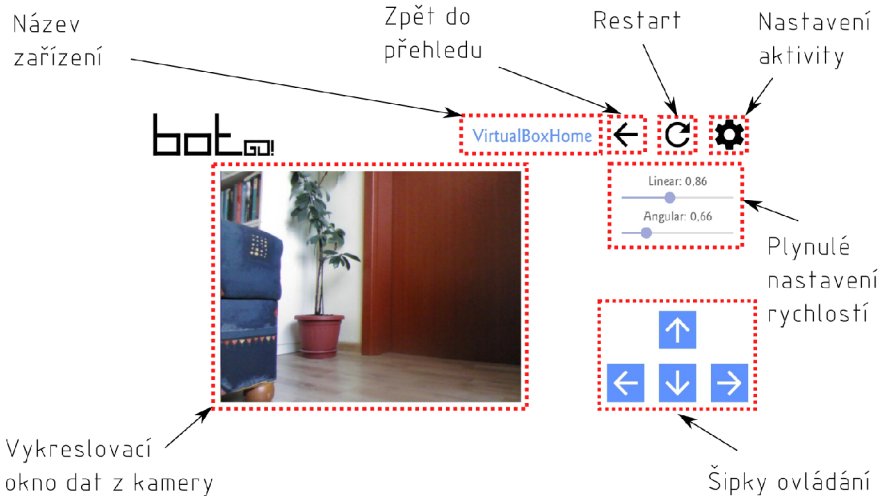
Je nutno zde upozornit na fakt, že aplikace bot je 2D vizualizace. V systému ROS jsou obecné souřadnice v osách x, y a z a otočení je definovaný pomocí kvaternionu $q(x, y, z, w)$. Aplikace úplně ignoruje souřadnici z a apriori předpokládá rotaci jen okolo osy z, kterou převádí na úhel ϕ!

Uživatel má možnost využít jednoduchý přehled o transformacích v systému pod tlačítkem v aplikaci. V tomto přehledu je možné zvolit i ty souřadnicové systémy, které mají být vykresleny.

Zobrazení uživatelského zařízení - je nutné se zde krátce zmínit o možnostech zobrazení samotného zařízení. V systému ROS je fyzický popis zařízení obvykle obsažen v souboru *.urdf*. Aplikace bot však neumožňuje jeho zisk a zobrazení. Uživatel má možnost suplovat fyzickou podobu zařízení s využitím zobrazení jeho souřadnicového systému, popř. zobrazením určitého vhodného vykreslovaného typu zprávy (ideálně např. *PoseStamped*).

7.1.4 Aktivita Jeď!

Jak již název napovídá, tato aktivita slouží k jednoduchému ovládání zařízení s využitím typu zprávy *geometry_msgs/Twist*. Umožňuje též uživateli zobrazovat data z kamery pomocí zprávy *sensor_msgs/Image*.



Obrázek 7.5: Aktivita Jeď!

Přednastavení aktivity - pro správnou funkci aktivity je nutné nastavit několik proměnných. Jedná se zejména o systémovou proměnnou *ROS_IP*, jména tzv. *topiců* a maximální hodnoty odesílaných rychlostí.

Parametr	**Popis**	**Formát**
ROS_IP	ROS_IP	*192.0.0.0*
Name Twist	Název *topicu* se zprávou typu *geometry_msgs/Twist*	*/teleop*
Name Image	Název *topicu* se zprávou typu *sensor_msgs/Image*	*/camera*
Max Linear	Maximální rychlost jízdy	*3.25*
Max Angular	Maximální rychlost otáčení	*2.58*

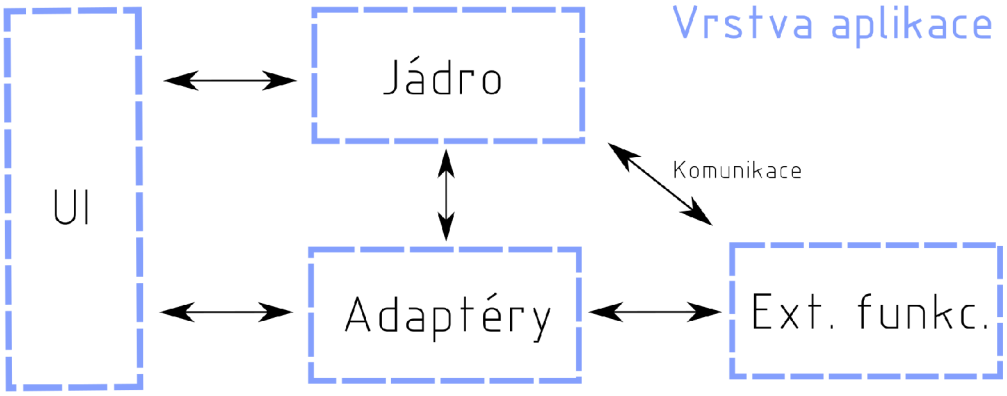
Tabulka 7.6: Nastavení aktivity Jeď!

Formát zprávy *sensor_msgs/Image* - aplikace je uzpůsobena pro odebírání zprávy s formátování odpovídajícímu typu *rgb8* (CV_8UC3) knihovny *cv_bridge*.

7.2 Struktura kódu

Struktura kódu vychází z idei oddělení jednotlivých částí systému. Výsledek se do určité míry blíží ke struktuře ideální aplikace formulované v páté kapitole.

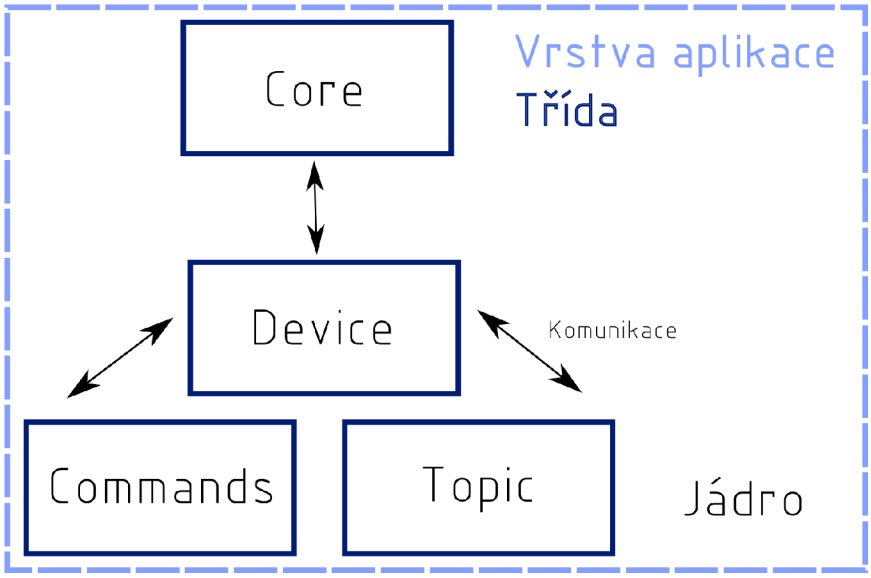
Celková jeho hierarchie je zobrazena v diagramu. Je členěn do tří hlavních vrstev, které navzájem komunikují. Jsou jimi části uživatelského rozhraní, jádro aplikace a část implementující konkrétní aplikační funkcionality. Mezi těmito velkými bloky často stojí tzv. *adaptéry*, speciální třídy, které zajišťují konverzi dat mezi jednotlivými částmi.



Obrázek 7.6: Struktura kódu

7.2.1 Jádro aplikace

Jádro je řídící centrum aplikace. Je složeno z několika tříd, které tvoří stromovitou strukturu. Struktura jádra aplikace je přehledně zobrazena v diagramu.



Obrázek 7.7: Struktura jádra aplikace

Obecným rysem této části je též správa dat aplikace. Jedná se zejména o ukládání, načítání, změnu a jejich poskytování (pozn. samotné ukládání je však provedeno externí funkcionalitou). Jádro samotné je tedy jakousi datovou strukturou, která je schopna mimo vlastnění dat i informovat jednotlivé části aplikace (ty, které to potřebují) o jejich změně a ty mohou díky této svázanosti adekvátně reagovat. Prakticky ve všech případech je tento mechanismus implementován asynchronně prostřednictvím tzv.rozhraní (*interface*) často statických.

Každý uzel stromu má na starost pouze sebe sama. Pokud je to nezbytné, volá i uzly, které na něj bezprostředně navazují (jedná se zejména o ukládání a načítání dat). Taková struktura aplikace tedy umožňuje jednoduchý přístup ke všem datům.

Třída *Core* je hlavní řídící třídou. Tvoří vždy kořen stromu a je inicializována při každém spuštění aplikace. Tím se liší od zbylých tříd v této části.

Jejím úkolem je správa všech uživatelem definovaných zařízení, tvorba logiky pro jednotlivé aktivity, navazování jednotlivých statických rozhraní po potřeby uživatelského rozhraní a externích funkcionalit.

Třída při každé inicializace načítá data z aplikační databáze.

Třída *Device* je spojena s každým, uživatelem definovaným, zařízením. Uchovává o něm data a uchovává navazující instance třídy *Topic* a *Command*.

Třída *Topic* definuje každý uživatelský tzv.*topic*, které dané zařízení při běhu ROSu odebírá nebo publikuje.

Třída *Commands* je třída reprezentující každý příkaz (uživatelský nebo předem přednastavený statický) pro *Rosbash* (ve formě instance vnořené třídy *Command*).

7.2.2 Uživatelské rozhraní

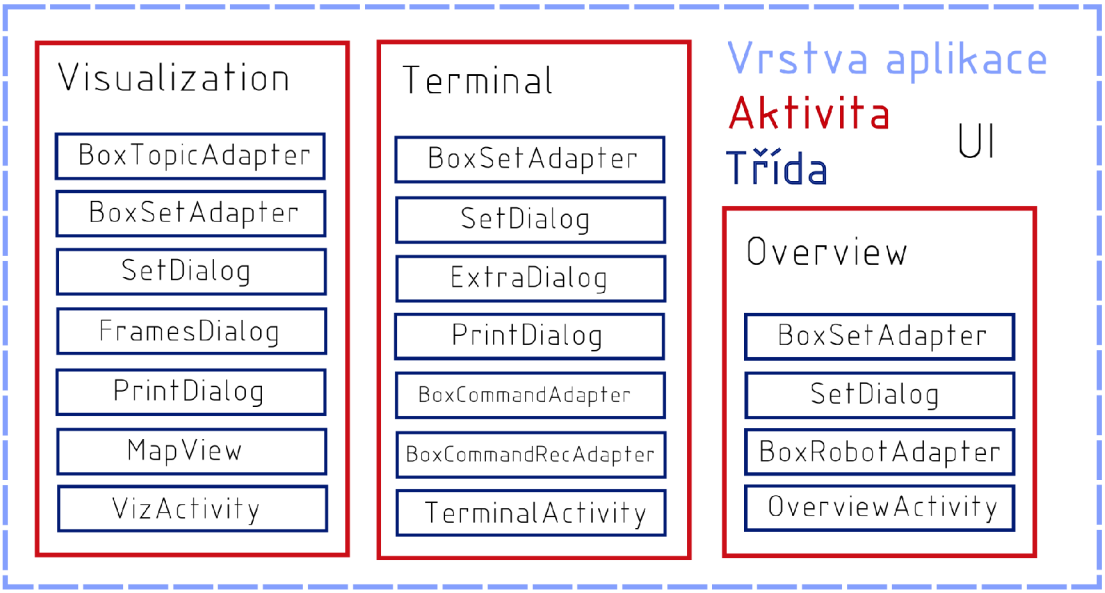
Uživatelské rozhraní je přítomno v kódu v několika podobách. Jedná se zejména o třídy hlavních aktivit a třídy, které definují funkcionalitu konkretních bloků layoutu aplikace. Kromě těchto tříd je podoba každého bloku layoutu aplikace vždy definovaná pomocí souborů ve formátu XML.

V aplikaci je použita vlastní grafika s výjimkou základních ikon, které jsou převzaty z Android Studia pod licencí *Apache Licence v.2*.

Každé uživatelské rozhraní, pokud to funkce vyžaduje, je spojeno pouze s jádrem aplikace a to pomocí asynchronních volání implementovaných rozhraních nebo přímým přístupem k daným proměnným. Reakci uživatelského rozhraní vždy definuje jádro aplikace (ve většině případů třída *Core*).

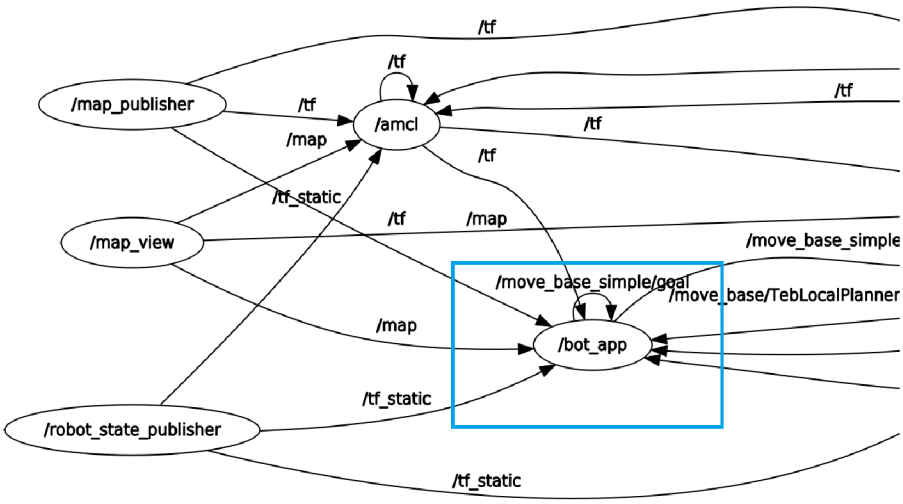
7.2.3 Externí funkcionality

Pod tímto abstraktním pojmem si lze představit každou implementaci interakce aplikace s okolím ať již fyzickým (komunikace s daným zařízením pomocí SSH, ROSu) nebo softwarovým na mobilním zařízení (zejména práce s daty).



Obrázek 7.8: Třídy uživatelského rozhraní

Třída *ROShandler* - zajišťuje komunikaci se systémem ROS. Třída při inicializaci vytvoří svůj vlastní tzv. *node* s názvem *bot_app*, který posléze naváže komunikaci s ostatními v systému. Provádí též základní zpracování dat z odebíraných zpráv jednotlivých definovaných *topiců* v zařízení.



Obrázek 7.9: *Node* aplikace v běžícím systému

Třída *SSHhandler* - zajišťuje komunikaci se zařízením využitím protokolu SSH. Aplikaci je též využívána ke zjištění připojitelnosti daného zařízení.

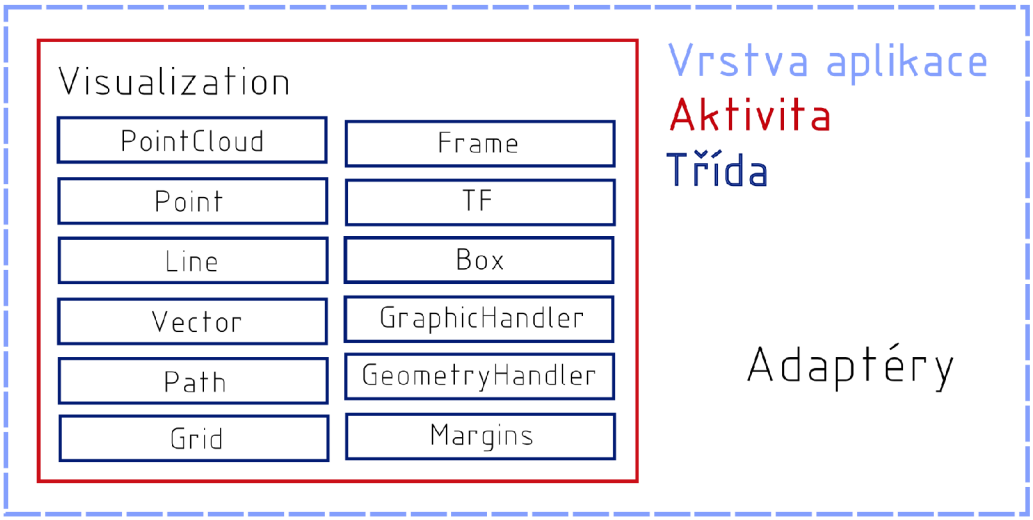
Komunikace probíhá formou vytvoření jednorázového spojení tzv. *session* (reprezentuje spojení s SSH serverem), ve které se vytvoří tzv. *channel*. Jeho prostřednictvím je možné zaslat daný příkaz a zaznamenat odpověď. To je implementováno

prováděním nekonečné smyčky reagující na přicházející proud znaků (*InputStream*) vytvořeného kanálu.

Třída *KeyValueDataHandler* - je třída provádějící ukládání a načítání dat aplikace. Využívá tzv. *Shared Preferences* systému Android, což je v podstatě jednoduchá slovníková databáze, tj. databáze ve kterém lze uložit pár údajů „*klíč + hodnota*“.

7.2.4 Adaptéry

Tato mezivrstva zahrnuje několik tříd zabývajících se zejména transformacemi uvnitř systému, vykreslováním dat a konverzí dat pro uživatelské rozhraní.

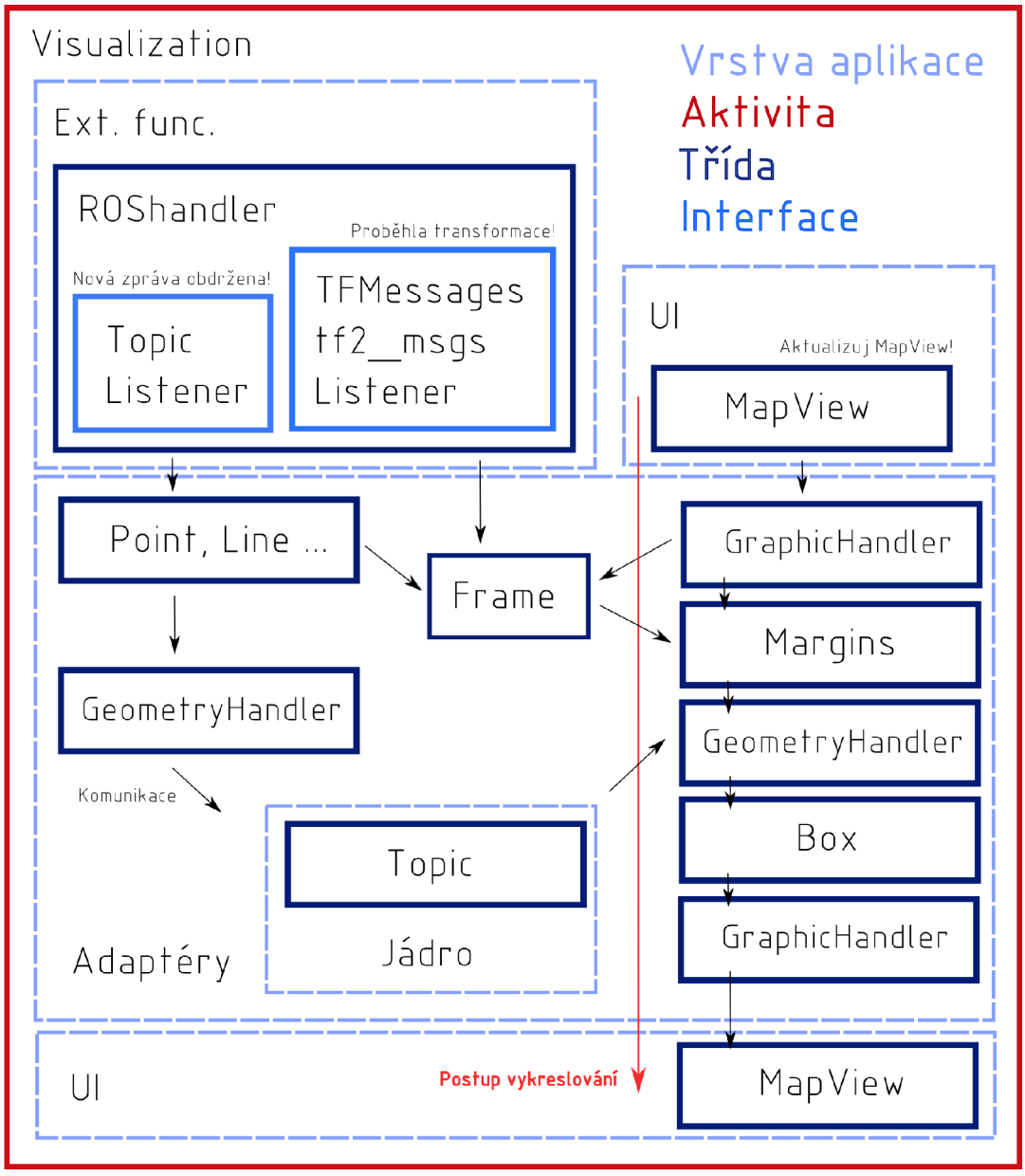


Obrázek 7.10: Adaptéry v aplikaci

7.3 Prostor, souřadnice a transformace

Vzhledem k tomu, že aplikace musí vykreslit některé typy zpráv, je nutné, aby obsahovala systém, který umí jednotlivé objekty (data ze zpráv umístěné do prostoru) sloučit do jednoho kartézského dvojrozměrném prostoru s osami x a y (příp. otočením ϕ okolo osy z).

Celý proces vykreslování je zobrazen na následujícím diagramu. Je v něm jasně patrné, že se jedná o proces paralelní, jednotlivé procesy jsou popsány dále.



Obrázek 7.11: Vykreslování dat

Odebírání transformačních zpráv - jak již bylo zmíněno dříve, pro správnou funkci systému je nutné definovat tzv. *subcribera* transformačních zpráv *tf2_msgs/TF-*

Message. Tyto zprávy obsahují informaci o transformaci (posunutí, otočení) jednoho souřadnicového systému vůči druhému v trojrozměrném prostoru.

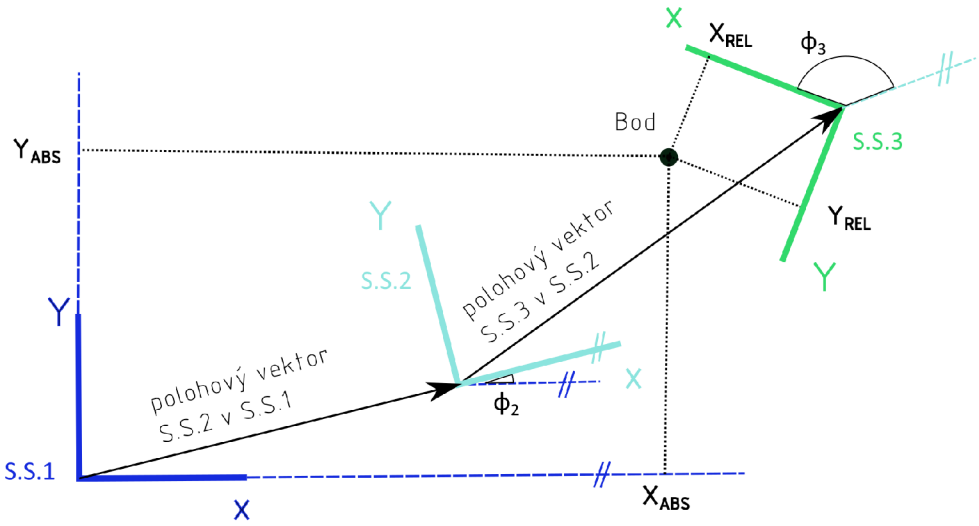
Každému souřadnicovému systému je vytvořena samostatná instance třídy *Frame*, která obsahuje i jméno jeho přímého předchůdce.

Přepočet transformací z odebírané zprávy - probíhá přímo ve třídě *ROShandler*, systém z odebírané zprávy separuje souřadnice x a y. Rotaci v podobě kvaternionu q (x, y, z, w) převádí na rotaci ϕ s užitím vztahu:

$$\phi = 2 \cdot sign(z) \arccos(w) \tag{7.1}$$

Tyto tři souřadnice se posléze předají třídě *TF*, která je přiřazena každému souřadnicovému systému reprezentovanému třídou *Frame*.

Správa souřadnicových systémů - probíhá automaticky. Všechny souřadnicové systémy, které jsou aplikací zaznamenány, jsou uspořádány do stromovité struktury ve statické části třídy *Frame*.



Obrázek 7.12: Řetězení souřadnicových systémů a souřadnice

Geometrické struktury - každý vykreslitelný druh zprávy je převáděn na základní geometrický útvar, kterému je přiřazen jeho souřadnicový systém. V systému je definováno několik těchto struktur. Jsou jimi bod, úsečka, vektor, lomená čára, mrak bodů a mapa.

Pokud má každý tento útvar definovaný souřadnicový systém, lze určit jeho relativní a absolutní souřadnice v prostoru. Relativní určuje pozici ve vlastním souřadnicovém systému. Absolutní určuje pozici k souřadnicovému systému ve vrcholku stromu (obvykle reprezentovanému např. mapou).

Algoritmus přepočtu relativních souřadnic na absolutní - probíhá formou sestavení transformační cesty procházením stromu od konkretního souřadnicového systému (listu) k počátku (ke kořeni stromu) a následnou postupnou transformací.

0.	**Vstup**: *Bod* $[x_{rel}, y_{rel}]$ *v souřadnicovém systému A s polohou [x, y,* ϕ*] vůči B*
1.	Prázdný seznam *Cesta*
2.	A = Aktuální souřadnicový systém
3.	přidej A do *Cesty*
4.	B = Předchozí souřadnicový systém
5.	dokud (B != prázdný){
-	přidej B do *Cesty*
-	A = B
-	B = A.zjisti předchozí souřadnicový systém
-	}
6.	S = Poslední přidaný souřadnicový systém (kořen stromu)
7.	pro každý další souřadnicový systém X z *Cesty* {
-	$S_x = S_x + X_x \cos(S_\phi) - X_y \sin(S_\phi)$
-	$S_y = S_y + X_x \sin(S_\phi) + X_y \cos(S_\phi)$
-	$S_\phi = S_\phi + X_\phi$
-	}
8.	$x_{abs} = S_x + x_{rel} \cos(S_\phi) - y_{rel} \sin(S_\phi)$
9.	$y_{abs} = S_y + x_{rel} \sin(S_\phi) + y_{rel} \cos(S_\phi)$
10.	**Výstup**: *Absolutní souřadnice bodu* $[x_{abs}, y_{abs}]$

Tabulka 7.7: Algoritmus přepočtu relativních souřadnic na absolutní

Vykreslování dat - celý tento proces je určen především pro vykreslování dat v ve třídě *MapView*. *MapView* přidává svůj vlastní souřadný systém *botViz*, který se pohybuje v prostoru společně se zobrazovacím oknem aplikace. Vždy je připojen ke kořeni transformačního stromu.

Pokyn pro překreslení *MapView* může vyjít ze tří zdrojů, prvním z nich je obdržení transformační zprávy, druhým z nich je obdržení vykreslovaného tzv. *topicu* nebo je překreslení vyvoláno přímo z uživatelského rozhraní (např. posunem mapy).

7.4 Testování aplikace

Pro úplnost je nutné se zde zmínit o způsobu testování aplikace. Byla prováděna zejména ručně na simulaci a to z důvodu velkého zastoupení funkcionalit reagujících na vstup z uživatelského rozhraní a obtížným simulováním příchozích dat ze systému ROS.

Jiné progresivní způsoby testování aplikace (jednotkové testy atd.) se bohužel při vývoji nestihly uplatnit a bude nutné je nahradit při dalším vývoji zvýšeným úsilím při testování.

Závěr

Cílem této práce byl vývoj mobilní aplikace, která bude pracovat s frameworkem ROS.

Po provedení průzkumu dostupných mobilních aplikací, ať již vyvinutých na této platformě nebo jiné, komerční a krátkém teoretickém okénku, které se týkalo popisu struktury systému ROS a komunikace v něm, byly formulovány požadavky ohledně podoby a funkce vyvíjené aplikace. Měla by umožňovat jednoduchou správu systému ROS pomocí terminálu a vizualizovat data v podobě, která se blíží programu RViz.

Výsledná práce má podobu čtyřaktivitové mobilní aplikace podporované systémem Android. Skládá se ze tří funkcionalit. První z nich je implementace terminálu v samostatné aktivitě s použitím knihovny pracující s protokolem *SSH*. Uživatel má možnost spravovat běžící systém ROS pomocí příkazů shellu *Rosbash* nebo svých vlastních.

Druhou z nich je implementace funkcí blížící se programu RViz. To je provedeno v aktivitě Vizualizace, která pracuje s uživatelem definovanými zprávami, které dokáže vypsat, vykreslit v dvojrozměrném prostoru nebo i posílat. Aplikace má implementované větší množství zpráv než samotný RViz.

Poslední funkcí aplikace je možnost jednoduchého ovládání zařízení a současné zobrazování dat z kamery. Je nutné poznamenat, že při odebírání zpráv typu *sensor_msgs/Image* dochází ke znatelnému zpoždění při vykreslování. To je nejspíše způsobeno kombinací rychlosti zasílání dat v ROSu a implementací funkcionality v aplikaci. Bohužel se tento problém nepodařilo odstranit.

I přes některá úskalí při vývoji lze závěrem konstatovat, že všechny cíle práce byly splněny. Aplikace je připravena být užitečným nástrojem uživatelů systému ROS.

Část 9

Seznam použitých zdrojů

[1] MARTIN, Robert C. *The Clean Architecture* [online]. 2012-08-13 [cit. 2020-04-13]. Dostupné z: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[2] *Model—view—viewmodel* [online]. 2020-05-10 [cit. 2020-04-12]. Dostupné z: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

[3] *Model—view—presenter* [online]. 2020-1-24 [cit. 2020-04-12]. Dostupné z: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter

[4] *Model-view-controller* [online]. 2020-2-20 [cit. 2020-04-12]. Dostupné z: https://cs.wikipedia.org/wiki/Model-view-controller

[5] GLASSMAN, Elena, ed. *UI Software Architecture* [online]. 2017 [cit. 2020-04-12]. Dostupné z: http://web.mit.edu/6.813/www/sp17/classes/05-ui-sw-arch/

[6] FOWLER, Martin. *GUI Architectures* [online]. 2006-7-18 [cit. 2020-04-12]. Dostupné z: https://martinfowler.com/eaaDev/uiArchs.html#Model-view-presentermvp

[7] BSD Licence: SPDX short identifier: BSD-2-Clause. *Opensource.org* [online]. [cit. 2020-04-04]. Dostupné z: https://opensource.org/licenses/bsd-license.php

[8] Robot Operating System. *Wikipedia.org* [online]. [cit. 2020-04-04]. Dostupné z: https://en.wikipedia.org/wiki/Robot_Operating_System

[9] OLADEPO, Habib, *Nodes* [online]. 2018-12-04 [cit. 2020-02-15]. Dostupné z: http://wiki.ros.org/Nodes

[10] FOOTE, Tully, *Topics* [online]. 2019-02-20 [cit. 2020-02-03]. Dostupné z: http://wiki.ros.org/Topics

[11] HENDRIX, Austin, *ROS/Messages* [online]. 2019-01-13 [cit. 2020-02-06]. Dostupné z: http://wiki.ros.org/msg

[12] MARTINEZ ROMERO, Aaron, ROS/Concepts. *ROS* [online]. 2014-06-21 [cit. 2020-02-01]. Dostupné z: http://wiki.ros.org/ROS/Concepts

[13] MAYORAL VILCHES, Víctor, *ROS/Technical Overview* [online]. 2014-06-15 [cit. 2020-02-01]. Dostupné z: http://wiki.ros.org/ROS/Technical%20Overview

[14] MAGNUSSON, Gudjon, *Slave API* [online]. 2014-03-19 [cit. 2020-02-13]. Dostupné z: http://wiki.ros.org/ROS/Slave_API

[15] MILLER, Brad, *Parameter Server* [online]. 2018-11-08 [cit. 2020-02-03]. Dostupné z: http://wiki.ros.org/Parameter%20Server

[16] MARSHALL, Dave. *Remote Procedure Calls (RPC)* [online]. 1999-5-1 [cit. 2020-02-02]. Dostupné z: http://users.cs.cf.ac.uk/Dave.Marshall/C/node33.html

[17] WINER, Dave, *XML-RPC: Specifications* [online]. 1999-6-15 [cit. 2020-04-03]. Dostupné z: http://xmlrpc.com/spec.md

[18] YANQING, Wu, 2018. *Master* [online]. [cit. 2020-02-03]. Dostupné z: http://wiki.ros.org/Master

[19] *UDPROS* [online], 2013-04-20 [cit. 2020-03-12]. Dostupné z: http://wiki.ros.org/ROS/UDPROS

[20] *TCPROS* [online]. 2013-04-15 [cit. 2020-02-8]. Dostupné z: http://wiki.ros.org/ROS/TCPROS

[21] CONLEY, Ken, *Parameter Server API* [online]. 2009-11-02 [cit. 2020-04-03]. Dostupné z: http://wiki.ros.org/ROS/Parameter%20Server%20API

[22] CONLEY, Ken, *Master Slave API* [online]. 2012-02-09 [cit. 2020-04-03]. Dostupné z: http://wiki.ros.org/ROS/Master_Slave_APIs

[23] VARAS, Nicolas. *Rosbash* [online]. 2019-12-11 [cit. 2020-02-15]. Dostupné z: http://wiki.ros.org/rosbash

[24] SAITO, Isaac. *ROS Command-line tools* [online]. 2015-08-13 [cit. 2020-02-22]. Dostupné z: http://wiki.ros.org/ROS/CommandLineTools

[25] OBERHAUSER, Tim. *RViz: User Guide* [online]. 2015-08-14 [cit. 2020-02-15]. Dostupné z: http://wiki.ros.org/rviz/UserGuide

[26] KOUBAA, Anis, ed., 2016. *Robot Operating System (ROS): The Complete Reference (Volume 1)*. London: Springer International Publishing, 728 s. Studies in Computational Intelligence. ISBN 978-3-319-26052-5.

[27] GUTENKUNST, Alexander, *Master API* [online]. 2014-11-18 [cit. 2020-02-03]. Dostupné z: http://wiki.ros.org/ROS/Master_API

[28] ZIMMERMAN, Adam, *Rviz for Android: Overview* [online]. 2012-08-28 [cit. 2020-04-03]. Dostupné z: https://bitbucket.org/zimmrmn3/rviz-for-android/wiki/Home

[29] WILLOW GARAGE, *RViz for Android: Promotion video*, Dostupné také z: https://www.youtube.com/watch?v=u8LaIIb0COQ

[30] BRUNSON, Michael, et all., *Robot CA: Overview* [online]. [cit. 2020-04-03]. Dostupné z: https://mtbii.github.io/RobotCA/

[31] *RobotCA: Git* [online], 2017-08-06 [cit. 2020-04-03]. Dostupné z: https://github.com/SCCapstone/RobotCA

[32] FOOTE, Tully, *Android Sensors Driver: Package Summary* [online]. 2018-03-27 [cit. 2020-04-03]. Dostupné z: http://wiki.ros.org/android_sensors_driver

[33] ROCKEY, Chad, *ROS Driver for Android Sensors* [online]. 2013-6-4 [cit. 2020-04-03]. Dostupné z: https://github.com/ros-android/android_sensors_driver

[34] *Android Sensors Driver: Play Store* [online], [cit. 2020-04-03]. Dostupné z: https://play.google.com/store/apps/details?id=org.ros.android.sensors_driver

[35] KOHLER, Damon, *Android libraries for rosjava* [online]. 2019-7-17 [cit. 2020-04-03]. Dostupné z: https://github.com/rosjava/android_core

[36] *Omron MobilePlanner: Play Store* [online], [cit. 2020-04-03]. Dostupné z: https://play.google.com/store/apps/de

[37] *Monitor and control on the go: MobilePlanner Tablet Edition* [online], Omron, 2018, 2 [cit. 2020-04-03]. Dostupné z: https://assets.omron.com/m/77184d1f18760b85/original/MobilePlanner-Tablet-Edition-tails?id=com.omron.MobilePlannerTablet

[38] *Omron Mobile Planner* [online], [cit. 2020-04-03]. Dostupné z: https://automation.omron.com/en/us/products/family/Mobile%20Planner

[39] *IRobot Home: Play Store* [online], [cit. 2020-04-03]. Dostupné z: https://play.google.com/store/apps/details?id=com.irobot.home&hl=en_US

[40] BOVET, Daniel Pierre a Marco CESATI. *Understanding the Linux kernel*. 3rd ed. Sebastopol: O'Reilly, c2006. ISBN 05-960-0565-2.

[41] *Android: operating system* [online]. 2020-04-05 [cit. 2020-04-11]. Dostupné z: https://en.wikipedia.org/wiki/Android_(operating_system)

[42] *Android Architecture* [online]. 2020-02-24 [cit. 2020-04-11]. Dostupné z: https://source.android.com/devices/architecture

[43] YAGHMOUR, Karim. *Embedded Android*. Sebastopol: O'Reilly, 2013. ISBN 978-1-449-30829-2.

[44] *Android Kernel Features* [online]. 2015-6-24 [cit. 2020-04-09]. Dostupné z: https://elinux.org/Android_Kernel_Features

[45] *Android Logging System* [online]. 2015-6-24 [cit. 2020-04-09]. Dostupné z: https://elinux.org/Android_Logging_System

[46] *Android HAL* [online]. 2020-01-06 [cit. 2020-04-08]. Dostupné z: https://source.android.com/devices/architecture/hal

[47] MUSCHKO, Benjamin. *Gradle in action*. Shelter Island: Manning, c2014. ISBN 978-1-617291-30-2.

[48] KUROSE, James F. a Keith W. ROSS. *Počítačové sítě*. Brno: Computer Press, 2014. ISBN 978-80-251-3825-0.

[49] CHIESA, Lucas. *Rosjava: Official Packages* [online]. 2014-01-13 [cit. 2020-04-10]. Dostupné z: http://wiki.ros.org/rosjava/Official%20Dependency%20Graph

[50] *Android Platform: Guide* [online]. 2019-12-27 [cit. 2020-04-08]. Dostupné z: https://developer.android.com/guide/platform

[51] BARRETT, Daniel J. a Richard E. SILVERMAN. *SSH: kompletní průvodce*. Brno: Computer Press, 2003. Security (CP Books). ISBN 80-722-6852-X.

Část 10

Seznam použitých zkratek a symbolů

ROS	*Robot Operating System*
PC	*Personal computer*
API	*Application Programming Interface*
GUI	*Graphic user interface*
UI	*User interface*
BSD	*Berkeley Software Distribution*
GNU	*General Public License*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
IP	*Internet Protocol*
XML	*Extensible Markup Language*
S.S.	*Souřadnicový systém*

Část 11

Seznam příloh

K práci je přiložen soubor, který obsahuje zdrojový kód programu.

\
    bot *složka se zdrojovým kódem*

bot	*idea.*		*složka IntelliJ IDEA*
		\...	
	gradle.		*složka systému Gradle*
		\...	
	gradle		*složka systému Gradle*
		\...	
	app		*složka aplikace*
		libs	*složka s importovanými knihovnami*
		res	*složka s prvky def. v XML*
		src	*složka s třídami aplikace*
	...		*automaticky generované soubory aplikace*
	README		*soubor README*