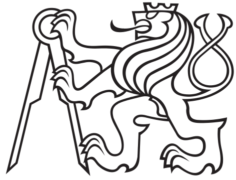
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Prototypování dotykových uživatelských rozhraní v automobilech

Markéta Hejná

Vedoucí: Ing. Ladislav Čmolík, Ph.D.
Obor: Interakce člověka s počítačem
Studijní program: Otevřená informatika
Leden 2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hejná** Jméno: **Markéta** Osobní číslo: **435011**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Interakce člověka s počítačem**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Prototypování dotykových uživatelských rozhraní v automobilech

Název diplomové práce anglicky:

Prototyping of touch user interfaces in cars

Pokyny pro vypracování:

Analyzujte různé typy prototypů, způsoby jejich vytváření a jejich možnosti pro ověření vzhledu a chování (look-and-feel) návrhu uživatelského rozhraní. Na základě analýzy zvolte typ prototypu, který umožní vytvořit a ověřit vzhled a chování uživatelského rozhraní na dotykové obrazovce umístěné na palubní desce automobilu. Prototyp musí podporovat dotykové ovládání, musí být schopen ukládat záznamy o akcích uživatele do strukturovaného textového souboru a musí být schopen odesílat a přijímat zprávy přes TCP protokol. Zprávy přes TCP budou odesílány při provedení specifických akcí uživatele. Na přijaté zprávy bude uživatelské rozhraní reagovat změnou stavu definovaných uživatelských prvků (např. změna stavu přepínače či posunutí vybraného prvku v seznamu). Formát odesílaných a přijímaných zpráv navrhněte. Zvolte nástroj pro vytváření prototypů, do kterého lze takovou funkčnost doimplementovat. Dále se zaměřte na to, aby způsob vytváření prototypu uživatelského rozhraní a definice jeho vzhledu a chování byly co nejjednodušší a bylo možné prototyp vytvářet vizuálně pomocí WYSIWYG (What you see is what you get) metody. Funkčnost výsledného nástroje ověřte vytvořením alespoň tří prototypů uživatelských rozhraní, jejichž vzhled a chování bude odpovídat uživatelským rozhraním ve třech rozdílných automobilech. Pomocí každého z nich bude možné splnit alespoň 3 středně složité úlohy (např. najít v adresáři definovanou osobu a zahájit s ní telefonický hovor).

Seznam doporučené literatury:

Arnowitz J., M. Arent, and N. Berger, Effective Prototyping for Software Makers. Elsevier Science & Technology, 2007.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Ladislav Čmolík, Ph.D., Katedra počítačové grafiky a interakce

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

Datum zadání diplomové práce: **14.02.2019** Termín odevzdání diplomové práce: **07.01.2020**

Platnost zadání diplomové práce: **20.09.2020**

Ing. Ladislav Čmolík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.

Datum převzetí zadání	Podpis studentky

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování

Děkuji vedoucímu mé práce Ing. Ladislavu Čmolíkovi, Ph.D., za odborné vedení a cenné rady. Také děkuji celé rodině za podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 7. ledna 2020

Abstrakt

Tato práce se zabývá návrhem způsobů pro zjednodušení tvorby prototypů dotykových rozhraní automobilů. Na základě seznamu požadavků na vlastnosti takových prototypů jsem analyzovala několik technologií pro návrh uživatelských rozhraní a vybrala takovou, pomocí které se dají navržené způsoby implementovat. Výsledkem je knihovna tříd a sada doporučení, které prototypování usnadňují. Vzniklé poznatky jsou demonstrovány na implementaci prototypu rozhraní reálného automobilu.

Klíčová slova: prototyp, automobilové rozhraní, uživatelské rozhraní, auto, JavaFX, FXML

Abstract

The aim of this thesis is to design ways to simplify prototyping of touch car interfaces. Based on a set of requirements of these prototypes' properties, I analysed several technologies used for user interface design and chose the most suitable one. The resulting library and recommendations offer easier prototyping. The results are demonstrated on an implementation of a prototype of real car's user interface.

Keywords: prototype, touch interface, user interface, car, JavaFX, FXML

Obsah

1 Úvod 1

1.1 Cíle práce 1

1.2 Struktura práce 3

2 Analýza a návrh 5

2.1 Požadavky na prototyp dotykového rozhraní v automobilu 5

2.1.1 Analýza podkladů k prototypu 7

2.2 Prototyp 8

2.2.1 Rozdělení prototypů podle úrovně věrohodnosti 9

2.2.2 Rozdělení prototypů podle úrovně zpětné vazby 12

2.2.3 Shrnutí 13

2.3 Rešerše technologií 14

2.3.1 HTML5 14

2.3.2 XAML 16

2.3.3 JavaFX 17

2.3.4 QML 18

2.3.5 Shrnutí technologií 19

2.3.6 Vybraná technologie 19

2.4 Návrh architektury 20

2.5 Návrh komunikace s hardwarem 21

2.6 Návrh vlastních UI komponent . 23

3 Implementace 25

3.1 Použitý software 25

3.2 Struktura JavaFX FXML aplikace 26

3.3 Knihovna pro zjednodušení výroby prototypů 30

3.3.1 Server 30

3.3.2 Client.java 32

3.3.3 AudioController.java 34

3.3.4 StageController.java 34

3.4 Implementace vlastních komponent 35

4 Práce s nástrojem Scene Builder 39

4.1 Použití Scene Builderu při tvorbě obrazovek prototypu 41

5 Výsledky **47**

5.1 Shrnutí poznatků a postupů 49

5.1.1 Finální podoba zasílaných zpráv 50

6 Závěr **53**

6.1 Možnosti rozšíření 54

A Obsah přiloženého CD **55**

B Literatura **57**

Obrázky

1.1 Simulátor automobilu [1] 2

2.1 Příklad Low-Fidelity prototypu [2] 10

2.2 Příklad papírového Low-Fidelity prototypu [3] 11

2.3 Příklad High-Fidelity prototypu [2] 12

2.4 Zaškrtávací pole před a po změně jeho "look-and-feel" na switch button. 14

2.5 Příklad architektury s více klienty 21

3.1 Struktura UI prvků JavaFX aplikace 27

3.2 GUI serveru 31

4.1 GUI nástroje Scene Builder 39

4.2 Nabídka pro vložení vlastních UI komponent 40

4.3 Defaultní radio button 42

4.4 Radio button po otočení o 180 stupňů 42

4.5 Radio button po otočení textu o 180 stupňů 42

4.6 Radio button po aplikování všech stylů 43

4.7 Ukázka polí pro zadání hodnoty property v nástroji Scene Builder . 44

4.8 Hiearchie prvků komponenty *SingleSliderControl* 45

4.9 Vlastní komponenta *SingleSliderControl* 45

4.10 Vlastní komponenta *SingleSliderControl* bez úpravy vzhledu pomocí CSS 45

4.11 Celý layout obrazovky *CarSettings2.fxml* 46

Tabulky

2.1 Shrnutí analyzovaných technologií a jejich vlastností v porovnání s požadavky na prototyp 19

Kapitola 1

Úvod

1.1 Cíle práce

V této práci se budu zabývat návrhem způsobů, jak zjednodušit a urychlit vytváření prototypů dotykových rozhraní v automobilech. Nejprve zanalyzuji vybrané technologie, které se využívají k tvorbě uživatelských rozhraní, a poté je vyhodnotím na základě předem daných požadavků, které výsledná technologie musí splňovat, aby vyhovovala potřebám této práce. Pomocí nejlépe hodnocené technologie naleznu postupy, které mohou návrh rozhraní zjednodušit a zaznamenám je, čímž vznikne sada doporučení pro usnadnění vytváření podobných prototypů. Dále vytvořím knihovnu, která bude některé postupy zjednodušující návrh uživatelského rozhraní implementovat. Výsledky práce demonstruji na prototypu existujícího automobilového dotykového rozhraní.

Během práce se ukázalo, že nebude možné obdržet podklady pro tři různá automobilová rozhraní, jak je avizováno v zadání práce. Po dohodě s vedoucím bude místo prototypů tří menších rozhraní realizován prototyp pouze jednoho rozhraní, avšak rozsáhlejšího a se složitějšími prvky.

Dotyková rozhraní jsou široce využívána v moderních technologiích. Kromě chytrých telefonů, tabletů, počítačových monitorů a hodinek se používají i u tiskáren, praček, lednic a dalších přístrojů. Dotykové obrazovky se rozšířily také do interiérů automobilů, kde se jejich pomocí mohou ovládat různé funkce automobilu, jako je GPS navigace, audiosystém, klimatizace, parkovací kamera

a další. Moderní funkce v automobilech mají nesporné výhody. Dotyková obrazovka umožňuje využití více funkcí a zároveň zabraňuje přehlcení palubní desky tlačítky, pomocí kterých by se jinak tyto funkce musely ovládat. Takové funkce mohou být spíše nadstandardní, ale některé z nich, jako např. parkovací kamera, mohou lidem výrazně ulehčovat řízení. Zároveň je však nutné si uvědomit problémy, které používání dotykových obrazovek během řízení způsobuje. Při návrhu rozhraní se musí dbát na to, aby nedošlo ke zlepšení funkcí automobilu na úkor bezpečnosti řidiče a spolujezdců. Protože interakce s dotykovou obrazovkou poskytuje pouze vizuální odezvu a nikoli haptickou, řidič potřebuje vizuálně ověřit správnost jeho interakcí s rozhraním, což odvádí jeho pozornost od vozovky a rozptyluje ho od řízení. Nevhodně umístěná obrazovka či prvky na obrazovce, nebo i jen nevhodně zvolená velikost textu může mít negativní vliv na kognitivní zátěž řidiče. Tím se prodlužuje jeho interakce s rozhraním a zvyšuje se tak riziko dopravní nehody. Z tohoto důvodu je nutné správnost návrhu rozhraní a interakci řidiče s rozhraním otestovat v nějakém bezpečném prostředí, které co možná nejvíce napodobuje prostředí při řízení. K tomu slouží simulátory automobilu. Zpravidla obsahují volant a nějaké zobrazovací zařízení, které simuluje okolní venkovní prostředí. Mohou také napodobovat kabinu řidiče včetně přístrojové desky a řadicí páky. Příklad takového simulátoru se nachází na obrázku 1.1. Tento simulátor nabízí realistické simulované prostředí řízení auta včetně vizuálního a zvukového systému. Používá se pro trénování začínajících řidičů, kteří si mohou osvojit základy řízení bez obav, že případné chyby budou mít následky [1].

V našem případě simulátor obsahuje také dotykovou obrazovku se spuštěným prototypem, který simuluje dotykové rozhraní ovládající nejrůznější funkce automobilu, například rádio, nastavení klimatizace a podobně. Dále obsahuje hardwarová tlačítka, která některé z těchto funkcí ovládají také. V simulátoru uživatel-řidič řídí a přitom dostává úkoly, které má v uživatelském rozhraní provést. Testuje se, zda je rozhraní pro uživatele dostatečně srozumitelné a zda se při ovládání rozhraní projeví zvýšená zátěž způsobená řízením. V případě nalezení nějakých problémů je potřeba je po ukončení testování opravit, k čemuž je vhodné, aby technologie použitá pro výrobu prototypu takové úpravy umožňovala.

Obrázek 1.1: Simulátor automobilu [1]

1.2 Struktura práce

Následující text práce je strukturován do pěti kapitol. Druhá kapitola se zabývá představením požadavků na prototyp dotykového rozhraní v automobilu a typů prototypů, z kterých na základě požadavků vyberu jeden vhodný pro tuto práci. Dále analyzuji možné technologie pro implementaci prototypů, přičemž vybraná technologie bude splňovat dané požadavky. Také v kapitole navrhnu architekturu aplikace a komunikaci prototypu s hardwarovými zařízeními simulátoru. Ve třetí kapitole blíže popíši vybranou technologii a postup při implementaci. Ve čtvrté kapitole demonstruji, jak lze pomocí zvolené technologie vizuálně tvořit grafická uživatelská rozhraní a využívat přitom vlastní vytvořené UI[1] komponenty. V páté kapitole rekapituluji získané poznatky o vývoji prototypu a závěrem v poslední kapitole shrnu výsledky celé práce.

[1] UI = user interface, česky uživatelské rozhraní

Kapitola 2

Analýza a návrh

V této kapitole analyzuji různé technologie s ohledem na požadavky popsané v předchozí kapitole. Na základě této analýzy vyberu takovou technologii, která všechny požadavky splňuje. Následně navrhnu architekturu aplikace a podobu komunikace s hardwarem.

2.1 Požadavky na prototyp dotykového rozhraní v automobilu

Cílem práce je vytvořit metodu pro snadnou a jednoduchou výrobu prototypů dotykových rozhraní využívaných v automobilech. Tato metoda by měla být co nejuniverzálnější, aby mohla být opakovaně použita pro různé prototypy a adaptovatelná na jejich individuální vlastnosti (tzv. "look-and-feel") s pouze minimálními úpravami. Na cílovou technologii máme následující požadavky:

- **Look-and-feel** Zvolená technologie musí umožňovat výrobu prototypů různých automobilových rozhraní. Proto musí být možné měnit jejich "look-and-feel", stejně jako se liší v různých automobilech. Pojem "look-and-feel" (vzhled a dojem) obsahuje mnoho aspektů uživatelského rozhraní prototypu. Mezi vzhledové patří například barvy, rozložení prvků nebo typ užitého písma. Pocitové aspekty jsou ty, které se týkají chování uživatelského rozhraní, dynamických prvků prototypu a jejich interaktivity. Daná technologie by proto měla umožňovat snadnou změnu vzhledu

základních elementů uživatelského rozhraní, jako jsou tlačítka, posuvníky, seznamy, zaškrtávací pole apod. Tyto základní prvky by buď měly být u vybrané technologie k dispozici, nebo by mělo být snadné je vytvořit.

- **WYSIWYG** S ohledem na jednoduchost by další velkou výhodou byla možnost navrhovat jednotlivé prvky vizuálně v nějakém WYSIWYG (What You See Is What You Get) editoru, bez nutnosti kódování. Takový editor by byl příhodný například pro snadnější a přesnější pozicování elementů a celkové urychlení prototypování.

- **Windows OS** Prototypy vzniklé vytvořenou metodou budou testovány v simulátoru na zařízení s operačním systémem Windows a dotykovým monitorem. Zvolená technologie proto musí fungovat na této platformě. Preferovaná je technologie umožňující výrobu prototypů nezávislých na platformě.

- **Vícedotyková gesta** Důležitou vlastností vytvářených prototypů je detekce dotykových událostí. Výsledné prototypy musí být ovladatelné na dotykové obrazovce a být schopné implementovat potřebná gesta, což kromě běžného kliknutí na tlačítko může být přibližování a oddalování dvěma prsty. To může být potřeba například u integrované GPS navigace pro manipulaci s mapou. Pro takové případy bude muset vybraná technologie umět detekovat více dotyků najednou a tato gesta správně interpretovat. Preferovaná je taková technologie, která tato vícedotyková gesta umí rozpoznat automaticky bez nutnosti další implementace.

- **Multimédia** Podstatnou součástí automobilových rozhraní je audiosystém. Prototyp proto bude muset umět přehrávat hudební soubory. Přehrávání audia by nemělo být přerušeno při přechodu na jinou obrazovku rozhraní, stejně jako tomu tak je v reálných automobilech.

- **HW komunikace** Dalším požadavkem je schopnost prototypu komunikovat s hardwarovými zařízeními, která jsou také součástí simulátoru. Tato zařízení jsou schopna komunikovat pomocí protokolu TCP zasíláním zpráv. Některé prvky obrazovek prototypu budou ovládány nejen dotykem přímo na obrazovce, ale také hardwarovými ovladači na palubní desce či na volantu. Elementy proto musí měnit své vlastnosti v reakci na dotyk i na zprávy přijaté z takového zařízení. Mělo by být možné prototyp ovládat pomocí šipek klávesnice, protože akce provedené použitím ovladačů mohou být namapovány na stisky kláves.

- **Logování** Pro účely testování bude také významné vytváření logů o akcích provedených v prototypu. Technologie by proto měla umožňovat ukládání zpráv o uživatelských akcích do souboru.

2.1.1 Analýza podkladů k prototypu

Jako podklady k práci jsem obdržela PDF soubor s fotografiemi automobilového rozhraní. Fotografie byly pořízeny v reálném automobilu. Z jednotlivých prvků na fotografiích vedou šipky k dalším fotografiím. To naznačuje stav rozhraní poté, co uživatel s daným prvkem interaguje. Změnou stavu může být přesun na jinou obrazovku nebo změna hodnoty vlastnosti nějakého prvku, např. hodnoty posuvníku nebo text tlačítka. Ne všechny prvky na fotografiích byly takto označeny a některé obrazovky se typově opakovaly, proto jsem z poskytnutých fotografií vybrala podčást, kterou využiji v návrhu a následné implementaci.

Po prohlédnutí podkladů jsem dospěla k několika závěrům. Rozhraní obsahuje prvky, které na všech obrazovkách, na kterých se vyskytují, vypadají naprosto totožně. Dále obsahuje prvky, které jsou typově stejné, ale na každé obrazovce mají jiný obsah. V rozhraní se také vyskytuje několik speciálních prvků, které se navíc opakují na více místech. Pro usnadnění prototypování by proto bylo vhodné tyto prvky rozšířením existujících UI komponent vytvořit jako samostatné komponenty (tzv. custom components), které se budou na obrazovku přidávat stejným způsobem jako ony existující prvky, např. tlačítko nebo posuvník.

Také bude třeba vytvořit mechanismus, kterým se budou přepínat obrazovky prototypu a zároveň zachovávat jejich aktuální nastavení. Pokud na některé obrazovce např. zaškrtneme zaškrtávací pole, chceme, aby během jednoho chodu aplikace zůstalo toto pole zaškrtnuté i po odchodu a následném návratu na tuto obrazovku. Stejný požadavek platí i pro přehrávání audio souborů.

Protože musíme umožnit ovládání prototypu pomocí šipek, nestačí pouze vytvořit obrázky daných obrazovek, ale budou muset být naprogramovány prvky, s nimiž se dá interagovat. V rámci přípravy na implementaci jsem fotografie z PDF souboru uložila a z každé fotografie za pomoci grafického editoru vyřezala obrázky a ikony tlačítek. Protože se u prototypů nepředpokládá obzvlášť pokročilá funkcionalita, prvky jako např. horní lišta s časem nebo analogové hodiny mohou být řešeny pouze obrázkem. Pokud by uživatel během testování v simulátoru měl od moderátora testu za úkol zjistit, kolik je hodin, je podstatné, aby uživatel věděl, kde má takovou informaci v prototypu hledat, nikoliv, jaký je právě aktuální čas. Implementace takových prvků by oslabila výhodu prototypování, která spočívá v rychlé produkci řešení problému bez nutnosti toho, aby vše bylo stoprocentně funkční a obsahovalo vnitřní logiku.

2.2 Prototyp

Prototyp v základním významu znamená první příklad něčeho, nějakého produktu, ze kterého vycházejí jeho pozdější verze [4]. Za prototyp tedy můžeme označit počáteční verzi produktu, která slouží k ověření klíčových parametrů produktu.

V oblasti návrhu softwarových produktů je prototypování velice silným nástrojem. Je klíčové k vytvoření úspěšného softwarového produktu a k dosažení kvalitní uživatelské zkušenosti (anglicky *user experience*). Prototypy zpravidla neobsahují mnoho skutečné funkcionality, pouze ji napodobují. To umožňuje (v závislosti na rozsahu daného projektu) relativně rychlou a především levnou výrobu prototypů, na kterých se dají otestovat různé funkcionality, aniž by se věnovalo mnohem více času na výrobu plně funkčního, avšak potenciálně nedostatečného a chybného řešení. Daleko méně času a finančních prostředků se věnuje na změnu částí prototypu či vytvoření úplně nového než na reimplementaci špatně navrženého a již dokončeného finálního produktu. Prostřednictvím prototypu můžeme vizualizovat softwarové požadavky a dát tak lepší náhled na jejich správnost a vhodnost zpracování. Prototypování je dobrý způsob pro rychlé otestování různých nápadů, z nichž posléze jeden nebo více vybereme a rozpracujeme dál, a pro lepší představu toho, jak bude vypadat konečný produkt.

Dalším důvodem k vytváření těchto prototypů je snaha ověřit použitelnost aplikace (snadnost používání aplikace a dosažení požadovaných cílů), způsoby interakce, informační architekturu (způsob rozmístění informací v aplikaci) a celkovou uživatelskou zkušenost [5, 6]. Pro zjištění nedostatků v těchto oblastech je vhodné prototypy testovat s uživateli. K testování prototypu produktu by měli být přizváni reální uživatelé, tedy lidé, kteří již podobný produkt používají, nebo u kterých se předpokládá, že by reálný produkt skutečně využívali, protože jejich vlastnosti, zájmy a potřeby jsou shodné s cílovou skupinou. Kromě uživatele, tzv. participanta, je u testování přítomen také moderátor. Uživatelé dostávají od moderátora pokyny, jaké úkoly mají v testovaném prototypu provést. Takovými úkoly jsou reálné akce, které by uživatel mohl v systému provádět. Příkladem může být vytvoření nového záznamu, nalezení záznamu nebo i změna hesla. Účastníci testování jsou moderátorem požádáni k tomu, aby tzv. "přemýšleli nahlas", to znamená, aby nahlas popisovali, co právě v systému provádějí, co plánují udělat a jaké mají ohledně prototypu myšlenky. To napomáhá tvůrcům systému k tomu, aby odhalili části, které mohou být uživatelům nesrozumitelné nebo nepohodlné, a navrhli je lépe [7].

Hlavním cílem vytváření prototypů je odhalit případné chyby v návrhu produktu v co možná nejranější fázi vývoje. Oprava chyb je tím nákladnější, čím později jsou během vývoje odhaleny [8]. Ideální je proto nalézt nedostatky již ve fázi návrhu, tedy ještě předtím, než jsou vůbec do výsledného produktu implementovány. K tomu je proto vhodné při vytváření produktu využívat prototypy a těmto problémům se tak vyvarovat.

U prototypů můžeme rozlišovat dva základní způsoby rozdělení: podle úrovně věrohodnosti na

- Low-Fidelity prototyp, neboli málo věrohodný
- High-Fidelity prototyp, neboli vysoce věrohodný

nebo podle úrovně zpětné vazby na

- statický prototyp
- interaktivní prototyp.

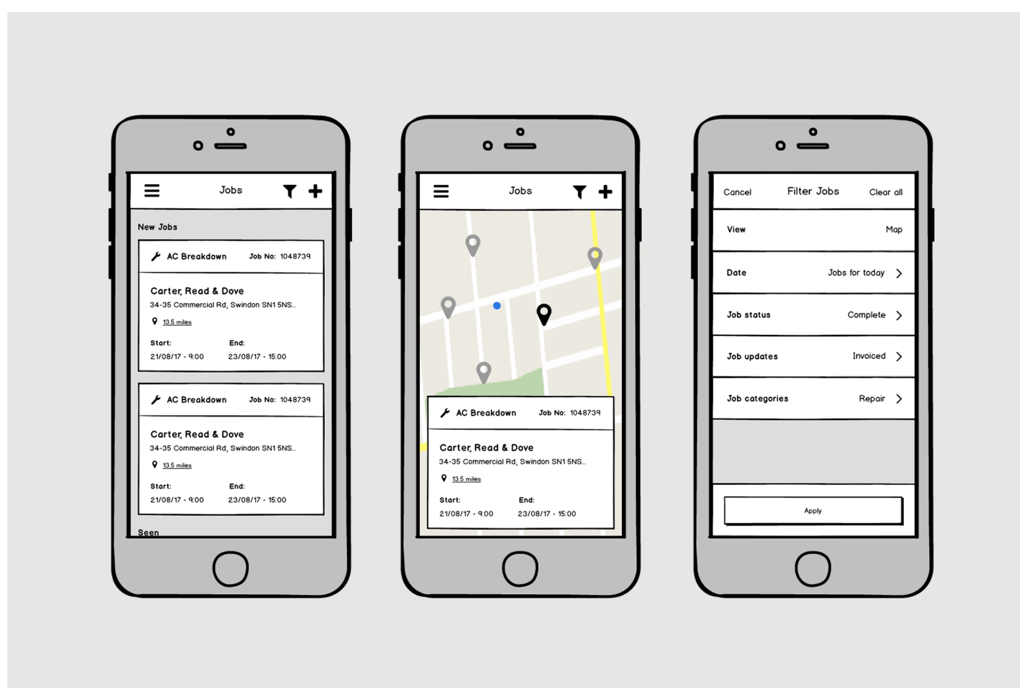
Tyto typy prototypů blíže popíši v následujících podkapitolách.

2.2.1 Rozdělení prototypů podle úrovně věrohodnosti

Věrnost nebo přesnost u prototypu představuje, jak moc prototyp odpovídá budoucí výsledné aplikaci, co se týče vzhledu a celkového pocitu, který uživatel z prototypu má. Tento ”look-and-feel” (vzhled a dojem) obsahuje mnoho aspektů uživatelského rozhraní prototypu. Mezi vzhledové patří například barvy, rozložení prvků nebo typ užitého písma. Pocitové aspekty jsou ty, které se týkají chování uživatelského rozhraní, dynamických prvků prototypu a jejich interaktivity [6].

Low-Fidelity prototyp. Low-Fidelity prototyp nepřipomíná skutečný produkt. Jeho podstatou je, aby byl snadno a rychle vytvořitelný, ideálně během několika hodin či dní, a bylo tak možné vyrobit více verzí, které se mezi sebou liší a mohou se tak v rané fázi vývoje otestovat různé přístupy k produktu.

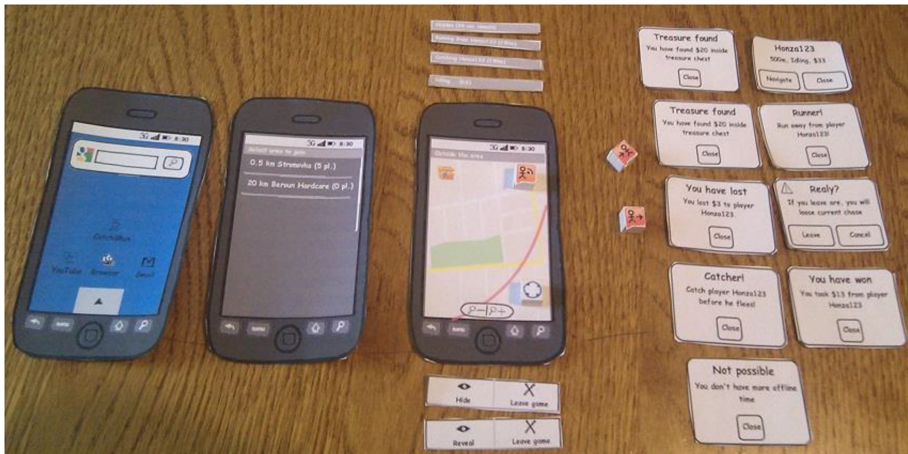
Takové prototypy mívají "načrtnutý vzhled", nebo jsou přímo vyrobené z papíru nebo z jiného vhodného levného materiálu. K vytváření softwarových low-fidelity prototypů slouží například nástroj Balsamiq[1]. Na obrázku 2.1 je příklad onoho "načrtnutého" prototypu. Není kladen důraz na design, cílem je zjistit, co všechno má aplikace obsahovat, zda na sebe obrazovky správně navazují, zda se uživatel nedostane do situace, kde nemůže jít zpět ani vpřed apod [9, 2].



Obrázek 2.1: Příklad Low-Fidelity prototypu [2]

Na obrázku 2.2 je příklad papírového prototypu. Tento prototyp může napodobovat základní interakce ručním vkládáním menších papírových částí, představující dialogová okna, do velkých částí, představující obrazovku telefonu.

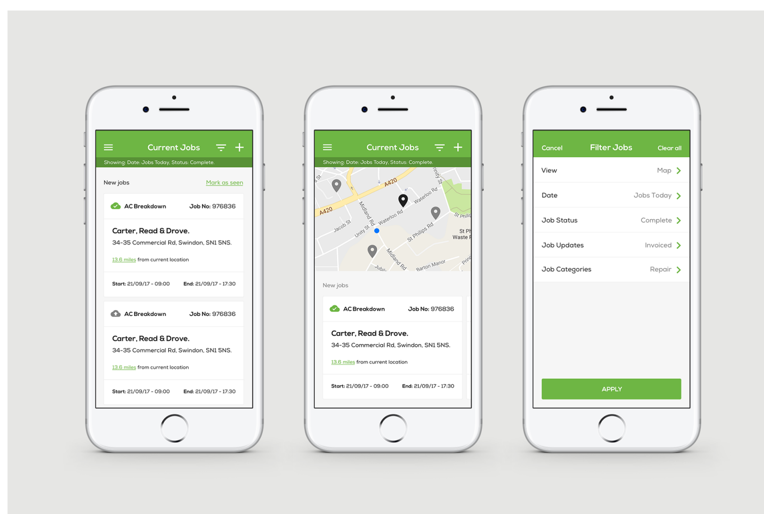
[1]https://balsamiq.com/



Obrázek 2.2: Příklad papírového Low-Fidelity prototypu [3]

High-Fidelity prototyp. High-Fidelity prototyp vytváří iluzi finálního vizuálního a interakčního designu. Aplikační logika nemusí být nutně implementována, jen nasimulována, mohou být použita ilustrační data. Při uživatelském testování je možné použít metodu zvanou Wizard of Oz (viz podkapitola Statický prototyp). Měla by však být implementována hlavní část uživatelského rozhraní aplikace, vizuální a interakční design by měl splňovat zásady doporučované cílovou platformou. Aplikace by též měla být testována na cílovém zařízení a rovněž implementovat gesta používaná k interakci na tomto zařízení, tedy mobilní aplikace bude testována na mobilním telefonu s případným využitím gest používaných na dotykových obrazovkách, nikoli na počítači a ovládaná myší. Tento typ prototypu je vhodný nejen k uživatelskému testování v laboratoři, ale i k testování v terénu v praxi. Protože výroba High-Fidelity prototypů je nákladnější, měla by jim předcházet výroba Low-Fidelity prototypů, na kterých se ověří zásadní vlastnosti aplikace, jak jsem již popsala v předchozí sekci [9, 10].

Na obrázku 2.3 se nachází High-Fidelity varianta prototypu z obrázku 2.1. Prototyp již nemá "načrtnutý" vzhled, vizuální design je zde propracovanější a připomíná finální produkt. Mapa na druhé obrazovce již není jen naznačená, ale jsou na ní vyobrazeny skutečné ulice. Prototyp také obsahuje více detailů oproti Low-Fidelity verzi.



Obrázek 2.3: Příklad High-Fidelity prototypu [2]

Hlavní rozdíly. Low-Fidelity prototyp je finančně i časově méně náročný na výrobu, proto je vhodný pro vytvoření mnoha potenciálních verzí vyvíjeného produktu a vyzkoušení různých možností. Poté se vytvoří jen několik či dokonce jeden High-Fidelity prototyp, který již využívá pouze nejvhodnější, ověřené přístupy otestované v předchozích Low-Fidelity prototypech. Vizuální a interakční design obou typů prototypů se liší v obsažení detailu a celkové propracovanosti, kdy Low-Fidelity prototyp nepřipomíná finální produkt a High-Fidelity prototyp naopak ano.

2.2.2 Rozdělení prototypů podle úrovně zpětné vazby

Statický prototyp. Statické prototypy obvykle obsahují několik vyobrazení uživatelského rozhraní, například nákresů. Zpětnou vazbu zde nezajišťuje prototyp sám od sebe, ale je ovládán někým, kdo je s jeho předpokládanou funkcionalitou dobře obeznámen, což může být například designer produktu. Tato osoba při uživatelském testování představuje "počítač" a reaguje na akce vykonávané uživatelem. Představím zde dvě takové metody, které se u testování statických prototypů využívají. Jedna z metod se nazývá Wizard of Oz, pojmenovaná podle postavy z románu Čaroděj ze země Oz. Stejně jako v tomto příběhu, kdy čaroděj zpoza opony interaguje se světem, je součástí uživatelského testování "čaroděj", někdo, kdo pracuje na vývoji testovaného produktu, a z jiné místnosti vzdáleně ovládá testovací zařízení. Na uživatelovy akce přímo manuálně reaguje a simuluje tak interaktivitu prototypu. Pokud například uživatel klikne na nějaké tlačítko v prototypu, "čaroděj" vyhodnotí, co se má stát a příslušnou obrazovku či data uživateli zobrazí. Uživatel přitom

neví, že neinteraguje přímo s počítačem [11].

Další metodou statického prototypování je vytváření papírových prototypů. Osoba zastupující "počítač" je přítomna u uživatelského testování, uživatel prstem poklepává na papírové obrazovky a "počítač" na uživatelovy akce reaguje přímou manipulací papírových částí prototypu [9].

Interaktivní prototyp. Interaktivní prototypy umožňují interakci uživatele s rozhraním. Běžné prototypovací nástroje (např. Sketch[2], Axure[12]) nabízí základní interakce, například přechod na jinou obrazovku po kliknutí na tlačítko a základní animace. Jsou-li potřeba otestovat pokročilejší interakce, nabízí se vyrobit kódovaný prototyp, který již může umožňovat zadávání uživatelského vstupu, validaci vstupu či gesta pro ovládání dotykových obrazovek.

Kódovaný prototyp. Kódované prototypy jsou vytvořené pomocí některého programovacího jazyka. Zpravidla obsahují high-fidelity uživatelské rozhraní a jsou vhodné pro zachycení pokročilejších vlastností prototypů. Dovolí uživateli s aplikací doopravdy interagovat bez nutnosti pouze si představovat některé prvky interakce, které v běžném prototypovacím nástroji nejsou možné zachytit (například umožňují vkládat skutečná a individuální data z uživatelského vstupu). Kódovaný prototyp je zpravidla programovaný v cílovém jazyce a umožňuje tak vývojářům opětovně použít části kódu ve finálním produktu [6].

2.2.3 Shrnutí

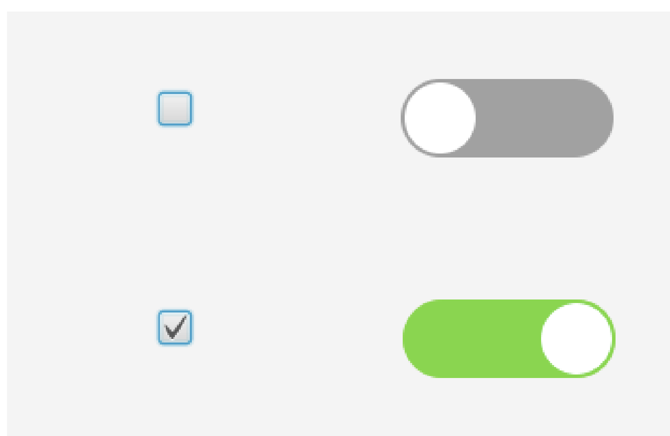
Protože budou prototypy využívány v simulátorech za účelem co nejpřesněji nasimulovat skutečné podmínky, ve kterých se řidiči při jízdě v automobilu nacházejí, včetně nejrůznějších interakcí s dotykovou obrazovkou, nejvhodnější volbou bude v tomto případě tvorba kódovaných High-Fidelity prototypů. Při tvorbě prototypů automobilových rozhraní je potřeba zajistit věrohodnost prototypu, což zajistí High-Fidelity prototyp.

[2] https://www.sketch.com/

2.3 Rešerše technologií

Cílem práce je vytvořit metodu pro snadnou a jednoduchou výrobu prototypů dotykových rozhraní využívaných v automobilech. Tato metoda by měla být co nejuniverzálnější, aby mohla být opakovaně použita pro různé prototypy a adaptovatelná na jejich individuální vlastnosti (výše zmíněný ”look-and-feel”) s pouze minimálními úpravami.

Z různých technologií, které existují, jsem vybrala několik nejvhodnějších k podrobnějšímu prozkoumání. Všechny tyto technologie je možné využít pro tvorbu kódovaných prototypů. Pro každou z nich ověřím, zda je schopna naplnit výše popsané požadavky, a nejvhodnější z nich použiji pro vytvoření prototypu. Pro srovnání u technologií přikládám úryvek kódu, jímž lze změnit look-and-feel prvku zaškrtávacího pole na tzv. switch button. Jak taková změna vypadá můžeme vidět na obrázku 2.4. K práci jsou přiloženy i kompletní kódy a spustitelné programy.



Obrázek 2.4: Zaškrtávací pole před a po změně jeho ”look-and-feel” na switch button.

2.3.1 HTML5

HTML (Hyper Text Markup Language) je značkovací jazyk používaný pro tvorbu webových stránek, výsledný prototyp by proto byl spustitelný v podporovaném webovém prohlížeči na jakékoliv platformě včetně operačního systému Windows. Jednotlivé elementy jsou reprezentovány pomocí HTML tagů, jsou zde již obsažené tagy např. pro tlačítka a label, dále tag *input* (vstup) s různými nastavitelnými typy (například *range* pro posuvník, nebo *checkbox* pro

zaškrtávací pole). Jsou k dispozici různé WYSIWYG editory. Look-and-feel prvků jde pomocí HTML změnit v kombinaci s kaskádovými styly (CSS[3]). HTML podporuje multimediální soubory, ale po přepnutí stránky přestane hrát. Asynchronní přehrávání a funkce jako vícedotyková gesta, logování a HW komunikace by bylo možné implementovat pomocí JavaScriptu. Problémová je navigace pomocí šipek. Přechod mezi prvky je prováděn pomocí stisku klávesy TAB nebo SHIFT+TAB. Navigace pomocí šipek by se musela zvlášť implementovat v JavaScriptu. Také nelze nastavit přesné rozměry aplikace a skrýt rozhraní webového prohlížeče [13].

Ukázku kódu pro vytvoření zaškrtávacího pole jako switch button jsem vytvořila podle tutoriálu na webu W3Schools [14].

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<label class="switch">
 <input type="checkbox">
 <span class="slider"></span>
</label>
</html>
```

Listing 2.1: Implementace zaškrtávacího pole jako switch button

```
.switch {
   position: relative;
   display: inline-block;
   width: 114px;
   height: 42px;
}

.switch input {
  opacity: 0;
  width: 0;
  height: 0;
}

.slider {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-image: url("toggle_off.png");
}

input:checked + .slider {
  background-image: url("toggle_on.png");
  width: 114px;
}
```

Listing 2.2: Implementace zaškrtávacího pole jako switch button - style.css

[3] https://www.w3.org/Style/CSS/

2.3.2 XAML

XAML (Extensible Application Markup Language) je značkovací jazyk vyvíjený společností Microsoft a je tedy určený pro operační systém Windows. Je založený na jazyku XML a stejně jako u HTML jsou jednotlivé prvky reprezentovány tagy. Je vhodný k tvorbě uživatelského rozhraní WPF aplikací (Windows Presentation Foundation). K tomu napomáhá nástroj Blend for Visual Studio[4], který nabízí knihovnu základních elementů, možnost manuálně elementy pozicovat na stránce pomocí myši ve WYSIWYG editoru a další nástroje. Funkcionalita, neboli "feel" vytvořeného uživatelského rozhraní může být řešena v jazyce C#, C++ nebo Visual Basic. Stejně tak všechny prvky vytvořené pomocí XAML mohou být nahrazeny kódem napsaným v jednom z těchto jazyků. Taktéž je možná detekce dotykových událostí, komunikace s hardwarem, přehrávání audio souborů bez přerušení po přechodu na jinou stránku i logování [15].

Následuje ukázka kódu pro změnu zaškrtávacího pole na switch button jazyce XAML. V jazyce XAML se změna prvků provádí vytvořením šablony, která je definovaná značkou *Style*. Tato šablona se pak u daného prvku, na který ji chceme aplikovat, přiřadí k atributu *Style*. [16]

```
<Window.Resources>
        ...
        <Style x:Key="CheckBoxStyle1" TargetType="{x:Type
            CheckBox}">
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="{x:Type
                        CheckBox}">
                        <StackPanel Orientation="Horizontal">
                            <Image x:Name="imageCheckBox"
                          Width="100"
                          Height="100"
                          Source="ToggleOFF.png"/>
                            <ContentPresenter VerticalAlignment
                                ="Center"/>
                        </StackPanel>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsChecked" Value
                                ="False">
                                <Setter TargetName="
                                    imageCheckBox"
                                Property="Source"
                                Value="ToggleOFF.png"/>
                            </Trigger>
                            <Trigger Property="IsChecked" Value
                                ="True">
```

[4]https://docs.microsoft.com/en-us/visualstudio/xaml-tools/creating-a-ui-by-using-blend-for-visual-studio?view=vs-2019

```
                                    <Setter TargetName="
                                        imageCheckBox"
                                    Property="Source"
                                    Value="ToggleON.png"/>
                                </Trigger>
                            </ControlTemplate.Triggers>
                        </ControlTemplate>
                    </Setter.Value>
                </Setter>
            </Style>
        </Window.Resources>
        <Grid>
            <CheckBox x:Name="checkBox" Content=""
                HorizontalAlignment="Right" Margin="0,178,330.6,0"
                VerticalAlignment="Top" Style="{DynamicResource
                CheckBoxStyle1}" Checked="CheckBox_Checked"
                IsChecked="False" Width="108" Height="41"/>

        </Grid>
```

Listing 2.3: Implementace zašrktávacího pole jako switch button v jazyce XAML - MainWindow.xaml

2.3.3 JavaFX

JavaFX je sada grafických a mediálních knihoven sloužící pro vývoj bohatých klientských aplikací. Je následníkem knihovny Java Swingy[5]. Aplikace využívající JavaFX jsou psány v Javě, jsou proto multiplatformní a kompatibilní nejen s operačním systémem Windows. JavaFX může využívat jakoukoli Java knihovnu, existují zde tedy prostředky pro logování a komunikaci s hardwarem. JavaFX aplikace podporují různá dotyková gesta, například rotaci, vertikální i horizontální scrollování, swipe ("švihnutí") a gesta pro přibližování a oddalování. Umí pracovat s multimediálními soubory a přehrávat audio bez přerušení po přechodu na jinou stránku. V neposlední řadě nabízí sadu běžných elementů, jejichž vlastnosti mohou být vyjádřeny pomocí JavaFX nebo v jazyce FXML, což je jazyk založený na XML určený pro definování uživatelských rozhraní JavaFX aplikací. Vytvoření prvku pomocí FXML tak odděluje jeho vzhled ("look") od funkcionality ("feel"). FXML soubor popisující dané uživatelské rozhraní lze vytvořit a upravovat textově v textovém editoru nebo vývojovém prostředí, či pouze vizuálně, bez psaní kódu, pomocí nástroje Scene Builder [17]. V něm se pouhým tažením myší mohou přidávat nové elementy a umisťovat na požadovanou pozici. Vzhled prvků v JavaFX lze snadno změnit pomocí JavaFX CSS[6]. Pro základní elementy jsou již definované výchozí CSS třídy, např. pro zaškrtávací pole je to *.check-box.* [19].

[5]https://docs.oracle.com/javase/8/docs/technotes/guides/swing/index.html

[6]JavaFX CSS jsou kaskádové styly upravené pro JavaFX, které vznikly z běžných kaskádových stylů používaných pro popis HTML prvků [18]

Listing 2.4 obsahuje ukázku kódu pro vytvoření zaškrtávacího pole. Pro ukázku jsem zvolila variantu s prvkem vytvořeným v FXML. Listing 2.5 pak ukazuje úpravu vzhledu pole na switch button pomocí kaskádových stylů.

```
<CheckBox mnemonicParsing="false" prefHeight="45.0" prefWidth="
    118.0" stylesheets="@../css/styles.css" xmlns="http://
    javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml
    /1" fx:controller="checkbox.CheckboxController" />
```

Listing 2.4: Implementace tlačítka s obrázkem v jazyce FXML - Checkbox.fxml

```
.check-box {
    -fx-background-image: url('../img/ToggleOFF.png');
    -fx-background-repeat: no-repeat;
}
.check-box:selected {
    -fx-background-image: url('../img/ToggleON.png');
    -fx-background-repeat: no-repeat;
}
.check-box .box, .check-box .mark {
    -fx-background-color: transparent;
}
```

Listing 2.5: Implementace tlačítka s obrázkem v jazyce FXML - styles.css

2.3.4 QML

QML (Qt Modeling Language) spojuje popis grafického uživatelského rozhraní a zpracování událostí, který je součástí Qt frameworku pro tvorbu GUI a multiplatformních aplikací (včetně Windows OS). Logiku aplikace je možné psát v JavaScriptu, také se dá rozšířit kódem napsaným v jazyce C++. Díky propojení s C++ je možné zalogované informace ukládat do souboru a posílat a přijímat zprávy přes sériový port. QML nabízí základní typy prvků (tvary, tlačítko, checkbox, slider apod.), obsahuje modul pro práci s multimediálními soubory a podporuje vícedotykové ovládání. Je možné nastavit přesné rozměry okna aplikace. Pro jazyk QML je k dispozici návrhové a vývojové prostředí QML Design Studio, které umožňuje vizuální návrh uživatelského prostředí. V placené verzi podporuje import návrhů z bitmapového editoru Adobe Photoshop[7], k dispozici je také verze zdarma bez této funkcionality [20].

```
Item {
    id: firstPage
    CheckBox {
        id: checkbox
        x: 280
        y: 183
        checked: false
```

[7] https://www.adobe.com/products/photoshop.html

```
            background: Rectangle {
                Image {
                    source: checkbox.checked ? "toggleON.png" :
                        "toggleOFF.png"

                     MouseArea {
                        anchors.fill: parent
                        onClicked: {
                            checkbox.checked = checkbox.checked
                                ? checkbox.checked = false :
                               checkbox.checked = true
                        }
                    }
                }
            }

            indicator: Rectangle {}
    }
}
```

Listing 2.6: Úryvek kódu pro vytvoření switch buttonu ze zaškrtávacího pole v jazyce QML - main.qml

2.3.5 Shrnutí technologií

Výsledky rešerše vhodných technologií v závislosti na požadavcích jsou shrnuty v tabulce 2.1. Splnění podmínky je označeno ✓, nesplnění znakem ×.

	HTML	XAML	JavaFX	QML
Změna ”look-and-feel”	✓	✓	✓	✓
WYSIWYG editor	✓	✓	✓	✓
Windows OS	✓	✓	✓	✓
Vícedotykové události	✓	✓	✓	✓
Multimédia (bez přerušení)	×	✓	✓	✓
HW komunikace	×	✓	✓	✓
Logování do souboru	×	✓	✓	✓

Tabulka 2.1: Shrnutí analyzovaných technologií a jejich vlastností v porovnání s požadavky na prototyp

2.3.6 Vybraná technologie

Cílem tohoto projektu je vytvořit způsob k tvorbě prototypů dotykových rozhraní v automobilech. Takováto rozhraní mohou umožňovat gesta typická

pro ovládání dotykových obrazovek, například ”pinch-to-zoom”[8]. Požadavky popsané výše obsahují některé pokročilejší způsoby interakce, které nejsou dosažitelné použitím standardních prototypovacích nástrojů. Z těchto důvodů jsem pro toto zadání vybrala tvorbu kódovaných High-Fidelity prototypů.

Jak je vidět z tabulky 2.1, technologie, které jsou schopny splnit všechny vytyčené požadavky, jsou XAML, JavaFX a QML. Z těchto tří možností jsem si proto pro práci vybrala JavaFX, mimo jiné i z toho důvodu, že již mám zkušenosti s programováním v jazyce Java. JavaFX je také jednodušší na nastavení, např. QML zabíralo po instalaci 41.95 GB místa na disku, zatímco JavaFX pouze 345 MB.

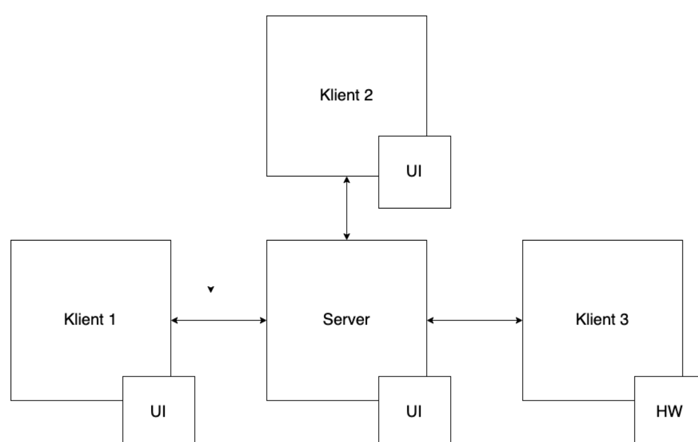
2.4 Návrh architektury

Při návrhu architektury je nutné vzít v úvahu, jak má prototyp fungovat. Prototyp automobilového rozhraní bude aplikace s množstvím panelů, kdy každý panel obsahuje nějaké UI elementy. Pro účely testování prototypu v simulátoru chceme, abychom při změně hodnoty vlastnosti nějakého prvku v prototypu dostali o této změně informaci. Také musíme zajistit, aby akce provedené pomocí hardwarového ovladače byly prototypem zaznamenány.

Vhodnou architekturou pro tento problém je klient-server architektura, kde server zastupuje libovolný počet hardwarových ovladačů automobilu a klient představuje samotný prototyp. Prostřednictvím serveru také bude možné monitorovat vybrané vlastnosti prvků rozhraní, jejichž změny se budou ukládat do textového souboru.

Možným budoucím rozšířením architektury je varianta s jedním serverem a více klienty. K serveru by se mohl připojit libovolný počet klientů, kteří představují buď hardwarové ovladače nebo např. další obrazovku s rozhraním, pokud by to bylo potřeba. Tímto způsobem by se daly prostřednictvím serveru zasílat zprávy mezi jednotlivými rozhraními. Akce provedená na jednom rozhraní by se tak mohla projevit na jiném. Příklad architektury s jedním hardwarovým ovladačem a dvěma obrazovkami s grafickým uživatelským rozhraním je znázorněna na obrázku 2.5. V této práci bude použita architektura klient-server popsaná v předchozím odstavci.

[8] pohyb dvěma prsty od sebe pro přiblížení objektu a k sobě pro oddálení



Obrázek 2.5: Příklad architektury s více klienty

2.5 Návrh komunikace s hardwarem

Protože není známo, jaké hardwarové ovladače budou v simulátoru přítomné, v práci pracuji s předpokladem, že obsahuje joystick pro navigaci po obrazovce nahoru, dolů, doprava a doleva a potvrzovací tlačítko, které simuluje stisknutí aktuálního tlačítka v prototypu. Také beru v úvahu tlačítko zpět. Joystick se může nacházet např. na volantu. Zprávy ze serveru budou zasílané v několika formátech, které představím v následujícím textu.

Existuje několik možných scénářů použití serveru v komunikaci s hardwarovými prvky.

- Prvním scénářem je situace, kdy je třeba monitorovat změny hodnot vlastností prvků během testování prototypu. Musí se proto zaslat danému prvku zpráva, která monitorování zajistí.
- Dalším případem je navigace v prototypu pomocí pohybu joystickem. To zahrnuje i tlačítka pro potvrzení a návrat na předchozí obrazovku.
- Posledním typem akce, která se dá serverovými zprávami provést, je samotná změna hodnoty vlastnosti prvku.

Monitorovací zprávy. Je potřeba, aby prototyp dokázal zaznamenat změnu hodnoty vlastnosti určitého prvku a informovat o tom server. V případě, že by úkolem při testování bylo změnit hodnotu prvku, moderátor díky zprávě se

serveru může zjistit, že participant úkol splnil. K tomuto účelu JavaFX nabízí tzv. posluchače změn (ChangeListener), které lze přihlásit k nějakému objektu, případně je můžeme odhlásit. Tento posluchač pak na každou změnu hodnoty reaguje předepsaným způsobem, v tomto případě zápisem záznamu o změně do textového souboru. Monitor by mělo být možné zaregistrovat na prvky na jakékoliv obrazovce, ne pouze na aktuálně zobrazené. Abychom mohli UI prvek pro registraci posluchače najít, zpráva musí obsahovat identifikátor obrazovky, na které se daný prvek nachází. Dále musí zpráva obsahovat identifikátor samotného prvku, který chceme monitorovat a název konkrétní vlastnosti, jejíž změna nás zajímá. Příkladem může být text štítku nebo hodnota posuvníku. Tyto vlastnosti prvků jsou v JavaFX implementovány jako objekty, na které můžeme přidávat posluchače [21]. Jako poslední je třeba sdělit informaci, zda se posluchač přidává, aby monitoroval změny, nebo odstraňuje, protože informace o změnách už nejsou potřeba. Výsledné zprávy by se tedy mohly skládat z těchto částí:

- označení obrazovky, na které se daný prvek nachází
- označení monitorovaného prvku
- vlastnost prvku, jejíž změny mají být zaznamenány
- informace, zda je posluchač přidáván nebo odebírán

Navigační zprávy. Zprávy generované joystickem bude možné používat pouze na aktuální obrazovce. Musí tedy pouze určovat směr, kterým se joystick pohnul. Jejich obsahem proto bude tento směr. Pro potvrzovací tlačítko a tlačítko zpět bude též dostatečná jednoslovná zpráva sdělující, která akce se má provést. Konkrétně by tyto zprávy mohly vypadat následovně:

- **Up** pro pohyb nahoru
- **Right** pro pohyb doprava
- **Down** pro pohyb dolů
- **Left** pro pohyb doleva
- **Click** pro stisknutí tlačítka
- **Back** pro návrat na předchozí obrazovku

Zpráva pro změnu hodnoty vlastnosti prvku. Tyto zprávy se značně podobají monitorovacím zprávám. Stejně jako u nich, změnu lze zjišťovat i u prvků z nenačtené obrazovky a tedy i zde je potřeba znát obrazovku, na které se prvek nachází, označení prvku i název jeho měněné vlastnosti. Protože můžeme snadno také zjišťovat aktuální stav vlastnosti, zde zprávu rozdělíme na dva případy. Pokud chceme měnit stav prvku, dalším parametrem zprávy bude klíčové slovo pro nastavení a nová hodnota. Pokud se na stav prvku tážeme, použijeme klíčové slovo pro získání hodnoty prvku. Výsledná zpráva má tedy následující složky:

- označení obrazovky, na které se daný prvek nachází
- označení prvku, jehož vlastnost se bude měnit/získávat
- vlastnost
- klíčové slovo ”set” pro nastavení hodnoty vlastnosti, případně ”get” pro získání
- nová hodnota vlastnosti, v případě, že předcházelo slovo ”set”

Pomocí zpráv zasílaných ze serveru mohou moderátoři před testem změnit některé hodnoty nebo text, aniž by museli zasahovat do kódu. Mohou si tak například ověřit vhodnost popisku některých tlačítek bez nutnosti přepisovat texty v programu a v případě, že se po dokončení testu ukážou jako nevyhovující, je zase nemusí přepisovat zpět. Tyto změny pomocí serverových zpráv tak mohou ušetřit čas strávený přepisováním daných hodnot. Pro snadné zadání by bylo možné vytvořit konfigurační soubor a načíst ho na server. Podobným souborem by se také mohly nastavit monitory prvků.

2.6 Návrh vlastních UI komponent

JavaFX umožňuje implementaci vlastních UI komponent rozšířením již existujících základních komponent, jako je např. tlačítko, posuvník nebo zaškrtávací pole. Takto vytvořené komponenty umožňují přidat k elementům funkcionalitu, které v základu nenabízejí.

Na základě analýzy podkladů k prototypu jsem vybrala několik prvků, ze kterých by bylo vhodné vytvořit komponenty. Nejrozšířenějším prvkem v prototypech bude pravděpodobně tlačítko pro přechod na jinou obrazovku.

Vytvořením navigačního tlačítka ušetříme kód pro navigaci mezi obrazovkami, který bychom jinak museli řešit pro každou obrazovku zvlášť. V JavaFX můžeme ve třídě nové komponenty definovat její vlastnosti pomocí tříd typu Property [22]. Takto můžeme například rozšířením objektu tlačítka přidat vlastnost, jež bude obsahovat název obrazovky, na níž se má po kliknutí na tlačítko přesměrovat. Návrh rozložení složitějších komponent s využitím existujících UI prvků lze zjednodušit pomocí nástroje Scene Builder. Proces vytvoření komponenty s využitím nástroje předvedu v kapitole 4.

Kapitola 3

Implementace

V této kapitole blíže představím použitý software a základní principy JavaFX aplikací. Dále popíši můj postup při implementaci knihovny pro zjednodušení výroby prototypů a na několika vlastních komponentách demonstruji proces jejich tvorby.

Při využití výsledků této práce pro vývoj prototypů se předpokládá, že alespoň jedna osoba zapojená do vývoje má zkušenosti s programováním. Implementační postupy popsané v této kapitole mají sloužit jako pomůcka pro tvorbu nových komponent, použití mnou vytvořené knihovny a implementaci dalších funkcionalit a vzhledu (pomocí CSS) jednotlivých obrazovek v prototypech libovolných dotykových rozhraní v automobilech. Návrh struktury grafického uživatelského rozhraní prototypu je díky využití značkovacího jazyka FXML a vizuálního nástroje Scene Builder možný i bez znalosti programování.

3.1 Použitý software

Sada knihoven JavaFX 8. K implementaci jsem použila knihovnu JavaFX 8 a značkovací jazyk FXML, který je určen k popisu hierarchie prvků uživatelského rozhraní. JavaFX 8 je poslední verzí, která je součástí vývojářského balíku JDK (Java Development Kit). Jejím použitím tak odpadá nutnost složitější instalace, stačí pouze nainstalovat JDK 8. V současné době již existuje

JavaFX 13[1]jako externí knihovna, která je určena pro použití s jazykem Java 13. Protože jsem však ve své práci plánovala využívat program Scene Builder, který je poskytován pouze pro Java 8 a 11, nemohla jsem JavaFX 13 ani Java 13 použít. Při rozhodování, zda zvolit JavaFX ve verzi 8 nebo 11 jsem z důvodu jednoduchosti instalace vybrala verzi 8.

Značkovací jazyk FXML. FXML je jazyk založený na XML používaný k deklaraci uživatelského rozhraní JavaFX aplikací. Odděluje UI aplikace od logiky, která se nachází v tzv. kontroleru.

Programovací jazyk Java 8. Z důvodů vysvětlených výše byla použita Java ve verzi 8.

Vývojové prostředí NetBeans 8.2. Vývojové prostředí NetBeans[2] podporuje různé technologie včetně Java a JavaFX. Z NetBeans lze snadno otevírat FXML soubory v programu Scene Builder dvojitým poklepáním na daný FXML soubor v projektu. Kód FXML se zobrazí volbou Edit z kontextového menu souboru.

WYSIWYG nástroj Scene Builder 8.5.0. Scene Builder je vizuální nástroj pro tvorbu grafického uživatelského rozhraní JavaFX aplikací. Umožňuje rychlé vytváření GUI bez nutnosti použití kódu. Po otevření FXML souboru v Scene Builderu mohou uživatelé tažením myší přidávat na obrazovku různé komponenty a umisťovat je na požadované místo, měnit jejich vlastnosti a přidávat k nim CSS styly. Změny provedené v nástroji Scene Builder jsou generovány do daného FXML souboru v podobě FXML kódu. Stejně tak změny provedené přímo v kódu se po uložení projeví i v Scene Builderu. Je tak možné pracovat na jednom FXML souboru v Scene Builderu i přímo v NetBeans zároveň [17]. Tento nástroj byl v práci využit pro navržení všech obrazovek. Podrobněji Scene Builder a práci s ním popíši v kapitole 4.

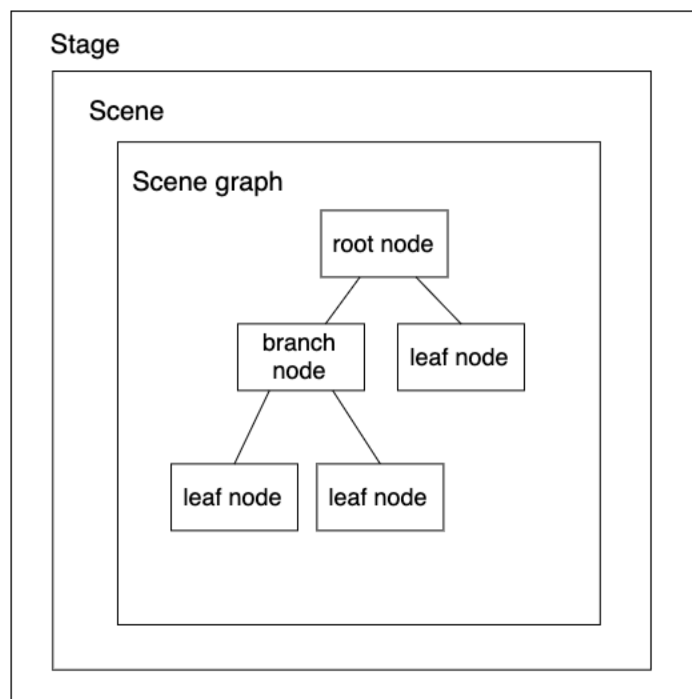
3.2 Struktura JavaFX FXML aplikace

JavaFX aplikace má v zásadě tři hlavní komponenty - *Stage*, *Scene* a objekty typu *Node* (uzel). *Stage* (jeviště) představuje samotné okno aplikace a obsahuje

[1]https://openjfx.io/
[2]https://netbeans.org/downloads/8.2/

všechny objekty JavaFX aplikace. Objektu *Stage* nastavujeme objekt *Scene* (scéna) a metodou *show()* *Stage* zobrazíme.

Scene je kontejner pro veškerý vykreslovaný obsah. Tento obsah je reprezentován pomocí tzv. scene graph (graf scény). Scene graph představuje stromovou strukturu obsahu na scéně, tedy všech vizuálních elementů uživatelského rozhraní aplikace. *Node* neboli uzel je element grafu scény. Jak je vidět na obrázku 3.1, každý uzel má maximálně jednoho rodiče, uzel bez rodiče se nazývá root (kořen) a je nejvyšším prvkem grafu scény. Root se předává konstruktoru scény a v grafu scény může být jen jeden jediný. Dalšími druhy uzlů jsou tzv. branch nodes (uzly větve) nebo také parent nodes (rodičovské uzly), které mají jednoho nebo více potomků, a leaf nodes (listy), které nemají potomka žádného [23].



Obrázek 3.1: Struktura UI prvků JavaFX aplikace

Vývojové prostředí NetBeans poskytuje možnost založení šablony pro JavaFX FXML aplikaci. Tento projekt obsahuje tři následující soubory:

- *JavaFXApplication.java*
- *FXMLDocument.fxml*
- *FXMLDocumentController.java*

Na tomto jednoduchém vzorovém projektu představím základní strukturu JavaFX FXML aplikace.

JavaFXApplication.java. Tato třída rozšiřuje třídu *Application*, která poskytuje framework pro spravování JavaFX aplikace [24]. Dále soubor obsahuje metodu *main()* a přetíženou metodu *start()*. V metodě *main()* se volá statická metoda *Application.launch()*, která přebírá argumenty z příkazové řádky a spouští aplikaci. V metodě start se načítá FXML soubor pomocí metody *load()*, která vrací objekt typu, jež odpovídá kořenovému prvku v souboru. V tomto případě se vrácený objekt načte jako objekt Parent, což je základní třída pro všechny možné kořenové či branch uzly. Tento objekt se přidá na scénu jako kořen a scéna se nastaví objektu Stage, která obsah scény s grafem definovaným v FXML zobrazí. Při zavření okna aplikace se zavolá metoda *stop()* a tím životní cyklus aplikace končí. Metodu *stop()* je možné přetížit a před ukončením aplikace v jejím těle zavolat kód, který je třeba před zavřením aplikace provést.

```
public class JavaFXApplication extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root =  FXMLLoader.load(getClass().getResource("
            FXMLDocument.fxml"));

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Listing 3.1: Obsah souboru JavaFXApplication.java ze vzorového projektu JavaFX FXML Application v NetBeans

FXMLDocument.fxml. Tento soubor obsahuje FXML kód, viz 3.2. Ten popisuje stromovou hierarchii prvků dané obrazovky. Kořenovým prvkem je kontejner *AnchorPane*, který má děti - *Button* a *Label*. Jedním z atributů kořene je *fx:controller*, který specifikuje soubor s funkcionalitou pro danou obrazovku. Atribut *xmlns:fx* je povinný a speficikuje jmenný prostor *fx*. Popisy elementů obsahují atribut *fx:id*, jehož hodnota jednoznačně identifikuje daný prvek v rámci jednoho FXML souboru. Přiřazením *fx:id* hodnoty k elementu se vytvoří proměnná ve jmenném prostoru dokumentu a k této proměnné je poté možné přistupovat odjinud z kódu. Prvek Button má také vlastnost *onAction*,

jejíž hodnotou je název metody *handleButtonAction()* uveden znakem "#". Tato metoda je definovaná v kontroleru a volá se při dané události vzniklé na elementu.

```
<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
    xmlns:fx="http://javafx.com/fxml/1" fx:controller="
    javafxapplication.FXMLDocumentController">
    <children>
        <Button layoutX="126" layoutY="90" text="Click␣Me!"
            onAction="#handleButtonAction" fx:id="button" />
        <Label layoutX="126" layoutY="120" minHeight="16"
            minWidth="69" fx:id="label" />
    </children>
</AnchorPane>
```

Listing 3.2: Obsah souboru FXMLDocument.fxml ze vzorového projektu JavaFX FXML Application v NetBeans

FXMLDocumentController.java. Tento soubor obsahuje kód v jazyce Java. Implementuje rozhraní *Initializable* a přetěžuje metodu *initialize()*. FXML loader volá metodu i*nitialize()* po načtení FXML dokumentu. která slouží k inicializaci prvků. a Do tohoto souboru patří metody pro úpravu vlastností prvků z FXML souboru, v jehož kořeni byl daný kontroler specifikován. V kontroleru také dochází ke zpracování událostí, které na těchto prvcích vznikají. Každý prvek, který má v FXML souboru přiřazenou hodnotu *fx:id*, může být pomocí anotace *@FXML* a vytvoření proměnné pod stejným názvem jako hodnota *fx:id* zpřístupněn v kontroleru.

V kódu 3.3 je implementována metoda *handleButtonAction()*, která byla v FXML souboru nastavena tlačítku v atributu *onAction*. Metoda, stejně jako reference na objekt label, jsou označeny anotací *@FXML*. Při události typu *ActionEvent* vyvolané na tlačítku se do konzole vypíše text "You clicked me!" a label se nastaví na text "Hello World!".

```
public class FXMLDocumentController implements Initializable {

    @FXML
    private Label label;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You␣clicked␣me!");
        label.setText("Hello␣World!");
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
    }
}
```

Listing 3.3: Obsah souboru FXMLDocument.Controller.java ze vzorového projektu JavaFX FXML Application v NetBeans

3.3 Knihovna pro zjednodušení výroby prototypů

V této části popíši třídy, které jsem v rámci práce implementovala a které jsou znovupoužitelné pro prototypy různých rozhraní.

Projekt TouchInCarServer obsahuje serverovou logiku a GUI serveru pro zasílání zpráv.

Balíček *controllers* v projektu Prototype obsahuje tyto třídy pro usnadnění vývoje prototypů.

- *Client.java*
- *AudioController.java*
- *StageController.java*

Postup při implementaci tříd popíši v následujícím textu.

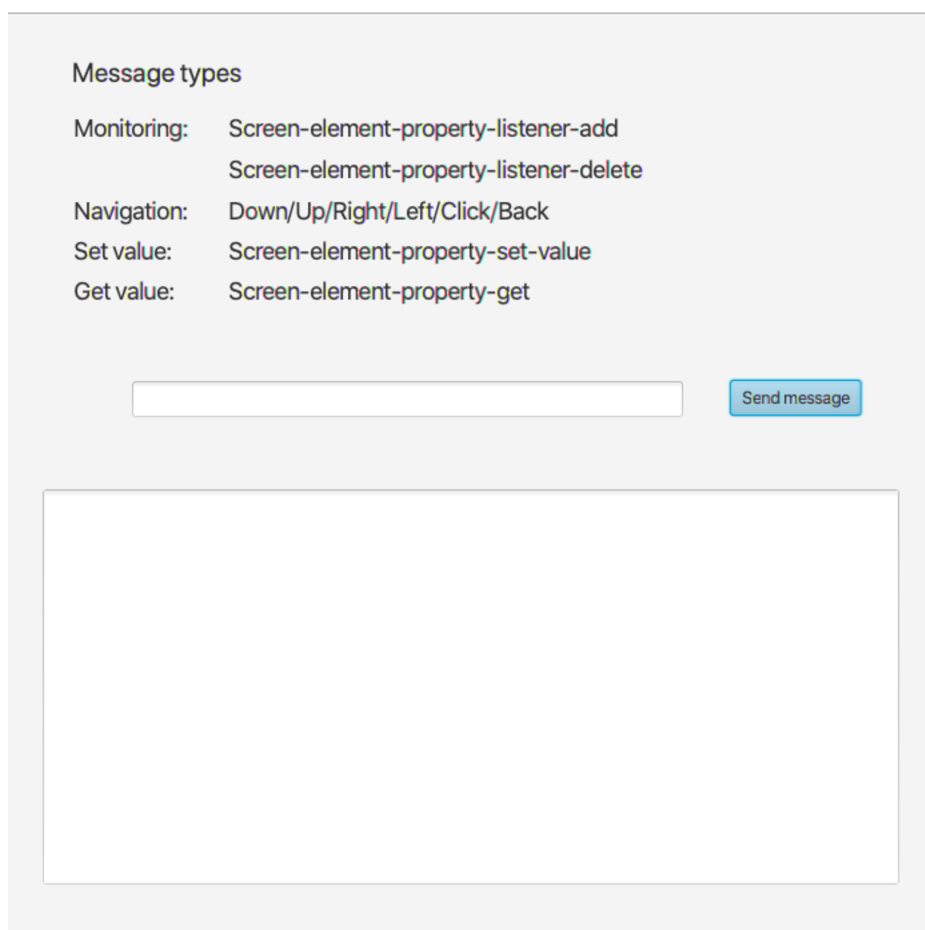
3.3.1 Server

Prvním krokem implementace bylo vytvořit klient-server architekturu. Dva vzniklé programy spolu komunikují pomocí TCP socketů[3] mezi. Klientská aplikace je aplikace prototypu samotného. Třída *Client.java* slouží k připojení k serveru, k zasílání zpráv serveru a ke zpracování zpráv přijatých ze serveru, tedy zpráv zastupujících akce hardwarových ovladačů, monitorování změn a samotnou změnu hodnot vybraných prvků. Bylo vytvořeno grafické uživatelské

[3] Soket je jeden z koncových bodů komunikace mezi dvěma programy běžícími na síti. Koncový bod je určen IP adresou a číslem portu, které rozlišuje aplikaci v rámci počítače [25].

rozhraní (GUI) serverové aplikace, aby simulovalo hardwarové komponenty simulátoru. GUI je k vidění na obrázku 3.2. Obsahuje textové pole pro zapsání zprávy a potvrzovací tlačítko, kterým se tato zpráva zašle klientovi. Taktéž obsahuje legendu s obsahem typů povolených zpráv podle návrhu z předchozí kapitoly. Posledním prvkem tohoto GUI je víceřádkové textové pole, které slouží k výpisu zpráv přijatých od klienta.

Jako parametr přijímá server číslo portu, ke kterému se má připojit. Po připojení klienta server čeká na zprávy od klienta pro vypsání do GUI, nebo může prostřednictvím textového pole odeslat klientovi jednu ze tří typů navržených zpráv. Tato logika je implementována ve třídě *Server.java*. Projekt dále obsahuje třídu *DPServer.java* s metodou *main()*, FXML soubor s definicí GUI a kontroler, který při inicializaci nastaví server a poté obsluhuje zasílání zpráv klientovi.



Obrázek 3.2: GUI serveru

3.3.2 Client.java

Jak bylo zmíněno v předchozí sekci, tato třída obsahuje logiku klienta. Při spuštění program požaduje dva parametry - IP adresu a číslo portu, které musí být shodné s portem zadaným při spouštění serveru.

Pokud třída přijme zprávu ze serveru, pošle ji do metody *parseMessage()*, která se stará o rozklíčování typu přijaté zprávy a následné reakce. Třída rozeznává typy zpráv, jež byly navrženy v předchozí kapitole - monitorovací, navigační a zprávy pro změnu/získání hodnoty vlastnosti prvku. Metoda přijatou zprávu rozdělí podle pomlček a na základě vzniklého počtu parametrů rozhoduje, která zpráva byla přijata.

Zpráva s jedním parametrem určuje navigační zprávu. Navigací mezi prvky nastavujeme aktivnímu prvku tzv. focus (česky zaměření). Tato změna focusu se nazývá "focus traversal". Všechny prvky, mezi kterými chceme navigovat pomocí šipek, musí mít property *focusTraversable* nastavenou na true, což způsobí, že mohou být zaměřeny. Každý Node má property *focused*, která říká, zdá tento prvek má či nemá focus. Nemůžeme přímo nastavit hodnotu *focusedProperty* ani např. Stagi nastavit konkrétní prvek, který má focus mít. Manuálně můžeme focus nastavit tak, že přímo na elementu zavoláme metodu *requestFocus()*. Pokud je *focusTraversableProperty* nastavena na true, *focusedProperty* elementu se nastaví na true a prvek získá focus [22].

Běžně se focus traversal mezi prvky řeší klávesami TAB pro posun po stránce dolů a SHIFT+TAB pro posun nahoru. Hardwarový joystick v simulátoru ale lze namapovat na klávesové šipky. Mezi prvky proto navigujeme pomocí kombinace klávesy SHIFT a šipek. Je nutné ke stiskům šipek přidat i SHIFT z toho důvodu, jak objekt *ScrollPane* na stisky šipek reaguje. ScrollPane při stisknutí kláves šipek nahoru a dolů posouvá obsah a při stisknutí kláves šipek a klávesy SHIFT mění focus mezi prvky, které obsahuje. Většina prvků stisknutí klávesy SHIFT ignoruje, ne však *Slider* (posuvník), který šipkami do stran mění hodnotu a na události se stisknutou klávesou SHIFT nereaguje. V případě potřeby se musí událost zachytit a chování posuvníku zvlášť nastavit. To jde například použitím kódu 4.9 v kontroleru dané obrazovky, kde se posuvník nachází.

```
slider.addEventHandler(KeyEvent.KEY_PRESSED, (KeyEvent event)
    -> {
            KeyCombination kbRight = new KeyCodeCombination(
                KeyCode.RIGHT, KeyCombination.SHIFT_DOWN);
            KeyCombination kbLeft = new KeyCodeCombination(
                KeyCode.LEFT, KeyCombination.SHIFT_DOWN);
            if (kbRight.match(event)) {
                slider.increment();
```

```
            } else if (kbLeft.match(event)) {
                slider.decrement();
            }
        });
```

Listing 3.4: Kód pro zvýšení nebo snížení hodnoty posuvníku při stisknutí šipek do stran spolu s klávesou SHIFT.

Aby traversal správně fungoval, nesmí se prvky na obrazovce navzájem překrývat. Pořadí focus traversalu je závislé na pořadí prvků v FXML souboru. Pokud chceme, aby nějaký prvek byl zaměřen jako první, musíme tento prvek v FXML souboru definovat jako první. Pokud se prvek nachází v kontejneru, musí být také prvním prvkem v tomto kontejneru. Na správné pořadí komponent je třeba dávat pozor především při použití kontejnerů Pane a *AnchorPane*, které umožňují absolutní pozicování elementů a automaticky nemění pořadí prvků v hierarchii podle jejich pozice na obrazovce.

Parametr určující směr vyvolá událost typu *KeyEvent*, kliknutí nebo tlačítko zpět vyvolá událost typu *ActionEvent*, která ji spustí na aktuálně zaměřeném prvku. Tento typ zpráv vždy reaguje pouze na aktuálně zobrazenou obrazovku. Protože se událostmi modifikuje stav GUI, musí se zasílat přes aplikační vlákno JavaFX pomocí volání *Platform.runLater()*.

Větší počet parametrů značí monitorovací zprávu nebo zprávu pro změnu, respektive získání hodnoty. Tyto tři případy jsou rozlišeny klíčovým slovem "listener", "set" a "get". Tyto zprávy se mohou dotazovat na prvky, které se nachází i na dosud nezobrazených obrazovkách. Je nezbytné, aby všechny tyto prvky měly v FXML nastavené fx:id, pomocí kterého jsou na daných obrazovkách vyhledávány.

Aby šel posluchač přidat na libovolný prvek bez nutnosti přetypovávání, použila jsem princip reflexe[26] pro získání objektu typu *Property* daného prvku na základě názvu vlastnosti. Na získaný objekt se přidá posluchač, který při změně hodnoty této property vypíše zprávu o změně do GUI serveru a do logovacího souboru. Posluchače se ukládají do mapy, aby bylo možné je v reakci na zprávu odstranit.

Pro změnu hodnoty vlastnosti (property) prvku taktéž bez nutnosti přetypovávání na různé typy modifkovaných objektů jsem využila *BeanAdapter*.

Všechny reakce na zprávu zasílají informaci o změnách do GUI serveru a do logovacího souboru.

3.3.3 AudioController.java

Třída obsahuje metody pro spuštění a zastavení hudební skladby. Třídu jsem v prototypu využila při simulaci hrajícího rádia na obrazovce *Radio.fxml*[4]. Kliknutím na obrázek rádiové stanice se spustí, respektive zastaví přehrávání. *AudioController* je Singleton, vždy tedy bude existovat pouze jedna instance. Je zajištěno, aby vždy hrála pouze jedna skladba. Hudební soubory ve formátu .mp3 se musí nacházet v hlavním adresáři projektu. Pro implementaci audio kontroleru byla využita třída *MediaPlayer*, která nabízí možnosti pro další rozšíření.

3.3.4 StageController.java

Protože prototyp rozhraní automobilu obsahuje velké množství obrazovek, bylo potřeba implementovat co nejflexibilnější způsob, jak mezi obrazovkami přecházet. Běžný způsob změny obrazovky při nízkém celkovém počtu obrazovek je načtení nové scény v kontroleru odpovídající obrazovky. Při vysokém počtu obrazovek, který se předpokládá u prototypů rozhraní automobilů, by byl ale takový přístup zdlouhavý, nepřehledný a v případě změny by se musely opravit ve všech kontrolerech.

Řešením, které jsem zvolila, bylo vytvoření navigačního tlačítka s property "path", do které se uloží název FXML soubory obrazovky, na kterou tlačítko vede. Tlačítko při akci zavolá metodu *switchToScreen()* ze třídy *StageController*, která za parametr bere název FXML souboru následující obrazovky. Veškerá funkcionalita je tak obsažena v tlačítku a v případě potřeby stačí provést změny pouze na jednom místě.

Místo aby se vždy při přechodu na jinou stránku vytvořila nová scéna s nově načteným obsahem, na *Stage* je jedna scéna, které se pouze vyměňuje kořenový uzel. Kořenové uzly načtené z FXML souborů jsou uložené v mapě pro snadný přístup přes název FXML souboru dané obrazovky. Vyměňováním již načtených stránek se zachovává stav obrazovek i při přechodu jinam a zpět. Díky tomuto mechanismu je možné na základě názvu souboru s obrazovkou načíst stránku do mapy ještě předtím, než je fyzicky navštívena v prototypu. Tato funkcionalita je využita při změnách a monitorování prvků pomocí serverových zpráv.

[4] Použité hudební skladby byly získány ze stránky https://www.bensound.com/royalty-free-music

Navštívené obrazovky se ukládají na zásobník. Při stisku tlačítka zpět nebo zaslání zprávy ”Back” ze serveru se zavolá metoda *goToPreviousScreen()* a *StageController* zobrazí předchozí obrazovku.

3.4 Implementace vlastních komponent

Při implementaci prototypu jsem vytvořila několik vlastních komponent, k čemuž jsem využila existující JavaFX elementy. V této části podrobně představím způsob jejich implementace na příkladu navigačního tlačítka.

Nové komponenty se v FXML kódu definují pomocí značky *<fx:root>* v kořenovém uzlu a nastavení atributu type na požadovaný typ komponenty. Kontroler v tomto případě nenastavujeme.

```
<fx:root type="javafx.scene.control.Button" xmlns:fx="http://
    javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8.0.171"
    >
</fx:root>
```

Listing 3.5: Popis vlastní komponenty typu Button v jazyce FXML

Dále vytvoříme třídu kontroleru, která bude typ definovaný v kořeni rozšiřovat a načítat předchozí FXML kód v konstruktoru. Třída také sama sebe nastaví jako root a kontroler.

```
public class MyButton extends Button implements Initializable {

        public MyButton() {
                FXMLLoader fxmlLoader = new FXMLLoader(getClass
                    ().getResource("MyButton.fxml"));
                fxmlLoader.setRoot(this);
                        fxmlLoader.setController(this);

        try {
                fxmlLoader.load();
        }   catch (IOException exception) {
                throw new RuntimeException(exception);
        }
    }

        @Override
         public void initialize(URL url, ResourceBundle rb) {
        }
}
```

Listing 3.6: Kontroler vlastní komponenty typu Button

Po sestavení projektu můžeme takovou komponentu nahrát do Scene Builderu a používat ji v tvorbě rozhraní jako běžnou komponentu.

Nové vlastnosti lze ke komponentám přidávat dvěma způsoby. Prvním je deklarace atributů různých datových typů. Vezmeme-li za příklad navigační tlačítko, které potřebuje ukládat cestu na další obrazovku, kód by vypadal následovně:

```
private String path;

public String getPath() {
        return path;
}

public void setPath(String value) {
        path = value;
}
```

Listing 3.7: Přidání nového atributu třídy.

Takto je možné přidávat prvkům nové vlastnosti, které lze v FXML kódu i v Scene Builderu používat jako atributy a dávat jim hodnoty.

Přidávání nových vlastností komponent je možné také pomocí objektů typu *Property*. Pokud chceme být schopni monitorovat změny dané vlastnosti, pak musí být vytvořena za použití *Property*. Výsledná vlastnost je objekt, na který lze přidávat posluchače změn, což v předchozím popsaném případě nelze. Kód pro vytvoření takové vlastnosti je ukázán v 3.8. Při vytváření dalších vlastností je důležité zachovat tuto strukturu.

```
StringProperty path = new SimpleStringProperty();

public final StringProperty pathProperty() {
        return path;
    }

    public final String getPath() {
        return path.get();
    }

    public void setPath(String value) {
        path.set(value);
    }
```

Listing 3.8: Přidání nové vlastnosti třídy pomocí rozhraní *Property*

Na objektu takto vytvořené vlastnosti lze také volat metodu *bind()*:

```
objectA.propertyA().bind(objectB.propertyB());
```

Listing 3.9: Metoda pro provázání vlastností dvou objektů

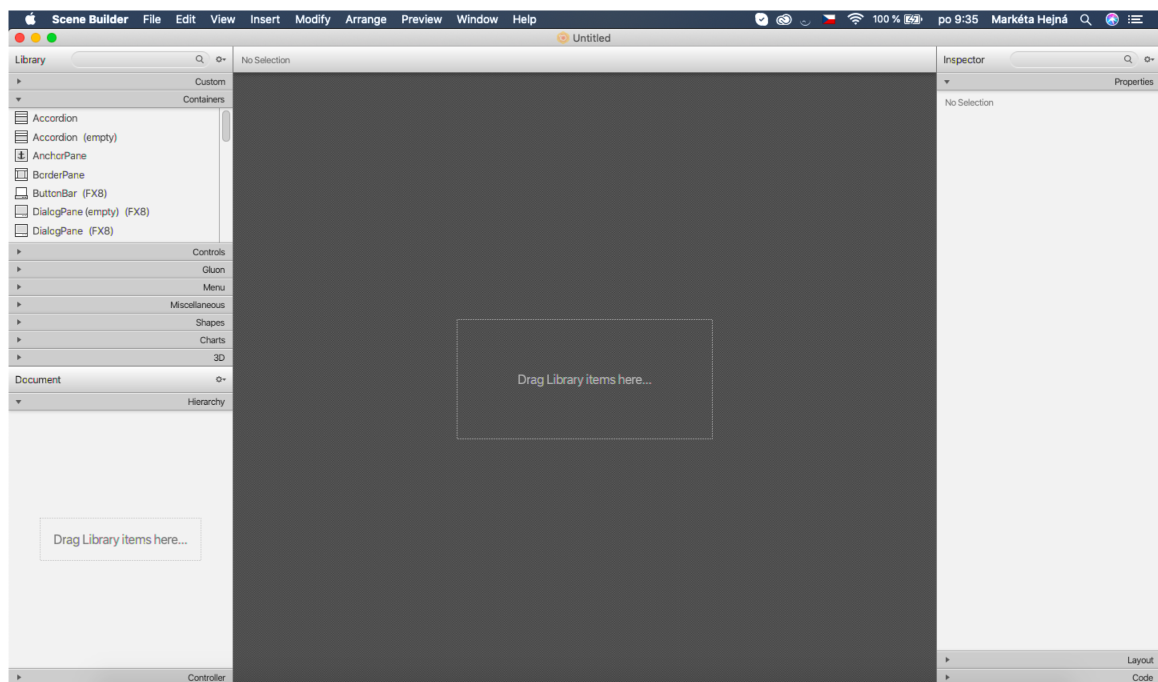
Metoda *bind()* způsobí, že při každé změně hodnoty propertyB se změní i hodnota propertyA. Pokud například máme komponentu, kde je tlačítko uvnitř panelu a pomocí serverových zpráv chceme změnit jeho text, musíme komponentě nastavit objekt *StringProperty* a na něj navázat *textProperty()* tlačítka. Poté zprávou ze serveru změníme hodnotu *StringProperty* a díky metodě *bind()* se změna projeví i na tlačítku.

Další typy property jsou např. *IntegerProperty*, *DoubleProperty*, *FloatProperty* či *BooleanProperty* či *ObjectProperty<T>*, která může být libovolného typu [22].

Kapitola 4

Práce s nástrojem Scene Builder

V této kapitole představím nástroj Scene Builder, vysvětlím jeho rozhraní a postupy pro vytváření obrazovek, mimo jiné i s využitím vlastních komponent.



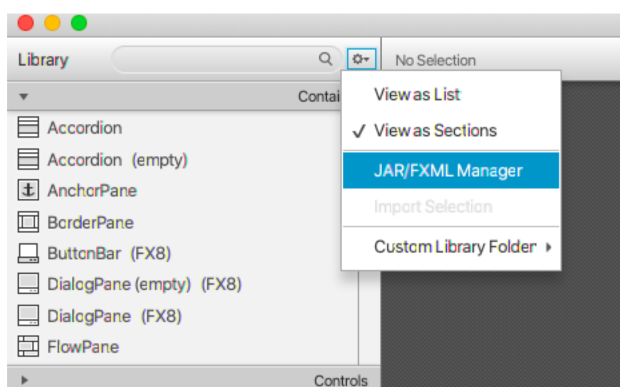
Obrázek 4.1: GUI nástroje Scene Builder

Na obrázku 4.1 se nachází rozhraní nástroje Scene Builder pro vizuální navrhování uživatelských rozhraní JavaFX FXML aplikací. Nástroj podporuje

tzv. "drag-and-drop" (táhni a pusť) mechanismus pro přidávání a pozicování jednotlivých elementů na pracovní ploše. Obrazovky se do Scene Builderu načítají jako FXML soubor a veškeré změny provedené v nástroji se převedou na FXML kód. Stejně tak změny provedené přímo v kódu se ihned projeví v nástroji.

Uprostřed se nachází pracovní plocha, která slouží k návrhu uživatelského rozhraní. Vlevo nahoře je panel s knihovnou, která obsahuje všechny elementy, jež můžeme použít pro návrh rozhraní. Panel dokument zobrazuje aktuální scene graph obsahu pracovní plochy a v záložce Controller můžeme vidět všechny objekty FXML souboru, které mají nastavené *fx:id*, nebo souboru přiřadit soubor s kontrolerem. Na pravé straně je panel Inspektor rozdělený na části Properties, Layout a Code. V záložce Properties můžeme nastavovat různé vlastnosti elementů včetně přidávání kaskádových stylů, v záložce Layout nastavovat velikost či pozici na obrazovce a v poslední záložce nastavit *fx:id* či handlery různých událostí. Poslední možností na kartě View v horním menu je "Sample Controller Skeleton", ve kterém se nachází vzorový kód kontroleru ke zkopírování, aby se všechna nastavená *fx:id* nemusela do kontroleru psát zvlášť.

Do Scene Builderu můžeme rovněž přidávat vlastní komponenty. Nejprve je třeba sestavit projekt v NetBeans pro vytvoření .jar souboru. Poté klikneme na tlačítko podle obrázku 4.2 a vybereme "JAR/FXML Manager". Otevře se okno, kde klikneme na odkaz "Add Library/FXML from file system". V průzkumníku souborů navigujeme do složky s vytvořeným .jar souborem, zvolíme ho a potvrdíme. Na panelu Library se zobrazí nová záložka Custom s nově přidanou komponentou.



Obrázek 4.2: Nabídka pro vložení vlastních UI komponent

Na jedné obrazovce nemusí být pouze jeden FXML soubor. Ve spodní části nabídky menu File jsou dvě možnosti, Import a Include. Import FXML souboru vloží FXML kód přímo do kódu aktuálního souboru. Použitím

Include se soubor do aktuální obrazovky pouze načte, v kódu je tento soubor označen značkami *<fx:include>*. Změny původního souboru se aplikují na všech obrazovkách, kde je tímto způsobem vložený. Kontroler hlavního souboru má navíc díky tomu přístup ke kontroleru vloženého souboru a může volat jeho metody. Vložený soubor musí mít nastavené *fx:id*. Řekněme, *fx:id* je nastaveno na "box", kořen vloženého souboru je *VBox* a kontroler toho souboru je ve třídě *MyController.java*. Pak obě hodnoty injektujeme do kontroleru hlavního souboru takto:

```
@FXML
private VBox box;
@FXML
private MyController boxController;
```

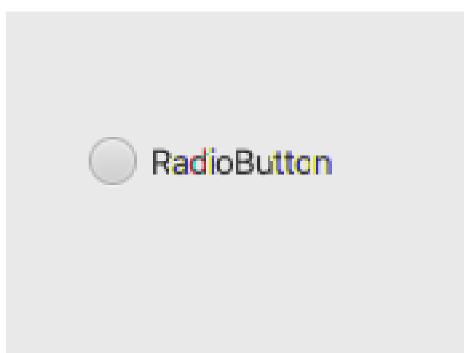
4.1 Použití Scene Builderu při tvorbě obrazovek prototypu

Všechny obrazovky výsledného prototypu byly navrženy pomocí tohoto nástroje. Soubory s FXML soubory obrazovek a jejich kontrolery se nachází v balíčku *prototype* projektu Prototype. Scene Builder výrazně ulehčuje stavbu grafu scény pomocí tažení potřebných komponent přímo na pracovní plochu, která představuje obsah obrazovky. Pro vytvoření nové obrazovky jsem ve vývojovém prostředí NetBeans kliknutím pravým tlačítkem na balíček v projektu zvolila nový prázdný FXML soubor a soubor s kontrolerem. FXML soubor jsem dvojklikem otevřela v Scene Builderu, kde se na pracovní ploše zobrazil defaultní kontejner *AnchorPane*. Z kontejnerů, které JavaFX nabízí, jsem pro prototyp používala základní *Pane* pro absolutní pozicování prvků, *AnchorPane*, *ScrollPane* pro skrolovatelný obsah, *HBox* pro prvky umístěné horizontálně vedle sebe a *VBox* pro prvky umístěné vertikálně nad sebou. *HBox* a *VBox*automaticky změní velikost prvků uvnitř, pokud jejich velikost překračuje velikost kontejneru.

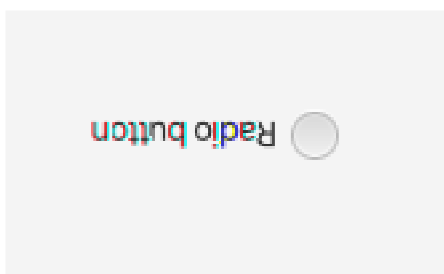
Na hlavní kontejner vpravo v menu Properties přidám CSS soubor *pane.css*, který byl použit pro nastylování celého prototypu. Mohli bychom soubor s kaskádovými styly přidávat přímo na scénu pomocí *scene.getStylesheets().add ("path/stylesheet.css");*, protože scéna je pro všechny obrazovky společná, ale v tom případě by se nezobrazovaly v Scene Builderu, pouze po spuštění aplikace. Přidáním CSS souboru na kořen obrazovky se styly aplikují i na zbytek elementů. Pokud bychom chtěli styly změnit, měl by se změnit obsah tohoto souboru. CSS třídy se objektům nastavují v pravém panelu Inspektor na záložce Properties, kam je možné i rovnou psát CSS vlastnosti a jejich hodnoty. JavaFX CSS obsahuje třídy pro defaultní prvky ve tvaru názvu třídy

malými písmeny. Víceslovné názvy se rozdělují pomlčkou, např. *.check-box*, *.scroll-pane*, či *.radio-button*. Jména všech vlastností jsou uvozena předponou *-fx-*. Aplikují se též pravidla pro kombinace selektorů. Na příkladu ukážu, jak je pomocí CSS možné změnit look-and-feel radio buttonu.

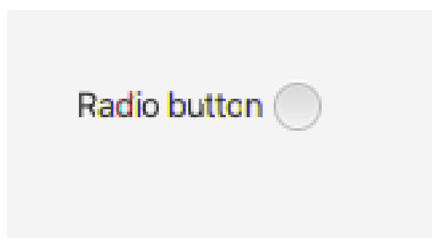
Radio button má defaultní vzhled jako na obrázku 4.3. Automobilové rozhraní z podkladů používá radio buttony s tmavým pozadím, které mají text napravo od vybíracího kolečka a celá komponenta více připomíná tlačítko. Těchto změn se dá snadno dosáhnout pomocí kaskádových stylů. Na obrázku 4.4 je radio button poté, co byl pomocí CSS otočen o 180 stupňů. Tím se kolečko dostalo na opačnou stranu. Na obrázku 4.5 je pak ještě přes výběr *.radio-button > .text* text znovu o 180 stupňů otočen a tím se dostal do původní pozice, ale na opačné straně kolečka. Všechny změny CSS se ihned zobrazují v Scene Builderu.



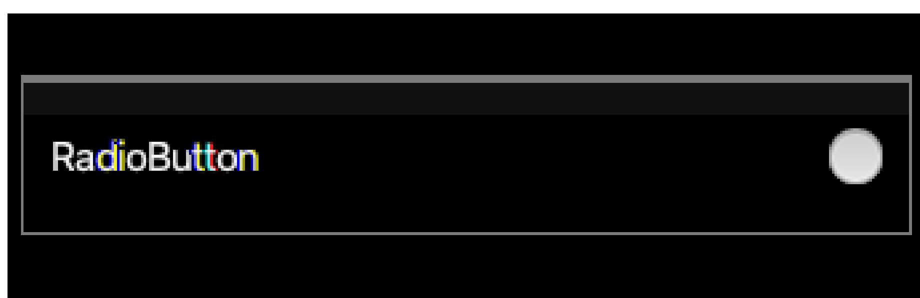
Obrázek 4.3: Defaultní radio button



Obrázek 4.4: Radio button po otočení o 180 stupňů



Obrázek 4.5: Radio button po otočení textu o 180 stupňů



Obrázek 4.6: Radio button po aplikování všech stylů

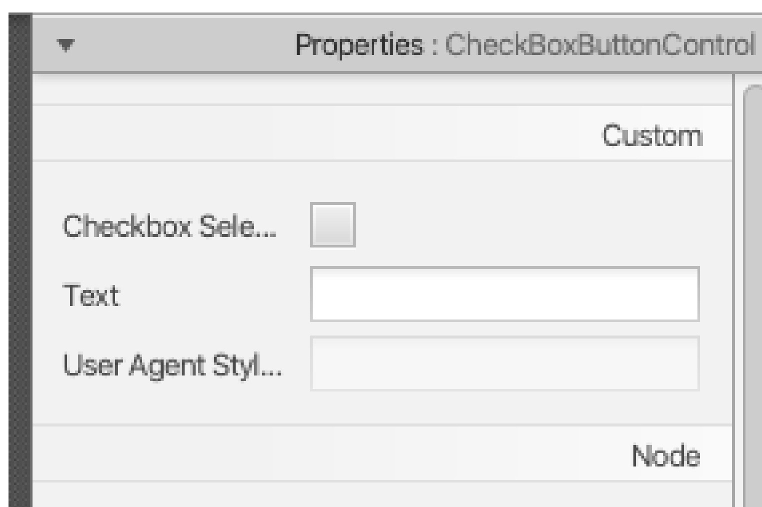
Pro ukázku zde popíšu postup při tvorbě obrazovky *CarSettings2.fxml*, která obsahuje tři mnou vytvořené komponenty.

Připravím si fotografii obrazovky rozhraní, podle které budu obrazovku navrhovat. Je na ní štítek s názvem Nastavení pneumatik, tlačítko zpět a několik tlačítek ve *ScrollPane*. První tlačítko má vlevo text a vpravo obrázek. Následující tlačítko na sobě má zaškrtávací pole. Poslední tlačítko s názvem a číslem je neaktivní, ale aktivuje se zaškrtnutím pole na tlačítku výše. Při rozkliknutí posledního tlačítka se zobrazí posuvník, kterým se mění číslo na tlačítku.

Protože tento typ obrazovky se v prototypu často opakoval, připravila jsem si šablonu *settings.fxml*, kterou importuji do Scene Builderu. V šabloně je předpřipravená sada komponent, které používám pro obrazovku typu nastavení. Obsahuje štítek pro název obrazovky, *Pane*, *ScrollPane* s *Pane* uvnitř a také tlačítko zpět, které je jednou z použitých vlastních komponent. Při stisknutí tlačítka se zavolá metoda *(*goToPreviousScreen()) ze *StageControlleru*. *Pane* umístěnou ve *ScrollPane* jako kontejner pro prvky jsem se rozhodla vyměnit za *VBox*, který automaticky polohuje vložené prvky vertikálně pod sebe. Na záložce Layout v inspektoru je možné mu nastavit mezery mezi všemi prvky a každému prvku zvlášť pak lze nastavit odsazení od každého kraje - margin. Protože tato obrazovka obsahuje dva štítky, které oddělují dvě sekce tlačítek od sebe, každému z nich jsem nastavila horní margin. První tlačítko na sobě má obrázek. Tlačítko jsem proto ještě dodatečně vložila na Pane, která umožňuje všechny elementy na ní absolutně polohovat. Pak jsem ze záložky Controls vložila ImageView, kterému jsem nastavila obrázek ze složky, který jsem měla připravený už z analýzy podkladů. Zbylá dvě tlačítka jsem už vytvořila jako vlastní komponenty.

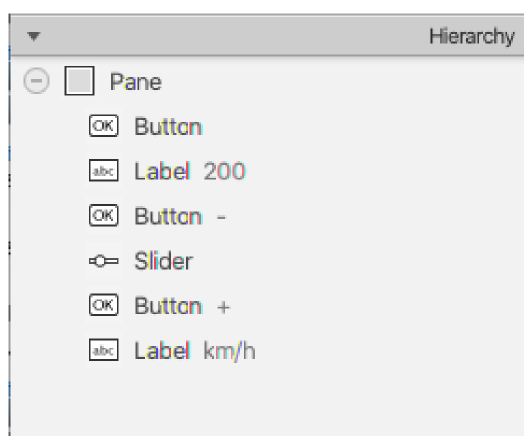
CheckBoxButtonControl tvoří Pane, na které je umístěné tlačítko a zaškrtávací pole. Rozložení komponenty jsem taktéž vytvořila v nástroji Scene Builder. Při stisku na tlačítko se pole zaškrtne a naopak. Komponenta má

přidané dva *Property* objekty - *BooleanProperty checkboxSelected* a *StringProperty text*, které můžeme vidět v rozhraní Scene Builderu (viz obrázek 4.7) a též tam měnit jejich nastavení. Property *checkboxSelected* je přes metodu *bindDirectional()* napojená na property *selected* zaškrtávacího pole, které je součástí komponenty. Použití metody *bindDirectional()* znamená, že změna *checkboxSelected* se projeví na zaškrtávacím poli a zároveň zaškrnutí nebo odškrtnutí pole změní hodnotu *checkboxSelected*. Tohoto mechanismu využívám pro změnu hodnoty pole přes serverové zprávy. Protože je komponenta rozšiřuje třídu Pane, máme přístup jen k metodám této třídy. Vlastnosti prvků uvnitř Pane se musí vytvořit jako vlastnosti celé komponenty, aby bylo možné k nim přistupovat zvenčí. Takto je možné ze serveru měnit hodnotu property *checkboxSelected*, čímž se změní i hodnota property *selected* a tato změna se projeví fyzicky na zaškrtávacím poli. Property *text* slouží k přístupu k textu tlačítka uvnitř komponenty a property *text* tlačítka je na ni jednosměrně vázaná metodou bind().

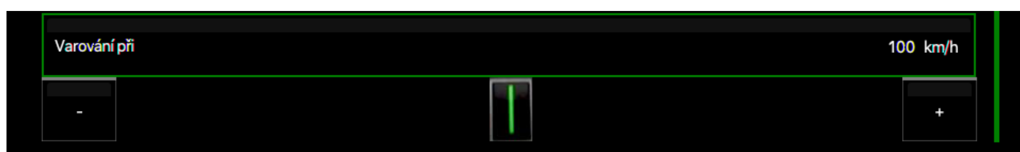


Obrázek 4.7: Ukázka polí pro zadání hodnoty property v nástroji Scene Builder

Poslední komponentou je *SingleSliderControl*. Protože dokud horní pole není zaškrtnuté, nemá být prvek aktivní, vpravo v záložce Properties jsem zaškrtla pole Disabled a tím prvek zneaktivnila. Podobně jako předchozí komponenta je vytvořena pomocí Pane, na které je umístěné tlačítko. V souboru *SingleSliderControl.java* je řešena funkcionalita komponenty, kdy po kliknutí na tlačítko vyjede zespoda posuvník a dvě tlačítka, která posuvník ovládají (viz obrázek 4.9). I tuto komponentu jsem vytvořila pomocí Scene Builderu. Hierarchii jednotlivých prvků této komponenty vidíme na obrázku 4.8. Komponenta má dvě property pro úpravu textu tlačítka a hodnotu posuvníku. Tyto hodnoty tedy můžeme měnit a naslouchat na změny pomocí serverových zpráv



Obrázek 4.8: Hiearchie prvků komponenty *SingleSliderControl*

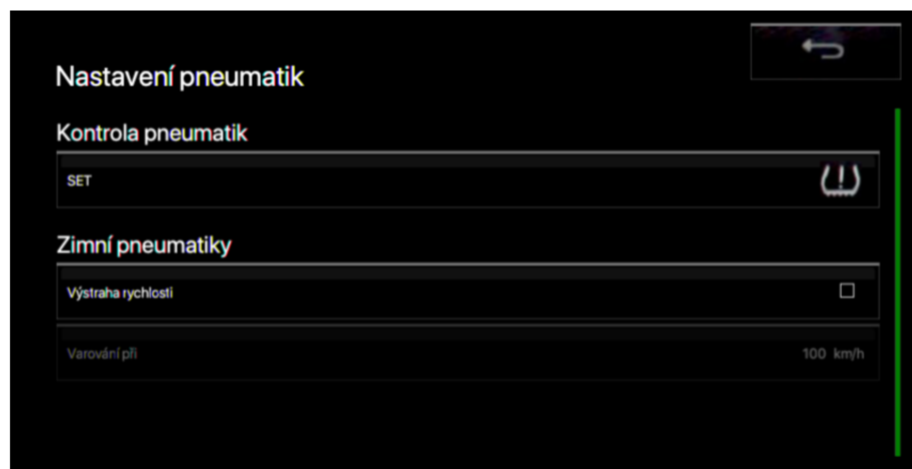


Obrázek 4.9: Vlastní komponenta *SingleSliderControl*

Na obrázku 4.10 je tato komponenta bez aplikovaných kaskádových stylů.

Obrázek 4.10: Vlastní komponenta *SingleSliderControl* bez úpravy vzhledu pomocí CSS

Konečné rozhraní obrazovky je zobrazeno na obrázku 4.11. Obrazovku je možné si ve Scene Builderu prohlédnout v náhledu v menu Preview - Show Preview in Window. Pokud je funkcionalita daného prvku implementována v kontroleru v anotované metodě reagující na nějakou akci, tato implementace se projeví i v náhledu.



Obrázek 4.11: Celý layout obrazovky *CarSettings2.fxml*

Kapitola 5

Výsledky

Výsledkem práce je knihovna tříd pro jednodušší tvorbu prototypů dotykových uživatelských rozhraní automobilů, server simulující hardwarové ovladače simulátoru automobilu a prototyp automobilového dotykového rozhraní, který používá postupy vytvořené pro zjednodušení tvorby takových prototypů. Vše bylo naprogramováno pomocí knihoven JavaFX. Rozhraní aplikace serveru a prototypu byla navržena WYSIWYG editorem Scene Builder, který umožňuje vizuálně navrhnout uživatelské rozhraní a využít při tom i vlastní UI komponenty. Změnu vzhledu zajišťují kaskádové styly, a dohromady tyto technologie umožňují změnit look-and-feel aplikace, jak bylo demonstrováno v předchozí kapitole. JavaFX umí pracovat s dotykovými událostmi a audio soubory. Výsledný prototyp byl otestován na dotykovém tabletu s operačním systémem Windows, aby se ověřilo chování na zařízení podobné cílovému. Aplikace na dotyky reagovala správně.

Oba programy se spouští jako aplikace ve formátu .jar. K jejich spuštění je třeba mít na počítači nainstalované JRE (Java Runtime Environment)[1] Přiložené audio soubory se musí nacházet ve stejné složce jako aplikace prototypu.

Aplikace se spouští z příkazové řádky s parametry. V příkazové řádce navigujeme do složky s .jar soubory a zadáme příkaz *java -jar TouchInCarServer.jar <port>*. Místo <port> zadáme číslo portu, na který se má server připojit. Celý příkaz může vypadat např. takto: *java -jar TouchInCarServer.jar 8888*. Následně v novém okně terminálu spustíme ve složce s .jar soubory příkaz

[1] https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

java -jar Prototype.jar <adresa> <port>. Místo <adresa> zadáme IP adresu a port serveru. Celý příkaz bude např. *java -jar Prototype.jar localhost 8888*.

Server zasílá zprávy v zadaném formátu, který prototyp umí rozpoznat a reagovat na ně. Může zasílat navigační zprávy pro pohyb v prototypu, monitorovací zprávy pro zjišťování změn hodnot prvků a také zprávy pro samotnou změnu prvků. Tyto zprávy jsou navržené pro využití pro testování podobných prototypů v simulátoru automobilu. Reakce na zaslané zprávy, jako např. informace o změně monitorovaného prvku, se zobrazují v GUI serveru a také se ukládají do logovacího souboru *log.txt*. Představím několik scénářů, které mohou být předmětem takového testování.

Scénář 1. Participant jede autem den po změně zimního času. Na rozhraní si všimne, že není nastaven aktuální čas. Chce to napravit.

- Moderátor před testem nastaví hodnotu zaškrtávacího pole "Automaticky nastavit letní čas" na false: "Settings2-checkbox-checkboxSelected-set-false"
- Nastaví listener na checkboxSelectedProperty tohoto prvku: "Settings2-checkbox-checkboxSelected-listener-add"
- Participant najde nastavení, otevře položku Čas a Datum a stiskne tlačítko "Automaticky nastavit letní čas", čímž se zaškrtne odpovídající pole. Do logovacího souboru a GUI serveru se zapíše zpráva o změně tohoto prvku. Participant úkol splnil.

Scénář 2. Participant si zapne rádio, ale připadá mu, že by se v hudbě mu zamlouvalo více basů. Chce si tedy zvýšit nastavení basů. Ke změně nastavení chce použít joystick.

- Moderátor před testem nastaví listener na stránce Sound2 na prvek třetího posuvníku, který odpovídá nastavení basů: "Sound2-thirdSlider-value-listener-add".
- Participant otevře stránku Radio a pustí nějakou rádiovou stanici.
- Potom pomocí joystickem naviguje do nastavení zvuku, provedou se tyto události: "Back, Click, Down, Down, Click, Down, Down, Down, Click, Click, Click". Do logovacího souboru a GUI serveru se zapíše zpráva o změně tohoto prvku. Participant úkol splnil.

Scénář 3. Participant má za úkol zavolat na číslo 123456789.

- Moderátor před testem nastaví listener na stránce PhoneCalling na prvek štítku, které zobrazuje volané číslo: "PhoneCalling-numberLabel-text-listener-add".
- Participant otevře menu Telefon a klikne na tlačítko Volat číslo.
- Na dotykové klávesnici participant zadá volané číslo a stiskne tlačítko s ikonou telefonu pro potvrzení hovoru.
- Otevře se nová stránka s rozhraním probíhajícího hovoru. Do GUI serveru a logovacího souboru se zapíše zpráva o změně daného prvku i s jeho novou hodnotou, kterou moderátor potvrdí jako správnou. Participant úkol splnil.

5.1 Shrnutí poznatků a postupů

Nespornou výhodou při navrhování uživatelského rozhraní je použití nástroje Scene Builder. Použití nástroje je doporučeno i pro tvorbu rozložení vlastní komponenty, jak je ukázáno v předchozí kapitole.

Je dobré zavést si nějakou konvenci k pojmenovávání jednotlivých souborů obrazovek a *fx:id* jednotlivých elementů. V implementovaném prototypu mají soubory hlavního menu základní názvy jako např. *Phone* pro menu Telefon. Soubory popisující obrazovky obsahů jednotlivých jsou buď dále očíslované podle pořadí tlačítka, které na stránku vede (např. *Settings2*, pokud na danou stránku odkazuje druhé tlačítko na obrazovce *Settings*) nebo podle názvu takového tlačítka (např. *CarSettings*). Pro přiřazování *fx:id* se nabízí vytvořit zkratku z prvního písmene každého slova v názvu prvku a čísla pořadí prvku stejného typu na dané obrazovce. První prvek daného typu bude mít k *fx:id* přiřazeno číslo 1, pokud se na aktuální obrazovce vyskytuje takových prvků víc, každý dostane číslo podle svého pořadí na obrazovce.

Názvy souborů obrazovek bez přípony.fxml pak budou sloužit jako parametry pro metody ve třídě *StageController* pro přesun mezi obrazovkami.

Pro účely testování je třeba se ujistit, že všechny monitorované prvky mají nastavené *fx:id*, aby bylo možné je serverovou zprávou vyhledat.

Navigace pomocí serverových zpráv probíhá pomocí klávesových šipek a klávesy SHIFT. Pro volbu prvku se používá mezerník. Aby bylo možné mezi prvky správně navigovat, nesmí se prvky navzájem překrývat.

Doporučený postup pro vytváření vlastních UI komponent je popsán v sekci 3.4. Jak již bylo řečeno, pro účely testování je nezbytné, aby vytvářené komponenty měly své vlastnosti implementovány pomocí rozhraní *Property* . Jedině tak lze na danou vlastnost přidat posluchač pro informování o změnách hodnot vlastnosti.

Při úpravě nějaké hodnoty, jejíž změna se má zobrazit i na jiné obrazovce, lze tento prvek v kontroleru původní stránky lokallizovat pomocí metody *getObjectFromNameSpace()*, která za parametry bere název obrazovky a *fx:id* prvku, na kterém se má změna projevit. Změny tomuto nalezenému prvku rovnou nastavujeme v kontroleru stránky, kde je tato změna iniciována. Tento postup je demonstrován na dvou místech implementovaného prototypu - v souboru PhoneCallController pro nastavení volaného čísla na obrazovku probíhajícího hovoru, a v souboru Settings2Controller

5.1.1 Finální podoba zasílaných zpráv

Jsou naimplementované následující typy zpráv zasílaných serverem:

- **Up, Right, Down a Left** pro pohyb do nahoru, dolů a do stran. Posun tlačítky se vypisuje do GUI serveru a do logovacího souboru.
- **Click** pro stisknutí tlačítka. Tato akce se vypisuje do GUI serveru a do logovacího souboru.
- **Back** pro návrat na předchozí obrazovku. Tato akce se vypisuje do GUI serveru a do logovacího souboru.
- **Obrazovka-objekt-property-set-hodnota** pro nastavení hodnoty vlastnosti (property) objektu, který se nachází na obrazovce. Do GUI serveru a logovacího souboru se vypíše potvrzení o změně hodnoty.
- **Obrazovka-objekt-property-get** pro získání hodnoty property objektu, který se nachází na obrazovce. Do GUI serveru a logovacího souboru se vypíše hodnota této property objektu.
- **Obrazovka-objekt-property-listener-add** pro přidání posluchače na property objektu, který se nachází na obrazovce. Do GUI serveru a logovacího souboru se vypisují změny hodnot dané property.

- **Obrazovka-objekt-property-listener-delete** pro odebrání posluchače z property objektu, který se nachází na obrazovce.

Závěr

Cílem této práce bylo navrhnout způsoby pro zjednodušení prototypování dotykových uživatelských rozhraní v autech. Prototypy vzniklé touto cestou byly určeny pro testování v simulátoru automobilu, kde participant testování řídí simulátor a přitom ovládá dotykové rozhraní a plní v rozhraní úkoly, které zadává moderátor testu. Simulátor také obsahuje hardwarové ovladače, kterými uživatel může prototyp rozhraní ovládat. Dotyková rozhraní v automobilech je třeba pečlivě otestovat, protože při interakci s nimi uživatel nedostává žádnou zpětnou vazbu kromě vizuální, narozdíl např. od tlačítek. Testování rozhraní s uživatelem umožní zjistit, do jaké míry současný návrh rozhraní ovlivňuje řidičovu pozornost a zda není při interakci s rozhraním nevhodně rozptylován od řízení. Pokud by v testovaném rozhraní vyšel najevo nějaký problém, je potřeba ho odstranit. Použití prototypů je časově i finančně méně náročně na výrobu i případné opravy. Při testování se může použít i více iterací prototypu, dokud se nedojde k závěrům, které jsou vyhovující pro implementaci do finálního rozhraní.

K práci mi bylo zadáno několik požadavků, které by prototyp dotykového rozhraní měl splňovat. Na jejich základě jsem zkoumala typy prototypů a vybrala kódovaný prototyp jako nejvhodnější typ prototypu pro účely testování. Dále jsem analyzovala různé technologie používané pro návrh uživatelských rozhraní a zjišťovala, zda splňují předepsané požadavky. Mezi požadavky patřila možnost snadné změny "look-and-feel" prototypů, možnost vytvářet rozhraní prototypu ve WYSIWYG editoru, kompatibilita prototypů s operačním systémem Windows, podpora dotykových gest, multimediálních souborů, schopnost komunikace s hardwarovými ovladači simulátoru přes protokol TCP a možnost ukládat data do textového souboru. S analyzovaných technologií jsem vybrala knihovnu JavaFX, která všechny požadavky splňovala.

Výsledkem práce je knihovna tříd a sada doporučení, jež usnadňují tvorbu prototypů dotykových uživatelských rozhraní automobilů. Dále server, který je schopen zasílat zprávy v předepsaném formátu a tím simulovat hardwarové ovladače simulátoru automobilu. Knihovnu tříd a doporučení jsem využila v implementaci kódovaného prototypu automobilového dotykového rozhraní, abych demonstrovala použití postupů pro zjednodušení tvorby takových prototypů vytvořených v rámci této práce. Prototyp se jako klient připojí k serveru a následně od něj přijímá zprávy, které zpracuje a na jejich základě provede odpovídající akci, např. začne monitorovat vlastnost zadaného prvku a změny jejích hodnot ukládat do souboru. Pomocí zpráv lze také simulovat pohyb v prototypu pomocí hardwarového joysticku či změnit hodnotu prvku na jiné obrazovce než na aktuální. Implementovaný prototyp byl otestován na dotykovém tabletu s operačním systémem Windows. Zároveň byl otestován i server na jiném zařízení na stejné síti a vzájemná komunikace s prototypem. Aplikace správně reagovala na dotyky a zpracovávala zprávy ze serveru.

Celá práce byla implementována použitím knihovny JavaFX. Rozhraní aplikace serveru a prototypu bylo navrženo pomocí vizuálního editoru Scene Builder. Tento nástroj také umožňuje import vlastních UI komponent, které se potom dají v návrhu rozhraní využít. Při implementaci prototypu jsem zjistila, že navržená knihovna společně s nástrojem Scene Builder velice usnadňuje tvorbu kódovaných prototypů.

6.1 Možnosti rozšíření

Nabízí se různé možnosti rozšíření. Je možné upravit architekturu aplikaci z klient-server na architekturu s více klienty, kdy každý klient může představovat buď hardwarový ovladač, nebo uživatelské rozhraní. Umožňovalo by to klientovi s grafickým uživatelským rozhraním ovládat hardwarové prvky, např. stisknutím tlačítka na obrazovce rozsvítit diodu. Zvolená technologie také umožňuje tvorbu prototypů s méně běžnými prvky, jako jsou velké dotykové obrazovky nebo webový prohlížeč přímo na obrazovce rozhraní automobilu. Do vytvořené knihovny je tak možné přidat i podporu těchto prvků a připravit ji tak na využití pro prototypování budoucích uživatelských rozhraní v automobilovém průmyslu.

Příloha A

Obsah přiloženého CD

- server_src.zip - zdrojové kódy serveru
- prototyp_src.zip - zdrojové kódy prototypu
- Prototypovani_dotykovych_uzivatelskych_rozhrani_v_automobilech.pdf - elektronická verze textu diplomové práce
- text_src.zip - zdrojové soubory textu diplomové práce
- analyza_src.zip

Příloha B

Literatura

[1] Virage Simulation Inc.: COST-EFFECTIVE CAR DRIVING SIMULATOR – VS300. [Online; cit. 2019-10-21]. Dostupné z: https://viragesimulation.com/vs300-cost-effective-car-driving-simulator/

[2] MCCADE, S.: Prototyping: an integral part of every designer's workflow. 2018, [Online; cit. 2019-10-21]. Dostupné z: https://uxdesign.cc/prototyping-an-integral-part-of-every-designers-workflow-1e7bceea48dd/

[3] MÍKOVEC, Z.: Prototyping. 2017, [Online přednáška, cit. 2019-10-21]. Dostupné z: https://moodle.fel.cvut.cz/pluginfile.php/69999/course/section/17557/nur_lecture03_prototyping_low-v07.pdf

[4] "prototype". Cambridge Advanced Learner's Dictionary & Thesaurus. Dostupné z: https://dictionary.cambridge.org/dictionary/english/prototype

[5] URIAS, E.: 10 Reasons Why You Should Prototype before Developing. 2018, [Online; cit. 2019-10-20]. Dostupné z: https://invidgroup.com/10-reasons-why-you-should-prototype-before-developing/

[6] ARNOWITZ, J.; Arent, M.; BERGER, N.: *Effective Prototyping for Software Makers*. Elsevier Science & Technology, 2007, ISBN ISBN 0120885689.

[7] MORAN, K.: Usability Testing 101. 2019, [Online; cit. 2019-10-21]. Dostupné z: https://www.nngroup.com/articles/usability-testing-101/

[8] WESTLAND, J. C.: The Cost of Errors in Software Development: Evidence from Industry. *Journal of Systems and Software*, ročník 62, č. 1,

Květen 2002: str. 1–9, ISSN 0164-1212, [Online; cit. 2019-10-20. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.1508&rep=rep1&type=pdf

[9] PERNICE, K.: UX Prototypes: Low Fidelity vs. High Fidelity. 2016, [Online; cit. 2019-10-21]. Dostupné z: https://www.nngroup.com/articles/ux-prototype-hi-lo-fidelity/

[10] MÍKOVEC, Z.: Prototyping & Evaluation. 2017, [Online přednáška, cit. 2019-10-21]. Dostupné z: https://moodle.fel.cvut.cz/pluginfile.php/69999/course/section/17557/nur_lecture04_prototyping_high_eval-v07.pdf

[11] GREEN, P.; WEI-HAAS, L.: The Wizard of Oz: a tool for rapid development of user interfaces. 1985. Dostupné z: https://deepblue.lib.umich.edu/bitstream/handle/2027.42/174/71952.0001.001.pdf?sequence=2

[12] Axure Software Solutions, Inc.: Axure. 2019, [Online; cit. 2019-11-2]. Dostupné z: https://www.axure.com/

[13] Web technology for developers: Mozilla. 2019, [Online; cit. 2019-11-2]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web

[14] W3Schools: How TO - Toggle Switch. 2019, [Online; cit. 2019-11-2]. Dostupné z: https://www.w3schools.com/howto/howto_css_switch.asp

[15] Tutorials Point: WPF - Overview. 2019, [Online; cit. 2019-11-5]. Dostupné z: https://www.tutorialspoint.com/wpf/wpf_overview.htm

[16] Microsoft: Styles and templates in WPF. 2019, [Online; cit. 2019-11-5]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/styles-templates-overview

[17] Gluon: Scene Builder. 2019, [Online; cit. 2019-11-5]. Dostupné z: https://gluonhq.com/products/scene-builder/

[18] Oracle: JavaFX CSS Reference Guide. 2015, [Online; cit. 2019-11-5]. Dostupné z: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

[19] Oracle: JavaFX Overview. 2014, [Online; cit. 2019-11-5]. Dostupné z: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm

[20] The Qt Company: Qt. 2019, [Online; cit. 2019-11-6]. Dostupné z: https://www.qt.io/

[21] Oracle: Interface ChangeListener<T>. 2015, [Online; cit. 2019-11-30]. Dostupné z: https://docs.oracle.com/javase/8/javafx/api/javafx/beans/value/ChangeListener.html

[22] VOS, J.; GAO, W.; WEAVER, J.; aj.: *Pro JavaFX 8: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients*. USA: Apress, první vydání, 2014, ISBN 1430265744.

[23] Oracle: IOverview (JavaFX 8). 2015. Dostupné z: https://docs.oracle.com/javase/8/javafx/api/overview-summary.html

[24] Friesen, J.: Exploring JavaFX's Application class. 2016, [Online; cit. 2019-12-13]. Dostupné z: https://www.javaworld.com/article/3057072/learn-java/exploring-javafxs-application-class.html

[25] HUNT, J.: *Java and Object Orientation*. Springer, apr 2002, ISBN 1852335696, str. 335.

[26] MCCLUSKEY, G.: Using Java Reflection. 1998. Dostupné z: https://www.oracle.com/technical-resources/articles/java/javareflection.html