VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO PROGRAMOVÁNÍ ROBOTICKÉHO RAMENE

MOBILE APPLICATION FOR ROBOTIC ARM PROGRAMMING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VOJTĚCH JURKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL KAPINUS**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Jurka Vojtěch**
Program: Informační technologie
Název: **Mobilní aplikace pro programování robotického ramene**
Mobile Application for Robotic Arm Programming
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s platformou Raspberry Pi a technologiemi používanými pro tvorbu moderních mobilních aplikací.
2. Navrhněte modul určený pro řízení zvoleného robotického ramene, správu vytvořených robotických aplikací a způsob komunikace mezi Raspberry Pi a mobilním telefonem.
3. Navrhněte sadu jednoduchých robotických instrukcí a uživatelské rozhraní určené pro tvorbu robotických programů pomocí těchto instrukcí.
4. Implementujte řídicí část systému na platformě Raspberry Pi.
5. Implementujte navržené uživatelské rozhraní formou mobilní aplikace.
6. Funkčnost celého systému ověřte, diskutujte dosažené výsledky a zvažte případná rozšíření či vylepšení.
7. Vytvořte video prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2, 3 a rozpracované body 4 a 5 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kapinus Michal, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 29. dubna 2021

Abstrakt

Práce popisuje návrh a tvorbu systému, který pomocí mobilní aplikace umožní ovládat a programovat robotické rameno. Systém se skládá především z mobilní aplikace na platformě Android a serveru implementovaném v jazyku Python, který zajišťuje komunikaci jak s robotickým ramenem, tak se samotnou mobilní aplikací. Díky tomuto přístupu je následně možné využít jedno mobilní zařízení pro práci s více roboty a jejich nezávislé ovládání.

Předložené řešení poskytuje uživateli možnost pomocí mobilního zařízení ovládat robotické rameno v reálném čase pomocí intuitivních ovládacích prvků a také vytvářet či upravovat robotické aplikace. Vytváření programů je uskutečněno skrze metody vizuálního programování přímo v aplikaci. Tento způsob ovládání robotického ramene je přínosem k zlepšení přístupnosti takových zařízení méně odborné veřejnosti, jež by je mohla upotřebit.

Abstract

This thesis describes the design and creation of a system which controls and programs a robotic arm through a mobile application. The system consists of an Android mobile application and a server. The server is implemented like a Python script to process the app communication and then send instructions to the robotic arm. This approach makes it easy to use one mobile device to control several robots independently.

This solution enables the user to control and program the robotic arm in real time using intuitive control elements and to create and edit applications for the robot. All through his mobile device. The programs for the robot are created with the methods of visual programming directly in the mobile application. This way of robotic arm control contributes to the accessibility of these devices to the less professional public, which could utilize these devices.

Klíčová slova

vizuální programování, Blockly, JavaScript, robot Dobot Magician, uživatelské rozhraní, mobilní aplikace, Android, Java, Kotlin, Python

Keywords

visual programming, Blockly, JavaScript, robot Dobot Magician, user interface, mobile application, Android, Java, Kotlin, Python

Citace

JURKA, Vojtěch. *Mobilní aplikace pro programování robotického ramene.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Kapinus

Mobilní aplikace pro programování robotického ramene

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Kapinuse. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vojtěch Jurka
4. května 2021

Poděkování

Chtěl bych poděkovat svému vedoucímu práce, Ing. Michalu Kapinusovi, za to, že mi byl vždy ochoten pomoci, poradit a konzultovat řešení. Dále bych rád poděkoval svojí rodině, která mě po celou dobu studia podporovala. Nakonec bych rád poděkoval svojí přítelkyni, která za mnou vždy stála a držela mě při světlejších myšlenkách.

Obsah

Kapitola 1

Úvod

Robotika je jedním z technologických odvětví, které čím dál více a čím dál rychleji ovlivňuje naše životy. Dříve se jednalo zejména o odvětví průmyslové výroby, v moderní době se však její využití protlačuje nejen do ostatních profesionálních pracovních prostředí, ale i do světa malorozměrných, nízkonákladových pracovních projektů a amatérských hobby aktivit. Toto má za příčinu hlavně technologický pokrok, který způsobuje, že robotické nástroje jsou čím dál levnější a dostupnější pro širokou veřejnost. Tato skutečnost má mimo jiné také neblahý následek názoru veřejnosti, podle kterého je tento vývoj zkázou pro lidi, které nahradí v jejich manuální práci. Většina odborníků na robotiku se však domnívá, že při správném přístupu se tyto technologie dají naopak využít k tomu, aby i obyčejnému manuálnímu pracovníkovi ulehčovaly práci tím, že za něj budou odvádět banální, opakované, či fyzicky náročné práce. Touto cestou mu umožní odvádět jeho práci mnohem kvalitněji a efektivněji.

Tato práce se nese v duchu víry v tuto budoucnost. Věřím v to, že přibližování možností využívání robotiky laickým uživatelům je logickým krokem ve vývoji a budoucností této technologie. Hlavně z tohoto důvodu jsem si tuto práci vybral. Jejím cílem je vytvořit softwarové komponenty, které jako celek tvoří ovládací rozhraní mezi koncovým uživatelem a ovládaným robotickým ramenem, přičemž není od uživatele vyžadovaná pokročilá znalost technologií, které se v praxi používají k operaci robotických ramen.

První část této práce popisuje relevantní teoretické informace o robotických ramenech, jejich kategorizaci, současný stav těchto technologií a představuje základní metody jejich programového řízení. Dále se věnuje popisu operačního systému pro mobilní zařízení Android, jeho historii a verzím, architektuře a aplikacím, které jsou pro tento systém vyvíjeny. Další částí kapitoly je sekce o vizuálním programování a poté následuje krátký popis paradigmatu programování koncového uživatele.

Další část se zabývá návrhem řešení této práce. Kapitola rozebírá existující podobná řešení zadaného problému a jejich výhody a nevýhody. Poté je již popsán samotný návrh systému, ovládání a programování robotického ramene a jeho jednotlivé součásti, zejména mobilní aplikace a její komunikace se serverem, ke kterému je robot připojen.

Třetí část této práce popisuje implementaci celého systému a jeho částí. Jsou zde popsány různé stěžejní nebo zajímavé implementační detaily programů a případné odchylky řešení od návrhu. Na závěr je v práci ještě obsažena kapitola o testování výsledného produktu.

Kapitola 2

Teorie

Následující kapitola souhrnně popisuje současný stav technologií, které s touto prací souvisí. V první sekci je popsaná problematika průmyslových robotů a robotických ramen. Ze začátku hlavně z hlediska teorie a rozdělení typů ramen. Dále je popsán pojem kolaborativního robota, jeho využití a nakonec shrnutí současného stavu ve vývoji a využití těchto ramen. Další část této kapitoly pojednává o operačním systému Android a o mobilních aplikacích, které jsou pro tento systém vyvíjeny. První část pojednává hlavně o systému jako samotném. O jeho historii, vývoji a architektuře. Dále jsou zkoumány samotné aplikace, respektive jejich složení ze základních komponent a jak díky tomuto složení funguje jejich implementace. Poslední část této kapitoly se zaměřuje na vizuální programování. Rozebírá jeho záměry, výhody a nevýhody. Dále popisuje několik hlavních druhů vizuálních programovacích jazyků a jejich využití. Nakonec je popsáno využití vizuálního programování u paradigmatu zvaného „end-user programming“, česky programování koncovým uživatelem.

2.1 Robotická ramena

Průmysloví roboti jsou multifunkční mechanická zařízení navržená k přenášení materiálu, součástek, nářadí, nebo specializovaných zařízení [9]. Průmyslový robotický systém obsahuje nejen průmyslové roboty, ale i jakákoli zařízení i senzory potřebné k tomu, aby robot mohl provádět svoje úkoly, stejně jako řídící či sledovací komunikační rozhraní. Roboti jsou všeobecně používáni pro provádění nebezpečné, repetitivní a člověku nepříjemné práce.

Robotické rameno je typem mechanického ramene, které mimikuje funkci lidské ruky. Většinou je programovatelné. Rameno může představovat celý mechanismus nebo může být součástí komplexnějšího robotického zařízení. Články takového manipulátoru jsou propojeny klouby, které jim umožňují rotační, či posuvný pohyb. Články tohoto manipulátoru se považují za takzvaný *kinematický řetězec* [29]. Ukončující prvek kinematického řetězce manipulátoru se nazývá *koncový efektor* [26] a je analogický k lidské ruce. V praxi je ale výraz „robotická ruka“ často přirovnáván k celému rameni, což je ovšem z hlediska odborné terminologie často označováno za špatné přirovnání.

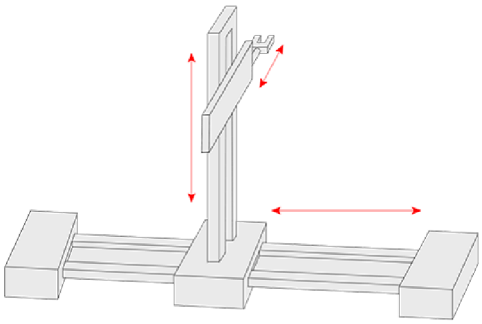
2.1.1 Kategorizace robotických ramen

I přes to, že se robotická ramena používají v různých průmyslových oborech a můžeme je dělit podle různých kritérií, jako například typ operace, typ průmyslu nebo velikost, dají se v základě kategorizovat do 6 základních typů [15][14]. Tato kategorizace se řídí typy kloubů a strukturou mechanismu ramene (kinematického řetězce).

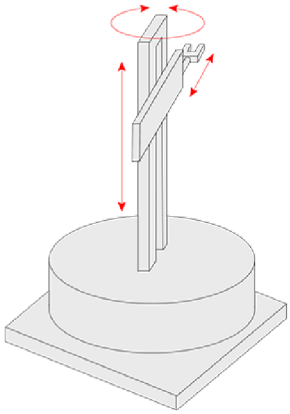
Základní dělení robotických ramen tedy vypadá následovně:

- **Kartézský robot** je robot, jehož rameno má tři posuvné klouby, jejichž osy jsou shodné s kartézským koordinátorem (jsou na sebe tedy kolmé). Používá se například pro přemisťování objektů, aplikaci těsnících látek, montáže, manipulaci s nástroji, obloukové svařování a 3D tisk. Mechanismus je znázorněn na obrázku 2.1a.
- **Válcový robot** je robot, jehož osy pohybu koncového efektoru tvoří souřadnicový systém ve tvaru válce. Byl hojně využívaný v počátcích éry robotického průmyslu. Používá se například pro montáže, manipulaci s nástroji, bodové svařování a manipulaci u odlévacích zařízení. Mechanismus je znázorněn na obrázku 2.1b.
- **Sférický/polární robot** je robot, jehož osy pohybu koncového efektoru tvoří souřadnicový systém ve tvaru koule. V raných dobách využívání robotiky v průmyslu patřil tento typ robotického mechanismu mezi nejvyužívanější. Používá se například pro manipulaci s nástroji, bodové sváření, odlévání a plynové nebo obloukové sváření. Mechanismus je znázorněn na obrázku 2.1c.
- **SCARA robot** je robot, který má dva souběžné rotační klouby, které zajišťují dodržení přesné pozice v rovině. Používá se například pro manipulaci s objekty, aplikaci těsnících látek, montáže, a manipulaci s nástroji. V porovnání s šestiosými kloubovými roboty umožňuje velmi rychlou a přesnou manipulaci se součástkami. Mechanismus je znázorněn na obrázku 2.1d.
- **Kloubový robot** je robot, jehož rameno má alespoň tři rotační klouby. Používá se například pro montáže, odlévání, tryskání, plynové svařování a sprejování barvy. Mechanismus je znázorněn na obrázku 2.1e.
- **Paralelní robot** je robot, jehož rameno má souběžné rotační nebo posuvné klouby. Používá se v situacích, kdy je prioritou rychlost pohybu. Mechanismus je znázorněn na obrázku 2.1f.

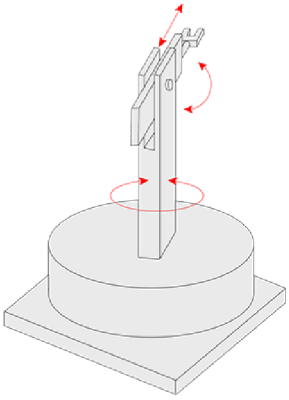
V dnešní době jsou nejběžněji využíváni Klouboví roboti. Neznamená to však, že tento typ mechanismu je vždy nejvhodnější. I přes to, že například válcové či polární robotické mechanismy nejsou v dnešní době už tolik populární, najdou se stále situace, kdy je nejvhodnější použít právě je.



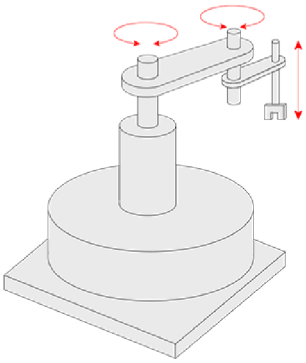
(a) Kartézský robot.



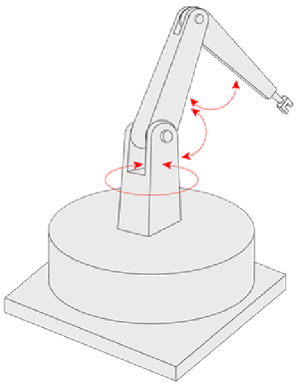
(b) Válcový robot.



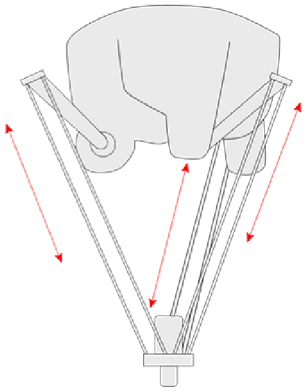
(c) Sférický/polární robot.



(d) SCARA robot.



(e) Kloubový robot.



(f) Paralelní robot.

Obrázek 2.1: Jednotlivé typy robotů.[1]

2.1.2 Kobot aneb Kolaborativní robot

Až do nedávné doby znamenalo používání robotických ramen při práci jednu hlavní věc. Na pracovišti byl pracovní prostor robota jednoznačně vyznačen a při jeho práci do tohoto prostoru nesmí žádný člověk zasahovat. Této izolace se dosahuje fyzickým oddělením prostoru (například klecí), či senzory, které v případě narušení prostoru člověkem okamžitě zastaví práci robota. Z čistě teoretického hlediska by použití takových senzorů činilo daného robota kolaborativním [13], avšak v moderním slova smyslu, kdy termín kolaborativní robot a kobot splývá v synonymum, jsou tito roboti též označováni jako nekolaborativní. Kolaborativní roboti jsou určeni pro přímou interakci mezi člověkem a robotem ve sdíleném pracovním prostoru, nebo v bezprostřední blízkosti [12]. Termín „cobot“ byl poprvé uveden již v samotném patentu [17], který patří pánům se jmény James Edward Colgate a Michael Peshkin, kteří kobota vynalezli v roce 1996.

Existuje mnoho způsobů, jakými lze dosáhnout požadované bezpečnosti při práci s koboty [13]. Může to být volba použitých konstrukčních materiálů, zaoblené hrany, omezení rychlosti pohybu a síly motorů nebo používání senzorů a softwaru, který se stará o bezpečné chování robota. Ukázku takového zařízení můžeme vidět na obrázku 2.2.

Obrázek 2.2: Ukázka cobota - kolaborativní robotické ruky.[2]

[1]Obrázky převzaty z https://robotics.kawasaki.com/ja1/xyz/en/1803-01/.

[2]Převzato z https://industryeurope.com/universal-robots-launches-ur16e-cobot-for-collaborative-automation/.

Mezinárodní federace pro robotiku - IFR[3], která je globálním sdružením pro výrobce robotů, stejně jako pro jednotlivá národní sdružení pro robotiku, definuje čtyři typy spolupráce robota s člověkem:

- **Koexistence:** Člověk i robot pracují souběžně vedle sebe, ale pracovní prostor mají oddělený.
- **Sekvenční kolaborace:** Člověk a robot sdílí určitou část, nebo celý pracovní prostor, avšak nepracují se součástkou, či nástrojem současně.
- **Kooperace:** Robot i člověk pracují ve stejný čas na jednom výrobku či s jedním nástrojem a oba jsou v pohybu.
- **Responzivní kolaborace:** Robot reaguje v reálném čase na pohyby člověka.

Ve většině průmyslových použití cobotů v současnosti sdílí člověk a cobot stejný pracovní prostor, ale svoje úkoly provádí nezávisle, nebo sekvenčně (koexistence, nebo sekvenční kolaborace). Kooperace, či responzivní kolaborace jsou současně méně běžné.

2.1.3 Současný stav

V současnosti by se vývoj používání technologií robotiky, respektive robotických ramen, dal shrnout několika trendy. Prvním je skutečnost, že celkově se používání těchto technologií čím dál více rozšiřuje a popularizuje, a to i v odvětvích, kde tomu tak dříve ani nebylo. Je to zejména díky technologickému pokroku, který způsobuje čím dál větší dostupnost robotů jako takových. Toto se samozřejmě týká zejména robotiky v průmyslu. Mimo to se v posledních letech začínají čím dál více objevovat případy užívání robotických technologií v běžném životě mimo průmysl. Z tohoto hlediska tedy IFR dělí roboty na dvě kategorie dle oblasti použití. První jsou průmysloví roboti, kteří se používají k automatizaci v oblasti průmyslu. Druhou kategorií jsou roboti používaní pro služby pro domácí a profesionální použití (mimo průmysl). Používání robotů v průmyslu je pochopitelně na mnohem vyšší úrovni, avšak v ostatních službách se užívání robotiky čím dál tím rychleji rozmáhá. Hlavní vliv na to má opět rozmach dostupnosti cobotů díky technologickému pokroku.

V dnešní době už není použití cobota pro malý domácí business projekt pouhým sci-fi, jelikož ceny takových robotických paží se pohybují už od 6500 amerických dolarů (zhruba 140 tisíc korun) [3]. K dispozici jsou komerčně však i coboti s cenovkou v řádu nižších desítek tisíc, například robotická ruka Dobot Magician (obrázek 2.3). I když se postupem času tito coboti čím dál více vyrovnávají průmyslovým robotům z hlediska přesnosti a rychlosti, přesto tato řešení nejsou vhodná pro průmyslové použití, jelikož nedisponují potřebnou spolehlivostí a výdrží konstrukce. Jsou to spíše univerzální zařízení určená pro výzkumné a edukativní účely. Vysoké ceny robotických ramen vhodných pro průmyslové použití způsobuje hlavně náročnost vývoje, prototypování a celkové kolaterální náklady spojené s vývojem, programováním, testováním a údržbou.

[3] https://ifr.org/

Obrázek 2.3: Robotická ruka Dobot Magician. Jedná se o kloubového čtyřosého robota, který je určen hlavně pro domácí a edukativní účely.[4]

2.1.4 Ovládání robotických ramen

Ve většině moderních robotických ramen se pro dosažení přesného pohybu kloubů používají *krokové motory*. Jsou to elektromotory, které jsou napájeny impulsy stejnosměrného proudu. Každý impuls způsobí, že se rotor otočí o přesný úhel, respektive krok [11]. Velikost tohoto kroku závisí na konstrukci motoru. Díky možnosti ovládat tyto kroky programově lze dosáhnout velice přesných pohybů kloubů. Další výhodou je, že je možné ovládat kloub tak, že se otočí do určité přesně definované pozice bez použití jakýchkoli senzorů pro zpětnou vazbu. Na základě této skutečnosti potom spočívá celá problematika ovládání robotického ramene pomocí elektroniky a softwaru v jednom principu. Je třeba určit každému kloubu ramene, do jakého úhlu se má natočit. V praxi to tedy znamená, že pohyb robota je abstrahován programem, který se skládá ze sekvence příkazů určujících natočení jednotlivých kloubů robotického ramene. V průmyslovém prostředí se tento program často vytváří pomocí nástroje zvaného „teach pendant“ [10], v češtině ruční ovládací panel, který můžeme vidět na obrázku 2.4. Je to přenosné zařízení, které umožňuje ručně nastavovat aktuální polohu vstupního efektoru robotického ramene. Aktuální úhel natočení všech kloubů se po nastavení dá uložit, čímž je možno vytvářet, nebo upravovat program pohybu ramene. Postupu posouvání polohy ramene robota se v praxi říká *jogging* [6].

Při vytváření programů pro ovládání robotických ramen programátorovi práci velice usnadní, pokud je schopný se v prostoru dosahu koncového efektoru orientovat pomocí kartézské soustavy souřadnic.

[4]Převzato z https://www.dobot.us/.

Obrázek 2.4: Ukázka teach pendantu.[5]

Každý bod v prostoru společně s orientací označuje přesnou polohu koncového efektoru. Je tedy potřeba údaje o ohybu jednotlivých kloubů převést na tyto souřadnice. Matematickému procesu, který tento problém řeší, se říká dopředná kinematika. V praxi je ale více užitečný proces opačný, respektive výpočet ohybu jednotlivých kloubů v kinematickém řetězci, přičemž vstupem jsou souřadnice kartézské soustavy o poloze a orientaci koncového efektoru daného řetězce [24].

2.2 Mobilní aplikace na platformě Android

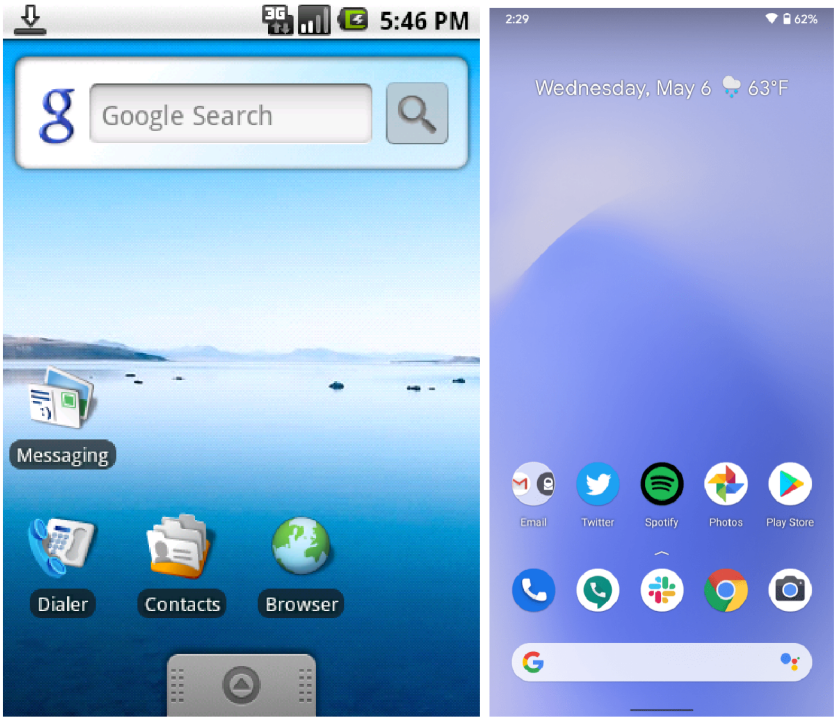
Mobilní aplikace je typ softwaru, který je navrhován a vyvíjen pro mobilní zařízení, jako například telefony, tablety nebo chytré hodinky. Android je operační systém pro mobilní zařízení, který funguje jako otevřená platforma pod licencí Apache[6]. Je založen na modifikované verzi Linux kernelu. Na jejím vývoji pracuje společnost Open Handset Alliance, která je konsorciem 84 společností a má za úkol rozvíjet tuto otevřenou platformu [5].

[5] Převzato z https://www.amazon.com/00313-1-Robot-Teach-Pendant-Backlight/dp/B00RKI4ALK.
[6] https://www.apache.org/licenses/LICENSE-2.0

2.2.1 Historie operačního systému

Jako počátek historie této platformy se dá označit rok 2005, kdy společnost Google odkoupila společnost Android, Inc. V roce 2007 vzniká zmiňovaná společnost Open Handset Alliance a Android je oficiálně uveden jako open source software. O rok později je vydána první verze Android SDK[7] a společnost HTC přichází na trh s prvním mobilním telefonem s tímto systémem.

V dalších letech postupně vycházejí další verze operačního systému a další mobilní zařízení, která ho podporují a jsou s ním prodávána. Popularita Androidu výrazně stoupá a již v roce 2009 je druhou nejprodávanější platformou pro chytré telefony na světě (hned za Blackberry) [19].



(a) Android 1.5. (b) Android 11.

Obrázek 2.5: Porovnání grafického uživatelského rozhraní jedné z nejstarších a nejnovějších verzí systému Android.[8]

Verze systému

Následuje přehled hlavních stabilních verzí systému společně s klíčovými vlastnostmi [1]:

- **verze 1**

  Je první verzí. Spolu s ní byl uveden i první telefon (G1). Systém obsahoval aplikace jako je Android Market, Gmail, YouTube, atd. Dále notifikace, Wi-fi a Bluetooth. Vydána v září 2008.

[7] https://www.techopedia.com/definition/4220/android-sdk

[8] Obrázky převzaty z https://www.theverge.com/2011/12/7/2585779/android-10th-anniversary-google-history-pie-oreo-nougat-cupcake a https://9to5google.com/2020/05/06/android-11-dp4-icon-shapes-launcher/.

- **verze 1.5 Cupcake**

  Přidány rotace okna na základě akcelerometru, sotfwarová klávesnice a podpora pro klávesnice třetí strany. Dále prvky uživatelského rozhraní jako je domovská obrazovka, widgety... Také podpora pro kameru. Vydána v dubnu 2009.

- **verze 1.6 Donut**

  Vylepšení vyhledávání v systému, usnadnění přístupu, převádění textu na řeč. Dále používání gest a aktualizace Linux kernelu. Vydána v září 2009.

- **verze 2 Eclair**

  Nově podporuje Google účty a synchronizaci s nimi. Dále vylepšení uživatelských aplikací, jako je prohlížeč, kamera, email, zasílání zpráv nebo kontakty. Vydána v říjnu 2009.

- **verze 2.2 Froyo**

  Tato aktualizace přináší vylepšení uživatelského rozhraní, zabezpečení zařízení a mobilní hotspot. Vydána v květnu 2010.

- **verze 2.3 Gingerbread**

  Vylepšení uživatelského rozhraní, funkce copy-paste, management stahování souborů. Přidána také podpora NFC. Vydána v prosinci 2010.

- **verze 3 Honeycomb**

  Přidána nová verze uživatelského rozhraní pro tablety. Dále podpora USB příslušenství, upravování domovské obrazovky, podpora vizuálního multitaskingu. Vydána v únoru 2011.

- **verze 4 Ice cream sandwich**

  Tato verze přináší hodně změn. Patří mezi ně nové uživatelské rozhraní, upravitelné widgety, funkce zamykací obrazovky, odemykání pomocí obličeje, nebo Wi-fi připojení klienta na klienta (v angličtině „peer to peer“). Vydána v říjnu 2011.

- **verze 4.1 - 4.3 Jelly bean**

  Tato aktualizace je rozdělena na 3 podverze. První byla vydána v červnu 2012. Přináší podporu pro většinu světových jazyků, vylepšení notifikací, widgetů na zamykací obrazovce, externích obrazovek. Dále HDR mód kamery nebo vylepšenou integraci grafického procesoru a podporu OpenGL[9].

- **verze 4.4 Kitkat**

  V této verzi najdeme zejména optimalizaci pro méně výkoná zařízení. Dále podporu bezkontaktních platebních karet skrz NFC, tisku přes IP konektivitu nebo infračervených vysílačů. Vydána v září 2013.

- **verze 5 Lollipop**

  Zde jsou obsaženy velké změny designu uživatelského rozhraní v podobě material designu[10]. Dále novou platformu Android TV, což je operační systém pro chytré

[9] https://www.khronos.org/opengles/
[10] https://material.io/

televize, nebo vylepšení práce se zvukovými vstupy a výstupy. Vydána v listopadu 2014.

- **verze 6 Marshmallow**

  Přidána podpora snímače otisku prstu, ovládání oprávnění aplikací za běhu, a další menší úpravy a vylepšení, například optimalizace využití baterie. Vydána v říjnu 2015.

- **verze 7 Nougat**

  Tato verze se zaměřuje na ještě větší optimalizaci využití baterie i výkonu zařízení. Dále přináší vylepšení usnadnění přístupu, jako například přiblížení obrazovky. Vydána v srpnu 2016.

- **verze 8 Oreo**

  Přidána podpora módu obrazu v obraze v aplikacích. Dále s sebou přináší Android Go - speciální distribuci systému, která je zaměřena na zařízení s nízkým výkonem. Vydána v srpnu 2017.

- **verze 9 Pie**

  Tato verze přináší podporu vykrojených displejů, několikanásobných kamer a ostatní vylepšení zabezpečení a uživatelského rozhraní. Vydána v srpnu 2018.

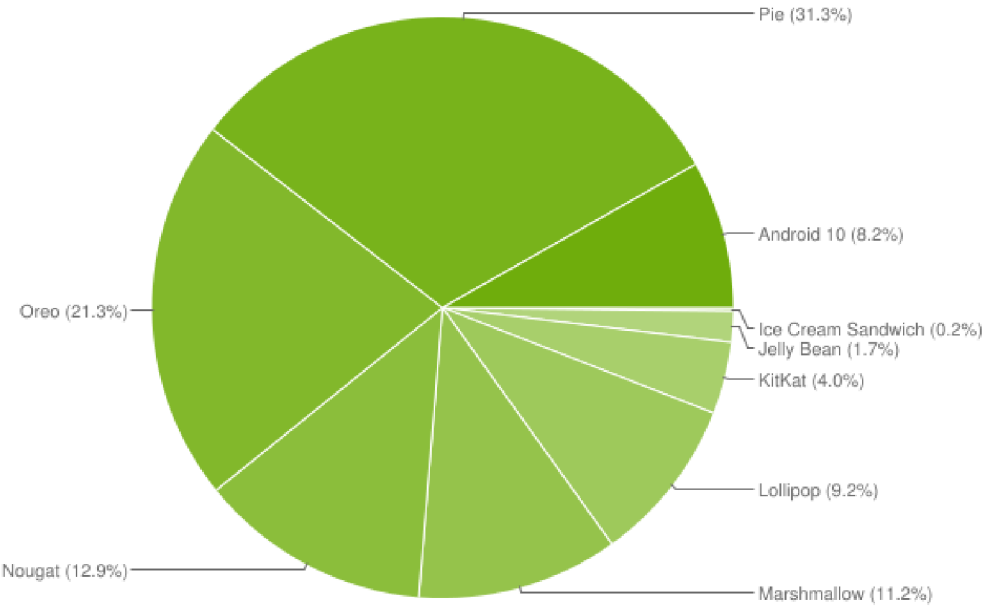
- **verze 10 Q**

  Přidán celosystémový noční režim uživatelského rozhraní. Dále velká vylepšení zabezpečení a soukromí a vylepšení notifikací, jako například adaptivní reakční tlačítka přímo v notifikaci. Vydána v září 2019.

- **verze 11 R**

  Tato zatím poslední aktualizace přináší plnou integraci aplikací s 5G sítěmi a další vylepšení notifikací a zabezpečení. Vydána v září 2020.

Využití jednotlivých verzí systému, které můžeme vidět například na obrázku 2.6, nám ukazuje, že většina zařízení nefunguje na nejnovější verzi. Toto je způsobeno jednak obrovským množstvím zařízení, ale také jejich variabilitou a faktem, že některé typy zařízení nejnovější verzi systému nepotřebují. Dalším důvodem je neochota výrobců aktualizovat starší zařízení.



Obrázek 2.6: Koláčový graf ukazující využití verzí systému Android ke květnu 2020.[11]

2.2.2 Architektura systému

Celá architektura tohoto operačního systému se skládá z několika vrstev, jak můžeme vidět na obrázku 2.7. Níže jsou jednotlivé vrstvy popsány [25][30].

- **Linux kernel (jádro)**

  Je nejspodnější vrstvou operačního systému. Zajišťuje základní funkce systému, jako je řízení procesů, přístupu do paměti a spolupráci hardwarových zařízení, jako je kamera, tlačítka, displej, a tak dále. Také nabízí různé ovladače, které zajišťují jednoduchou spolupráci s periferními zařízeními. Dále je zde implementováno zabezpečení systému, vstupně-výstupní operace, moduly ovladačů různých komunikačních sítí nebo správa napájení.

- **Libraries (knihovny)**

  Tato vrstva je situovaná nad úrovní jádra. Jsou v ní obsaženy různé knihovny, které jsou užitečné pro přímý přístup aplikací k různým komponentám systému. Jsou napsány v jazyce C/C++ a stavěny specificky pro tento systém.

  Mimo ostatní je mezi nimi například SSL pro zabezpečení internetové komunikace, Surface manager pro funkcionalitu dotykového displeje nebo OpenGL pro práci s grafikou.

- **Android Runtime**

  Tato vrstva obsahuje jednu z nejdůležitějších součástí systému, kterou je *Dalvik Virtual Machine*. Je to obdoba virtuálního stroje *Java Virtual Machine*, který je používán

[11] Převzato z https://9to5google.com/2020/04/10/google-kills-android-distribution-numbers-web/.

na klasických počítačích. Rozdíl mezi nimi je ten, že *Dalvik* je navržen a optimalizován specificky pro Android. Každá aplikace je samostatným procesem, který využívá vlastní instanci tohoto virtuálního stroje.

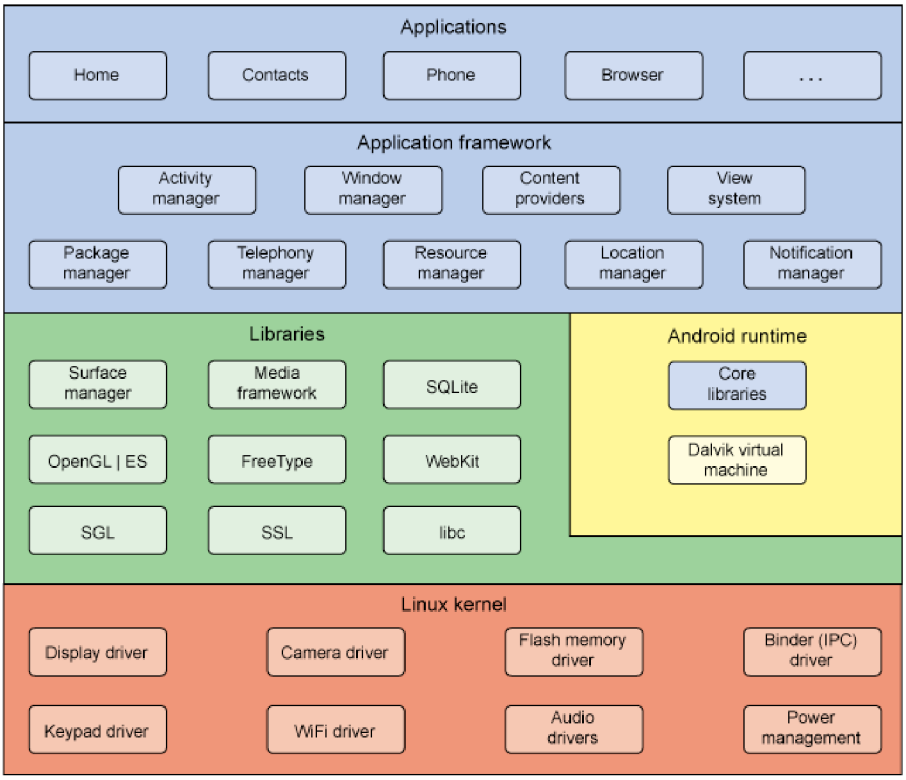
V této vrstvě jsou obsaženy také knihovny jádra, které umožňují psát aplikace pro android ve standardním programovacím jazyku Java.

- **Application Framework (Aplikační rámec)**

  Aplikace v systému Android přímo interagují s touto vrstvou. Aplikační rámec poskytuje aplikacím základní služby systému. Tato vrstva také nabízí mnoho služeb vyšší úrovně pro aplikace ve formě Java tříd. Vývojáři aplikací tyto služby ve svých aplikacích využívají. Mezi nejdůležitější dostupné služby se řadí například **Activity manager**, který se stará o kompletní cyklus života aplikace, **Package manager**, který udržuje seznam aktuálně nainstalovaných aplikací v systému, **Window manager**, který spravuje okna, která tvoří aplikace, či **View system**, který spravuje různé společné prvky grafického uživatelského rozhraní.

- **Applications (Aplikace)**

  Nejvyšší úrovní architektury jsou samotné aplikace. Aplikací může být například kniha kontaktů, prohlížeč, hry, navigace, posílání zpráv, kalendář a tak dále.



Obrázek 2.7: Schéma architektury systému Android.[12]

[12]Převzato z http://www.techplayon.com/android-os-architecture/.

2.2.3 Aplikace a její základní součásti

Základní součástí každé Android aplikace jsou tzv. *komponenty*. Jsou to v podstatě vstupní body, skrze které může uživatel nebo systém přistupovat k aplikaci [25][2].

Existují čtyři základní typy komponent:

- **Activities (Aktivity)**
- **Services (Služby)**
- **Broadcast receivers (Přijímače vysílání)**
- **Content providers (Poskytovatelé obsahu)**

Každý typ komponenty má jedinečný účel a životní cyklus, který definuje, jak je komponenta vytvořena a jak zaniká. Níže jsou popsány zmiňované čtyři základní typy.

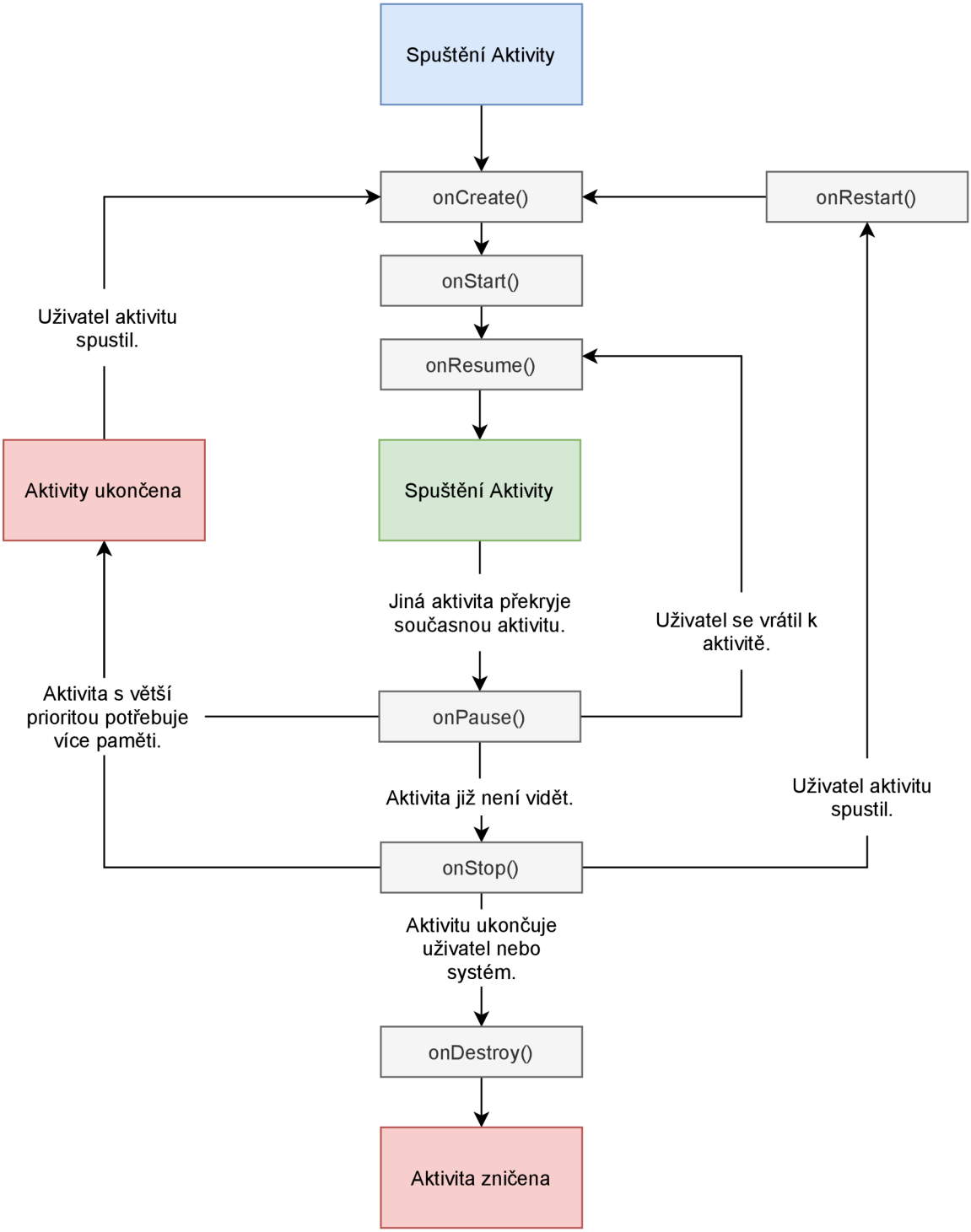
Aktivity

Aktivita je vstupním bodem pro interakci s uživatelem. Je jediným typem komponenty, která má uživatelské rozhraní, skrze které je viditelná pro uživatele. Reprezentuje jednu obrazovku, která obsahuje uživatelské rozhraní[18]. Například aplikace obsluhující e-maily může obsahovat aktivitu, která obsahuje seznam zpráv, dále aktivitu na vytvoření zprávy, další na přečtení zprávy a podobně. I když aktivity tvoří dohromady ucelené uživatelské rozhraní pro uživatele, každá aktivita je nezávislá na ostatních. Díky tomu může úplně jiná samostatná aplikace využít jakoukoli aktivitu z jiné, pokud jí to daná aplikace povolí. Například aplikace kamery může využít aktivitu zmiňované aplikace e-mailu pro vytvoření nové zprávy, aby umožnila uživateli odeslat obrázek přes e-mail.

Aktivity umožňují následující klíčové interakce mezi systémem a aplikací:

- Sledují všechny prvky rozhraní, které uživatel může aktuálně vidět na obrazovce, aby byl u systému zajištěn běh procesu dané aktivity.
- Určují vyšší priority procesům aktivit, ze kterých uživatel nedávno vystoupil, pokud tyto aktivity obsahují prvky, ke kterým se uživatel může vrátit (pozastavené aktivity).
- Pomáhají aplikaci obsluhovat ukončení vlastního procesu, aby se uživatel mohl vrátit k aktivitám, u kterých je obnoven jejich předchozí stav.
- Poskytují aplikacím možnost implementovat určité vazby mezi sebou, a systému umožňují tyto vazby koordinovat.

Uspořádání aktivit je hierarchické. To znamená, že po spuštění aplikace se spustí nejprve hlavní spouštěcí aktivita, ze které je možné spouštět aktivity další. Z hlediska kódu programu je aktivita implementována jako podtřída třídy `Activity`, která definuje metody reagující na stav aktivity. Celý životní cyklus aktivity s pořadím volání těchto metod můžeme vidět na obrázku 2.8.



Obrázek 2.8: Schéma životního cyklu aktivity.[13]

Služby

Služba je všeobecný vstupní bod, skrze který se může udržet aplikace v běhu na pozadí, a to z různých důvodů. Je to komponenta, která v pozadí provádí dlouhodobé operace, nebo práci pro vzdálené procesy. Služba neposkytuje uživateli žádné rozhraní. Může být například použita k přehrávání hudby v pozadí, zatímco je uživatel v jiné aplikaci, nebo může stahovat data přes síť, aniž by blokovala uživatele, který se nachází v jiné aktivitě. Jiný typ komponenty, jako je například *aktivita*, může zahájit *službu* a nechat ji běžet, nebo se na ni napojit, aby s ní mohla interagovat.

Existují dvě sémantiky, kterými služby říkají systému, jak má řídit aplikaci: Zahájená služba říká systému, aby ji nechal běžet, dokud nedokončí svoji práci. Například to může být za účelem synchronizace dat v pozadí. Druhým typem je situace, kdy je služba napojena na jiný proces. Pokud tedy například proces A je napojen na službu v procesu B, systém ví, že je potřeba udržet proces B (a jeho službu) v běhu.

Z hlediska kódu programu je služba implementována jako podtřída třídy `Service`.

Přijímače vysílání

Komponenta, která umožňuje systému, aby doručil události k aplikaci mimo regulérní akce uživatele, se nazývá *přijímač vysílání*. Tím umožňuje aplikaci reagovat na celosystémové oznámení událostí a události také vysílat. Díky tomu, že jsou přijímače dalším samostatným vstupním bodem do aplikace, systém může doručovat oznámení i do aplikací, které nejsou zrovna v běhu. Například tedy může aplikace oznámit notifikací, že do telefonu přišla nová SMS zpráva, aniž by tato aplikace musela po celý čas do doby notifikace běžet. Docílí se toho tak, že systém doručí hlášení notifikace do přijímače vysílání této aplikace. Mnohá hlášení pocházejí přímo od systému. Například vypnutí displeje, nízký stav baterie, a tak dále. Aplikace mohou také vytvářet hlášení.

Z hlediska kódu programu je přijímač implementován jako podtřída třídy `BroadcastReceiver` a každé hlášení je doručeno jako objekt třídy `Intent`.

Poskytovatelé obsahu

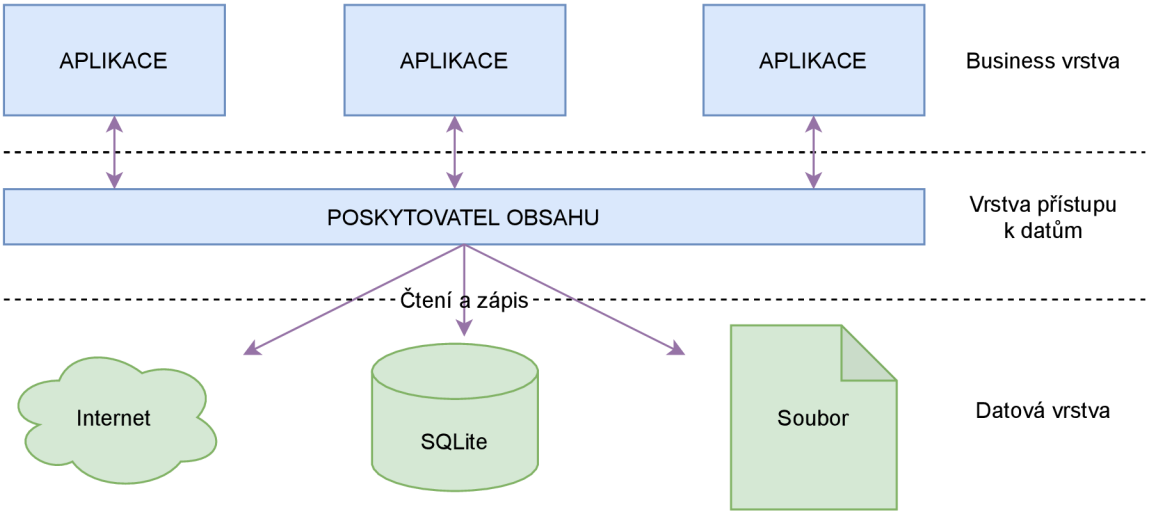
Poskytovatel obsahu řídí soubor aplikačních dat, která se mohou uložit v souborovém systému, SQLite databázi, na webu, nebo jakémkoli jiném persistentním úložišti, ke které může mít aplikace přístup. Skrze poskytovatele obsahu mohou ostatní aplikace modifikovat, nebo žádat o data, pokud to poskytovatel povolí. Například operační systém poskytuje poskytovatele obsahu, který řídí kontaktní informace uživatele. Každá aplikace s příslušnými právy potom může poskytovatele žádat o zápis, či čtení dat o jakémkoli kontaktu.

Při práci s poskytovatelem se využívá rozhraní podobného databázovému, poskytovatelé obsahu jsou ale víc než jen databáze. Pro systém jsou poskytovatelé obsahu vstupním bodem aplikace, skrze který lze publikovat pojmenované datové položky, které jsou identifikované skrze schéma jednotného identifikátoru zdroje (URI). Tím pádem si aplikace mohou samy rozhodnout, jak budou vlastní data mapovat na jmenný prostor URI. Poté tyto identifikátory rozdávají ostatním entitám, které je použijí pro přístup k jejich datům. Identifikátory potom přetrvávají, i když aplikace aktuálně není spuštěná. Tudíž systému stačí, aby aplikaci spustil až když potřebuje pracovat s daty přes určitý identifikátor. Navíc

[13] Převzato z https://developer.android.com/guide/components/activities/activity-lifecycle.html.

lze díky oprávněním na určitý identifikátor vytvářet velice specifický model bezpečnosti dat aplikace.

Z hlediska kódu programu je poskytovatel implementován jako podtřída třídy `ContentProvider` a musí implementovat standardní sadu metod pro provádění datových transakcí. Schéma napojení aplikací a dat na poskytovatele můžeme vidět na obrázku 2.9.



Obrázek 2.9: Schéma poskytovatele obsahu.[14]

2.3 Vizuální programování

Vizuální programování je způsob vytváření počítačových programů za pomoci *vizuálních programovacích jazyků*. Tyto jazyky umožňují uživateli vytvářet programové konstrukce pomocí manipulace grafických elementů místo vytváření textu zdrojového kódu. První pokusy o vytvoření takového jazyku započaly už v 70.letech [7], avšak až v současnosti začíná být jejich využití opravdu rozšířené. Oblast, ve které jsou tyto jazyky velice rozšířené a využívané je vzdělávání. Vizuální programovací jazyky jsou totiž velice intuitivní a díky tomu se skvěle hodí k výuce zásad programového myšlení a základních typů programových konstrukcí.

2.3.1 Záměry a výhody

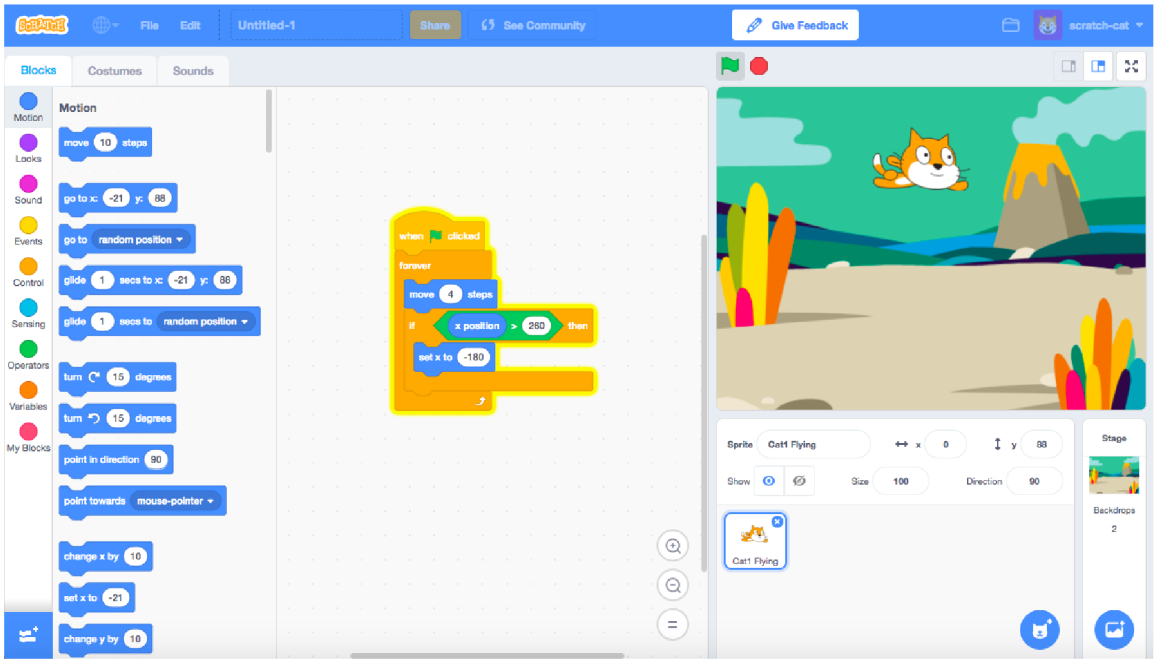
Hlavním záměrem většiny z těchto jazyků je učinit programování více přístupným, a to zejména snížením obtíží, se kterými se potýkají začátečníci při programování. Toto se týká hlavně těchto tří oblastí [28]:

- **Syntaktická** je o seskládání komponent programovacího jazyka do jednolitého programu.
- **Sémantická** se týká pochopení významu programu i jeho jednotlivých komponent.
- **Pragmatická** je hlavně o praktických záležitostech programovacích jazyků, jako je například pochopení významu různých programů zasazených do kontextu specifických situací.

[14] Převzato z https://www.tutorialspoint.com/android/android_content_providers.htm.

Jedna z hlavních pomocí je při udržování syntaktických pravidel. Při psaní programu v textové podobě je třeba dodržovat přísná pravidla syntaxu daného jazyka, aby bylo docílené požadované funkce programu. U vizuálního programování jsou základní komponenty programu většinou reprezentovány jako prvky s vizuální nápovědou jejich použití a spojování. Tudíž pro vytvoření programu skládá uživatel jednotlivé bloky jako stavebnici ve formě, která má určitou logiku. Syntakticky nesprávné výrazy je proto v podstatě nemožné vytvořit. Navíc jsou jednotlivé bloky přístupné z určitého centrálního katalogu. Uživatel tedy nemusí složitě pročítat dokumentaci, aby zjistil, jaké funkce má k dispozici, což je další hlavní překážkou pro začátečníka v programování. Další výhodou vizuálního programování je možnost intuitivně zobrazovat abstrakci. Programy jsou totiž definovány svými tvary, takže uživatel může jednoduchým pohledem na tyto základní tvary zjistit vlastnosti a členění programu.Tím pádem může daný program i jednodušeji vysvětlit ostatním uživatelům, kteří ani nemusí mít s daným jazykem zkušenosti. To také napomáhá udržitelnosti programu a snadné refaktorizaci.

Vývojové prostředí vizuálních programovacích jazyků je většinou propojeno se zjednodušeným prostředím pro provádění programu, takže uživatel může v rychlosti program zkoušet a vidět jeho výsledky. Všechny tyto výhody nejsou samozřejmě exkluzivními vlastnostmi vizuálních programovacích jazyků, avšak právě u nich jsou díky jejich vlastnostem nejvíce zvýrazněny. Příklad vizuálního programování můžeme vidět na obrázku 2.10.



Obrázek 2.10: Scratch - populární vizuální programovací jazyk určen zejména pro vzdělávání.[15]

[15] Převzato z https://medium.com/scratchteam-blog/3-things-to-know-about-scratch-3-0-18ee2f564278.

2.3.2 Nic není dokonalé

I přes všechny výhody nejsou vizuální programovací jazyky bezchybné a bez zádrhelů. Ostatně, kdyby tomu tak nebylo, používali bychom je dnes téměř všude. Je třeba zmínit hlavní fakt: I když vizuální, pořád se jedná o programování. I každá „stavebnice bloků“ vyžaduje precizní a jednoznačnou kontrolu toku programu a i přes to, že vizuální programovací jazyky můžou usnadnit učení a zmírnit náročnosti syntaxu klasických programovacích jazyků, každé programování vyžaduje po vývojáři, aby se naučil všeobecným, ale i jazykově specifickým konceptům programování (například koncept proměnné a všeobecnosti imperativního programování).

Další nevýhodou je špatná rozšířitelnost. Existuje jen určité množství barev a tvarů bloků, které budou pro uživatele rozlišitelné. Proto v případě, kdy si při konvenčním programování vývojář pomůže použitím jakékoli knihovny a k rozlišení proměnných, funkcí, objektů a metod mu stačí rozdíl pár písmen, vizuální programování silně zaostává. Další velkou nevýhodou je, že všeobecně mají vizuální programovací jazyky technické problémy. Hlavně z hlediska rychlosti provádění programu.

2.3.3 Typy vizuálních programovacích jazyků

Vizuální programovací jazyky se mohou všeobecně dělit dle mnoha kritérií, a nemají dělení nijak formálně určeno, jejich dělení je tedy debatabilní záležitostí. Následující příkladové typy dělí tyto jazyky pouze orientačně, tedy hlavně z hlediska nejmenšího počtu skupin, do kterých lze jejich většinu zařadit. [7].

Blokové stavebnice

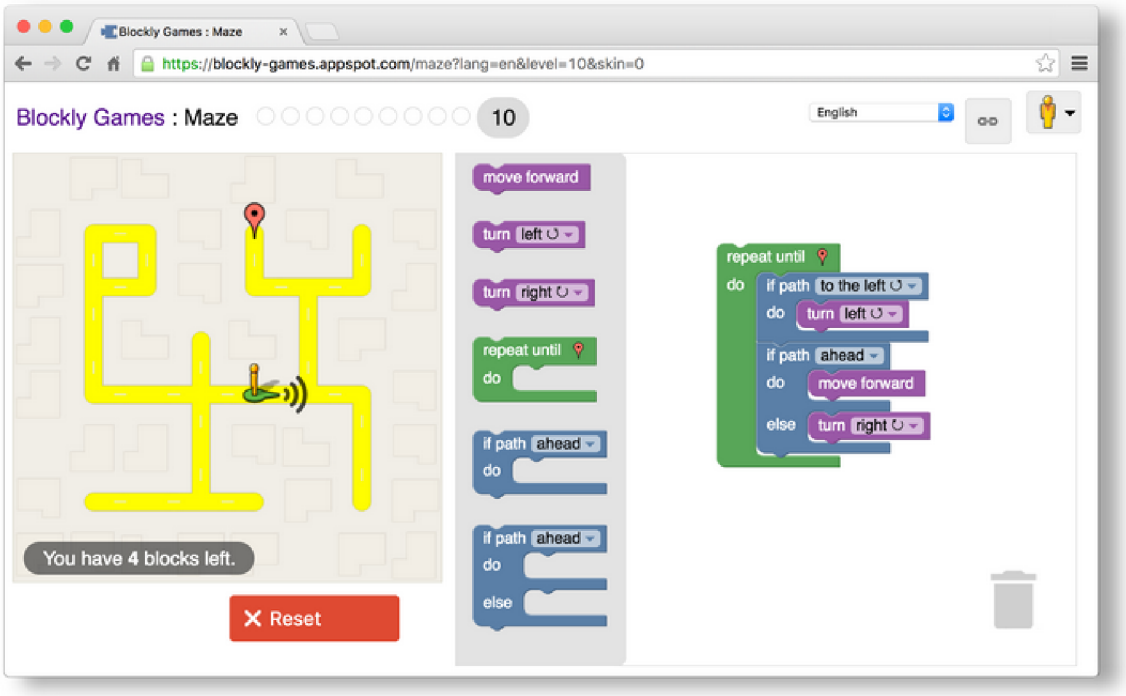
Tento typ vizuálního programovacího jazyku je to, co si pod tímto pojmem představí většina lidí. Jejich gramatika vychází z klasických imperativních programovacích jazyků. Ve stylu skládání bloků, které představují například proměnné, podmínky a ovladače toku programu, dochází k vytváření ucelené struktury. Textová pole slouží k definici proměnných, podmínek, atd. Grafický návrh vytváří nápovědu pro uživatele, aby intuitivně mohl skládat bloky do správné formy. Tyto jazyky jde většinou doplňovat o uživatelem definované bloky. Příkladem těchto jazyků může být Blockly[16], jehož uživatelské rozhraní můžeme vidět na obrázku 2.11.

V porovnání s ostatními vizuálními jazyky je tento typ velice blízko klasickému zdrojovému kódu imperativního jazyka.

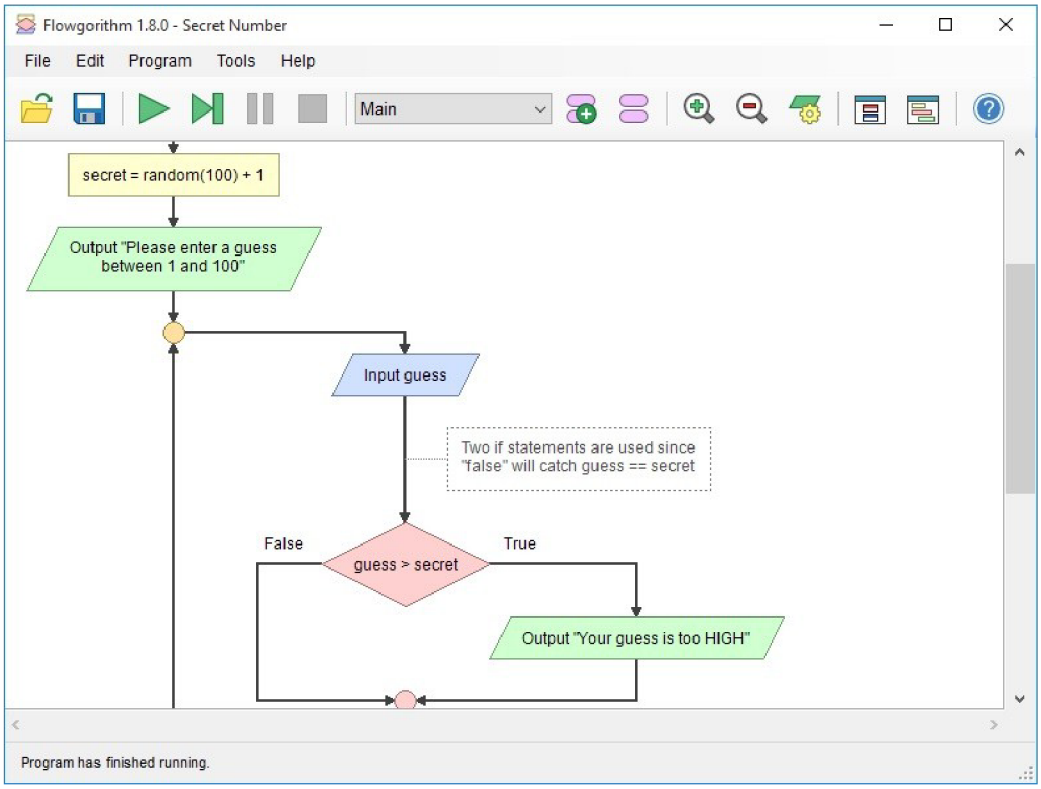
Vývojové diagramy

Tyto jazyky užívají vizuální reprezentace blízké vývojovým diagramům, aby popsaly tok řízení programu. Spoje mezi bloky reprezentují usměrněnou sekvenci provádění programu postupně skrze dané bloky. Tento styl vizuální gramatiky je velice jednoduchý na pochopení z hlediska syntaxu. Logické konstrukce, které se dají vytvořit skrze tento jazyk jsou však velmi omezené. Hodně záleží na obsahu jednotlivých bloků. Příkladem těchto jazyků je Flowgorithm[17], jehož uživatelské prostředí můžeme vidět na obrázku 2.12.

[16] https://developers.google.com/blockly
[17] http://www.flowgorithm.org/
[18] Převzato z https://developers.google.com/blockly.



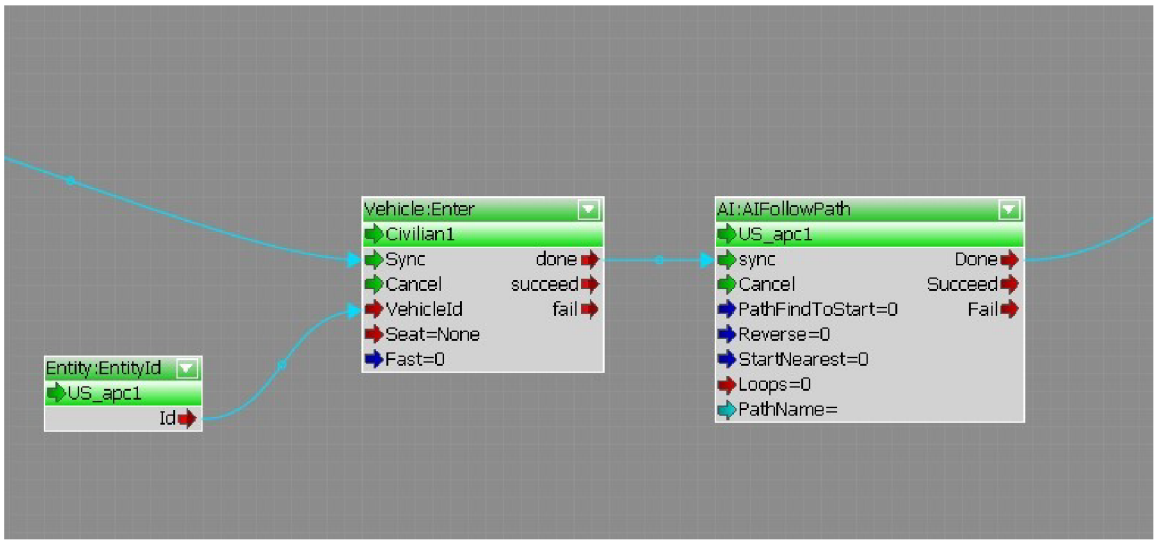
Obrázek 2.11: Vizuální programovací jazyk Blockly zastupuje kus kódu ve vybraném konvenčním jazyku jednotlivými bloky.[18]



Obrázek 2.12: Vizuální programovací jazyk ve stylu vývojového diagramu.[19]

Programování tokem dat

Tento formát je nejvíce používaný v profesionálních nasazeních [23]. Je zaměřen spíše na designéry, než na konečné uživatele, či pro výuku programování. Bloky zde reprezentují funkce a jsou navzájem propojeny skrze svoje vstupy s výstupy. Tato propojení reprezentují toky dat. Vizuální gramatika je tedy velice jednoduchá a nejvíce práce je odvedeno v samotných funkcích/blocích, které jsou většinou definovány klasickým programovacím jazykem. Ve většině případů jsou tyto bloky a jejich chování již zadány a jejich použití se opakuje. Uživatel pracuje pouze s propojením toků dat. Příkladem může být LabView[20]. Ukázku prostředí programování tokem dat můžeme vidět na obrázku 2.13.



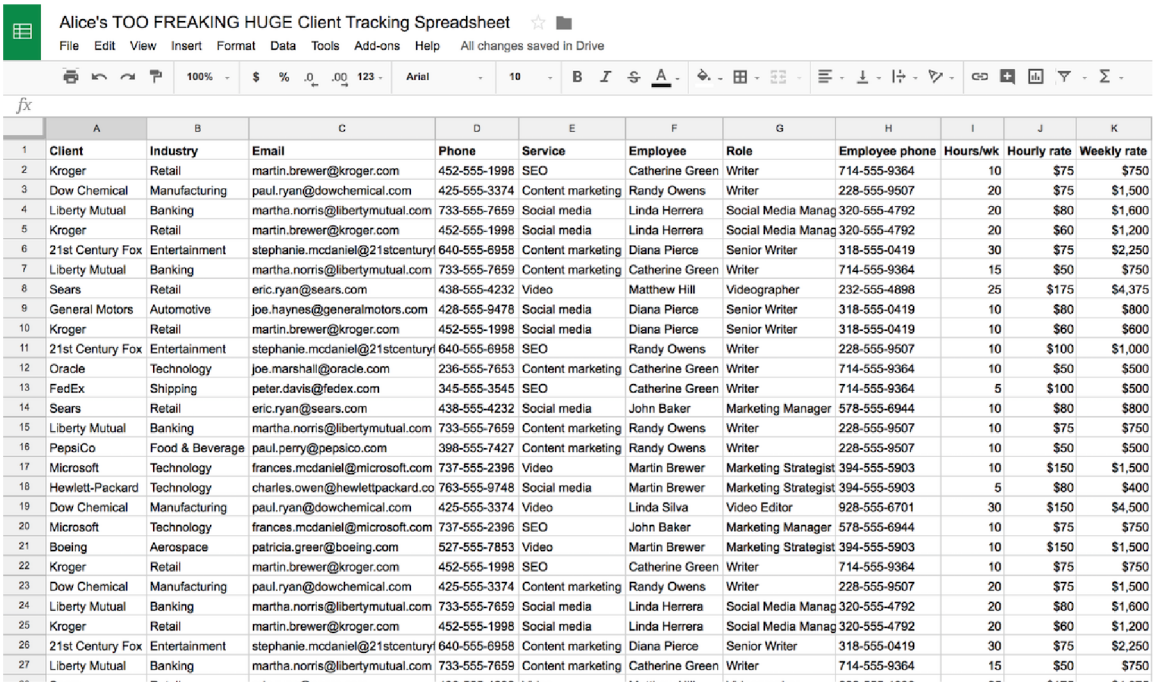
Obrázek 2.13: Programování tokem dat ve vývojovém prostředí CryEngine.[21]

2.3.4 Programování koncovým uživatelem

Nejrozšířenějším a nejuplatnitelnějším případem používání vizuálních programovacích jazyků je programování koncového uživatele. Tímto programováním rozumíme metody, aktivity a nástroje, které umožňují uživatelům, kteří nejsou považováni za profesionální softwarové vývojáře, aby mohli vytvářet, upravovat nebo rozšiřovat software nebo jeho části [16]. Tento přístup je nejvíce uplatnitelný v situacích, kdy si uživatel upravuje a ladí parametry a rozhraní svého pracovního nástroje, jelikož sám uživatel zná nejlépe detaily svého pracovního kontextu a změny svého pracovního prostředí. Dalším důvodem pro tento přístup je fakt, že koncoví uživatelé jsou několikanásobně početnější skupina, než samotní vývojáři, kteří pro ně software vyvíjí.

Za nejjednodušší příklad programování koncového uživatele se dá pokládat vytváření tabulek v tabulkovém procesoru [22]. Tento příklad se zdá být tím nejlepším na shrnutí celé situace ohledně tohoto paradigmatu. Ukazuje totiž největší výhody, kterými jsou dostupnost a jednoduchost použití, které jsou prokázané popularitou tohoto softwaru. Zároveň se zde ale projevují hlavní problémy tohoto přístupu. Hlavní překážkou, kterou je třeba překonat, je vysoká chybovost ve výsledném softwaru.

[19] Převzato z https://www.craft.ai/blog/the-maturity-of-visual-programming.
[20] https://www.ni.com/cs-cz/shop/labview.html
[21] Převzato z https://www.craft.ai/blog/the-maturity-of-visual-programming.


Alice's TOO FREAKING HUGE Client Tracking Spreadsheet

File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive

	A	B	C	D	E	F	G	H	I	J	K
1	Client	Industry	Email	Phone	Service	Employee	Role	Employee phone	Hours/wk	Hourly rate	Weekly rate
2	Kroger	Retail	martin.brewer@kroger.com	452-555-1998	SEO	Catherine Green	Writer	714-555-9364	10	$75	$750
3	Dow Chemical	Manufacturing	paul.ryan@dowchemical.com	425-555-3374	Content marketing	Randy Owens	Writer	228-555-9507	20	$75	$1,500
4	Liberty Mutual	Banking	martha.norris@libertymutual.com	733-555-7659	Social media	Linda Herrera	Social Media Manag	320-555-4792	20	$80	$1,600
5	Kroger	Retail	martin.brewer@kroger.com	452-555-1998	Social media	Linda Herrera	Social Media Manag	320-555-4792	20	$60	$1,200
6	21st Century Fox	Entertainment	stephanie.mcdaniel@21stcentury	640-555-6958	Content marketing	Diana Pierce	Senior Writer	318-555-0419	30	$75	$2,250
7	Liberty Mutual	Banking	martha.norris@libertymutual.com	733-555-7659	Content marketing	Catherine Green	Writer	714-555-9364	15	$50	$750
8	Sears	Retail	eric.ryan@sears.com	438-555-4232	Video	Matthew Hill	Videographer	232-555-4898	25	$175	$4,375
9	General Motors	Automotive	joe.haynes@generalmotors.com	428-555-9478	Social media	Diana Pierce	Senior Writer	318-555-0419	10	$80	$800
10	Kroger	Retail	martin.brewer@kroger.com	452-555-1998	Social media	Diana Pierce	Senior Writer	318-555-0419	10	$60	$600
11	21st Century Fox	Entertainment	stephanie.mcdaniel@21stcentury	640-555-6958	SEO	Randy Owens	Writer	228-555-9507	10	$100	$1,000
12	Oracle	Technology	joe.marshall@oracle.com	236-555-7653	Content marketing	Catherine Green	Writer	714-555-9364	10	$50	$500
13	FedEx	Shipping	peter.davis@fedex.com	345-555-3545	SEO	Catherine Green	Writer	714-555-9364	5	$100	$500
14	Sears	Retail	eric.ryan@sears.com	438-555-4232	Social media	John Baker	Marketing Manager	578-555-6944	10	$80	$800
15	Liberty Mutual	Banking	martha.norris@libertymutual.com	733-555-7659	Content marketing	Randy Owens	Writer	228-555-9507	10	$75	$750
16	PepsiCo	Food & Beverage	paul.perry@pepsico.com	398-555-7427	Content marketing	Randy Owens	Writer	228-555-9507	10	$50	$500
17	Microsoft	Technology	frances.mcdaniel@microsoft.com	737-555-2396	Video	Martin Brewer	Marketing Strategist	394-555-5903	10	$150	$1,500
18	Hewlett-Packard	Technology	charles.owen@hewlettpackard.co	763-555-9748	Social media	Martin Brewer	Marketing Strategist	394-555-5903	5	$80	$400
19	Dow Chemical	Manufacturing	paul.ryan@dowchemical.com	425-555-3374	Video	Linda Silva	Video Editor	928-555-6701	30	$150	$4,500
20	Microsoft	Technology	frances.mcdaniel@microsoft.com	737-555-2396	SEO	John Baker	Marketing Manager	578-555-6944	10	$75	$750
21	Boeing	Aerospace	patricia.greer@boeing.com	527-555-7853	Video	Martin Brewer	Marketing Strategist	394-555-5903	10	$150	$1,500
22	Kroger	Retail	martin.brewer@kroger.com	452-555-1998	SEO	Catherine Green	Writer	714-555-9364	10	$75	$750
23	Dow Chemical	Manufacturing	paul.ryan@dowchemical.com	425-555-3374	Content marketing	Randy Owens	Writer	228-555-9507	20	$75	$1,500
24	Liberty Mutual	Banking	martha.norris@libertymutual.com	733-555-7659	Social media	Linda Herrera	Social Media Manag	320-555-4792	20	$80	$1,600
25	Kroger	Retail	martin.brewer@kroger.com	452-555-1998	Social media	Linda Herrera	Social Media Manag	320-555-4792	20	$60	$1,200
26	21st Century Fox	Entertainment	stephanie.mcdaniel@21stcentury	640-555-6958	Content marketing	Diana Pierce	Senior Writer	318-555-0419	30	$75	$2,250
27	Liberty Mutual	Banking	martha.norris@libertymutual.com	733-555-7659	Content marketing	Catherine Green	Writer	714-555-9364	15	$50	$750

Obrázek 2.14: Typické použití tabulkového procesoru. Převzato z https://medium.com/the-fieldbook-blog/its-ok-to-hate-your-spreadsheet-ea67a7579073.

Jednou z posledních evolucí v této oblasti je používání mobilních zařízení pro aktivity spojené s vývojem softwaru koncovým uživatelem [27]. Pro tento případ je však potřeba navrhnout zcela odlišné systémy, než ty navržené pro stolní zařízení. Například omezená velikost obrazovek těchto zařízení vyžaduje promyšlený návrh prezentace obsahu a ovládacích prvků vývojového prostředí. Specifika moderních mobilních zařízení umožňují provádět vývoj koncovým uživatelem za pohybu v dynamickém pracovním prostředí, což přináší zcela nové možnosti.

Dalším novým směrem, kterým se toto paradigma ubírá, je využití ve vývoji internetu věcí. Záměrem je hlavně vylepšení personalizace nastavení jednotlivých zařízení i celkového systému skrze využití algoritmizace. Slibným přístupem je takzvané „Trigger-Action“ programování [20]. Samozřejmě jsou v cestě různé překážky. Jednou z nich je například nedostatek konzistentní terminologie, což má za výsledek různá pojmenování stejných konceptů v různých prostředích. Tato skutečnost uživateli přidává na obtížnosti systém pochopit.

Kapitola 3

Návrh řešení

Tato kapitola popisuje rozbor již existujících podobných řešení zadané problematiky. Poté je představen návrh řešení práce samotné. Zprvu je popsán celý systém a jeho součásti a následně je obsáhleji popsán návrh mobilní aplikace a jejího uživatelského rozhraní. Tato část návrhu je nejstěžejnější, protože představuje hlavní část zodpovědnou za jednoduchost použití koncovým uživatelem. Bylo jí tedy nutno věnovat zvláštní pozornost. Poslední část této kapitoly se věnuje návrhu komunikačního rozhraní mezi aplikací a serverem.

3.1 Existující řešení

Kolaborativní roboti jsou dnes komerčně dostupné nástroje s velikou variabilitou atributů a účelů, stejně jako software na jejich programování. Tyto aplikace jsou určeny pouze pro ovládání daných robotů. Umožňují jejich programování pomocí inverzní kinematiky, skládání programů z tzv. bloků a jiných intuitivních metod, tudíž není třeba pokročilejších znalostí programování k práci s nimi. Zaměření těchto aplikací je buď edukativní, nebo je jejich software určen pro domácí kutily, či automatizaci profesionální produkce v menší škále.

3.1.1 Dobot Magician

DOBOT Magician je multifunkční robotické rameno, které je určeno hlavně pro praktické edukativní účely. Jsou k němu k dispozici různé nástroje, se kterými může rameno pracovat a samozřejmě software pro jeho ovládání zvaný **Dobot Studio** [4], který můžeme vidět na obrázku 3.1.

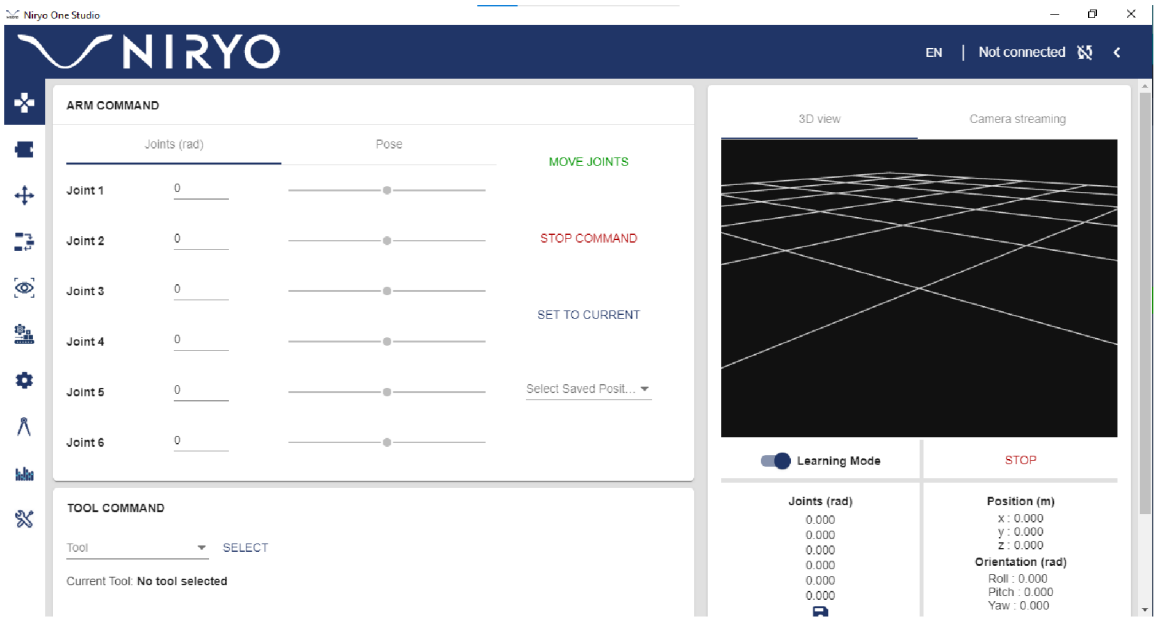
Ovládací systém je poměrně bohatý na různé funkce, ale kvůli zachování edukativního záměru celého projektu zůstává uživatelské prostředí čisté a přehledné. Je možné robotické rameno ovládat v reálném čase bočním panelem pomocí tlačítek. Je dostupné ovládání pomocí inverzní kinematiky, i skrz nastavování jednotlivých kloubů. Dále aplikace nabízí různé specifické funkce pro ovládání ramene. K dispozici je opakování ručně nastavené sekvence pozic ramene, vytváření programů pro robota skrz rozhraní Blockly, nebo i klasické psaní kódu skriptu. Dále je nabízena funkce 3D tisku, kreslení na 2D plochu, či gravírování laserem. Aplikace umožňuje také nekonvenční metody ovládání robota, například ovládání pohybem myši, či snímanými gesty lidské ruky.

Aplikace je dostupná pouze pro pro platformy Windows a Mac OS, nikoli pro mobilní platformy. Pro zajištění spojení aplikace a robotické ruky je třeba obě strany propojit sériovým USB rozhraním.

Obrázek 3.1: Dobot Studio - software určený pro ovládání robotického ramene Dobot Magician.

3.1.2 Niryo One

Platforma, která pracuje na stejném principu, ale s jiným zaměřením, se nazývá Niryo One. Obsahuje šestiosé kolaborativní robotické rameno a ovládací software. Tento systém je navržen pro pokročilejší vzdělávání v oblasti robotiky, odborné trénování a laboratoře pro výzkum a vývoj. Je založen na platformě ROS[1]. Aplikace pro ovládání ramene se nazývá **Niryo One Studio** [8].



Obrázek 3.2: Niryo One Studio - software určený pro ovládání robotického ramene Niryo One.

Na obrázku 3.2 můžeme vidět uživatelské rozhraní aplikace. Na rozdíl od předchozího případu, Niryo One Studio nenabízí tak velikou rozmanitost funkcí, ale zaměřuje se spíš na

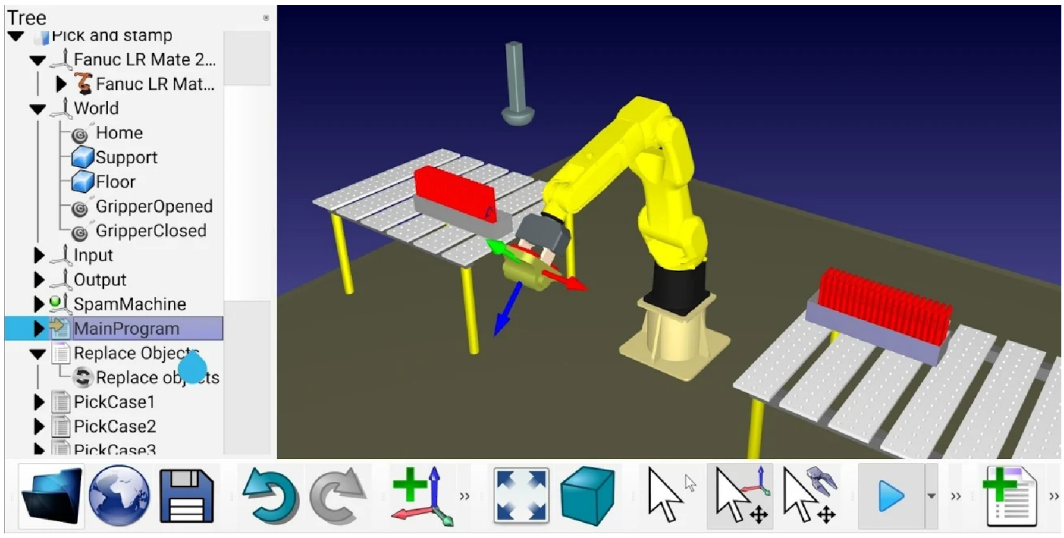
[1] https://www.ros.org/

kompletní funkčnost, reálnou použitelnost a preciznost. Umožňuje ovládat robota v reálném čase nastavováním kloubů i pomocí inverzní kinematiky. Zároveň má k dispozici pohled na 3D model ramene. Mezi další funkce se řadí vizuální programování ramene pomocí prostředí Blockly, ovládání pomocí externího gamepadu, nebo ukládání sekvencí a pozic ramene. Pro vývojáře je také dostupné Python aplikační rozhraní pro ovládání robota pomocí přímých příkazů. ROS kód platformy robotického ramene je navíc otevřený veřejnosti, tudíž nabízí další možnosti pokročilejším vývojářům.

Robotické rameno disponuje bezdrátovou Wi-fi konektivitou, a to jak přes externí vysílač, tak v tzv. „hotspot“ módu, kdy Wi-fi signál vysílá samo. Navíc je možné skrze digitální piny rameno propojit s ostatními zařízeními, jako je například Arduino, či Raspberry Pi. Jedná se o jedno z funkčně nejpokročilejších řešení.

3.1.3 RoboDK Mobile

RoboDK Mobile je mobilní verze univerzálního softwaru na programování a simulaci robotických ramen [21].



Obrázek 3.3: Uživatelské prostředí Android aplikace RoboDK Mobile. Převzato z https://play.google.com/store/apps/details?id=org.robodk.app.sharex.

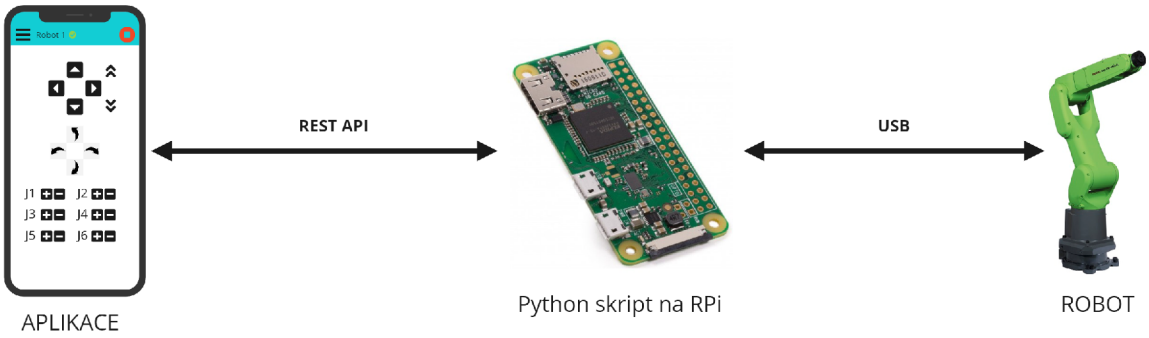
Podporuje více než 400 robotů od 40 různých výrobců ze své knihovny a umožňuje vytváření univerzálního programu pro robotická ramena a jejich následný export na specifickou platformu. Tento projekt je zaměřen na průmyslové a produkční použití. Základní způsob programování robotických ramen v této aplikaci spočívá v interaktivním provádění simulace v 3D prostoru, kde uživatel zadává robotickému rameni cíle a úkoly. Aplikace si potom jednotlivé instrukce ukládá do sekvence. Zároveň je ale pokročilejším uživatelům poskytnuta možnost přímo upravovat a vytvářet instrukce pro robota, nebo přistupovat k nástrojům pro obrábění materiálu, či 3D tisk.

Jedinou nevýhodou aplikace je její horší intuitivnost a přehlednost uživatelského rozhraní. Výhodou je naopak samotná podstata vytváření programů díky skvělému přísunu zpětné vazby vizualizací na 3D modelu. Tento jev je však očekáván kvůli zaměření na profesionální použití. Samotná mobilní verze není kvůli podstatě ovládacích prvků ideální jako samostatná jednotka pro programování robotického ramene, ale spíše se uplatní jako ob-

doba tzv. „teach pendantu“, tedy přenosného zařízení určeného pro finální menší úpravy programu robota.

3.2 Systém pro ovládání robotického ramene

Celý systém bude z pohledu uživatele fungovat tak, že po zapnutí aplikace v mobilním zařízení se bude moci přes síť bezdrátově připojit k robotickému rameni. Celou jeho funkčnost bude potom ovládat přes aplikaci. Mezi aplikaci a samotné robotické rameno je mezičlánek, který zajistí funkčnost. Znázornění všech komponent systému můžeme vidět na obrázku 3.4.



Obrázek 3.4: Schéma systému pro ovládání a programování robotického ramene.

Jak už schéma napovídá, na počátku cesty mezi uživatelem a robotem stojí mobilní aplikace, která mu poskytuje rozhraní k jeho ovládání. Uživatel v ní ovládá robota v reálném čase, nebo vytváří a spouští programy, dle kterých se robot řídí. Aplikace, respektive mobilní Android zařízení komunikuje skrz síťové připojení se serverem běžícím na malém zařízení s bezdrátovou síťovou konektivitou a zasílá mu jednoduché zprávy o instrukcích pro robotické rameno. Původně bylo v plánu použít zařízení typu „systém na čipu“ (SoC), jako je například platforma ESP8266[2]. V pozdějším průběhu návrhu celého systému však narůstaly obavy, že výkon tohoto zařízení nebude dostačující. Nejen z tohoto důvodu padla nakonec volba na alternativní řešení - **Raspberry Pi**[3]. Jedná se o minipočítač, který je schopný fungovat na bázi plnohodnotného operačního systému, jako je například Raspberry Pi OS, což je systém odvozený od distribuce Linuxu zvané Debian[4]. Cenově je toto řešení oproti systému na čipu rozdílné jen zanedbatelně (v případě modelu Raspberry Pi Zero W).

Hlavní výhoda spočívá v jednoduchosti implementace díky již zmiňovanému operačnímu systému, kde je možné server, se kterým bude naše aplikace komunikovat, implementovat jako například prostý skript v jazyku Python[5]. Tento způsob implementace serveru je přesně ten, který byl zvolen. Hlavní důvod pro toto rozhodnutí je fakt, že pro Python jsou dostupné různé aplikační rámce pro implementaci HTTP REST API serveru, jako je například Flask[6]. Dalším důvodem je existence velkého množství volně dostupných modulů, které implementují např. výpočet inverzní kinematiky[7], což výrazně rozšiřuje funkčnost celého systému.

[2] http://esp8266.net/
[3] https://www.raspberrypi.org/
[4] http://www.debian.cz/
[5] https://www.python.org/
[6] https://flask.palletsprojects.com/en/1.1.x/
[7] https://github.com/Phylliade/ikpy

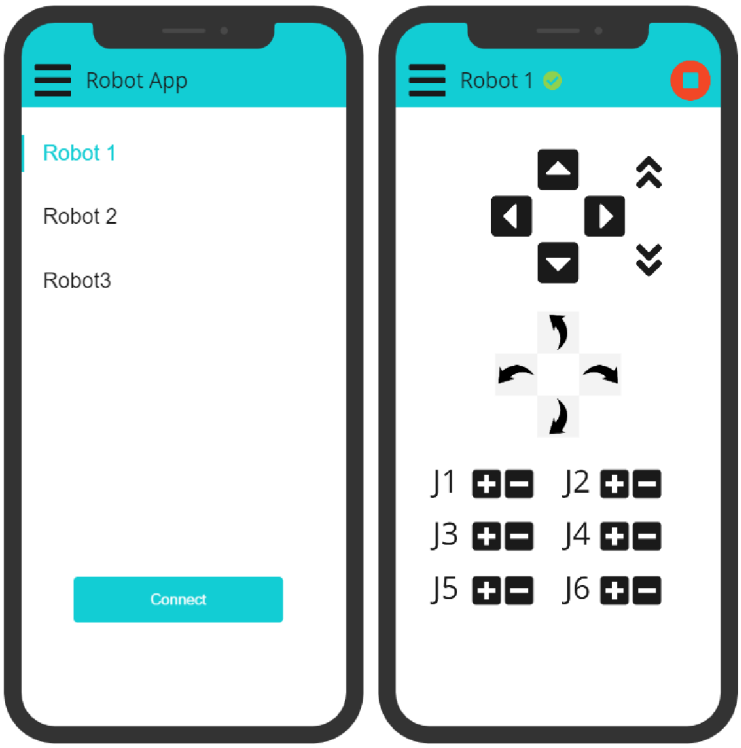
Dále je potřeba v rámci serveru zajistit implementaci jednotlivých instrukcí robotického ramene a implementaci následné komunikace s ovladačem servomotorů robota. Server bude navržen tak, že komunikace s ovladači robota bude probíhat skrz jednotné rozhraní, což umožní kompatibilitu s různými roboty. Samotný ovladač potom přijímá jednotlivé příkazy pro ovládání servomotorů robotického ramene, zpracovává je a skrze přímé spojení ke každému motoru vysílá impulsy pro jejich ovládání.

3.3 Aplikace

V této části kapitoly o návrhu je popsán návrh mobilní aplikace, která bude z pohledu uživatele hlavním pracovním nástrojem, se kterým bude interagovat. Jmenovitě jde o návrh grafického uživatelského rozhraní, editoru programů pro robotické rameno a způsobu komunikace s ramenem.

3.3.1 Uživatelské rozhraní aplikace

Návrh uživatelského rozhraní byl vytvořen formou sady grafických skic, takzvaných „wireframů“, které předvádějí hlavní součásti rozhraní aplikace. Prostředí aplikace bylo navrženo tak, aby bylo co nejčistší a nejpřehlednější. Inspiruje se zásadami tzv. material designu, tudíž jsou použita jednoduchá a kontrastní barevná schémata, stejně jako tvary ovládacích prvků aplikace.



Obrázek 3.5: Grafický návrh uživatelského prostředí aplikace s ovládáním robota v reálném čase (napravo).

Na první skice, která se nachází na obrázku 3.5 můžeme vidět prostředí, do kterého se uživatel dostane hned poté, co zapne aplikaci. V jednoduchém seznamu uprostřed vidí

výčet dostupných robotických ramen, ke kterým se může připojit. Příslušného robota může označit a stiskem tlačítka se k němu připojit. Ve vrchní barevné liště se potom nachází pouze text obsahující název aplikace a na levé straně tlačítko, které uživateli umožňuje odhalit boční výsuvný panel, který bude obsahovat jednoduchou navigaci mezi různými sekcemi prostředí aplikace.

Po připojení k robotickému rameni aplikace uživatele přesměruje do prostředí, ve kterém má k dispozici ovládací prvky ve formě tlačítek, které mu umožňují ovládat robotické rameno okamžitě v reálném čase. Ovládat ho může třemi způsoby. Prvním je uskutečněn za pomoci využití inverzní kinematiky, kdy uživatel pouze specifikuje rameni robota směr pohybu jeho koncového efektoru v pomyslné kartézské soustavě souřadnic, a to posunem ve směru jednotlivých os souřadnic. Dále může uživatel stejným způsobem upravovat náklon koncového efektoru všemi směry.

Poslední způsob ovládání je přímá změna ohybu jednotlivých kloubů robotického ramene. Ke každému kloubu robota má uživatel k dispozici tlačítka pro změnu ohybu oběma směry. Navíc si lze všimnout, že po připojení k rameni přibyl ve vrchní liště indikátor potvrzující připojení a červené tlačítko fungující jako takzvaná záchranná brzda, které slouží k okamžitému zastavení pohybu robota v jakékoli situaci.

Obrázek 3.6: Další skicy grafického návrhu grafického rozhraní aplikace. Nalevo výběr programů pro robota, napravo rozhraní pro spouštění a ovládání a uprostřed prostředí pro vytváření a úpravu programů.

Na dalších skicách, které můžeme vidět na obrázku 3.6, se na levé straně nachází rozhraní výběru programu, který může robot vykonat, stejně jako možnost vytvářet nové programy pomocí tlačítka na spodní straně. Po volbě požadovaného programu se uživatel dostane do prostředí, kde může program pomocí tlačítek spouštět, zastavovat, či pozastavovat. Dále

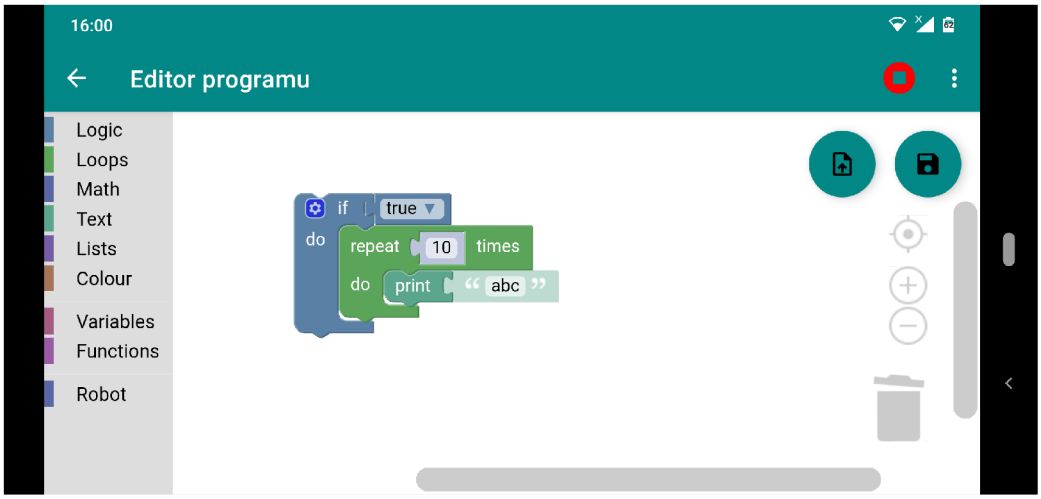
zde bude moci upravovat název programu a jeho popis, vytvářet vstupní signály programu za pomocí tlačítek, či program odstranit z aplikace.

Editor programů

Stiskem tlačítka ve tvaru puzzle součástky se uživatel dostane do prostředí editoru programů (stejně jako stiskem tlačítka pro přidání programu na skice vlevo), což je nejstěžejnější součást aplikace. Účelem tohoto editoru je poskytnout uživateli jednoduchý, rychlý a efektivní nástroj pro úpravu a vytváření programů díky možnostem vizuálního programování. Původním záměrem bylo navrhnout vlastní prostředí a vizuální programovací jazyk. Jeho skicu můžeme vidět na obrázku 3.6 uprostřed. Uživatel by skládal pod sebe jednotlivé instrukce a upravoval jejich operandy, čímž by vytvářel sekvenci, kterou by robot poté vykonával. Cílem bylo vytvořit co nejintuitivnější, nepřehlednější prostředí, což je obtížný úkol. Vzhledem k velikosti displeje běžného mobilního zařízení je pracovní plocha pro uživatele velice omezená. Zároveň bylo potřeba dosáhnout co největší možnosti tvorby komplexnějších programů, a to například pomocí větvení programů, smyček, či aritmeticko-logických instrukcí. Jako nejvýhodnější řešení se ukázalo využití existujících platforem pro vytváření vizuálních programů, které všechny požadované funkce nabízí.

3.3.2 Výběr platformy vizuálního editoru programů

Hlavním požadavkem byla možnost tvorby vlastních programových instrukcí, export zhotovených programů ve formě kódu a možnost integrace prostředí editoru do rozhraní aplikace a samozřejmě licence umožňující volné využití. Prvním kandidátem byla platforma Scratch Blocks. Ta však cílí spíše na edukativní stránku vizuálního programování a je tvořena spíše pro malé děti, než pro mírně pokročilejší uživatele, kteří by ji využívali pro práci a skutečnou tvorbu. Navíc využívá pro běh programů vlastní virtuální stroj a neumožňuje tedy generovat z vizuálního programu použitelný kód. Navíc je projekt její možnosti integrace do Android aplikace teprve v nezralém stádiu, mohlo by tedy toto řešení s sebou přinášet problémy při implementaci.



Obrázek 3.7: Vývojové prostředí Blockly integrované v Android aplikaci.

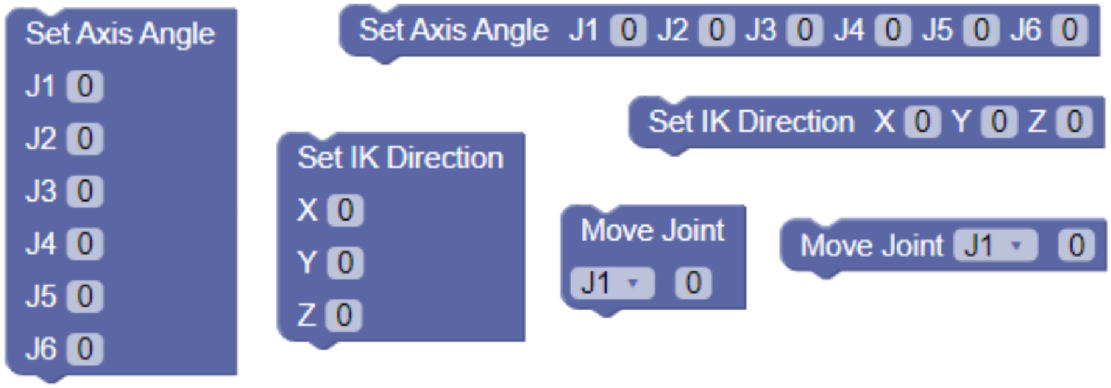
Jako lepší alternativou se ukázala být platforma **Blockly**, která je dostatečně rozvinutá a obsáhle dokumentovaná. Je zde umožněna tvorba vlastních instrukcí (bloků) a jednodu-

chá generace kódu z výsledného programu. Navíc je volně dostupná oficiálně podporovaná implementace webového Blockly rozhraní integrovaného do Android aplikace skrze `WebView` třídu, což výrazně zjednodušší postup integrace do vlastní aplikace.

Jak můžeme vidět na obrázku 3.7, na levé straně okna se potom nachází postranní panel, který obsahuje výběr typů jednotlivých programovacích bloků, které může uživatel přidávat do samotného pracovního prostoru. Dále se zde v dolním pravém rohu nachází ovládací prvky pro orientaci v pracovním prostoru (přiblížení, oddálení, vycentrování) a značka koše, která slouží k odstraňování nechtěných bloků z pracovního prostoru.

3.3.3 Návrh instrukcí robota

Platforma Blockly nabízí většinu potřebných instrukcí pro aritmeticko-logické operace i pro řízení toku programu. Tuto sadu bylo nutné doplnit o instrukce pro samotné robotické rameno, tj. ovládání jeho pohybu. Původní návrh instrukcí pro pohyb ramene je ukázán na obrázku 3.8. Můžeme na něm vidět horizontální, či vertikální bloky představující instrukce pro nastavování úhlů jednotlivých kloubů (`Set Axis Angle`, či `Move Joint`) a instrukci pro pohyb koncového efektoru ramene na určité souřadnice pomocí inverzní kinematiky (`Set IK Direction`). Dosazování operátorů je v tomto případě řešeno staticky určenou formou zadávacích políček a výběrů ze seznamů.

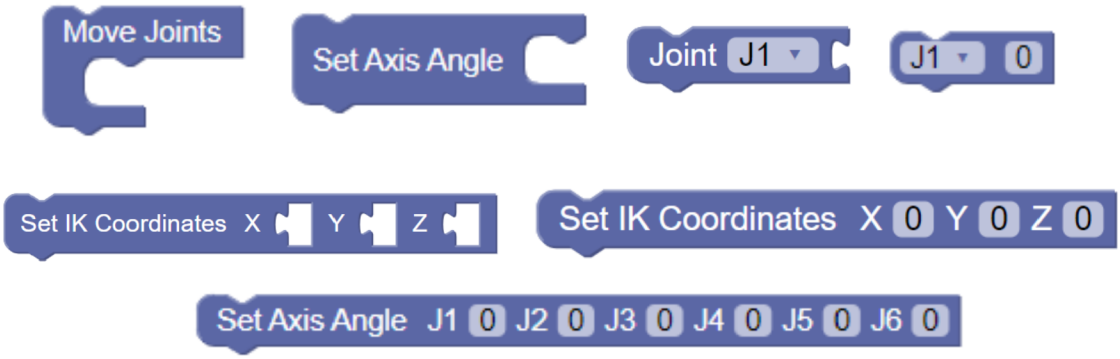


Obrázek 3.8: První návrh instrukčních bloků pro pohyb robotického ramene.

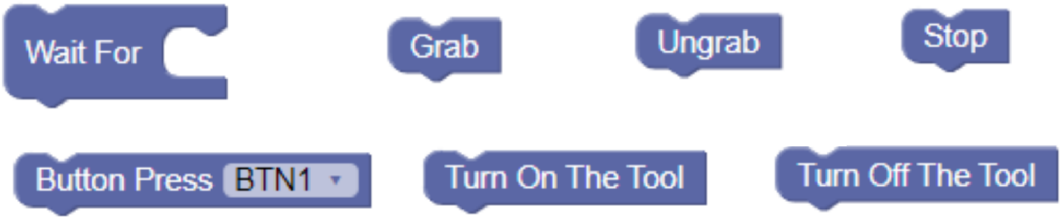
Bylo však třeba dosáhnout větší modularity při práci s těmito instrukcemi. Například zasazování proměnných jako operandů instrukcí, či proměnlivý počet operandů. Tím bude uživateli zajištěna opravdová svoboda ve tvorbě programů pro robota. Přišla proto na řadu druhá verze návrhu, kterou můžeme vidět na obrázku 3.9.

Nyní je možné do bloků instrukcí na pohyb vkládat operandy v libovolném množství. Operand pro určitý kloub je přidán jako samostatný blok (`Joint`), kterému lze určit číselnou hodnotu úhlu do zadávacího pole, či ve formě proměnné (stejně jako u instrukce pro použití inverzní kinematiky). Instrukce pro zadání všech úhlů kloubů ramene do vstupních polí zůstala zachována. Bude totiž používána pro použití v případě, kdy nastavujeme hodnoty operandů této instrukce čtením z aktuálních hodnot ohybu kloubů ramene.

Na obrázku 3.10 můžeme vidět příklady návrhu ostatních instrukcí pro ovládání ramene, jako je například zastavení jeho pohybu (`Stop`), ovládání jeho nástrojů (`Ungrab`, `Grab`, atd.), či vyčkávání na určitou událost (`Wait For`) a podpora vstupu (`Button Press`).



Obrázek 3.9: Druhý návrh instrukčních bloků pro pohyb robotického ramene.



Obrázek 3.10: Návrh instrukčních bloků pro jiná ovládání robotického ramene.

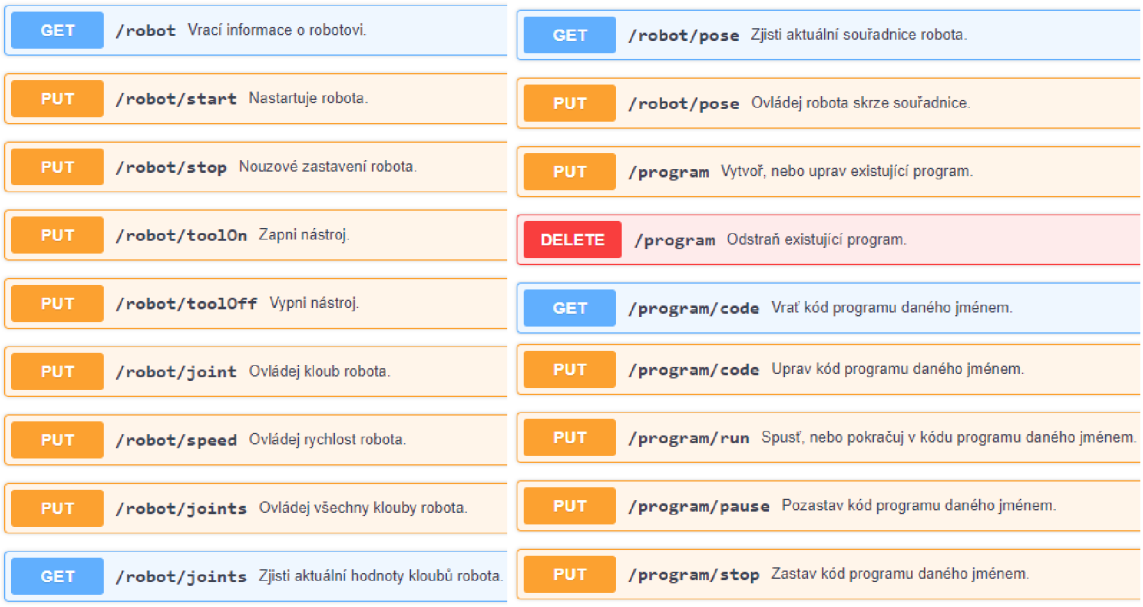
3.4 Komunikace se serverem

Jak už bylo řečeno, pro komunikaci aplikace se serverem bude využito rozhraní fungující na HTTP protokolu, inspirované REST[8] metodikou. Pro jeho návrh byla použita specifikace pomocí OpenAPI dokumentu. OpenAPI[9] je specifikace REST aplikačního programového rozhraní, která umožňuje jednotně vytvářet a upravovat jeho požadavky, přenášená data, cesty, apod. Vytvořený dokument ve formátu YAML[10] (který je přiložen ke zdrojovým kódům této práce jako soubor `openapi.yaml`) potom funguje jako vstup do vizualizačního programu, který graficky přehledně znázorní jednotlivé elementy komunikačního rozhraní, či do generátoru kódu, který vygeneruje jak klientskou knihovnu pro volání serveru, tak samotnou implementaci serveru, do které potom stačí jen doplnit programovou logiku pro jednotlivá volání. Konkrétně byly pro úkoly návrhu a generace kódu použity nástroje Swagger[11], které nabízí všechnu potřebnou funkcionalitu a jsou volně dostupné.

Celé aplikační rozhraní by se dalo rozdělit na dvě poloviny. Ta první se stará o práci s robotem jako takovým. Její jednotlivé zasílané zprávy, které můžeme vidět ve formě HTTP požadavků na obrázku 3.11, jsou v celku přímočaré.

Každý požadavek vyšle na server informaci o instrukci, kterou má robot provést. Jde například o nastavení jednotlivých, či všech kloubů ramene, zastavení robota, nastavení

[8] https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
[9] https://spec.openapis.org/oas/v3.1.0
[10] https://yaml.org/
[11] https://swagger.io/



Obrázek 3.11: Vizualizovaný návrh aplikačního rozhraní pro práci s robotem a jeho programy v editoru Swagger.

rychlosti, či ovládání pomocí inverzní kinematiky. Dále jsou tu i požadavky na získání informací o robotu a jeho aktuálním stavu, či aktuální poloze jeho kloubů.

Druhá část se týká práce s programy robotického ramene. Zde se nachází jednoduché požadavky pro operace vytváření, mazání a úpravu programových položek a jejich kódu. Dále také požadavky pro ovládání běhu programu, tudíž spouštění, pozastavování a zastavení.

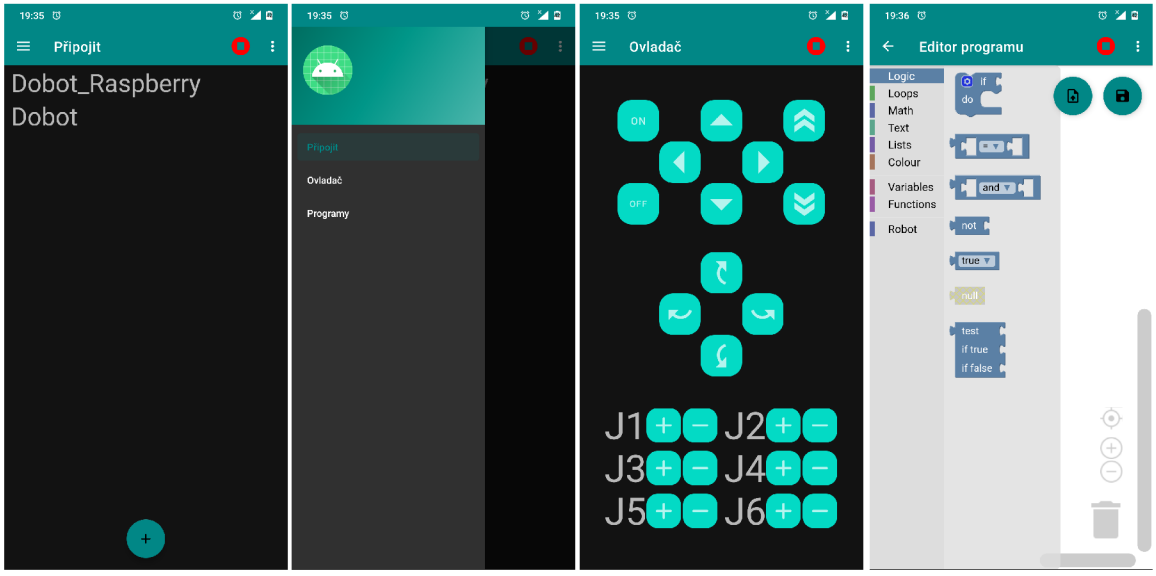
Kapitola 4

Implementace

Tato kapitola se zabývá způsobem implementace celého systému ovládání robotického ramene, tudíž hlavně důležitými detaily implementace jeho jednotlivých částí. První část se zabývá mobilní aplikací pro uživatele, se kterou pracuje. Druhá část popisuje rozhraní mezi robotickým ramenem a jeho ovladači a mobilní aplikací.

4.1 Mobilní aplikace

Aplikace pro Android určená k uživatelskému ovládání robotického ramene je implementována v rámci jedné Android aktivity, ve které probíhá pouze k přepínání fragmentů s veškerými částmi uživatelského prostředí. Některé z nich můžeme vidět na obrázku 4.1.



Obrázek 4.1: Snímky obrazovky uživatelského prostředí aplikace.

Jako implementační prostředí bylo zvoleno Android Studio[1] a jako programovací jazyk převážně Kotlin[2].

[1] https://developer.android.com/studio
[2] https://kotlinlang.org/

4.1.1 Aktivita

Uživatel se po celý životní cyklus běhu aplikace nachází v jedné stejné aktivitě. Atributy, metody a ovládací prvky uživatelského prostředí spjaté s instancí třídy této aktivity jménem `MainActivity` jsou tudíž dostupné během celého jejího životního cyklu. Jedná se například o atributy uchovávající informace o připojení k robotickému rameni, či pomocné atributy k přenášení dat mezi fragmenty. Prvky uživatelského prostředí, které aktivita uživateli neustále nabízí je vrchní lišta aplikace, která obsahuje tlačítko určené k nouzovému zastavení pohybu robota, či tlačítko pro otevření menu, které nabízí tlačítko pro navigaci do fragmentu uživatelského nastavení. Dále se zde nachází tlačítko ke zpětné navigaci mezi fragmenty, nebo tlačítko k vysunutí bočního ovládacího panelu (záleží na tom, v jakém fragmentu se uživatel nachází). Tento boční výsuvný panel, který můžeme vidět na obrázku 4.1 na druhém snímku obrazovky, obsahuje kromě záhlaví také položky navigačního menu, jejichž volbou se uživatel může přesouvat mezi hlavními fragmenty programu.

4.1.2 Fragmenty uživatelského prostředí

Jak již bylo zmíněno, uživatelské prostředí aplikace se dělí na fragmenty, mezi kterými uživatel naviguje jednak pomocí již zmíněného bočního výsuvného panelu, ale i pomocí různých akcí, které v aplikaci provádí. Patří mezi ně triviální součásti aplikace, jako je například sekce uživatelského nastavení, či přidávání, odebírání a upravování položek seznamů (například seznam robotických ramen k připojení, či seznam uložených programů). Tato podkapitola se však zaměřuje na výčet nejstěžejnějších fragmentů aplikace, které jsou jádrem její funkčnosti. Zároveň postupným popisem fragmentů dojde k průchodu běžného použití aplikace. Každý fragment je ve zdrojovém kódu reprezentován svojí třídou, a popisem rozložení jednotlivých prvků (takzvaný „layout“) v jazyce XML[3].

Připojení k robotovi

Po spuštění aplikace se uživatel dostane do fragmentu prostředí, který mu zobrazuje seznam připojení k robotům, který si sám vytváří a upravuje. Konkrétně se jedná o první snímek obrazovky na obrázku 4.1. Tento fragment je reprezentován třídou `HomeFragment`. Každá položka seznamu představuje jméno robotického ramene, které si uživatel zvolí, a IP adresu s portem serveru, na který se aplikace připojí. Tyto položky jsou ukládány v systému Android do souboru sdílených nastavení skrze rozhraní jménem `SharedPreferences`, které systém poskytuje aplikacím k ukládání persistentních dat ve formátu „klíč -> položka“. Pro tento účel je tento způsob dat jednoduchý a dostačující. Aplikace nenabízí uživateli seznam dostupných připojení, jelikož jak už bylo zmíněno v kapitole o návrhu, spojení se serverem není udržované. Klient pošle serveru požadavek protokolem HTTP[4] a pokud je to možné, server odpoví. Na toto je nutné myslet při celé implementaci aplikace. Po volbě robota ze seznamu se aplikace pokusí kontaktovat server na dané adrese. Pokud se jí to podaří, uloží si doručené informace o robotovi (jméno, seznam programů, ..) a přesune uživatele do fragmentu ovladače robota.

[3] https://www.w3schools.com/xml/xml_whatis.asp
[4] https://developer.mozilla.org/en-US/docs/Web/HTTP

Ovládání robota v reálném čase

Po úspěšném připojení se uživatel nachází ve fragmentu aplikace reprezentovaného třídou `ControllerFragment`. Tato část uživatelského prostředí, kterou můžeme vidět na předposledním snímku obrazovky obrázku 4.1, obsahuje sadu tlačítek, pomocí kterých může uživatel v reálném čase ovládat pohyb robotického ramene. To se děje formou zadávání instrukcí pohybu v prostoru kartézských souřadnic (k čemuž slouží tlačítka se šipkami v horní části fragmentu) a orientace koncového efektoru ramene (tlačítka uprostřed), kdy dochází k zadávání náklonu v ose X a Y. Dále je možno robotické rameno ovládat formou nastavování ohybu jednotlivých kloubů ramene (spodní část fragmentu) Je zde možno nastavovat maximálně 6 kloubů, jelikož běžní „domácí“ roboti jich více nemají. Aplikace je však snadno rozšiřitelná na větší počet kloubů. Po zmáčknutí každého tlačítka dojde k odeslání požadavku na server, který specifikuje jakým způsobem, směrem a o jak velký krok se má robot pohnout. Velikost tohoto kroku a rychlost pohybu robota si uživatel nastavuje v sekci nastavení aplikace.

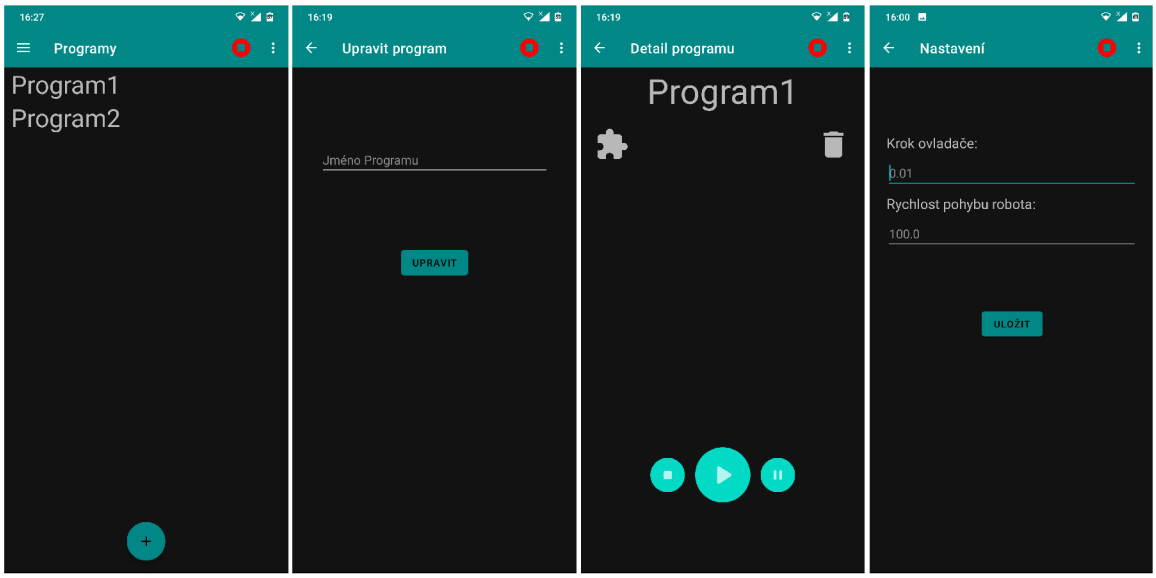
Komunikace aplikace a serveru, jejíž implementace byla při návrhu plánována formou generace kódu pomocí OpenAPI popisu komunikačního rozhraní, byla nakonec implementována ručně, což se ukázalo jako jednodušší řešení. Zasílání požadavků je tedy v rámci aplikace implementováno pomocí dvou knihoven jazyka Kotlin. První je knihovna `Volley`[5], která je použita pro téměř všechny požadavky, které aplikace zasílá. Tato knihovna implementuje zasílání požadavků asynchronně pomocí vláken, přičemž po každém vytvoření požadavek vloží do fronty, ze kterého jsou následně odesílány. Tímto způsobem je implementována i funkce tlačítek v tomto fragmentu. Jejich stisknutí tedy nijak neblokuje běh vlákna uživatelského prostředí aplikace a je vždy připravena přijímat vstup od uživatele. Druhá knihovna je použita z důvodu potřeby blokujícího zaslání požadavku. Tuto funkci tato knihovna také nabízí, avšak pouze pomocí ovládání běhu vláken, což by činilo běh programu i hledání chyb značně složitější. Této druhé knihovně se věnuje podkapitola o úpravě programů, která popisuje fragment, v němž je použita.

Spouštění programů

Další částí aplikace je vytváření a spouštění programů, které automatizují zasílání instrukcí pro robota. K této funkci se uživatel dostane po zvolení položky „Programy“ v bočním navigačním panelu. To ho přesune do fragmentu se seznamem programů, ve kterém může uživatel vybírat jednotlivé programy, vytvářet nové, či upravovat jejich název. Tento fragment představuje třída `ProgramFragment`. Prostředí tohoto fragmentu můžeme vidět na prvním snímku obrázku 4.2. Po klepnutí na položku programu v seznamu se uživatel přesune na další fragment, který obsahuje ovládací prvky programu a jeho běhu.

Toto prostředí můžeme vidět na třetím snímku obrázku 4.2. Spodní část fragmentu, který je reprezentován třídou `ProgramRunningFragment`, obsahuje tlačítka pro spouštění, pozastavování a zastavování běhu programu. Opět jde čistě o zaslání určitého požadavku na server. Vrchní část fragmentu obsahuje (mimo nadpisu s názvem programu) tlačítko pro odstranění programu a tlačítko pro jeho úpravu. Informace o jednotlivých programech si aplikace udržuje lokálně, avšak při každé změně názvu, přidání či odstranění programu si tyto informace aktualizuje ze serveru, na kterém je kód programu uložen. Tyto informace jsou společně s informacemi o robotovi uloženy v jednom JSON[6] objektu, který server po

[5] https://developer.android.com/training/volley
[6] https://www.json.org/json-en.html



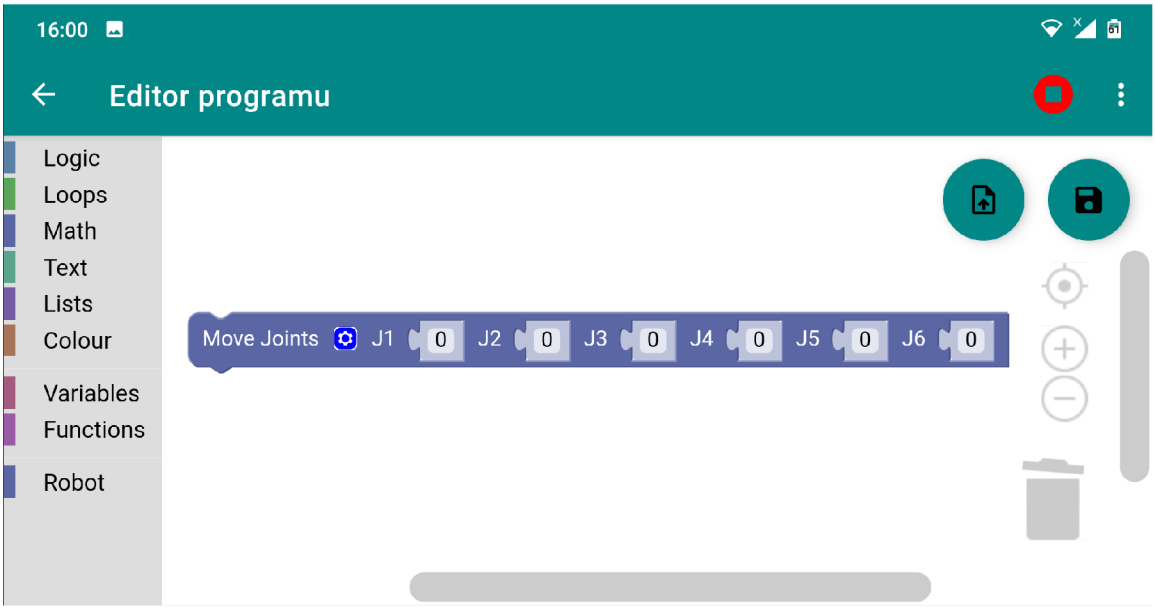
Obrázek 4.2: Další snímky obrazovky uživatelského prostředí aplikace. (Zleva: výběr programů, úprava jména programu, ovládání běhu programu a nakonec nastavení aplikace.

požádání posílá a při změnách si ho sám aktualizuje. Při stisku tlačítka pro úpravu programu, které se nachází na vrchní levé straně, se uživatel přemístí do posledního důležitého fragmentu aplikace.

Upravování programů

Jak již bylo zmíněno, programy samotné jsou ukládány trvale na serveru. Každý program je reprezentován samotným kódem, který je na serveru při spuštění programu vykonáván, a také XML popisem jednotlivých bloků vizuálního programovacího jazyka Blockly, který je pro vytváření programů v aplikaci použit. Integrace tohoto vývojového prostředí do aplikace je poměrně přímočará, jelikož tvůrci poskytují její volně dostupnou implementaci pomocí takzvaného „WebView“ fragmentu, který umožňuje v aplikaci zobrazovat webový obsah napsaný v jazyku HTML a JavaScript, ve kterém je uživatelské rozhraní Blockly implementováno. Z hlediska implementace v aplikaci se jedná o zasazení WebView fragmentu a definice vlastních bloků pro instrukce robota v JavaScriptu. Tento fragment zastupuje třída `BlocklyFragment`. Návrh těchto bloků je umožněn jednoduchou cestou skrze webový nástroj[7] od tvůrců Blockly. Ten umožňuje definovat jak samotné bloky, tak i postranní panel ve formátu XML, takzvaný „toolbox“, ze kterého uživatel nové bloky vybírá. Při definici tohoto panelu je možné vytvářet takzvané stínové bloky, které slouží jako výchozí bloky napojené na blok jiný (v našem případě pro blok proměnné hodnoty argumentů instrukcí pro pohyb robota), do kterých je možno zadávat vstup přímo, či je překrýt jiným blokem, například proměnnou. Tato funkce, jejíž podobu můžeme vidět na obrázku 4.3, umožňuje zuniverzálnit zadávání argumentů do bloků instrukcí robota a díky tomu snížit počet jejich variací. Následně už zbývá jen doplnit kód v zadaném jazyce, na který generátor bloky převede. V našem případě Python.

[7] https://blockly-demo.appspot.com/static/demos/blockfactory/index.html



Obrázek 4.3: Snímek prostředí Blockly v aplikaci s vloženým blokem obsahujícím stínové bloky jako argumenty instrukce.

Uživatel tedy skládáním bloků dohromady upravuje program, který načítá ze serveru a ukládá ho na něj. Této funkce je dosaženo za pomocí dvou plovoucích tlačítek v levém horním rohu obrazovky. Při stisku tlačítka pro načtení programu se odešle požadavek na server se jménem programu, na který server odpoví zasláním XML reprezentace seskládaných bloků programu, které si Blockly následně přeloží na bloky a zobrazí je. Při uložení dojde k přeložení sestavených bloků zpět do jejich XML reprezentace a zároveň se pomocí generátoru vytvoří kód programu v jazyce Python. Oba tyto kódy se odešlou zpět na server, kde se uloží.

Této komunikace mezi prostředí JavaScriptu v rámci WebView fragmentu a vnějšího prostředí kódu fragmentu aplikace v jazyku Kotlin je dosaženo pomocí rozhraní `JavaScriptInterface`. Toto rozhraní po jeho přidání na WebView fragment umožňuje vytvořit třídu, ve které jsou definovány metody, které lze v kódu JavaScriptu v rámci WebView volat a získávat z nich návratové hodnoty. V našem případě se jedná o třídu `WebInterface`, v jejíž metodách dochází k odesílání požadavků na server na ukládání, či načítání programů. Právě v těchto metodách je použita druhá knihovna jménem `OkHttp`[8], která umožňuje jednoduše zasílat blokující HTTP požadavky, které jsou zde kvůli tomu, že je data kódu programu z odpovědi požadavku nejprve potřeba získat a až pak je vrátit z metody zpět do kódu JavaScriptu.

Poslední metodou je funkce `getJoints()`, která je volána po stisknutí malého tlačítka s ozubeným kolečkem (můžeme ho také vidět na obrázku 4.3) na bloku pro nastavování všech kloubů robota zároveň. Tato funkce načte ze serveru aktuální hodnoty ohybu kloubů robota, které jsou následně vloženy do jednotlivých stínových bloků argumentů kloubů. Pomocí této funkce je možné jednoduše programovat pohyby robota tak, aby procházel cestu, kterou mu uživatel ručně postupně nastaví.

[8] https://square.github.io/okhttp/

Konečný výčet vlastních bloků s instrukcemi pro robota je tedy následující:

- **Move Joint** - blok pro nastavení ohybu specifického kloubu, který uživatel zvolí
- **Move Joints** - blok pro nastavení všech kloubů zároveň s tlačítkem pro načtení aktuálních hodnot kloubů ramene
- **Set Coordinates** - blok pro nastavení pozice a orientace koncového efektoru
- **Wait for button press** - blok, který pozastaví činnost programu, dokud nedojde k sepnutí tlačítka na určitém GPIO portu počítače Raspberry Pi
- **Turn On The Tool** - blok, který zapne činnost nástroje na robotickém rameni
- **Turn Off The Tool** - blok, který vypne činnost nástroje na robotickém rameni

4.2 Server na Raspberry Pi

Tato část kapitoly popisuje server vytvořený v jazyku Python, které slouží jako prostředník mezi robotem a uživatelskou aplikací. Zde je vhodné zmínit jeden z hlavních rysů tohoto systému, což je fakt, že server i klientská aplikace a komunikace mezi nimi jsou implementovány tak, aby byly co nejméně závislé na typu připojeného robota. Toho je dosaženo hlavně rozdělením aplikační logiky serveru na dvě části. Tou první je samotný server, který zpracovává požadavky zasílané klientskou mobilní aplikací. Jeho implementace je obsažena v souboru `server.py`. Druhou částí je třída `Robot`, jejíž metody server při ovládání robota volá. Samotná logika instrukcí robota je obsažena až v těchto metodách, což výrazně usnadňuje modularitu systému. Systém je tedy navržen tak, aby stačilo pouze změnit definice metod třídy `Robot`, pokud by vývojář chtěl implementaci přizpůsobit jinému robotickému rameni.

4.2.1 Python server

Pro implementaci serveru byl podle návrhu zvolen Flask. Tento aplikační rámec umožňuje velmi jednoduchou implementaci logiky serveru při přijmutí požadavku na určitý koncový bod. Konkrétní implementace probíhá tak, že na začátku skriptu je do proměnné uložena instance třídy `Flask` a následně je pro každý koncový bod, na který server odpovídá, implementována funkce s dekorátorem, kterému předáme jako parametry název koncového bodu a seznam přijímaných HTTP metod. V ukázce kódu níže můžeme vidět příklad.

```
api = Flask(__name__)

@api.route('/robot/pose', methods=['GET', 'PUT'])
def robot_pose():
    #implementace logiky
    return data_odpovedi
```

V jednotlivých funkcích jsou tedy implementovány příkazy pro přímé volání rozhraní `Robot`, či různou práci s programy, které uživatel vytváří.

Spouštění programů

Zajímavou částí implementace serveru je jeho práce s uživatelskými programy pro robota. Kromě koncových bodů a funkcí pro přidávání, mazání a úpravu programů, které si server

ukládá v podobě textových souborů, se zde nachází koncové body pro ovládání průběhu programů, respektive na jejich spuštění, pozastavení, či úplné zastavení.

Pro spouštění programů využívá server Python modul `subprocess`, který umožňuje programově vytvářet nové procesy. Při spuštění programu se tedy spustí nový proces, ve kterém se spustí Python skript s daným programem robota. Pozastavování a opětovné spouštění procesu probíhá pomocí zasílání UNIX řídících signálů `SIGSTOP` a `SIGCONT`. Aby nedocházelo k různým chybám, jako je například několikanásobný běh procesů programu, je třeba, aby si server udržoval informace o tom, jestli některý program právě běží. K tomu slouží globální proměnná `robot_info`. Jedná se o slovník, který kromě informací o robotovi a seznamu dostupných programů, které zasílá i klientské aplikaci, obsahuje i stavovou proměnnou uchovanou pod klíčem `running`. Tato proměnná uchovává řetězcový literál. Pokud je řetězec prázdný, znaměná to, že žádný program právě neběží. Pokud některý program běží nebo je pozastaven, obsahuje řetězec název daného programu s příponou `"_playing"`, či `"_paused"`. Při každém řízení programu si server tuto proměnnou ověřuje a po provedení zadaného úkonu ji také upravuje, aby reflektovala aktuální stav běhu programu.

Dále je také potřeba, aby byl server informován, když se spuštěný proces dokončí. Proto je úkon spouštění tohoto procesu prováděn ve funkci `runInThread()`, která je spouštěna v novém vlákně, aby neblokovala hlavní činnost serveru. V této funkci je po spuštění procesu zavolána funkce `wait()`, která blokuje běh programu, dokud se jeho proces neukončí. Když program dokončí svoji práci a proces se sám ukončí, může program upravit stavovou proměnnou.

4.2.2 Ovládání robota

Jak již bylo zmíněno, ovládání robota probíhá skrz oddělené rozhraní `Robot`, jehož metody server volá. Pro ukázkovou implementaci systému se jako nejvhodnější ukázalo robotické rameno Dobot Magician. K jeho ovládání je použito HTTP aplikační programové rozhraní od vývojové skupiny pod FIT VUT, které je volně dostupné[9]. Toto rozhraní poskytuje i pokročilé funkce, jako je například výpočet dopředné a inverzní kinematiky. Komunikace s tímto rozhraním probíhá zasíláním HTTP požadavků na jeho koncové body.

Zasílání požadavků je v tomto případě implementováno pomocí Python modulu `requests`. Tyto požadavky jsou blokující a například při příkazu pro pohyb ramene rozhraní odpovídá, až když je pohyb dokončen. Díky tomu není třeba v implementaci ovladače robota na serveru nijak řešit časování postupného zadávání příkazů robotovi. Kvůli tomu, že robot Dobot Magician nabízí pouze 4 osy volnosti pohybu, je koncový efektor vždy směrován dolů a je možné otáčet ho pouze kolem vlastní osy. V takovém případě tedy systém na ovládání páté a šesté osy pohybu nijak nereaguje.

[9] https://github.com/robofit/arcor2/

4.3 Testování

Tato sekce pojednává o testování výsledného produktu. Pro testování aplikace a systému pro ovládání robota jsem zvolil jednoduchou testovací proceduru, při které měl potenciální uživatel systému po krátkém seznámení se s aplikací za úkol sestavit jednoduchý program typu PickPlace, při kterém robot zvedne objekt pomocí přísavky z podložky a přemístí ho na jiné místo. Následně podává zpětnou vazbu k aplikaci a práci s ní.

Následně bude pomocí rozhovoru se subjektem získána zpětná vazba a ohodnotí se funkčnost produktu. Jako testovací subjekt byl vybrán mladý člověk, který se vzdělává a pracuje v technickém oboru automatizace, tudíž koncepty jako ovládání robotických ramen mu nejsou cizí. Byl vybrán proto, že právě takový typ člověka by v praxi produkt tohoto typu mohl využívat.

4.3.1 Výsledky

Testovací subjekt úkol zvládl v očekávaném čase. Během provádění úkolu nenarazil na žádné problémy, které by mu znemožnily dál pracovat. Při seznamování s prostředím aplikace měl však pár výtek k odlišnosti ovládacích prvků. Například fakt, že po podržení prstem na položce robota je možné připojení na robota odstranit, zatímco stejný úkon při položce programu uživateli umožní upravit jeho jméno. Jako další problém, který uživatel popsal, bylo chování prostředí aplikace při komunikaci se serverem, když při zvětšené prodlevě odpovědi serveru na akci neměl žádnou vizuální odezvu.

Vizuální styl a rozložení uživatelského prostředí hodnotil kladně. Malé problémy nastaly při druhém úkolu, než se subjekt seznámil s koncepty vizuálního programování, avšak po vysvětlení pár úvodních záležitostí se mohl pustit do práce. Při provádění tohoto úkolu ocenil zejména možnost načítání aktuálních hodnot kloubů do instrukce pro jejich nastavení. Hlavně pomocí této funkce program zhotovil.

Celkový dojem z jeho práce a funkce aplikace zhodnotil jako kladný a popsal tak, že podle jeho zkušenosti svůj účel aplikace i přes pár menších nedostatků splňuje.

Kapitola 5

Závěr

Cílem této práce bylo vytvořit mobilní aplikaci, pomocí níž lze ovládat a programovat robotické rameno. Pomocí spolupráce klientské mobilní aplikace a serveru, který robotické rameno skrz rozhraní řídí, lze rameno ovládat pomocí ovladače v reálném čase, vytvářet a upravovat programy pro jeho automatizovanou činnost, a ty lze následně spouštět a ovládat jejich průběh.

V teoretické části práce jsou popsána robotická ramena, jejich kategorizace, způsoby jejich ovládání a současný stav jejich technologií. Dále byl popsán operační systém pro mobilní zařízení Android a aplikace vyvíjené pro tento systém. Naposled je popsáno paradigma vizuálního programování.

Během návrhu byly prozkoumány současná řešení problematiky zadání této práce. Dále byly vytvořeny nákresy uživatelského rozhraní mobilní aplikace, instrukce pro robota společně s jejich grafickými verzemi pro vizuální programování v rozhraní Blockly a bylo navrženo i komunikační rozhraní protokolu HTTP mezi aplikací a serverem.

K implementaci mobilní aplikace bylo využito vývojové prostředí Android Studio a programovací jazyk Kotlin a Java. Pro vytváření programů pro robota byla využita platforma Blockly, která byla do aplikace integrována. Pro implementaci serveru komunikujícího s aplikací byl využit aplikační rámec Flask a skriptovací jazyk Python.

Výsledný systém byl otestován s pomocí subjektu, který disponuje znalostmi průměrného člověka, který by se systémem tohoto typu v praxi mohl pracovat. Po krátkém seznámení s funkcemi aplikace dostal subjekt za úkol vytvořit program, po jehož spuštění robot provede pár jednoduchých úkonů. Testování potvrdilo, že systém splňuje funkce, které jsou požadované. Zároveň taky odhalilo několik nedostatků zejména uživatelského prostředí aplikace.

Výsledný systém bude v jeho dalších verzích doplněn o několik dalších funkcí. Například doplnění grafického prostředí aplikace o animace, které by lépe reflektovaly aktuální stav systému a informovaly o něm uživatele. Dále doplnění nápovědy, která by uživatele provedla funkcemi systému. Hlavní věc, o kterou se dá systém dále rozšířit je kompatibilita s dalšími typy robotických ramen a jednoduché přepínání mezi jejich ovládáním.

Literatura

[1] *Android API levels* [online]. Google Inc. [cit. 2021-03-10]. Dostupné z: https://developer.android.com/guide/topics/manifest/uses-sdk-element.

[2] *Application Fundamentals* [online]. Google Inc. [cit. 2021-03-10]. Dostupné z: https://developer.android.com/guide/components/fundamentals.

[3] *Cobot To-Do List: Cheaper to Buy, Easier to Use* [online]. Asian Robotics Review [cit. 2021-01-24]. Dostupné z: https://asianroboticsreview.com/home274-html.

[4] *DOBOT Magician* [online]. Dobot.cc [cit. 2021-03-15]. Dostupné z: https://www.dobot.cc/dobot-magician/product-overview.html.

[5] *FAQ* [online]. Open Handset Alliance [cit. 2021-03-10]. Dostupné z: http://www.openhandsetalliance.com/oha_faq.html.

[6] *Jogging the Robot* [online]. Motion Controls Robotics [cit. 2021-01-24]. Dostupné z: https://motioncontrolsrobotics.com/jogging-the-robot/.

[7] *The maturity of visual programming* [online]. Craft ai [cit. 2021-03-11]. Dostupné z: https://www.craft.ai/blog/the-maturity-of-visual-programming.

[8] *Niryo One* [online]. Niryo [cit. 2021-03-15]. Dostupné z: https://niryo.com/product/niryo-one/.

[9] Industrial Robots and Robot System Safety. *Occupational Safety and Health Administration* [online]. US Department of labor [cit. 2021-01-21]. Dostupné z: https://www.osha.gov/dts/osta/otm/otm_iv/otm_iv_4.html.

[10] *Teach pendant* [online]. The Computer Language Co Inc. [cit. 2021-01-26]. Dostupné z: https://www.pcmag.com/encyclopedia/term/teach-pendant.

[11] *What is a Stepper Motor : Types Its Working* [online]. Elprocus [cit. 2021-01-24]. Dostupné z: https://www.elprocus.com/stepper-motor-types-advantages-applications/.

[12] *I, Cobot: Future collaboration of man and machine* [online]. The Manufacturer, 12. listopadu 2015 [cit. 2021-01-22]. Dostupné z: https://www.themanufacturer.com/articles/i-cobot-future-collaboration-of-man-and-machine/.

[13] *Demystifying Collaborative Industrial Robots* [online]. International Federation of Robotics, prosinec 2018 [cit. 2021-01-22]. Dostupné z: https://ifr.org/downloads/papers/IFR_Demystifying_Collaborative_Robots.pdf.

[14] *What Kinds of Industrial Robots Are There?* [online]. Kawasaki Heavy Industries, 4. října 2018 [cit. 2021-01-21]. Dostupné z: https://robotics.kawasaki.com/ja1/xyz/en/1803-01/.

[15] *Types of Robots* [online]. ROVer Ranch, 2021 [cit. 2021-01-21]. Dostupné z: https://prime.jsc.nasa.gov/ROV/types.html.

[16] Burnett, M. a Scaffidi, C. *The Encyclopedia of Human-Computer Interaction: End-User Development* [online]. Interaction Design Foundation [cit. 2021-03-11]. Dostupné z: https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/end-user-development.

[17] Colgate, J. E. a Peshkin, M. *Cobots.* U.S. Patent 5 952 796, 14. 10. 1999 [cit. 2021-01-22].

[18] Frank, J. *Lekce 8 - Android programování - Životní cyklus aktivity* [online]. ITnetwork.cz [cit. 2021-03-29]. Dostupné z: https://www.itnetwork.cz/java/android/zaklady/tutorial-programovani-pro-android-v-jave-zivotni-cyklus-a-novy-projekt.

[19] Gargenta, M. *Learning Android: Building Applications for the Android Market.* O'Reilly Media, březen 2011 [cit. 2021-03-10]. ISBN 978-1-449-39050-1.

[20] Ghiani, G., Manca, M., Paternò, F. a Santoro, C. *Personalization of Context-Dependent Applications Through Trigger-Action Rules* [online]. 2017 [cit. 2021-03-11]. DOI: 10.1145/3057861.

[21] Hill, A. *Finally! A Real Robot Programming Solution for Mobile* [online]. RoboDK Inc., 2020 [cit. 2021-03-15]. Dostupné z: https://robodk.com/blog/mobile-robot-simulation/.

[22] Hornsby, P. *Empowering Users to Create Their Own Software* [online]. UXmatters [cit. 2021-03-11]. Dostupné z: https://www.uxmatters.com/mt/archives/2009/08/empowering-users-to-create-their-own-software.php.

[23] Johnston, W., Hanna, P. a Millar, R. *Advances in dataflow programming languages* [online]. 2004 [cit. 2021-03-11]. DOI: 10.1145/1013208.1013209.

[24] Kucuk, S. a Bingul, Z. Robot Kinematics: Forward and Inverse Kinematics. In: Cubero, S., ed. *Industrial Robotics.* IntechOpen, 2006, kap. 4 [cit. 2021-01-26]. DOI: 10.5772/5015.

[25] Lacko Luboslav. *Vývoj aplikací pro Android.* Computer Press, 2015 [cit. 2021-03-10]. ISBN 978-80-251-4347-6.

[26] Monkman, G. J., Hesse, S., Steinmann, R. a Schunk, H. *Robot Grippers.* Wiley, říjen 2006 [cit. 2021-01-21]. ISBN 9783527406197.

[27] Paternò, F. *End User Development: Survey of an Emerging Field for Empowering People* [online]. 2013 [cit. 2021-03-11]. DOI: 10.1155/2013/532659.

[28] Repenning, A. *Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets* [online]. 2017 [cit. 2021-03-11]. DOI: 10.18293/vlss2017-010.

[29] Reuleaux, F. *The Kinematics of Machinery: Outlines of a Theory of Machines.* Alexander Blackie William Kennedy. Macmillan, 1876 [cit. 2021-01-21]. ISBN 0598975373.

[30] Singh, P. *Android OS Architecture* [online]. Techplayon [cit. 2021-03-10]. Dostupné z: http://www.techplayon.com/android-os-architecture/.

Příloha A

Obsah přiloženého paměťového média

- `py` - složka se soubory a skripty potřebnými pro běh serveru
- `py/info.json` - soubor pro ukládání pomocných dat serveru
- `py/server.py` - skript serveru
- `py/robot.py` - skript s rozhraním pro ovládání robota
- `arcor2_dobot.zip` - upravený balíček pro ovládací rozhraní robota na Raspberry Pi
- `MyRobot` - složka se zdrojovými soubory mobilní aplikace pro překlad v prostředí Android Studio
- `MyRobot.apk` - instalační balíček Android aplikace
- `openapi.yaml` - OpenAPI dokument popisující komunikační rozhraní aplikace a serveru
- `video.mp4` - video představující výsledný produkt práce
- `latex` - složka se zdrojovými soubory pro překlad PDF souboru tohoto textu
- `thesis.pdf` - soubor s tímto dokumentem ve formátu PDF
- `README.txt` - textový soubor s návodem k instalaci aplikace a serveru