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

AUTOMATICKÉ TESTOVÁNÍ UŽIVATELSKÝCH ROZHRANÍ

AUTOMATED TESTING OF USER INTERFACES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VOJTĚCH MELUZÍN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL KAPINUS**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Meluzín Vojtěch**
Program: Informační technologie
Název: **Automatické testování uživatelských rozhraní**
Automated Testing of User Interfaces
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s existujícími nástroji pro automatické testování software a uživatelských rozhraní.
2. Seznamte se s tvorbou uživatelských rozhraní v robotickém operačním systému (ROS), dále s nástroji a postupy používanými pro testování kódu.
3. Navrhněte řešení pro automatizované testování uživatelských rozhraní.
4. Realizujte navržené řešení.
5. Ověřte funkčnost řešení na uživatelském rozhraní ARTable.
6. Porovnejte a zhodnoťte dosažené výsledky. Navrhněte vhodné způsoby využití vašeho řešení.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Kapinus Michal, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato práce se zabývá řešením problému automatického testování uživatelského prostředí pro systém ARTable, který je tvořen pomocí frameworku ROS. Zvolený problém jsem vyřešil tvorbou rozšíření pro testovací nástroj. Rozšíření s názvem ARTBot nám umožňuje simulaci uživatelské interakce. Můžeme ho tak spolu s testovacím nástrojem využít k automatickému testování grafického rozhraní ARTable aplikací. V této práci nalezneme navrh, popis implementace a způsob testování ARTBota.

Abstract

The main theme of this thesis is automated testing of graphical user interfaces for the system ARTable, created using the ROS framework. My solution to this problem is an extension for a testing tool. The extension named ARTBot allows simulating user interaction. Its use can be combined with a testing tool to automate testing of garafical interface of ARTable applications. The thesis contains the design, description of the implementation and methods of testing the ARTBot.

Klíčová slova

ROS, ARTable, automatické testování, grafická uživatelská rozhraní, rozšířená realita, pytest, PyQt4

Keywords

ROS, ARTable, automated testing, graphical user interface, augmented reality, pytest, PyQt4

Citace

MELUZÍN, Vojtěch. *Automatické testování uživatelských rozhraní*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Kapinus

Automatické testování uživatelských rozhraní

Prohlášení

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

.....
Vojtěch Meluzín
15. května 2019

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé práce, Ing. Michalu Kapinusovi, za jeho pomoc, ochotu a věnovaný čas.

Dále bych chtěl poděkovat své rodině a přátelům za podporu při studiu.

Obsah

Kapitola 1

Úvod

V dnešní době je člověk nahrazován roboty stále více a více. Jsou nedílnou součástí ve výrobním prostředí a jejich „přešlapy“ tak mají velký vliv na zisk společností. Příchod rozšířené reality nabízí spoustu nových možností jak roboty ovládat a pracovat s nimi. Dělá je tak přístupnější i pro neodborné pracovníky, což sebou nese, že chyby které obsahují, mají dopad na spoustu lidí. Jako odpověď se nabízí automatické testování.

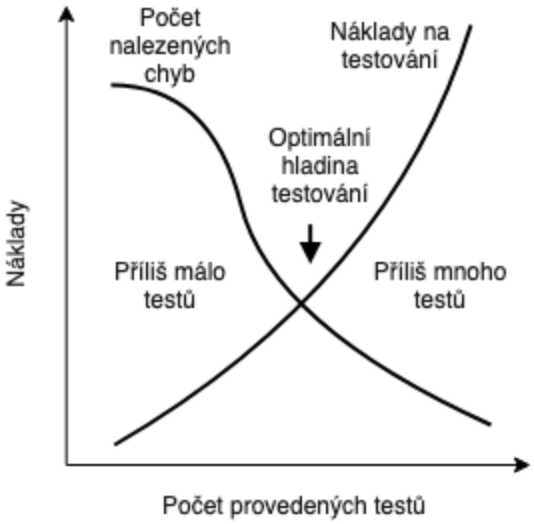
Tomu, jak jedno z takovýchto moderních řešení testovat, se věnuje tato práce. Prostředím určené pro otestování je ARTable, který představuje, určitou vizi budoucnosti ovládání robotů za pomoci projekce na pracovní plochu. Tento systém běží na robotickém operačním systému (zkráceně ROS). Cílem je tak navrhnout řešení automatického testování grafického prostředí rozšířené reality. Výsledkem této práce bude nástroj, který se dá použít k testování grafické části aplikací vytvořených pro ARTable.

V první kapitole se zaměřím na vysvětlení pojmu testování, jaký má pro nás při vývoji softwaru význam, na jaké se dělí druhy a kdy jaký typ použít. Dále popíši prostředí ROS a ARTable, pro které řešení cílí, a podíváme se na podobné projekty. Ukážeme si jakým stylem ARTable zpracovává doteky a jak řeší grafické prostředí v rozšířené realitě. Na základě těchto informací porovnám dostupná řešení pro automatické testování s našimi požadavky a navrhnu vlastní řešení, jehož implementací se budu zabývat. V závěru popíši jakým stylem probíhalo testování nástroje a zhodnotím výsledky celé práce.

Kapitola 2

Testování

Testování je proces, při kterém se ověřuje, zda reálné vlastnosti softwaru odpovídají vlastnostem očekávaným a požadovaným. Je důležitou součástí vývoje software. Jeho účelem je nalezení problémů - softwarových chyb a zjištění kvality. Čím v dřívější fázi vývoje chyby nalezneme, tím jsou náklady na jejich opravu menší. Touto skutečností se řídíme i při výběru stylu testování. Některé chyby z důvodu špatně zadaných požadavků nebo špatné analýzy není možné zachytit testy. Úspěšné splnění testů ještě neznamená bezchybný systém. Při tvorbě testů se snažíme dosáhnout tzv. bodu „Optimální hladiny testování“. Ten nasává když je software dostatečné pokryt testy, ale náklady na tvorbu těchto testů jsou v únosné mezi (viz. graf2.1).[17][1]



Obrázek 2.1: Graf efektivní hladiny testování[1]

[1]Obrázek převzat z[17]

2.1 Kvalita softwaru

K určení kvality softwaru existují normy, které popisují jak a z jakých pohledů hodnotit kvalitu. Příkladem takovéto normy je ISO/IEC 25010[11], která rozděluje kvalitu do osmi dimenzí.[1]

1. **Funkčnost** – Systém funguje správně za specifikovaných podmínek tak, jak je uvedeno ve funkční specifikaci.
2. **Výkon** – Hodnotí schopnost systému zvládat velké množství požadavků (například velké množství uživatelů současně) a jestli není pomalý pomalý, nebo i při splnění všech požadavků současně, si nebere příliš moc systémových zdrojů.
3. **Kompatibilita** – Možnost použití i s jinými programy nebo systémy.
4. **Použitelnost** – Určuje, jak dobře je systém pro uživatele přehledný, použitelný, přívětivý, náročný na učení se postupů k dosažení cíle a v poslední řadě i zda-li lze požadovaného cíle dosáhnout.
5. **Spolehlivost** – Schopnost systému se umět chovat za všech okolností stejně. Stavy které mohou nastat jsou přetížení, chyba nebo výpadek. Spolehlivý systém by měl umět tyto stavy detekovat a ohlásit. Při výpadku je hodnoceno, jak dobře se dokáže systém obnovit.
6. **Bezpečnost** – V jakém rozsahu jsou systém a jeho data zabezpečeny vůči ostatním osobám nebo systémům a jaké jsou úrovně autorizace potřebné k jejich přístupům.
7. **Udržovatelnost** – Jestli je a v jaké míře možné systém efektivně upravovat, rozšiřovat, přizpůsobovat prostředí nebo podmínkám a jak moc velký vliv zmíněné změny mohou na systém mít.
8. **Přenositelnost** – Možnost instalace na jiný operační systém, na jiný hardware nebo schopnost nahradit jiný systém.

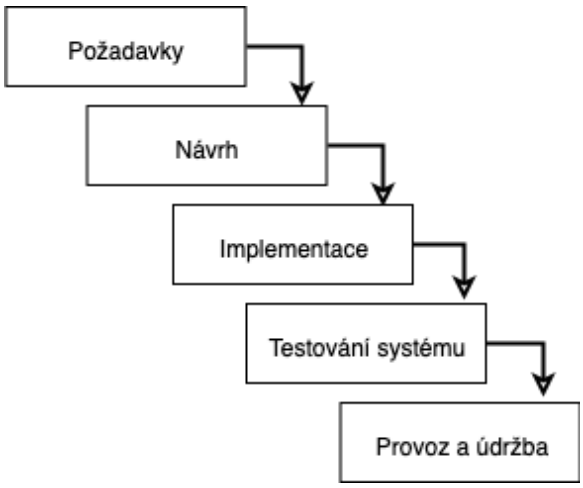
2.2 Modely životního cyklu software

„Model životního cyklu softwaru definuje etapy vývoje softwaru a pro každou etapu dále definuje nutné činnosti a její vstupy a výstupy.“[9]

Pro tvorbu systému znamená volba modelu jedno ze zásadních rozhodnutí. Určuje, jakým stylem budou zpracovány jednotlivé požadavky na funkčnost, zda budou zákazníkovi prezentovány až na konci nebo během vývoje a jak by se reagovalo na případné změny ve specifikaci. Z pohledu testování jsou pro nás tyto modely zajímavé, protože nám na základě etap pomáhají určit, jaký styl testování je vhodné použít.

Vodopádový model

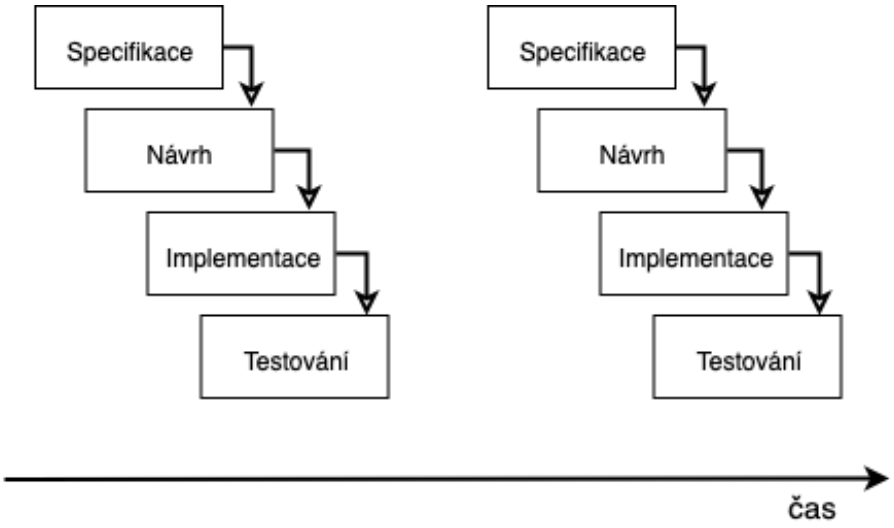
Vodopádový model (obrázek2.2) je základním modelem životního cyklu software. Znázorňuje idealizovaný stav, kdy jsou jednotlivé etapy vývoje řazeny postupně za sebou - následující etapa začíná až po etapě předcházející. Neobsahuje cykly a nelze se v něm vracet na předchozí etapu. Pokud proběhne špatná analýza nebo jsou špatně zadány požadavky zákazníkem, musí se celý vývoj zastavit a začít od začátku.[10][9]



Obrázek 2.2: Vodopádový model životního cyklu software

Iterativní model

Model založený na opakování - iteraci (obrázek2.3). Snaží se o odstranění hlavního nedostatku vodopádového modelu, kterým je postrádání možnosti ukázat zákazníkovi spustitelnou aplikaci během vývoje. Tento nedostatek se snaží vyřešit rozdělením vývoje do iterací. Software tak vzniká postupně s tím, že na konci každé iterace je spustitelná aplikace s neúplnou funkčností. Zákazník se tak stává součástí vývojového týmu.[10][9]



Obrázek 2.3: Iterativní model životního cyklu software

Inkrementální model

Iterativní a inkrementální modely jsou velice podobné. Jediným rozdílem je, že při specifikaci jsou stanoveny části do kterých bude systém rozdělen. Projekt je potom tvořen po částech, které má postupně k dispozici zákazník.[9]

Spirálový model

Model, který je odvozen od vodopádového modelu s dvěma odlišnými vlastnostmi: podrobně analyzuje rizika, která vznikají při vývoji a jednotlivé etapy se opakují. Opakováním připomíná iterativní nebo inkrementální model. Výstupem na konci jednotlivých cyklů není software s omezenou funkčností, ale pouhý prototyp.[10]

Prototyp se vždy po použití zahodí a v dalším cyklu se vytváří nový. Systém je čistější a jednodušší, je to ale na úkor časově náročné.[9]

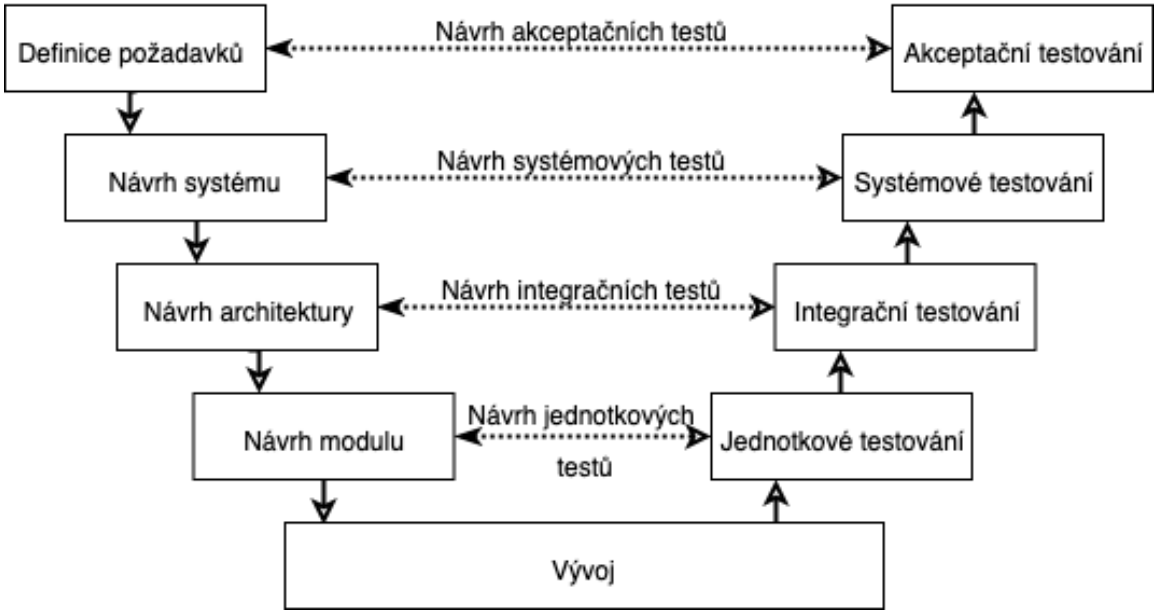
Přejití do další etapy je u spirálového modelu doprovázeno analýzou rizik, určující nám další směřování projektu. Analýza rizik je prováděna vždy na konci každého cyklu. Rizikem v případě vývoje může být jakákoliv událost nebo situace, která by mohla nějak ovlivnit projekt.[10] Spirálový model je zobrazen na obrázku2.4.

Obrázek 2.4: Spirálový model životního cyklu software[2]

V model

Model zakládající si na neustálém testování s cílem zajištění vysoké kvality software. Každá etapa využívá jiného druhu testovaní a testy vznikají při vzniku požadavků na ověřování jednotlivých kroků.[10][1] V model je zobrazený na obrázku2.5.

[2]Obrázek převzat z[27]



Obrázek 2.5: V model životního cyklu software

Levá část modelu obsahuje jednotlivé etapy vývoje, pravá část typy testování náležící etapě. Testování probíhá od malých částí přes integrační testy až po kompletní aplikaci. Podmínkou pro posun do další etapy vývoje je splnění všech testů současné etapy.

Agilní vývoj

Při vytváření zejména menších softwarových systémů, jsou klasické modely spíše ke škodě. Přinášejí spoustu pravidel, která vývoj prodlužují a tím i prodražují.[9] Agilní vývoj by měl být řešením zmíněných problémů. Dá se nejlépe definovat třemi základními principy, kterými se agilní vývoj řídí: (Tento seznam je parafrází z[10])

1. Použití modelu přírůstkového vývoje s velmi krátkými iteracemi, kdy jsou nejprve tvořeny nejdůležitější části software, které jsou vyzkoušeny zákazníkem. Po přijetí části zákazníkem se pokračuje na další.

2. Dobrá komunikace mezi zákazníkem a vývojářem. Zástupci zákazníka by měli být členy vývojového týmu a podílet se na návrhu systému. Díky krátkým iteracím sdělují své návrhy a poznámky vývojářům.

3. Přísné automatizované testování za pomoci komplexní sady testů, které jsou sestavené a prověřené pro každou verzi software.

Nejlepší využití najde tento styl vývoje u menších projektů nebo u projektů, kde se často mění požadavky na systém. Důraz je zde kladen na člověka jako určující faktor pro kvalitu výsledného software.[9]

2.3 Způsoby testování

Způsoby testování jsou děleny do skupin podle stylu provádění testů (statické a dynamické), míře znalosti kódu (černá a bílá skříňka) a fázi vývoje ve kterém se projekt nachází (například jednotkové a integrační). Výběr druhů testů je ovlivněn zvoleným vývojovým modelem2.2 a také typem samotného produktu. Pro zajištění co možná nejvyšší kvality softwaru, je doporučené mít již předem určeno, jaké typy testů a v jaké fázi vývoje budou použity. Nesprávná volba testů může mít za následek odhalení chyb až v pozdější fázi vývoje, ve které je oprava daleko nákladnější.

Úrovně testování

Testování můžeme rozdělit do pěti základních úrovní, také nazývaných jako fáze. Rozdělení je provedeno v závislosti na časovém odstupu od sepsání kódu a stádia v jakém se software právě nachází. (Tato podkapitola je parafrází článku[26] od autora Hlava T.)

1. **Programátorská** - Kontrola je provedena vzápětí po vytvoření kódu a v ideálním případě není prováděna programátorem, který kód psal, ale někým jiným. Takovému testu se potom říká tzv. "test čtyř očí". Kontrola probíhá na úrovni zdrojového kódu. Chyby zjištěné v této fázi jsou ihned opraveny s malými náklady.

2. **Jednotková** - Jedná se o testování samostatné části vytvářeného software tzv. jednotky. Testy těchto jednotek se zapisují ve formě programového kódu. Jsou z pravidla psány programátory. Jednotkovými testy je vhodné se zabývat již při návrhu aplikace, zde je ideální doba jestli je budeme chtít použít. Zpětné doplnění těchto testů je velice nákladné a investice se u menších projektů nevyplatí.

3. **Funkční** - Tyto testy probíhají na straně tvůrce software. Ověřují jestli aplikace správně plní všechny úkoly, pro které je určena. Předmětem testů jsou obecně všechny implementované funkce. Ověřuje se jejich správná funkčnost a jestli odpovídají požadavkům zákazníka.

4. **Integrační** - Na této úrovni se testuje integrace dosud jednotlivě ověřených částí. Předmětem testování je kontrola bezchybné komunikace mezi jednotlivými komponenty. Integrační testy mohou být ruční i automatizované. Nejsou připravovány programátorem, ale především testovacím týmem. Tyto testy jsou někdy označovány jako „testy vnitřní integrace“. Při správném provedení dalšího druhu testování - Systémového, není integračních testů potřeba, z tohoto důvodu se u menších projektů mnohdy vypouští.

5. **Systémové** - Ověřuje aplikaci jako funkční celek z pohledu zákazníka, podle připravených scénářů. Jednotlivé scénáře jsou tvořeny kroky, které mohou při používání aplikace nastat. Tento typ testování probíhá obvykle v několika iteracích. Nalezené chyby jsou opraveny a jejich opravy jsou otestovány v další iteraci. Systémová úroveň testování slouží jako výstupní kontrola.

6. **Akceptační** - Testy které probíhají na prostředí u zákazníka. Podklady pro testování jsou scénáře, které si mohl zákazník a dodavatel předem určit. V případě chyb nebo nesrovnalostí aplikace a její specifikace, je problém nahlášen dodavateli. Opravené chyby jsou implementovány rovnou k zákazníkovy.

Statické a dynamické

Podle nutnosti spuštění testovaného softwaru můžeme testy rozdělit do dvou skupin: statické a dynamické. Statické typy testů nepotřebují aby vytvářená aplikace běžela. Provádějí se ještě při raných fázích vývoje a využívají se k analýze kódu. Dynamické pro svůj běh potřebují spuštění testovaného software. Testování potom probíhá pomocí poskytnutí předem definovaných vstupů software a kontrole jeho výstupů.[5][1]

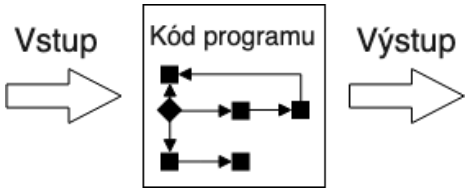
Způsob provádění

Testy lze rozlišit podle toho, zda jsou prováděny softwarem, nebo člověkem. Pokud test vyžaduje lidské ohodnocení a úsudek, je vhodnější využít manuální testování. Pro opakované spouštění velkého počtu testů s velkým množstvím generovaných dat je vhodnější použít automatické testování.[17]

Znalost zdrojového kódu

Podle znalosti kódu dělíme metody testování na Černou, Bílou nebo Šedou skříňku.

U metody Bílé skříňky je nám k dispozici testovaný kód. Tester tak na základě těchto znalostí vybírá vstupy a určuje očekávané výstupy. Tato metoda se dá použít na jednotkové, integrační i systémové testování. Nákres je možno vidět na obrázku2.6.[5][23]



Obrázek 2.6: Bílá skříňka

Černá skříňka vnímá testovaný software jako jeden celek. Nemá k dispozici jeho zdrojový kód a nezná jeho implementaci. Jediné co vidí je jak vypadá a analýzou může zjistit jak se i chová. V této metodě se zaměřujeme na vstupy a výstupy.Příkladem testů tohoto typu jsou akceptační testy a testy pomocí scénářů.[1] Nákres této metody je na obrázku2.7.

Obrázek 2.7: Černá skříňka

Na rozhraní mezi Bílou a Černou skříňkou se nachází Šedá. Oproti černé skříňce máme k dispozici navíc informace o implementaci, jako je například použitý algoritmus nebo datové struktury. Informací není dostatek, aby se dala považovat za Bílou.[1][23]

Kapitola 3

Shrnutí dosavadního stavu

3.1 Robotický operační systém

Robotický Operační Systém (zkráceně ROS) je framework vytvořený k psaní software pro roboty. Framework znamená soubor nástrojů, knihoven a konvencí. Těmi chce ROS docílit komplexního a robustního chování napříč velkým množstvím různých robotických platforem. Poskytuje nám hardwarovou abstrakci, ovladače pro zařízení a správce balíčků.[22]

Tento systém je neustále vyvíjen a vydáván ve formě distribucí. Distribuci můžeme chápat jako verzované soubory ROSovských balíčků. Aktuálně nejnovější distribucí je Melodic Morenia.[12][22] ARTable využívá starší verzi Indigo Igloo. Cílovou platformou pro ROS je Linux (na př. Ubuntu, Debian). Provozovat lze ale i in na jiných platformách jako jsou Windows, Mac OS X nebo Android.[22]

Základem architektury systému ROS jsou uzly, zprávy, pojmenované roury, služby a Master. Data do systému poskytuje každý z těchto prvků svým daným způsobem.[22][12]

Uzly

Uzly jsou paralérně běžící procesy, které provádějí výpočty a práci s daty. Každý uzel má své označení - jméno, pod kterým je registrovaný v ROS Masteru. ROS je navržen s myšlenkou co největší modularity. Systém je tak složen většinou z více uzlů, kde každý má na starosti pouze specifický úkol. Příkladem rozdělení systému pro robota složeného z kamery, ruky a kol, může být uzel zajišťující obsluhu ruky, uzel obsluhující kamera a uzel ke kontrole kol. Komunikace mezi jednotlivými uzly zajišťují pojmenované roury nebo služby. Rozdělení aplikace do uzlů nám zajišťuje snížení komplexnosti kódu - jednotlivé uzly řeší pouze specifické úkony a větší toleranci vůči chybám - pády jsou omezené pouze na jednotlivé uzly. Jednotlivé uzly mohou být programovány v různých jazycích.[12][22]

Master

„Hlavní uzel pojmenovaný ROS Master nám zajišťuje registraci a přístup ke službám pro ostatní uzly v ROS systému“.[22] Slouží jako prostředník pro uzly, aby se dokázali lokalizovat a vzájemně spolu komunikovat.[12]

Zprávy, pojmenovane roury a služby

„Uzly mezi sebou vzájemně komunikují pomocí publikování zpráv na pojmenovane roury nebo služby. Zpráva je jednoduchá datová struktura, která může být složena z primitivních datových typů (na př. celé číslo, boolean) a polí.“[22]

Pojmenovaná roura je jednosměrná sběrnice, která slouží k posílání zpráv mezi uzly. Přístup k rourám je anonymní, uzly tak nevědí, s kým na rouře komunikují. Pro posílání zpráv na pojmenovanou rouru se uzel musí přihlásit jako vydavatel. Pro příjem zpráv z roury se uzel přihlásí k dané rouře jako odběratel. K jedné rouře může být přihlášeno neomezeně vydavatelů a odběratelů.[22]

Pro obousměrnou komunikaci mezi uzly slouží služby, které fungují na principu žádost/odpověď. Služba je definovaná názvem a typem zpráv, které se po ní dají posílat - jedna pro žádost a jedna pro odpověď.[12][22]

3.2 ARTable

„ARTable je soubor zařízení, které představují vizi, jak by mohla v budoucnu vypadat spolupráce mezi člověkem a robotickými zařízeními na pracovišti.“[2] Zobrazen na obrázku3.1. Využívá rozšířené reality, pomocí které se snaží přiblížit interakci s roboty i nespecializovaným pracovníkům. Systém ARTable se skládá z dotekové plochy na kterou je pomocí projektorů zobrazeno grafické prostředí, Kinectu, počítače a robota PR2.

Programová část ARTable pracuje pod systémem ROS3.1, systém je rozdělen do uzlů. Popisu funkčnosti jednotlivých uzlů je věnována podkapitola Objektová architektura. Uzly jsou naprogramované pomocí jazyka Python ve verzi 2.7.

Obrázek 3.1: ARTable vývojové pracoviště[1]

[1]Obrázek převzat z[3]

Objektová architektura

Hlavním uzlem, který vše řídí je takzvaný Mozek. Pomocí služeb (viz. ROS zprávy a služby3.1) komunikuje s ostatními částmi - uzly a kontroluje jejich stav. Pro testování grafického prostředí jsou zajímavé pouze dva uzly:

- Dotekový ovladač - zpracovává doteky stolu
- Grafické prostředí - vytváří grafickou scénu

K předávání informací o dotecích využívají pojmenované roury a zpráv *Doteku*3.2.

Zprávy Dotek

Zpráva *Dotek* slouží ke sdílení informací mezi uzly o provedeném fyzickém doteku na stole. Struktůra zprávy:

- **id** - identifikátor doteku
- **touch** - stav doteku - pravda pokud se prst dotýká, nepravda pokud je prst zvednut z plochy
- **point** - bod s hlavičkou - obsahuje polohu doteku a hlavičku, ve které se nachází čas doteku a identifikátor rámce

Dotekový ovladač

Uzel starající se o zpracování doteků z dotekového stolu, které jsou sdíleny formou tzv. událostí. Událost představuje vše co se odehraje na dotekové ploše stolu.

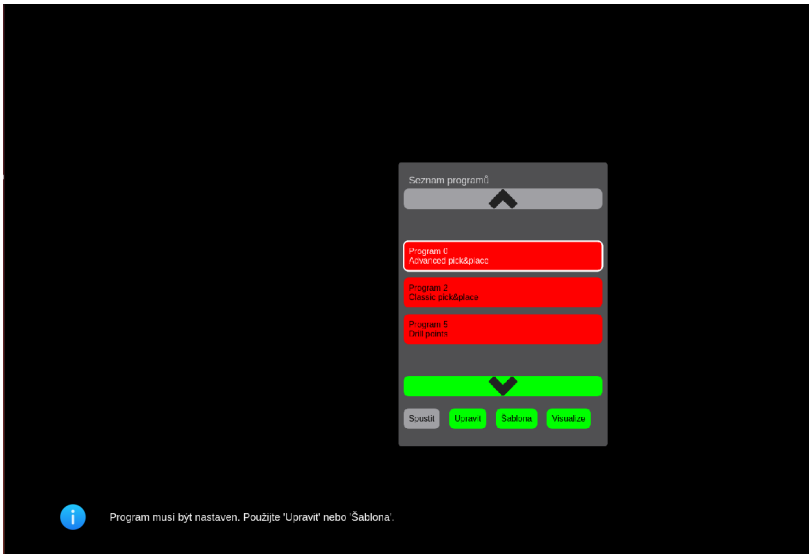
Dotekový ovladač události zpracuje, na jejich základě vytvoří a pošle zprávy *Doteku*. Grafické prostředí3.2 zprávy příjme a provede interakci s prvky (na pří. posun elementu nebo kliknutí na tlačítko).

Grafické prostředí

Z pohledu testování grafického prostředí nejzajímavější uzel. Zde je tvořena viditelná část ARTable, která se pomocí projektorů zobrazuje na dotekový stůl.

Grafická scéna je tvořena pomocí knihovny PyQt4[2] a jazyka Python[3]. PyQt4 obsahuje předem vytvořené elementy (na příklad tlačítka, menu, textová pole), určené k ovládání myší a klávesnicí, najdeme je v běžných okenních aplikacích. ARTable je ovládán doteky, scénu tak tvoří vlastními elementy přizpůsobené k ovládáním z dotekového stolu. Na obrázku3.2 je zobrazeno prostředí aplikace.

[2] https://www.riverbankcomputing.com/static/Docs/PyQt4/
[3] https://www.python.org/



Obrázek 3.2: Ukázka grafického prostředí ARTable ve vývojovém režimu na počítači

3.3 Práce s roboty v rozšířené realitě

Rozšířená realita zkráceně AR (z anglického Augmented reality) je termín používaný pro počítačem vytvořenou grafiku, text a 2D nebo 3D objekty, které překrývají skutečný svět. Virtuální informace jsou vloženy do reálného světa, ten se tak stává rozšířeným.[6]

Je s námi od 90. let 20. století, ale teprve s příchodem chytrých telefonů, tabletů a zájmu velkých firem se jí dostává větší popularity.[16] Jeden z prvních fungujících systémů využívajících AR byl Virtual fixture vyvinutý v roce 1992 Armstrongovou laboratoří pro Americké letectvo (obrázek3.3).[29]

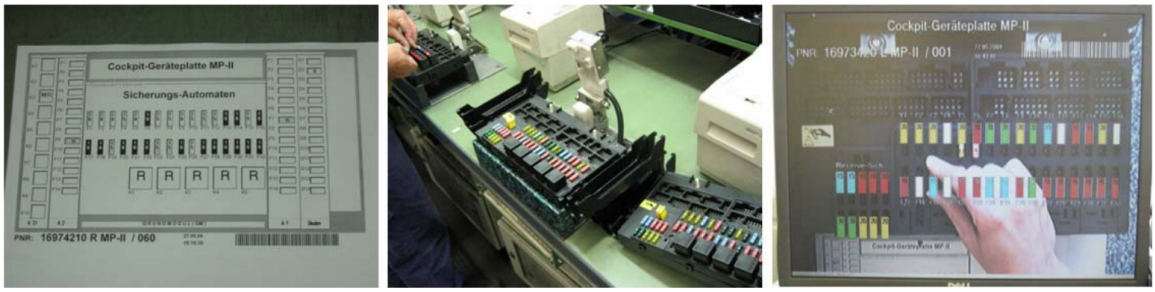
Obrázek 3.3: Ukázka Virtual fixtures použitého ke zlepšení schopnosti míření[4]

[4]Obrázek převzat z[28]

Za začátek používání AR v průmyslovém kontextu můžeme brát projekt společnosti Boeing zabývající se vývojem a výrobou letadel. V projektu firma využila AR pro asistovanou dálkovou montáž svazků vodičů.[14][19]

Na přelomu 21. století začaly vznikat i pokusy o využití AR při navrhování nového produktu, jako pomoc při skládání součástek na desku nebo pro trénování pilotů či řidičů automobilů. Obrázek3.4 zobrazuje příklad použití AR při montování pojistek.[19]

Z důvodu velkých nároků na odbornost pracovníků při běžné údržbě nebo i malé změně ve výrobním pochodu vznikly projekty, které se snaží přiblížit ovládání robotů pomocí AR širší veřejnosti. Řešení většinou využívá brýlí s rozšířenou realitou nebo projektoru.



Obrázek 3.4: Montování pojistek za pomocí AR[5]. Plán zapojení pojistek (Levý obrázek), Pracovní prostor (Prostřední obrázek) a AR pohled (Pravý obrázek)

Jak ovládat roboty za pomocí AR

Programování robotů za pomoci AR je nová disciplína, u které zatím nejsou určeny pevné základy. Vznikají tak studie a projekty, které se zabývají řešením interakce uživatele s robotem (na př. publikace[15]). Jednou z nich je studie[4] od Araiza-Illan D., De San Bernabe A., Hongchao F., Shin L.Y., která popisuje problémy a možná řešení při tvorbě systémů pro programování roborů za pomoci AR. Řešení využívá HoloLens od Microsoftu, které jsou k vidění na obrázku3.5.

Obrázek 3.5: Ukázka HoloLens od Microsoftu[6]

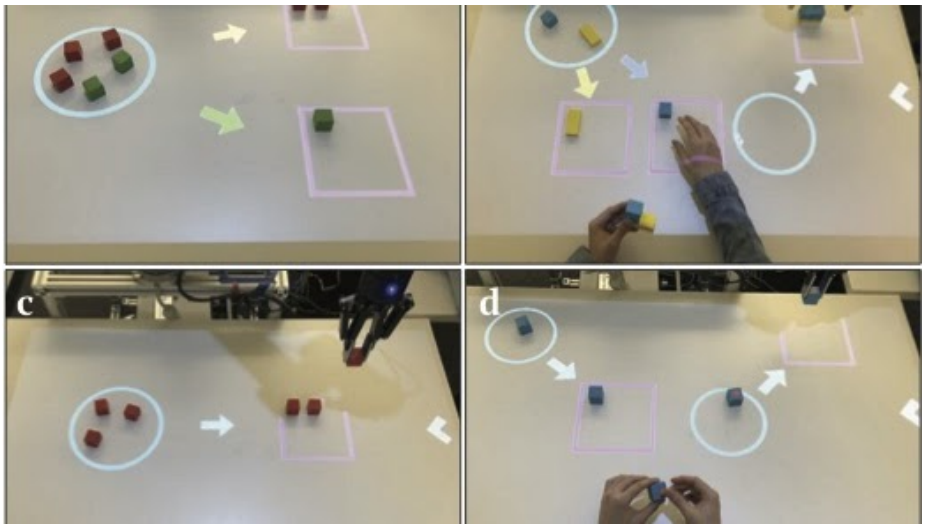
Další možností jak vyřešit problém uživatelské interakce s robotem je projekce. Menu a další grafické prvky jsou zobrazeny projektorem přímo na výrobním pásu nebo pracovní desce. Příklad prostředí můžeme vidět na obrázku3.6. Použití není omezeno brýlemi a více pracovníků má možnost bez další nástrojů prostředí používat. Jsme zde ale omezeni z důvodu použití projektoru pouze na 2D zobrazení.

[5]Obrázek převzat z[19]
[6]Obrázek převzat z[13]

PATI

PATI[7] je projekt využívající projektoru a Kinectu. Ovládání grafického prostředí je zde prováděno pomocí gest, která jsou zpracována Kinectem. Jak tento systém vypadá si můžeme prohlédnout na obrázku3.7. PATI byl vytvořen jako součást výzkumu „Programování za pomoci demonstrace“ (anglický originál Programming by Demonstration), který se zabýval zkoumáním metod a uživatelských rozhraní v rozšířené realitě, používaných při programování robotů.

Obrázek 3.6: Ukázka systému PATI[7]



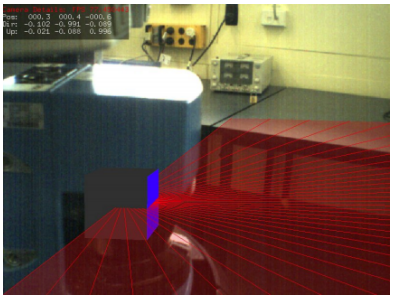
Obrázek 3.7: Ukázka rozhraní při programování v PATI[8]

[7]Obrázek převzat z[7]
[8]Obrázek převzat z[7]

ARDev

Abychom předešli chybám při programování robotů, potřebujeme vědět, jaké předměty je robot schopný v určitý okamžik vnímat.

Touto otázkou se zabývá studie[25] od T. H. J. Colletta a B. A. MacDonalda. V této studii je popsán nástroj *ARDev*, který zobrazuje v AR pomocí 2D objektů rozsah, jaký může robot pomocí svých senzorů vnímat. Ukázku použití *ARDev* můžeme vidět na obrázcích3.83.9, kde je ukázán zdroj laserů získávajících data o okolí a ohraničení kam dosáhly.



Obrázek 3.8: Zobrazení zdroje datových laserů[9]

Obrázek 3.9: Zobrazení hranic datových laserů[10]

[10]Obrázek převzat z[25]
[10]Obrázek převzat z[25]

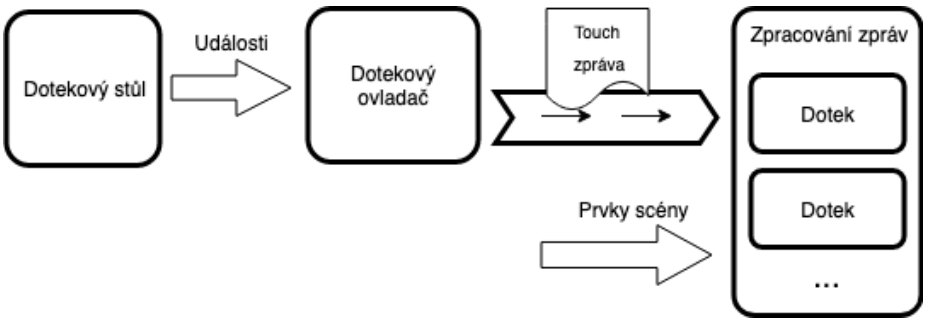
Kapitola 4

Návrh řešení

Začátek této kapitoly je věnován popisu, jakým stylem ARTable zpracovává doteky a jak řeší problematiku tvorby grafického prostředí. S těmito znalostmi se podívám na dostupné testovací nástroje a zhodnotím jestli se dají využít pro řešení problému automatického testování grafického prostředí ARTable. Na konci kapitoly uvedu návrh vlastního řešení.

4.1 Zpracování doteku v ARTable

Před začátkem hledání testovacích nástrojů bylo nutné se seznámit s ARTable a jeho stylem zpracování doteků. Obrázek4.1 nám znázorňuje, jak jsou doteky zpracovány.



Obrázek 4.1: Postup zpracování doteků v ARTable

Jednotlivé úrovně zpracování doteku:

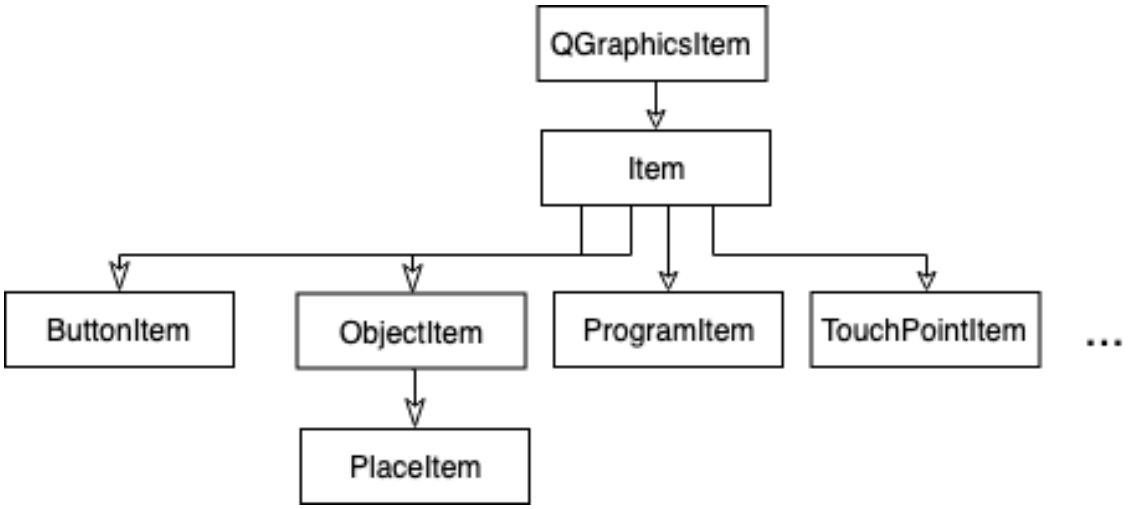
1. **Dotek na stole** - Zpracování doteku začíná u fyzického doteku na dotekovém stole, který pošle tzv. události přes rozhraní do systému ROS.

2. **Dotykový ovladač** - Příchozí událost typu dotek, převede na zprávu *Dotek*, kterou pošle na pojmenovanou rouru *Touch*. Pozice doteku je určená v souřadnicovém systému dotekového stolu.

3. **Grafická scéna** - *TouchTableItemHelper* přepošle zprávu *Doteku* pomocí Qt signálů do *TouchTableItem*. Zde jsou zprávy převedeny (včetně souřadnicových systému) na doteky reprezentované *TouchPointItemem*, který již mění stavy (na pří. kliknutí a zvýraznění) prvků grafické scény.

4.2 Vykreslení grafické scény

Grafické prostředí je tvořeno za pomoci knihovny PyQt ve verzi 4. Jedná se o vazbu na multiplatformní soubor nástrojů Qt, který se využívá k tvorbě grafických uživatelských prostředí. PyQt slouží k tvorbě tzv. okenních aplikací, nemá tak v základu podporu pro dotekovou interakci. Tento nedostatek řeší ARTable vytvořením obecného prvku - *Item* grafické scény, který disponuje metodami reagujícími na zpracované doteky. Všechny grafické prvky, které jsou použity k tvorbě scény, jsou objektovými potomky *Itemu*, který je potomkem PyQt objektu *QGraphicsItem*. Objektová relace mezi prvky scény je zobrazena na obrázku4.2.



Obrázek 4.2: Relace mezi objekty grafické scéna ARTable

4.3 Dostupné testovací nástroje

Se znalostmi o zpracování doteků a způsobu vykreslování na ARTable, jsem blíže specifikoval požadavky na testovací nástroj. Na základě této specifikace jsem vytvořil seznam atributů, který byl použit k hodnocení využitelnosti daného nástroje pro řešení problému. Funkčnost některých vybraných nástrojů jsem pro přesnější specifikování využitelnosti otestoval na jednoduché PyQt aplikaci.

Níže uvedené nástroje jsou pouze výběrem z dostupných nástrojů. Každý z těchto nástrojů představuje určitý druh přístupu k testování grafického prostředí.

Specifikace požadavků

Předmětem hledání je testovací nástroj na testování aplikace využívající PyQt verze 4 naprogramované v Pythonu verze 2.7, který dovoluje změnit nebo rozšířit metodu, která simuluje uživatelský vstup. Nástroj musí být možno zapojit do ROSu.

Na základě specifikace volím tento seznam atributů, který budu využívat k hodnocení použitelnosti nástrojů.

Seznam atributů:

- **Python/PyQt** - Podporovaná verze Pythonu a PyQt
- **Rozšíření/Dotek** - Možnosti změny nebo rozšíření funkcí pro simulaci kliknutí nebo simulace stisknutí
- **ROS zapojení** - Možnost nástroj zapojit do ROS
- **Licence** - Pouze volně dostupné nástroje

Testovací aplikace

Před hledáním jsem vytvořil s využitím znalostí o ARTable jednoduchou aplikaci za pomoci PyQt4, která simuluje zjednodušené grafické prostředí ARTable. Aplikace je složena z okna a scény obsahující tlačítko a prvky vytvořené pomocí ARTable *Itemu*. Za pomoci aplikace jsem testoval možnosti nástrojů a jejich využitelnost. Aplikace je k nalezení v příloze A.

QTest

QTest[1] je jeden z výchozích modulů knihovny PyQt. Slouží k simulaci uživatelských vstupů v podobě klávesových stisků nebo klikání myší. Může být využit k psaní testů společně s unittestem. Simulace probíhá pouze nad prvky typu QWidget. ARTable scéna je složená z prvků vycházejících z QGraphicsItem, tím se stává nástroj nevhodným. Rozšíření nástroje vidím jako neefektivní. Využít by se dala jeho část, konkrétně řešení pasivního čekání, při čekání na Qt signály při zpracování doteku (viz. podkapitola 4.1)

Atribut	Splněno	Poznámka
Python/PyQt	Ano	
Rozšíření/Dotek	Ne	
ROS zapojení	Ano	součástí PyQt
Licence	Ano	GNU General Public License[2]

Pytest-qt

Jedná se o rozšíření pro pytest, které usnadňuje psaní testů pro PyQt a PySide. Pytest je framework použitelný k testování python programů. Vyznačuje se lehce pochopitelnými a čitelnými testy.[18] Pytest-qt[3] je jednoduchý na použití z důvodu pěkné dokumentace a intuitivních názvů funkcí. Bez rozšíření o simulaci nepoužitelný. Po přidání schopnosti simulace doteků by obsahoval spoustu nevyužité funkčnosti a byl by pro budoucí úpravu nepřehledný.

Atribut	Splněno	Poznámka
Python/PyQt	Ano	Python 2 a více a PyQt4, PyQt5, PySide, PySide2
Rozšíření/Dotek		zbytečná funkčnost navíc
ROS zapojení		pytest se po změně výstupního formátu dá zapojit do ROSu
Licence	Ano	MIT[4]

[1] https://www.riverbankcomputing.com/static/Docs/PyQt4/qttest.html
[2] https://www.gnu.org/licenses/old-licenses/gpl-2.0.txt
[3] https://pytest-qt.readthedocs.io/

PyAutoGUI

PyAutoGUI[5] je rozšíření které simuluje uživatelský vstup z myši a klávesnice. Dal by se použít v rostestech, ale při jeho spuštění by musela být zobrazena grafická scéna ARTable a tester by nemohl dělat jinou práci, jelikož by přišel o možnost používat myš a klávesnici.

Atribut	Splněno	Poznámka
Python/PyQt	Ano	Python 2 a více
Rozšíření/Dotek	Ano	bez rozšíření, při spuštěném testu nelze dělat nic jiného
ROS zapojení	Ano	
Licence	Ano	BSD 3-Clause "New"or "Revised"[6]

PyQtTester

PyQtTester[7] je nástroj který nejprve nahraje testovací vstup do tzv. scénářů, které později reprodukuje.

Atribut	Splněno	Poznámka
Python/PyQt	Ne	Python 3
Rozšíření/Dotek	Ano	bez rozšíření, při spuštěném testu nelze dělat nic jiného
ROS zapojení		
Licence	Ano	GNU General Public License v3.0

Další nástroje

Pro testování existují i další nástroje. Jedním z příkladu je Sikilix [8], který pomocí technologie OpernCv[9] na rozpoznání obrázků hledá elementy, se kterými pracuje, a kontroluje jestli se aplikace dostala do požadovaného stavu. Zmíněné řešení by bylo vhodné za předpokladu, že by byl přístup ke zdrojovým kódům omezený.

Výsledek hledání

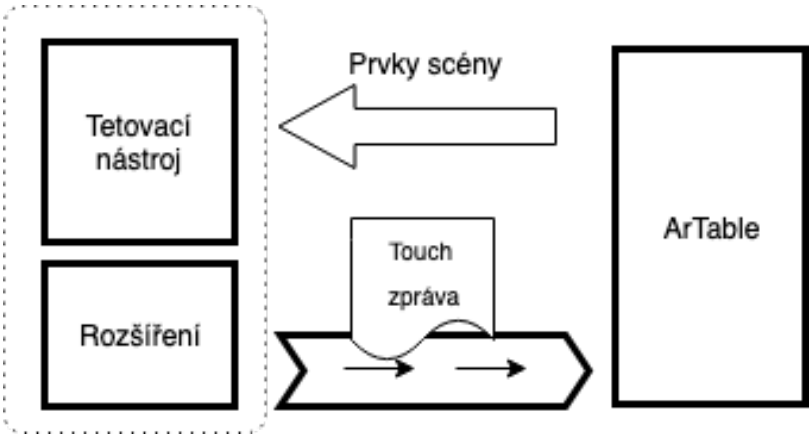
Vhodný nástroj řešící problém testování grafického rozhraní na ARTable se najít nepodařilo.

[4] https://opensource.org/licenses/MIT
[5] https://pyautogui.readthedocs.io/
[6] https://spdx.org/licenses/BSD-3-Clause.html
[7] https://github.com/biolab/PyQtTester
[8] http://sikulix.com/
[9] https://opencv.org/

4.4 Návrh nástroje

Na základě nabitých znalostí o dostupných testovacích nástrojích, jsem se rozhodl, že ideálním řešením je vytvořit nový nebo rozšířit již existující testovací nástroj o schopnost simulovat doteky.

Navrhuji vytvoření rozšíření testovacího nástroje za použití jazyka Python, které bude simulovat doteky pomocí posílání zpráv *Doteku*. Schéma navrženého řešení je na obrázku4.3.



Obrázek 4.3: Schéma navrženého testovacího nástroje

Navrhuji `pytest`[10] vhodným testovácím nástrojem k rozšíření. Ze subjektivního hlediska disponuje přehlednějšími testy a pohodlnějším vytvářením rozšíření v porovnání s `rostestem` a `unittestem`[11]

Rozšíření bude disponovat rozhraním, které odstíní testera od logiky posílání zpráv. Použití by mělo být intuitivní s možností používat oba souřadnicové systémy a určit místo doteku odkazem na prvek scény.

4.5 Postup práce

Práci rozděluji na čtyři části, které na sebe vzájemně navazují. Na konci každé z částí je testování, které má za úkol ověřit její úspěšnost. Výstupem bude nástroj použitelný k testování grafického prostředí ARTable.

1. **Zapojení pytestu do ROSu**
   Výchozím nástrojem použitím k testování v ROSu je rostest, který může být nahrazen jiným nástrojem za předpokladu, že je výstup reportů nástroje ve formátu JUnit XML[8].

   Otestování proběhne ruční kontrolou zda-li jsou reporty testů z `pytestu` správně viditelné v ROSu.

[10] https://pytest.org/
[11] https://docs.python.org/2/library/unittest.html

2. **Základ rozšíření**
   Rozšíření bude mít název ARTBot a bude pod tímto názvem dostupné v pytestech. K docílení bude využito pytestových fixtures, který jsou používány k vytváření fixních výchozích bodů pro opakované testování.[18]

   Kontrola proběhne ručně dostupnosti ARTBota v pytestech, následována tvorbou automatických testů, kontrolujících jeho dostupnost.

3. **Podpora pro odesílání zprávy `Doteku`**
   ARTBot bude rozšířen o možnost vytvoření a odeslání zpráv *Doteku*, které obsahují vlastní souřadnice. Zde bude nutná podpora obou typů souřadnic.

   Testování této funkčnosti proběhne ve dvou fázích. První fáze bude ruční, kdy se vytvoří testovací uzel v ROSu, který bude přijímat zprávy `Doteku` a vypisovat je na standartní výstup. Zde se provede kontrola dostupnosti odeslaných zpráv v ostatních uzlech. Ve druhé fázi vznikne test, který bude přijímat zprávy a kontrolovat jejich obsah, konkrétně souřadnicový systém.

4. **Tvorba rozhraní**
   Do ARTBota přibudou metody, které zjednoduší práci při psaní testů. Inspirací může být například `QtBot` z `pytest-qt` rozšíření a jeho metody pro simulování kliknutí.

   K otestování funkčnosti nad každou metodu vznikne test na extrémní hodnoty argumentů a zároveň i test, který bude, stejně jako v předchozí částí, kontrolovat zprávy *Doteku* a jejich obsah.

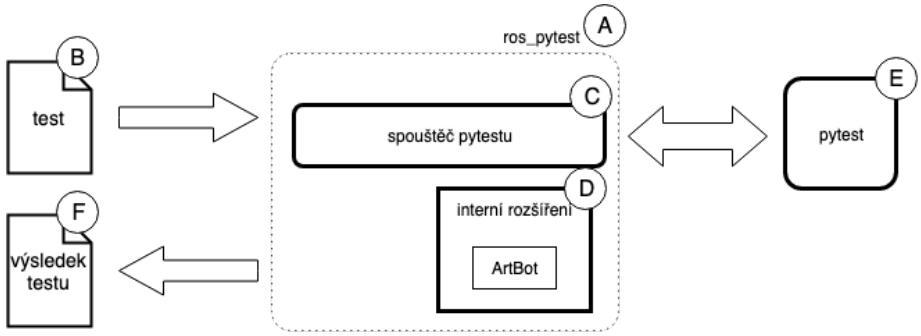
Kapitola 5

Implementace

Tato kapitola popisuje implementaci návrhu řešení testovacího nástroje grafického prostředí pro ARTable. Inspirací při vytváření mi byl nástroj pytest-qt (viz. testovací nástroj4.3), ze kterého jsem využil i jeho část, konkrétně zpracování čekání na signály. Testovací nástroj je realizován jako jeden uzel, který obsahuje rozšíření a spouštěč pro pytest. První část se zabývá architekturou nástroje jako uzlu v ROS a popisuje jeho obecnou funkčnost. Následuje řešení zapojení pytestu a popis ARTBota s jeho simulací doteků a rozhraní. Ukázka použití nástroje spolu s testováním a jeho průběhem se nachází na konci kapitoly.

5.1 Architektura nástroje

Obrázek5.1 znázorňuje architekturu řešení a jakým stylem probíhá vyhodnocování testů. V textu se budu odkazovat na jednotlivé části obrázku, které jsou označeny písmeny A-F.



Obrázek 5.1: Schéma testovacího nástroje

Nástroj se skládá z balíčku a uzlu s názvem *ros_pytest* (část A). Jehož vstupem je kód testu (část B) ve formátu pro pytest, ve kterém je možnost použít ARTBota. Ke kódu má tester možnost připojit i vlastní soubor obsahující další rozšíření, která budou použitá pouze pro tento test. Spouštěč pytestu (část C) načte interní a externí rozšíření (část D) a spustí pytest (část E) se vstupním kódem testu. Výstupem je zpráva o výsledku testu ve formátu pro rostest (část F).

5.2 Zapojení pytestu

Výchozí nástroj používaný ROSem pro testování je rostest[1], který umožňuje spuštění „roslaunch“ souborů jako základní bod pro testování.[22] Výsledky testů jsou ve formátu JUnit XML[8], který je standardním typem formátu výstupů pro testovací frameworky. ROS umožňuje použití i jiných testovací nástrojů, jejich výstup ale musí být ve formátu JUnit XML.[22] Inspirací při řešení mi byl internetový článek[24] od A. Rössler, který popisuje jak zapojit pytest do ROSu.

Uzel se spustitelným kódem

Základem nástroje je balíček obsahující jeden uzel použitelný k testování jiných uzlů (viz. 5.1). K vytvoření balíčku se jménem *ros_pytest* jsem použil nástroj *catkin*[2], který je základní částí ROSu a slouží ke správě balíčků. Součástí *ros_pytest* je spustitelný kód, který spouští pytest (viz. 5.2) a je přístupný v „roslaunch“ souborech z ostatních uzlů.

Spustitelný kód

Spustitelný kód můžeme najít ve složce `catkin_ws/src/ros_pytest/scripts` pod názvem `ros_pytest_runner.py`, kde `catkin_ws` představuje složku obsahující pracovní prostor projektu. Složky `scripts/` a `nodes/` jsou doporučovanými, pro ukládání spustitelných kódů.

Soubor `ros_pytest_runner.py`(ukázka kódu5.1) obsahuje volání funkce *run_pytest* z uzlu s argumenty příkazové řádky, jež provádí provolání pytestu5.2.

Ukázka kódu 5.1: Část obsahu souboru pro volání spouštěče pytestu

```
from ros_pytest import run_pytest

if __name__ == '__main__':
    sys.exit(run_pytest(sys.argv))
```

Globální přístup

K zajištění možnosti spuštění `ros_pytest_runner.py` i z jiných balíčků proběhla tzv. instalace. Nainstalované spustitelné kódy jsou potom v ostatních balíčcích přístupné pod názvem kódu a balíčku. Název kódu tak musí být unikátní pouze v rámci balíčku.

Instalace je složena ze dvou částí: první v souboru `CMakeLists.txt` a druhé v souboru `setup.py`. Instalace provedená v souboru `CMakeLists.txt` je za pomocí funkce *catkin_install_python*. Její parametry určují který kód a kam bude instalován. Umístěním je složka `bin/` představující globální prostor.[21]

Soubor `setup.py` (ukázka kódu5.3) obsahuje python definici balíčku nezbytnou pro instalaci do pythnovského ekosystému. Definice balíčku je již popsána v souboru `package.xml`, aby nedošlo k duplikaci informací byl použit catkin modul *catkin_pkg.python_setup*, který definici ze souboru načte. Balíček se instaluje automaticky. Instalace mám zajišťuje dostupnost balíčku ve vývojovém prostoru - devel v ROSu.

[1] http://wiki.ros.org/rostest
[2] https://github.com/ros/catkin

Použití definičního souboru `setup.py` je nutné uvést i v `CMakeLists.txt` (ukázka kódu5.2) pomocí funkce *catkin_python_setup()*, jinak by nedošlo k instalaci.[20]

Ukázka kódu 5.2: Instalační část souboru CMakeLists.txt

```
#...
# použít soubor setup.py
catkin_python_setup()
#...
catkin_install_python(PROGRAMS
    # cesta ke spustitelnému kódu
    scripts/ros_pytest_runner
    # kam bude kód instalován
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

Ukázka kódu 5.3: Obsah souboru setup.py zajišťující dostupnost v devel prostoru

```
from distutils.core import setup
from catkin_pkg.python_setup import generate_distutils_setup

# načtení informací o balíčku z~package.xml
setup_args = generate_distutils_setup(
    packages=['ros_pytest'],
    package_dir={'': 'src'},
)

setup(**setup_args)
```

Spuštění pytestu

Výchozí výstup pytestu je ve speciálním formátu, který není v ROSu podporován. Změnu formátu můžeme provést použitím parametru *junitxml*.

Pro spuštění pytestu je zapotřebí určit vstupní soubor a cestu k výstupnímu souboru.

Určení vstupního souboru

Spouštění testů v rosluch souborech probíhá pomocí značky *<test>*, která definuje jaký uzel s jakými parametry má ROS spustit k testu. Řešení ale využívá této značky ke spuštění kódu `ros_pytest_runner.py`. Cesta ke vstupnímu souboru je tak předávána pomocí ROS parametru s názvem *test_module* (ukázka kódu5.4).

Ukázka kódu 5.4: Určení vstupního souboru a spuštění testu v rosluch

```
<param name="test_module"
       value="$(find uzel_k_otestovani)/tests/test_.py" />
<test pkg="ros_pytest" type="ros_pytest_runner"
      test-name="pytest_test_ui_item_move" />
```

Určení výstupního souboru

Výstupní soubor předává ROS automaticky ve formě argumentu při spuštění uzlu pomocí značky *<test>*. Argument se nazývá *gtest_outpu* a je používám ve formátu *–gtest_outputx̄ml cesta k souboru*.

Volání pytestu

S připraveným vstupním kódem testu a cestou k výstupu je zavolána funkce *pytest.main*, která umožňuje spustit pytest přímo z kódu (ukázka kódu5.5). Při volání je k dispozici i možnost předat další argumenty, této vlastnosti využijeme u importování rozšíření (viz. část5.2).

Ukázka kódu 5.5: Volání pytestu z kódu s parametry pro výstup ve formátu JUnit xml

```
pytest.main([cesta_k_testu, '--junitxml={}'.format(vystupni_soubor)])
```

Rozšíření

Rozšíření jsou python moduly, které jsou vkládány automaticky před začátkem testování do pytestu. Díky tomuto si můžeme definovat vlastní fixture, které se používají k vytváření fixních výchozích bodů pro opakované testování.

Přidávání rozšíření může probíhat přímo v kódu vstupního testu nebo ještě před začátkem testu pomocí parametru *plugins*. V implementovaném řešení vkládáme rozšíření do pytestu ještě před spuštěním a to ze dvou umístění - interního a externího.

- **Interní** - jsou umístěny přímo v uzlu *ros_pytest* ve složce `/src/ros\_pytest/fixtures`. Z této složky se automaticky importují všechna rozšíření. Nachází se zde z důvodů testovatelnosti a přehlednosti - rozšíření, které se používají častěji jsou na jednom místě.
- **Externí** - mohou se nacházet kdekoliv, jejich umístění si určujeme pro každý test zvlášť.

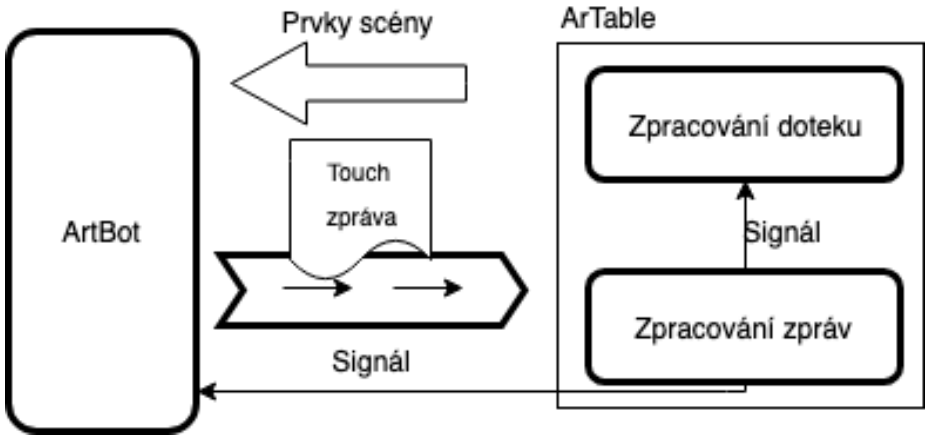
5.3 ARTBot

ARTBot je rozšířením pro `pytest`, které simuluje uživatelskou interakci pro `ARTable` pomocí posílání zpráv *Doteku*. Při tvorbě mi bylo inspirací rozšíření pytest-qt, ze kterého jsem převzal řešení čekání na signály Qt, které jsem upravil pro fungování s `ARTable`.

Simulace doteků

K simulaci doteků používá ARTBot pojmenovanou rouru „/art/interface/touchtable/touch“ (dále jen rouru *Touch*) po které posílá zprávy *Doteku* (viz. část5.2). Poloha doteku je buď určena souřadnicovým systémem nebo prvkem ze scény ARTable.

Proces simulace, ke kterému se vztahuje obrázek5.2, probíhá ve třech krocích: určení polohy, odeslání zprávy *Doteku* a nakonec čekání na zpracování zprávy.



Obrázek 5.2: Schéma testovacího nástroje

1. **Určení polohy doteku**
   Určit polohu doteku můžeme pomocí prvků z grafické scény ARTable, které máme k dispozici. Z nich získáme polohu pomocí metod *_get_item_art_position* nebo *_get_item_position”* pro polohu v pixelech. Polohu můžeme určit i ručně pomocí souřadnic.

2. **Odeslání zprávy**
   Když je určena poloha doteku je vytvořena zpráva `Doteku`. Jako identifikátor zprávy použijeme buť parametrem určené číslo nebo náhodně vygenerované. Bod ve zprávě, který určuje polohu doteku, musí být zadán v souřadnicovém systému pro ARTable. Pokud je poloha určena v pixelech převedeme pomocí metody *pix2m*. Přes pojmenovanou rouru `Touch` potom zprávu odešleme a na konec uložíme do pole zpracovávaných doteků – „_touches“.

3. **Čekání na zpracování zprávy**
   Zpracování zpráv `Doteku` je v ARTable rozděleno do dvou části. První naslouchá na rouře `Touch` a přeposílá přijaté zprávy pomocí Qt signálů druhé, které zprávu ze signálů zpracovává. Na zpracování zprávy ze signálů čekáme. K tomu slouží metoda *wait_for_table_to_process_touches*5.3, která pomocí PyQt metody „QCoreApplication.processEvents“, počká až budou všechny čekající události (na př. signály) zpracovány.

Čekání na signály

Před použitím ARTable je potřeba provést kalibraci dotekové plochy, projektorů a počkat než se ostatní uzly připraví k použití. Jakmile je stůl připraven změní Mozek3.2 stav aplikace na „Není používána“ a poté můžeme začít testovat. Informace o změně stavu je poslána přes pojmenovanou rouru *state* a převedena na signál Qt, na který je potřeba počkat.

Problém čekání řeší využití části rozšíření *pytest-qt* s názvem *wait_signal*[3], které po menší úpravě umožňuje připojení na ARTablem využívané signály.

Metody

ARTBot obsahuje metody pro simulování doteků nad prvky scény - začínající *item_*, k simulaci doteků na určité pozici – začínající *touch_* a pomocné metody (na př. čekání na zpracování doteků. Podle druhu simulovaného doteku (stisknutí, uvolnění a posunutí) jsou metody pojmenovávány. Pro stisknutí *_press*, uvolnění doteku *_release* a posun *_move*.

Identifikátor doteku je náhodně generovaný, pokud není použit parametr *touch_id*, který má přednost, a bude tak použit jako identifikátor.

Nejdůležitějšími metodami pro simulaci doteků jsou *touch_press*, *touch_release* a *touch_move*, řeší odesílání zprávy `Doteku` při změně stavu a polohy doteku. Ostatní metody řešící simulaci doteků už jen využívají těchto metod. Ze zmíněného důvode zde nejsou všechny popsány. Příkladem mohou být *item_move* a *item_press*, které navíc zjišťují polohu prvků zadaných ve vstupních parametrech metod.

Způsoby určení polohy

Polohu doteku můžeme určit ve dvou typech souřadnicových systému, buď v pixelech - `pos` parametr nebo jednotkách pro ARTable - `art_pos` parametr. Vstupní poloha pro ARTBota může být určena pomocí:

- **Místa** (`pos`, `art_pos`, `dest_pos`, `dest_art_pos`)
  Bod grafické scény.
- **Vzdáleností** (`by_art_pos` a `by_pos`)
  Od aktuální polohy o určenou vzdálenost. Využívá se u posunutí prvku scény - *item_move*.
- **Prvku** (`dest_item`)
  Polohou prvku který je součástí grafické scény. Přesnější polohu na cílovém prvku můžeme určit pomocí `dest_item_pos` nebo `dest_item_art_pos`.

touch_press - započetí doteku

```
touch_press(pos | art_pos [, touch_id]) return touch_id
```

Odešle zprávu `Doteku` s atributem touch nastaveným na True na pozici z argumentu. Pokud je zadán parametr „touch_id“ bude využit jako identifikátor pro zprávu. Dotek je pod svým identifikátorem uložen. Návratová hodnota je nově vytvořený identifikátor pro dotek nebo „touch_id“.

[3] https://github.com/pytest-dev/pytest-qt/blob/master/pytestqt/wait_signal.py

touch_release - ukončení doteku

```
touch_release(touch_id) return touch_id
```

Odeslání zprávy `Doteku` s atributem touch nastaveným na False a identifikátorem nastavený na hodnotu parametru touch_id. Dotek s identifikátorem touch_id je odstraněn z uložených doteků.

touch_move - změna polohy dotek

```
touch_move(touch_id, pos | art_pos) return touch_id
```

Zkontrolujeme jestli se dotek s identifikátorem `touch_id` nachází v uložených dotecích. Pokud je v uložených, najde si starou zprávu *Doteku* podle indentifikátoru, změní v ní polohu, časové razítko a odešle ji.

item_move - tažení prvku

```
item_move(item, by_pos | by_art_pos | dest_pos | dest_art_pos
| (dest_item [, dest_item_pos | dest_item_art_pos]) [, release_touch = True])
```

Přesun prvku na polohu určenou parametry. Na konci přetažení je prvek uvolněn. Uvolnění prvku se dá zabránit parametrem „release_touch“ nastaveným na False.

wait_for_ui_to_process_touches

```
wait_for_ui_to_process_touches(time_out=5000)
```

Metoda pomocí *QCoreApplication.processEvents*[4] počká na zpracování všech čekajících událostí (na př. pohyby elementů scény, kliknutí na prvky, signály a další) v aktuálním vlákně.

wait_for_ui_to_be_ready

```
wait_for_ui_to_be_ready(ui [, time_out=5000])
```

Zkontroluje jestli je `ui` - grafické prostředí ARTable ve stavu připraveném k použití. Když není, čeká pomocí *wait_signal* z pytest-qt na signál o změně stavu a když je nový stav typu „připraven k použití“, dovolí pokračovat v testování dále.

Čekání je neblokující a je maximálně po dobu pěti sekund nebo dobu určenou parametrem `time_out`.

[4] https://doc.qt.io/qt-5/qcoreapplication.html#processEvents

5.4 Použití nástroje

Nástroj lze využít na více úrovních testování (2.3), vše záleží pouze na zvoleném `roslunch` souboru pro testy. Zde si můžeme definovat zda-li je jedná o unit testování, kde bude testován pouze jeden uzel, nebo o integrační testování při kterém bude testováno zapojení nového uzlu do ARTable a další.

Jak postupovat při instalaci nástroje a jak ověřit jeho správné zapojení najdeme v souboru `README` v příloze A.

Níže uvedené příklady jsou pouze vymyšlené ukázky použití, mají pouze demonstrovat jak by se dal nástroj využít. Reálné příklady testů jsou k nalezení v příloze A.

Příklad

Máme vytvořenou novou komponentu jménem *ProgramoveMenu*. Jedná se o menu, které obsahuje seznam programů možných ke spuštění na ARTable. Potřebujeme otestovat zda menu správně reaguje na výběr prvků ze seznamu a ukazuje nám adekvátní tlačítka.

Kód testu

Níže jsou uvedeny pouze důležitější části kódu s vysvětlením jaký problém při testování nastává a jak je řešen.

Definujeme si část testu s názevem „vyber_prvniho_prvku“ a vyžádáme si po pytestu ARTBota.

```
1  def test menu_vyber_prvniho_prvku(ARTBot):
```

Inicializujeme grafické rozhraní `ARTable`.

```
2      rospy.init_node('projected_gui', anonymous=True)
3      ui = UICoreRos(InstructionsHelper(), 'cs_CZ')
```

Zde nastává problém, protože inicializace *UICoreRos* je již dokončená, ale systém ještě není ve stavu k možnému použití. Pomocí čekání na signál o změně stavu a kontrole typu nového stavu vyřešíme tento problém. Zmíněnou funkčnost zajišťuje metoda *wait_for_ui_to_be_ready*

```
4      ARTBot.wait_for_ui_to_be_ready(ui, 15000)
```

Předáme ARTBotu grafickou scénu, aby mohl mezi prvky vyhledávat a převádět souřadnicový systém.

```
5      ARTBot.set_art_table_scene(ui.scene)
```

Vybereme ze scény prvek typu *ProgramoveMenu* a jeho první položku.

```
6      program_list = ARTBot.find_item(type=ProgramoveMenu)
7      assert program_list is not None
8
9      item = program_list.list.items[0]
10     assert item is not None
```

Simulujeme stisknutí a uvolnění první položky.

```
11      ARTBot.item_press_and_release(item)
```

Poslání zprávy o stisknutí a zpracování není okamžité. Zde nastává další problém, nemůžeme testovat stav menu před zpracováním doteku. Metoda *wait_for_ui_to_process_touches* řeší problém se zpracováním a čeká než jsou zpracovány všechny Qt signály (viz. zpracování doteku4.1).

```
12      ARTBot.wait_for_ui_to_process_touches()
```

Nyní když jsou všechny doteky zpracovány, můžeme otestovat vybranou položku z menu a zda-li tlačítko „spustit program“ vidíme.

```
13      assert program_list.list.selected_item_idx == 0
14      assert program_list.start_program.isEnabled()
```

Rosclunch

V `roslunch` souboru (ukázka kódu5.6) definujeme pomocí „test_module“ hodnoty parametru umístění souboru obsahujícího kód testu. Dále spustíme pytestový test pomocí značky *<test>* a atributů *type* a *pkg*, které určují název balíčku s názvem spustitelného kódu.

Ukázka kódu 5.6: Určení vstupního souboru pro testování

```
<param name="test_module"
       value="$(find art_projected_gui)/tests/test_ui_program_menu.py" />
<test  type="ros_pytest_runner" pkg="ros_pytest"
       test-name="pytest_test_ui_program_menu" />
```

5.5 Testování

Řešení je složeno z více částí (spouštěč pytestu, ARTBot), testování tak probíhalo v několika iteracích. Jednotlivé iterace zde představují přidání nové vlastnosti nebo zapojení nástroje. Na konci každé iterace byl proveden ruční test a pokud bylo možné, byl doplněn i o test automatický. Výsledkem je soubor testů, které testují správné zapojení nástroje a jeho správnou funkčnost.

Rozšíření ARTBot vyžaduje ke své funkčnosti nástroj pytest a části ARTable. Před spuštěním je tak provedena kontrola, která má za úkol zjistit zda-li je k dispozici pytest a jestli se nachází ve stejném pracovním prostředí ROS i ARTable, konkrétně jeho definice zprávy `Doteku` a obecný prvek grafické scény - Item.

Spouštěč pytestu

K testování správné funkčnosti spouštěče pytestu byl vytvořen automatický test, který kontroluje:

- **Zpracování argumentů** - Argumenty mohou být omylem zadány špatně nebo nemusejí obsahovat určení výstupního souboru.
- **Přidávání rozšíření** - Vytvořená rozšíření mohou obsahovat chyby, nástroj by měl umět na takový stav reagovat.
- **ROS parametry** - Umístění externích rozšíření nemusí existovat nebo obsahovat chybu.

Testování spouštěče dopadlo úspěšně, splněním všech vytvořených testů.

ARTBot

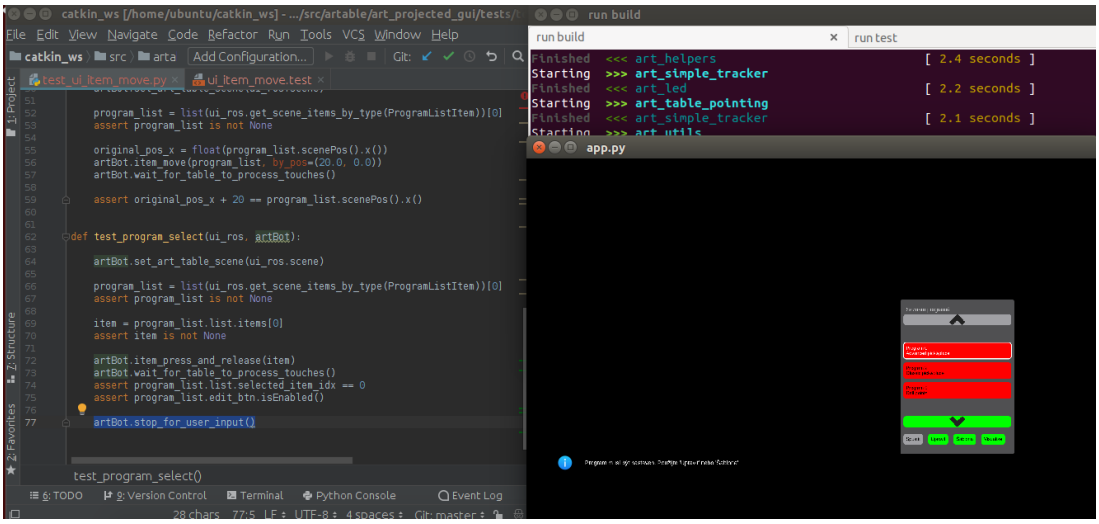
Kontrola správné funkčnosti ARTBota je rozdělena do tří částí:

- **Simulace doteků** - K ověření byl vytvořen soubor testů, který využil potomka ARTBota, upraveného aby neposílal zprávy. Kontrola tak probíhá na základě porovnání uložených dotyků v `_touchs` atributu ARTBota.
- **Hledání prvků** - Pomocí vytvořené grafické scény je ověřováno jestli je ARTBot schopný najít prvky s různými vlastnostmi.
- **Čekání na ARTable** - Testování probíhá pomocí simulace odesílání signálů podobným z ARTable.

Zmíněné části byly ověřeny automatickými testy, které jsou k nalezení v příloze A. Testování ARTBota dopadlo úspěšně.

Zapojení nástroje do ARTable

Správná funkčnost nástroje při zapojení do ARTable byla ověřena ručně pomocí ukázkových testů a kontroly správnosti jejich provedení. Jak takovéto testování probíhalo je k vidění na obrázku 5.3. Na levé straně se nachází připravený spuštěný test, na straně pravé můžeme vidět grafickou část ARTable. Testy jsou přiloženy v příloze A.



Obrázek 5.3: Ukázka ručního testování zapojení ARTBota do ARTable

Kapitola 6

Závěr

Práce si kladla za cíl nalezení řešení pro automatické testování uživatelského prostředí pro ARTable. V práci jsem analyzoval chování při zpracování dotyků a na základě těchto zkušeností navrhl a implementoval rozšíření, které simuluje uživatelskou interakci na dotekovém stolu.

Při této práci jsem se seznámil s tvorbou softwaru pro roboty za pomoci frameworku ROS, jakým stylem ARTable zpracovává doteky a vytváří uživatelské prostředí. Grafické rozhraní je z důvodu použití dotykového stolu tvořeno velice specificky, nebylo tak možné použít samostatně žádný testovací nástroj. Na základě této skutečnosti jsem navrhl a implementoval rozšíření s názvem ARTBot. V tomto rozšíření jsem vyřešil problematiku simulace uživatelské interakce. Je tak spolu s testovacím nástrojem možné využít k testování grafického rozhraní aplikací v ARTable. Jeho funkčnost a zapojení byly ověřeny automatickým a ručním testováním.

Pokračováním této práce by dále mohlo být přidání schopnosti simulovat různá dotyková gesta, která využívají více prstů zároveň. Příklady takovýchto gest známe třeba z dotykových telefonů. Jedním z nejpoužívanějších je Pinch-to-zoom, kdy k zvětšení fotky používáme dva prsty, které od sebe vzdalujeme.

Díky této práci jsem nabyl nových vědomostí nejen v oblasti testování, ale i tvorby softwaru pro roboty a mírně jsem nahlédnul do možné budoucnosti, jak by se mohli ovládat roboti bez odborných znalostí.

Literatura

[1] A. Havlíčková, a. P. R.: *Řízení kvality softwaru*. Computer Press, 2013, ISBN 978-80-251-3816-8.

[2] *ARTable github*. [Online; navštíveno 09.04.2019].
URL https://github.com/artable-dev/artable

[3] *ARTable na VUT FIT - obzázek*. [Online; navštíveno 07.05.2019].
URL http://www.fortes.cz/portfolio/artable-na-vut-fit-2/

[4] Blankemeyer, S.; Wiemann, R.; Posniak, L.; aj.: Intuitive Robot Programming Using Augmented Reality. *Procedia CIRP*, ročník 76, 2018: s. 155 – 160, ISSN 2212-8271, doi:https://doi.org/10.1016/j.procir.2018.02.028, 7th CIRP Conference on Assembly Technologies and Systems (CATS 2018).
URL http://www.sciencedirect.com/science/article/pii/S2212827118300933

[5] Borovcová, A.: *Testování webových aplikací*. [Online; navštíveno 31.03.2019].
URL https://is.cuni.cz/webapps/zzp/detail/46536/

[6] E., A. A.; Akan, B.; Çürüklü, B.: Augmented Reality Meets Industry: Interactive Robot Programming. In *SIGRAD 2010*, November 2010, s. 55–58.
URL http://www.es.mdh.se/publications/2241-

[7] Gao, Y.; Huang, C.-M.: PATI: a projection-based augmented table-top interface for robot programming. 03 2019, doi:10.1145/3301275.3302326.

[8] *JUnit definice*. [Online; navštíveno 11.05.2019].
URL https://www.ibm.com/support/knowledgecenter/en/SSQ2R2_14.0.0/com.ibm.rsar.analysis.codereview.cobol.doc/topics/cac_useresults_junit.html

[9] Ing. Bohuslav Křena, P., Ph.D. a Ing. Radek Kočí: *Úvod do softwarového inženýrství - Studijní opora*. [Online; navštíveno 28.03.2019].
URL https://wis.fit.vutbr.cz/FIT/st/cfs.php?file=%2Fcourse%2FIUS-IT%2Ftexts%2FIUS_opora.pdf&cid=10355

[10] Mgr. Jiří MARTINŮ a doc. Ing. Petr ČERMÁK, P.: *Metodiky vývoje software*. [Online; navštíveno 04.04.2019].
URL https://mvso.cz/wp-content/uploads/2018/02/Metodiky-v%C3%BDvoje-software-studijn%C3%AD-text.pdf

[11] ISO/IEC: *Definice normy ISO/IEC 25010*. [Online; navštíveno 30.03.2019].
URL http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[12] Joseph, L.: *Mastering ROS for Robotics Programming*. Packt Publishing, December 2015, ISBN 9781783551798.
URL https://www.packtpub.com/hardware-and-creative/mastering-ros-robotics-programming

[13] *HoloLens od Microsoftu - obzázek*. [Online; navštíveno 10.05.2019].
URL https://www.microsoft.com/cs-cz/p/microsoft-hololens-development-edition/8xf18pqz17ts

[14] Mizell, D.: Boeing"s Wire Bundle Assembly Project. 01 2001, [Online; navštíveno 10.04.2019].
URL https://www.researchgate.net/publication/245588238_Boeing''s_Wire_Bundle_Assembly_Project

[15] Ong, S. K.; W. S. Chong, J.; Nee, A.: Methodologies for immersive robot programming in an augmented reality environment. *IJVR*, ročník 6, 01 2007: s. 237–244, doi:10.1145/1174429.1174470.

[16] Pai, Y. S.; YAP, H. J.; Singh, R.: Augmented reality–based programming, planning and simulation of a robotic work cell. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, ročník 229, 05 2014, doi:10.1177/0954405414534642.

[17] PATTON, R.: *Testování softwaru*. Computer Press, 2002, ISBN 80-7226-636-5.

[18] *Pytest dokumentace*. [Online; navštíveno 25.03.2019].
URL
https://buildmedia.readthedocs.org/media/pdf/pytest/latest/pytest.pdf

[19] Regenbrecht, H.; Baratoff, G.; Wilke, W.: Augmented reality projects in the automotive and aerospace industries. *IEEE Computer Graphics and Applications*, ročník 25, č. 6, Nov 2005: s. 48–56, ISSN 0272-1716, doi:10.1109/MCG.2005.124.

[20] *Handling of setup.py and Using package.xml in setup.py*. [Online; navštíveno 10.11.2018].
URL
http://docs.ros.org/jade/api/catkin/html/user_guide/setup_dot_py.html

[21] *Installing Python scripts and modules*. [Online; navštíveno 10.11.2018].
URL
http://docs.ros.org/api/catkin/html/howto/format2/installing_python.html

[22] *ROS Dokumentace*. [Online; navštíveno 09.02.2019].
URL http://wiki.ros.org/Distributions

[23] *Testování software - Wikipedie*. [Online; navštíveno 30.03.2019].
URL https://en.wikipedia.org/wiki/Software_testing

[24] Rössler, A.: *Testing ROS powered Robots with pytest*. [Online; navštíveno 10.11.2018].
URL https://machinekoder.com/testing-ros-powered-robots-pytest/

[25] Toby Hartnoll Joshua Collett, B. A. M.: *An Augmented Reality Debugging System for Mobile Robot Software Engineers*. [Online; navštíveno 19.04.2019].
URL http://joser.unibg.it/index.php/joser/article/view/7

[26] Tomáš, H.: *Úrovně testování*. [Online; navštíveno 31.03.2019].
URL http://testovanisoftwaru.cz/tag/urovne-testovani/

[27] *Obrázek spirálového modelu - Wikipedia*. [Online; navštíveno 04.04.2019].
URL https://commons.wikimedia.org/wiki/File:Software_Development_Spiral_cz.svg

[28] *Obrázek Virtual Ficture - Wikipedia*. [Online; navštíveno 10.05.2019].
URL https://commons.wikimedia.org/wiki/File:Louis_Rosenberg_Augmented_Reality_Rig.png

[29] *Virtual fixture - Wikipedia*. [Online; navštíveno 10.04.2019].
URL https://en.wikipedia.org/wiki/Virtual_fixture

Příloha A

Obsah přiloženého paměťového média

- `/ros_pytest` - zdrojové kódy ROSovského uzlu nástroje ARTBot
- `/ARTable_test_example` - Příklady testů pro ARTable s využitím ARTBota
- `/thesis` - zdrojové kódy této práce
- `/thesis.pdf` - text práce