FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Python GUI: Tkinter API pro Qt
Student:	Bc. Matěj Schuh
Vedoucí:	Ing. Miroslav Hrončok
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Analyzujte API knihovny Tkinter ze standardní knihovny jazyka Python.
Analyzujte API frameworku Qt a jeho nadstaveb pro jazyk Python (PyQt, PySide).
Analyzujte možnosti, jak mapovat Tkinter API na GUI komponenty frameworku Qt.
Navrhněte implementaci těch možností, které dávají smysl.
Navrhněte kvantifikátory pro měření úspěšnosti mapování Tkinter API na GUI komponenty frameworku Qt.
Zvolte jednu z možností implementace a implementujte prototyp jako knihovnu pro Python pod permisivní svobodnou licencí. Součástí implementace musí být automatické testy a anglická dokumentace.
Implementaci otestujte na aplikaci Python IDLE podle definovaných kvantifikátorů.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 24. února 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Python GUI: Tkinter API pro Qt

Bc. Matěj Schuh

Katedra softwarového inženýrství
Vedoucí práce: Ing. Miroslav Hrončok

6. května 2021

Poděkování

Na tomto místě bych chtěl přednostně poděkovat svému vedoucímu Ing. Miroslavu Hrončokovi, který mi vždy pomohl s implementací, návrhem i směrem mé Diplomové práce. Dále bych chtěl poděkovat rodině a přátelům, kteří mi byli během zpracovávání oporou.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. května 2021 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Schuh, Matěj. *Python GUI: Tkinter API pro Qt*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021. Dostupný také z WWW: ⟨`https://github.com/SohajCZ/dumpTk`⟩.

Abstrakt

Tato Diplomová práce je zaměřena na tvorbu grafického uživatelského rozhraní v jazyce Python. Grafické rozhraní využívají grafických možností počítačů a dalších zařízeních pro zjednodušení komunikace mezi počítačem a uživatelem, díky čemuž může být počítač ovládán i méně zkušeným uživatelem.

Teoretická část práce je věnována vlastnímu grafickému rozhraní a jeho komponentům, s důrazem na možnosti implementace grafického rozhraní v programovacím jazyku Python, pomocí knihovny Tkinter, využívající Tk komponenty, nebo balíčky PyQt a PySide, využívajících komponenty z frameworku Qt.

Cílem praktické části je analýza a návrh řešení, které by umožnilo aplikacím používajícím knihovnu Tkinter využívat k vykreslení grafického rozhraní komponenty z frameworku Qt. K navrženému řešení je implementován prototyp, na kterém je zhodnocena úspěšnost návrhu.

Práce poskytuje přehled o možnostech grafického uživatelského rozhraní, jeho stavebních prvcích a o jeho možnostech implementace v Pythonu. Práce může být přínosná nejen pro ty, kteří se chtěji dozvědět více o vnitřních funkcích a závislostech Tkinteru, PyQt a PySide, ale rovněž těm, kterým nevyhovují k tvoření rozhraní Tk komponenty, ale chtějí zachovat možnosti programování pomocí Tkinteru.

Klíčová slova API, GUI, PyQt, Python, Qt, Tcl, Tk, Tkinter

Abstract

This diploma thesis is focused on the creation of graphics Python user interface. Graphical interfaces are using graphical capabilities of computers to simplify communication between the computer and the user, making it possible the computer is controlled even by a less experienced user.

The theoretical part of the work is devoted to graphical interface and its components, with emphasis on the possibility of implementing a graphical interface in the Python programming language, using the Tkinter library with Tk components or PyQt and PySide bindings using components from the Qt framework.

The aim of the practical part is the analysis and design of solution, which would allow applications using the Tkinter library to draw a graphical interface components from the Qt framework. To the designed solution is then prototype implemented and evaluated.

This diploma thesis provides an overview of the possibilities of graphical user interface, its components and its implementation options in Python. The work can be beneficial not only for those who seek how Tkinter, PyQt or PySide work internally, but also for those that don't like to create an interface with Tk components, but want maintain programming capabilities of Tkinter library.

Keywords API, GUI, PyQt, Python, Qt, Tcl, Tk, Tkinter

Obsah

Seznam obrázků

Úvod

Tato Diplomová práce je zaměřena na tvorbu grafického uživatelského rozhraní v jazyce Python. Grafické rozhraní využívají grafických možností počítačů a dalších zařízeních pro zjednodušení komunikace mezi počítačem a uživatelem, díky čemuž může být počítač ovládán i méně zkušeným uživatelem.

Teoretická část této Diplomové práce se věnuje obecně pojmu grafického uživatelského rozhraní (GUI), popisuje jeho komponenty a možnosti interakce s uživatelem. Dále teoretická část práce popisuje nutné pojmy související s tvorbou GUI, současně s představením existujících sad nástrojů, jako jsou Tk a Qt, knihovny Tkinter a balíčky PyQt a PySide pro programovací jazyk Python. Tyto nástroje a balíčky teoretická část detailně popisuje a mezi sebou vzájemně srovnává. Teoretická část je zakončena shrnutím motivace ke zpracování tohoto tématu.

Praktická část této Diplomové práce se věnuje implementaci nástroje, který by umožnil aplikacím napsaným pomocí knihovny Tkinter využívat komponenty ze sady nástrojů Qt místo sady Tk. Praktická část se dělí na čtyři části – analýzu, návrh, realizaci a testování.

Analýza se blíže věnuje knihovně Tkinter a balíčkům PyQt a PySide. Probírá detailně správce rozložení, události, dialogy a dalším funkce GUI. Část návrhu je věnována návrhům mapování, pomocí kterých by mělo být dosaženo cíle práce. Navržená mapování Tkinteru na Qt komponenty jsou zkoumány z hlediska navržených parametrů, dle kterých dojde k výběru mapovaní pro implementaci. Návrh je zakončen definováním kvantifikátorů, které změří úspěšnost implementovaného mapování.

Část realizace popisuje implementovaný prototyp mapování, včetně všech jeho možností, vysvětlení návrhových rozhodnutí a dalších detailů. V poslední části je popsán proces testování implementovaného prototypu a prototyp je testován na aplikaci IDLE. Část testování je ukončena zhodnocením implementovaného prototypu dle definovaných kvantifikátorů.

Kapitola 1

Cíl práce

1. Analyzujte API knihovny Tkinter ze standardní knihovny jazyka Python.
2. Analyzujte API frameworku Qt a jeho nadstaveb pro jazyk Python (PyQt, PySide).
3. Analyzujte možnosti, jak mapovat Tkinter API na GUI komponenty frameworku Qt.
4. Navrhněte implementaci těch možností, které dávají smysl.
5. Navrhněte kvantifikátory pro měření úspěšnosti mapování Tkinter API na GUI komponenty frameworku Qt.
6. Zvolte jednu z možností implementace a implementujte prototyp jako knihovnu pro Python pod permisivní svobodnou licencí. Součástí implementace musí být automatické testy a anglická dokumentace.
7. Implementaci otestujte na aplikaci Python IDLE podle definovaných kvantifikátorů.

Kapitola 2

Teoretická část

Teoretická část práce je nejprve věnována bližšímu přiblížení pojmu grafického uživatelského rozhraní a jeho standardním komponentům. V dalších kapitolách této části se věnuji možnostem implementace GUI v Pythonu, pomocí knihovny Tkinter a balíčků PyQt a PySide.

2.1 Python

Python je moderní programovací jazyk vyvinutý Guido van Rossumem. Je to interpretovaný, platformně nezávislý, interaktivní a objektově orientovaný jazyk. Představuje širokou paletu možností, díky čemuž umožňuje rychle vytvářet aplikace rychleji než v tradičních jazycích, jako jsou C, C++, nebo Java[1]. Python je od počátku vyvíjen jako otevřený software pod licencí Python Software Foundation Licence (PSFL) [1, 3-14].

Shrnutí vlastností Pythonu:

- Python disponuje automatickou správou paměti.
- Jednoduchá syntaxe a sémantika.
- Použitelný pro skriptování i rozsáhlé programy.
- Vyspělá asociativní pole, rozšiřitelná syntaxe tříd.
- Výkonné knihovny pro numerické výpočty, uživatelská rozhraní, manipulace s obrázky a další.
- Podpora široké komunity uživatelů.
- Snadná integrace s jinými programovacími jazyky [1, 14].

[1] Dle mého názoru si také získal mnoho příznivců díky své syntaxi odpovídající pseudokódu.

2.2 Interaktivní přístup

Python má od verze 1.5.2 (rok 1999) 2 možnosti spuštění:

1. Základní režim – Spuštění pod Unixem pomocí příkazu ‘python‘.
2. IDLE – Integrated DeveLopment Environment – Po vstupu do tohoto nástroje je uživateli zpřístupněn základní režim, ale v aplikaci, bez nutnosti zapnutí terminálu. IDLE nabízí možnosti jako automatické řádkování, zvýraznění syntaxe a pohyb ve zdrojovém kódu pomocí obvyklých ovládacích prvků a Emacs standardu [1, 27].

2.3 GUI

GUI je zkratka z anglického Graphical User Interface, česky grafické uživatelské rozhraní. Toto rozhraní umožňuje lidem interagovat a komunikovat s počítači a dalšími zařízeními, v dnešní době například s mobilními telefony. K tomu GUI využívá grafických možností zařízení pro zjednodušení komunikace zakrytím detailů programovacího jazyka uživateli.

GUI využívá různých prvků – okna, ikony, nabídky, tlačítka, dialogy a další prvky – jako prostředníky pro komunikaci mezi uživatelem a počítačem. Těmto prvkům se také říká „widgety“. Uživatel je obvykle aktivuje nebo s nimi manipuluje pomocí myši nebo jiným ukazovátkem (např. dotykem na dotykovém displeji). Pro zjednodušení některých častých operací může být využito klávesových zkratek.

GUI je většinou řízeno pomocí událostí, jakýkoliv úkol je tedy zpracován na základě vygenerované a detekované události, kterou mohl (ale nemusel) spustit uživatel. K uložení rozepsaného textu tak může dojít automaticky na základě časovače, stisknutím klávesových zkratek uživatelem a nebo pomocí akce uložení v menu aplikace. V případě výstupu (např. otevření dialogu) je výstup zobrazen uživateli na obrazovku.

Pro porovnání je v případě CLI – z anglického Command Line Interface, česky rozhraní příkazové řádky – uživateli prezentována prázdná obrazovka a příkazový řádek. Komunikace se zařízením probíhá pomocí příkazové řádky, kam uživatel zadává příkazy. Příkazová řádka předpokládá uživatelovu znalost programovacího jazyka a interpretuje zadaný příkaz jako úkol. V minulosti byl takto realizován např. operační systém MS-DOS. Příkazová řádka je v současné době stále v operačních systémech dostupná, nicméně je obvykle zakrytá grafickým prostředím [2].

2.3.1 Komponenty GUI

Výsledný výstup grafického uživatelského rozhraní záleží na operačním systému, platformě a účelu GUI. Většina GUI má ale podobné komponenty, jak popsáno v [2]:

- **Okna** jsou jedním z hlavních grafických mechanismů používaných ke seskupování a izolování funkcí softwaru založeného na grafickém uživatelském rozhraní. Okno rozdělí obrazovku na oddělené oblasti, kde může uživatel spustit jiný program, zobrazit adresáře, zobrazit jiný graf atd.

- **Desktop** je hlavní oblastí ve většině operačních systémů, také nazývaný plochou. Dalo by se to považovat za hlavní okno, kde se obvykle zobrazují ikony.

- **Ikony** jsou grafikou nebo obrázky, které představují úkoly, programy, složky, soubory, okna a další. Zjednodušují tak označení akce, která je po nimi skrytá. Obvykle jsou uspořádány na ploše, na panelech nástrojů, v adresářích a nabídkách. Hlavním účelem ikon je sdělit uživateli užitečné informace o úkolu, který se vyvolá kliknutím na ikonu.

- **Ukazovací zařízení** – jedná se obvykle o myš, trackball nebo dotykovou podložku. Polohovací zařízení pohybuje kurzorem na obrazovce. Umožňuje uživateli vybrat objekty na obrazovce a klepnutím aktivovat komponenty uživatelského rozhraní, přetáhnout soubory nebo objekty, změnit velikost oken atd. Důležitou funkcí, které umožňuje grafické rozhraní, je naznačení možnosti akce pod tlačítkem či ikonou, po najetí kurzorem, pomocí jeho změny.

- **Nabídka** případně menu se používá ve velké většině programů, které mají grafické rozhraní. Nabízí seznam dostupných možností, ze kterých si uživatel může vybrat. Často jsou tyto akce kombinovány se zkratkami zadaných pomocí klávesnice.

- **Zrychlená nabídka** – tento typ nabídky se aktivuje kliknutím pravým tlačítkem myši. Objeví se různé nabídky a možnosti v závislosti na oblasti, kde ke kliknutí dochází. Komponenta je tedy svázána s kontextem, také označována jako kontextová nabídka.

- **Rozbalovací seznam** se používá v případě výběru z velkého množství možností. Ty obvykle zobrazují aktuální hodnotu pro uživatelské rozhraní a mají šipku směřující dolů, což naznačuje, že je k dispozici více možností. Uživatel může kliknutím na šipku zobrazit nabídku nebo seznam dalších možností. Některé nabídky nabízejí vyhledávání či filtrování možností pomocí psaní do nabídky.

- **Dialogová okna** nebo vyskakovací okna – od těch, které poskytují pouze informace, po ty, které uživatele požádají o zadání něčeho nebo o výběr. Jedním příkladem dialogového okna je okno, ve kterém má uživatel možnost vybrat soubor jako přílohu, vyskakující okno může být jakákoliv chybová hláška, kterou je třeba zobrazit uživateli.

- **Tlačítka** jsou použité v případě, kdy vývojář chce uživateli poskytnout diskrétní možnosti, kde po stisknutí tlačítka dojde k nějaké akci okamžitě. Může existovat několik tlačítek nabízejících celou řadu možností, kdy by mělo být výchozí tlačítko zvýrazněno (např. potvrzení formuláře). Ve formulářích může uživatel místo kliknutí na tlačítko stisknout klávesu Return; v takovém případě se zvýrazněná možnost provede.

- **Přepínače a zaškrtávací políčka** jsou podobná tlačítkům v tom, že nabízejí uživateli způsoby výběru z krátkého seznamu možností. Přepínače a zaškrtávací políčka jsou obvykle seskupeny a definovány tak, aby se možnosti vzájemně vylučovaly. Jinými slovy, uživatel může provést pouze jeden výběr.

- **Textová pole** nebo textové oblasti, umožňují uživateli zapsat text. Ve formulářích jsou užitečné, když má uživatel nekonečné množství možností. Příkladem tohoto typu rozhraní je pole adresy URL ve většině prohlížečů.

2.3.2 Interakce

Navrhování vizuální kompozice a chování grafického uživatelského rozhraní je důležitou součástí programování softwarových aplikací v oblasti interakce člověka s počítačem. Jeho cílem je zvýšit efektivitu a snadnost použití pro základní logický design uloženého programu, designovou disciplínu nazvanou použitelnost. Používají se metody designu zaměřené na uživatele, aby bylo zajištěno, že vizuální jazyk zavedený v návrhu je dobře přizpůsoben úkolům.

Zařízení lidského rozhraní (anglická zkratka HID) je počítačové zařízení, které je obvykle používání lidmi a které přijímá vstup od lidí a poskytuje výstup lidem. Pro efektivní interakci s grafickým uživatelským rozhraním HID pro vstup zahrnují počítačovou klávesnici, zejména používanou společně s klávesovými zkratkami, ukazovací zařízení pro ovládání kurzoru (nebo spíše ukazatele): myš, ukazovátko, touchpad, trackball, joystick. Vstupem může být i mikrofon pro hlasové ovládání. Pro výstup jsou použité obrazovky, reproduktory a případně vibrace, ovlivňující lidský hmat.[3]

2.3.3 Sada nástrojů

Sada nástrojů, neboli widget toolkit, je knihovna grafických komponentů (widgetů), ze kterých je GUI vytvářeno. Několik z těchto sad lze použít v různých programovacích jazycích, pokud jsou pro ně napsány balíčky s vazbami pro daný toolkit, jako je tomu v případě frameworku Qt v C++ a balíčku PyQt pro Python.

Vzhled a chování grafických ovládacích prvků lze oddělit, což umožňuje modifikaci grafických ovládacích prvků. GUI programu je obvykle konstruováno kaskádovitě, přičemž grafické ovládací prvky jsou přidávány přímo na sebe. S konstrukcí mohou pomoci vhodné editory pro daný toolkit, které je možné

použít WYSIWYG způsobem s využitím značkovacího jazyka. Vytvořený zdrojový soubor se značkovým jazykem pak může být načten při konstrukci GUI v samotném programu.[4]

Většina sad nástrojů pro widgety používá programování založené na událostech jako model interakce. Sada nástrojů zpracovává uživatelské události, například když uživatel klikne na tlačítko. Když je zjištěna událost, je předána aplikaci, kde je událost řešena.[5]

Níže uvádím několik příkladů toolkitů s jejich možným řazením dle [4]:

- Integrované v operačních systémech
  - Cocoa (OS X)
  - Windows API (Microsoft Windows - do roku 2006)
- Oddělená vrstva nad operačním systémem
  - X Window systém s Xt widgety, používáno toolkity OLIT, Motif, Xaw
    - X Window systém je okenní systém pro bitmapové displaye, častý v systémech založených na Unixu.
    - Periferie komunikují s X serverem, aplikace jsou připojeny k serveru jako X klienti.
- Závislé na operačním systému
  - Amiga – BOOPSI, MUI, ReAction, Zune
  - Mackintosh – Cocoa, MacApp, PowerPlant
  - Microsoft Windows – MFC (obalení Windows API v C++), WTL, QWL, VCL, Windows Forms, Windows Presentation Foundation
  - Unix – LessTif, MoOLIT, Motif, OLIT, Xaw, XView
- Multiplatformní – vždy vychází z nějaké existující technologie, programovacího jazyka:
  - C – Elementary, GTK+, IUP, Tk, XForms, XVT
  - C++ – CEGUI, FLTK, FOX, GLUI, gtkmm, Juce, Qt, wxWidgets
  - OpenGL – Clutter
  - Flash – Adobe Flash, Adobe Flex
  - Go – Fyne
  - XML – GladeXML, XAML, XUL
  - JavaScript – Cappuccino, jQuery Ui, React, Google Web Toolkit
  - SVG – Raphaël

- C# – Gtk#, QtSharp, WindowsForms
- Java - Abstract Window Toolkit, Swing, Apache Pivot, JavaFX/-FXML, Standard WIdget Toolkit
- Object Pascal – FireMonkey, IP Pascal, Lazarus LCL, fpGUI, CLX
- Objective-C – GNUstep, Cocoa
- Ruby – Shoes

2.4 Tkinter

Tkinter je standardizovaný balíček pro programování GUI pomocí Tk toolkitem v Pythonu. Byl vytvořen Steen Lumholtem a Guido van Rossumem. Jak Tk i Tkinter jsou dostupné na většině Unixových platformách, stejně tak na Windowsových systémech. Samotné Tk není součástí Pythonu.

Tkinter kromě samotného mapování na Tk objekty nabízí také další moduly. Nejdůležitější z nich je modul `tkinter.constants`, který obsahuje konstanty používané v Tkinteru. Tkinter si jej však importuje samostatně. Hlavní třída Tk lze vytvořit bez argumentů – je obvykle hlavním oknem aplikace. Každá instance má vlastní přidružený interpret Tcl.

Mezi další moduly, které tkinter nabízí, patří:

- `tkinter.colorchooser` – Dialog umožňující uživateli vybrat barvu.
- `tkinter.commondialog` – Základní třída pro dialogy definované v dalších zde uvedených modulech.
- `tkinter.filedialog` – Běžné dialogy umožňující uživateli určit soubor, který se má otevřít nebo uložit.
- `tkinter.font` – Obslužné programy usnadňující práci s písmy.
- `tkinter.messagebox` – Přístup ke standardním dialogovým oknům Tk.
- `tkinter.scrolledtext` – Textový widget se zabudovaným vertikálním posuvníkem.
- `tkinter.simpledialog` – Základní dialogy a praktické funkce.
- `tkinter.dnd` – Podpora přetažení (drag and drop). Experimentální – mělo by se stát zastaralým, až bude nahrazeno Tk DND.
- `turtle` – Turtle grafika v okně Tk.

Tkinter je možné importovat v Pythonu jako balíček ze standardní knihovny. Tento balíček je napsaný v Pythonu. Při volání rozdělí příkazy a argumenty

příkazu a konvertuje je do podoby, která odpovídá formě pro Tk. Tyto zpracovaná data pošle do modulu `_tkinter`, napsaném CPythonu. Funkce v programovacím jazyce C může dále volat ostatní moduly v C, tudíž i funkce pro Tk v C. Ty jsou implementovány v C a v Tcl, které Tk widgetů přiřazuje určité výchozí chování. Tk v jazyce C dále implementuje mapování do Xlib, knihovny, která obsahuje funkce pro komunikaci s X serverem.[6]

2.4.1 Tcl/Tk

Tcl Tcl je jednoduchý programovací jazyk, obsahující všechny obvyklé součásti programovacího jazyka. Tcl skripty jsou tvořeny příkazy oddělených pomocí středníků nebo odřádkování. Příkazy jsou složeny ze slov oddělených pomocí mezer. V Tcl je možné definovat proměnné, podpříkazy, řetězce a další. Běh programu je možné řídit pomocí obvyklých řídících struktur – `if, switch, for, while` a dalších. Dále umožňuje manipulaci se soubory, I/O operace, práci s časem a události. Obsahuje také struktury jako jsou listy a pole.[7]

Jazyk Tcl byl navržen jako flexibilní jazyk s malým jádrem, jednoduše upravitelný, díky čemuž byl používán například v Cisco routerech. Využití jazyka je několik:

- Webové aplikace – Díky síťovým možnostem umožňuje tvorbu webových aplikací. V Tcl byl vytvořen web server AOLServer, který nabízel i komerční řešení.
- Tvorba aplikací s GUI – Widget toolkit Tk rozšíří popularitu Tcl poskytnutím rozhraní pro jednoduché psaní aplikací s grafickým rozhraním pod Unixem a X11.
- Testování – Tcl přichází s testovacím frameworkem `tcltest`, na kterém jsou postaveny další testovací frameworky (např. `DejaGnu`).
- Databáze – Tcl poskytuje rozhraní pro připojení do SQL databáze, pomocí TDBC. Nabízí ovladače pro databáze `SQLite, MySQL, PostgreSQL` a další.
- Vývoj vestavěných řešení – Tcl je velmi kompaktní jazyk a je tak snadno integrovatelný se speciálním hardware, může tak být volbou pro vestavěný vývoj.[8]

Tk Tk je platformně nezávislou sadou nástrojů vytvořenou pro Tcl. Tk widgety jsou vysoce modifikovatelné, a to jak při vytvoření widgetu, tak i v době jeho existence, pomocí volání konfiguračních příkazů. Další funkce je možné přidávat pomocí příkazů, které je možné uložit a později volat. Existuje několik jazyků, pro které existují balíčky pro využití Tk widgetů.

Příkazy Tk pro Tcl mají vždy pevně danou strukturu. Pokud je widget vytvářen, skládá se z těchto částí – `classCommand newPathname options`, postupně:

1. `classCommand` – Žádaná třída widgetu, například `button`, tedy tlačítko.
2. `newPathname` – Název dané instance widgetu, např. `button1`.
3. `options` – Nastavení vlastností widgetu, vždy po dvojicích název vlastnosti a hodnota, ve formátu `-název hodnota`, tedy například `-fg red -text "Hello world!"`. Tímto nastavením bychom docílili vykreslení červeného textu obsahující zprávu „Hello world!“ na zadaném tlačítku.

Pokud je widget již vytvořen, je možné ho měnit pomocí příkazu složeného z následujících částí – `.button1 someAction someOptions`, postupně:

1. `.button1` – Jméno instance widgetu, kterou konfigurujeme, uvozené tečkou (použita instance tlačítka z ukázky založení widgetu).
2. `someAction` – Akce, která má být aplikována na instanci widgetu. Hodnoty tohoto parametru příkazu záleží na dané třídě, jejíž instancí je widget.
3. `someOptions` – Argumenty akce aplikované na instanci. Některé akce nepotřebují žádné argumenty, jiné mohou mít více argumentů.[6]

2.5 PyQt, PySide

Pro použití toolkitu Qt se nejčastěji v Pythonu používají balíčky PyQt nebo PySide. Oba balíčky mají vazby do Qt frameworku, každý z nich ale využívá jiný generátor vazeb.

2.5.1 Qt

Qt je multiplatformní toolkit, který umožňuje vytvoření aplikací pro Windows, Mac OS X, Linux a další systémy založené na Unixu. Velká část Qt toolkitu je věnována vytvoření platformně neutrálnímu rozhraní pro grafické prvky. Qt je licencováno duálně, komerční licencí pro komerční vývoj a open source licencí – GPL a za určitých podmínek LGPL licencí [9].

Qt bylo původně vyvinuto pro `C` a C++, existují však vazby pro mnoho dalších jazyků. Trolltech, společnost, která Qt vytvořila, poskytuje oficiální vazby pro C++, Javu a JavaScript. Třetí strany poskytují vazby pro další jazyky, jako je právě Python, ale i Ruby, PHP a .NET.

Qt přináší do C++ signály a sloty, které umožňují propojení objektů, zatímco usnadňují návrh a zvyšují opětovné použití kódu. Signál je metoda, která je místo volání emitována. Z pohledu programátora jde spíše deklarací prototypů

signálů, které by mohly být emitovány. Naopak slot je funkce, která je volána jako důsledek emitování signálu. Je možné připojit libovolný počet signálů na libovolný počet slotů. Když je vysílán signál, všechny sloty s ním spojené jsou vyvolány – pořadí není definováno.[10]

Qt je více než sada nástrojů, zahrnuje také abstrakce síťových soketů, podprocesů, Unicode, regulárních výrazů, databází SQL, SVG, OpenGL, XML, plně funkční webový prohlížeč, systém nápovědy, multimediální rámec a bohatou sbírku widgetů GUI.[11]

2.5.2 PyQt

PyQt je balíček, množina vazeb pro framework Qt. Vazby jsou implementovány jako moduly pro Python. PyQt je podporováno na všech platformách, na kterých je podporováno Qt.

Licence PyQt je duální, a to licencí GNU GPL verze 3 a licencí Riverbank Commercial License, která je určena pro komerční použití.

Pro Qt existuje program `Qt Designer` pro návrh grafického uživatelského rozhraní. Návrh je uložen do souboru, ze kterého PyQt umožňuje vygenerovat zdrojový kód pro Python.

PyQt kombinuje výhody Qt a Pythonu, umožňuje tedy uživateli využít Qt v plné síle pomocí jednoduchosti Pythonu.[11]

Balíček PyQt je sestaven pomocí generátoru vazeb SIP. Tento generátor vazeb byl původně vytvořen pouze pro PyQt, ale může být použit pro jakékoliv jiné C a C++ knihovny. Je použit například i pro wxPython toolkit.[12]

2.5.3 PySide

PySide je balíček vazeb Pythonu na framework Qt, který poskytuje přístup k úplnému rámci Qt a také k nástrojům generátoru pro rychlé generování vazeb pro všechny knihovny C++. Projekt PySide je vyvíjen jako open source se všemi funkcemi, které byste očekávali od jakéhokoli moderního projektu otevřeného software, jako je veškerý kód v úložišti git, otevřený prostor pro hlášení chyb a otevřený proces návrhu.[13]

PySide je sestaven pomocí generátoru vazeb Shiboken. Ten extrahuje všechny informace z hlavičkových souborů C/C++ a generuje zdrojový soubor v CPythonu, který umožňuje využívat C/C++ projekty v Pythonu. Při tom využívá knihovnu ApiExtractor, který vnitřně využívá Clang. Shiboken dále slouží jako Pythonní modul napsaný v CPythonu, který vystavuje nové typy, funkce a další v Pythonu.[14]

2.5.4 Rozdíly PyQt a PySide

Rozdíly mezi PyQt a PySide můžeme hledat ve třech úrovních – z hlediska rozdílného rozhraní, syntaxe a funkcionality vazeb na Qt.

Rozhraní PySide podporuje pouze PyQt „API 2“. API 2 poskytuje automatickou konverzi mezi třídami v Qt a odpovídajícím datovým typem či třídou v Pythonu. „API 1“ spočívá v poskytnutí typů jako je `QString, QVariants` a další, jako je tomu v Pythonu. Použité API je možné změnit, vy výchozím stavu je API 1 používáno v Pythonu 2, API 2 je ve výchozím stavu použito v Pythonu 3. V PySide je tak nutné použít nativní Python datové typy.

Syntaxe Nový styl signálů a slotů PyQt využívá názvy metod a dekorátorů specifické pro jejich provádění. PySide tak využívá `QtCore.Signal`, respektive `QtCore.Slot` místo `QtCore.pyqtSignal`, respektive `QtCore.pyqtSlot`. Podobně tomu je pro vlastnosti z `QtCore.Property`, pro které má PyQt umístění `QtCore.pyqtProperty`. Rozdílné názvy mají také balíčky s nástroji.

Funkcionality Funkce, které byly zastaralé před Qt verze 4.5 nejsou pro PySide vygenerované. PySide nemůže správně využít funkce `sender()` pro získání odesílatele signálu, pokud je jako slot použita `lambda`. PySide také očekává, že při dědění bude volán přímý konstruktor rodiče.[15]

Existují projekty, jako je například QtPy, které tyto rozdíly zabalují přes společné rozhraní a vykonávající balíček je zvolen podle dostupnosti nebo podle uživatelova nastavení [16].

2.6 Srovnání

V této části se pokusím shrnout a tak porovnat Tkinter, PyQt a PySide. V některých kritériích nebudu uvažovat PySide, jelikož není příliš rozdílný, jak jsem uvedl v předchozí části 2.5.4.

Tkinter i PyQt jsou užitečné při vývoji grafických uživatelských rozhraní, ale zároveň se liší z hlediska přizpůsobivosti a funkčnosti. Zatímco v Tkinteru má vývojář možnost samostatného vyvíjení GUI, programování svých nastavení nebo definování funkcí ve stejném skriptu, v PyQt je obvyklé oddělení grafického uživatelského rozhraní ve skriptu, který používá funkcionality z jiného skriptu pro Python.

Místo vytváření vlastního kódu pro uživatelské rozhraní v PyQt je možné využít Qt Designer pro návrh rozhraní. Výstupní soubor z Qt Designeru je možné rovněž zpracovat v PySide, pomocí podobné knihovny. Pro Tkinter také existují programy pro návrh rozhraní, jako je program PyGubu. Jeho použití je však náročnější, než jeho ekvivalent pro Qt.

Tkinter je součástí Pythonu a pro jeho komerční využití není definována rozdílná licence (PSFL licence). Stejně tak Tcl a Tk jsou licencovány jako opensource, zdarma, bez nutnosti uvádět svůj zdrojový kód [17]. Qt je však licencováno duálně, pro komerční využití, kde není vhodné uvádět zdrojový kód, je nutné zakoupit komerční licenci. To samé platí pro duální licencování PyQt. PySide je licencován licencí LGPL verze 2.1.

Díky tomu, že Tkinter je součástí Pythonu jako vlastní knihovna, není třeba žádná instalace dalších balíčků a závislostí. Nevýhodou této distribuce je nutnost vydat novou verzi Pythonu pro zpřístupnění změn v Tkinteru. Tkinter má poměrně jednoduché API, které má své limitace. Na rozdíl od PyQt nemá pokročilé widgety, ty jsou z části doplněny pomocí rozšíření `ttk`.

PyQt či PySide jsou samostatnými balíčky. Qt těmto balíčkům přináší funkcionalitu signálů a slotů k propojení komunikace mezi objekty, která umožňuje dekompozici kódu a určitou míru flexibility při zpracování událostí v grafickém uživatelském rozhraní.

Začít vytvářet grafické uživatelské rozhraní v Tkinteru je jednodušší díky jeho jednoduchému API. Na druhou stranu široké množství možností v PyQt je vhodné pro tvorbu větších aplikací, vyžaduje ale od uživatele vyšší míru pochopení funkcí PyQt. Navíc pro PyQt chybí dokumentace pro Python. Dokumentace PyQt většinou odkazuje na dokumentaci zdrojových tříd v C++. Dokumentace pro PySide však pokrývá i specifické části pro Python, a vzhledem k tomu, že mezi PyQt a PySide není příliš rozdílů, je možné ji využít i pro vývoj s využitím PyQt.

2.7 Motivace

Zdrojem motivace ke zpracování tohoto tématu je několik. Jako první může být podoba Tkinteru, který je často za nehezký, ošklivý, a těžce se nastavuje tak, aby vypadal dle odpovídající platformy. Abych pojem „ošklivý“ uvedl korektně, Tkinter se sice snaží používat nativní komponenty, pokud je to možné (také za pomoci modulu `ttk`, viz. 3.1.4), rozdíl je v integraci s cílovou platformou. Qt je vždy nativní.

Hlavní důvod je však složitější. Již v dřívější kapitole (2.3.3) jsem psal o X Window Systému (také jen „X“). Poté, co bylo umožněno mít více procesů pro uživatelské rozhraní, dalším technickým problémem, který je třeba vyřešit, je získání výstupů každého z procesů na displej. Historicky se přistupovalo k používání X a psaní svého vlastního správce oken. Správce obrazovky X je dnes nahrazován Waylandem. Ten je jednodušší na správu a rozšíření. Je moderním přístupem ke správě oken v souvislosti v dnešním hardwarem.[18]

Rozdíly Mezi Waylandem a X je dle [19] několik rozdílů:

- **Architektura** – Správce skladby (zásobník oken mimo obrazovku) je v X samostatná, přidaná funkce. Wayland spojuje zobrazovací server a správce skladby jako jednu funkci. Wayland také zahrnuje některé z úkolů správce oken, což je v X samostatný proces na straně klienta.
- **Kompozice** – Kompozice je v X volitelná, ale v Waylandu povinná. Kompozice v X je ”aktivní”; to znamená, že správce skladby musí načíst všechna pixelová data, díky čemuž dochází k latenci. U Waylandu je

kompozice „pasivní“, což znamená že správce skladby přijímá pixelová data přímo od klientů.

- **Vykreslování** – Samotný server X je schopen provádět vykreslování, pokyn k vykreslení mu také může být odeslán klientem. Naproti tomu Wayland nevystavuje žádné rozhraní API pro vykreslování, ale deleguje takové úkoly na klienty (včetně vykreslování písem, widgetů atd.). Dekorátory okna lze vykreslit na straně klienta (např. pomocí sady grafických nástrojů) nebo na na straně serveru (od správce skladby).

- **Bezpečnostní** – Wayland izoluje vstup a výstup každého okna a dosahuje důvěrnosti, integrity a dostupnosti v obou případech. Originální X design postrádá tyto důležité bezpečnostní prvky, k čemuž musela být vyvinutá rozšíření pro zmírnění dopadů absence této funkcionality.

- **Komunikace mezi procesy** – Server X poskytuje základní metodu komunikace mezi klienty X. Základní protokol Wayland nepodporuje komunikaci mezi klienty Wayland vůbec a odpovídající funkce by měly být implementovány v desktopových prostředích.

- **Síťové funkce** – X Window System je architektura, jejíž jádro bylo navrženo pro provoz přes síť. Wayland sám o sobě nenabízí transparentnost sítě; správce skladby však může implementovat jakýkoli protokol vzdálené plochy. Kromě toho probíhá výzkum streamování obrazů z Waylandu.

Wayland a Qt Dřívější verze Qt poskytují Qt Window System (QWS), který sdílí některé z návrhových cílů Waylandu. QWS však podporuje pouze klienty založené na Qt a jeho architektura není vhodná pro moderní hardware založený na OpenGL. S Qt 5 žačala práce na Waylandu. Spouštění aplikací Qt jako klientů Waylandu je oficiálně podporována od Qt 5.4.[18]

Wayland a Tk V roce 2016 bylo vypsán návrh na téma pro stipendijní akci GSoC, které se mělo zabývat implementací přímé podpory Tk pro Wayland.[2]. V roce 2017 byl proveden port Tcl/Tk do Waylandu, pomocí jeho podpory SDL, v rámci projektu `undroidwish` (varianta portu Tcl/Tk do Androidu), nicméně nikdy nebyl přidán do hlavní vývojové větve.[20]

HiDPI Další motivací jsou dnes rozšiřující se HiDPI obrazovky. HiDPI je pojem pro vysokou hustotu pixelů na palce obrazovky. Přesněji, HiDPI monitor je takový, který má více než dvojnásobné DPI oproti standardnímu monitoru. Díky tomu od aplikací vyžaduje škálování, a to hlavně v případě velikosti

[2]Návrh tématu je dostupný na `https://wiki.tcl-lang.org/page/GSoC+Idea%3A+Tk+Backend+for+the+Wayland+Display+Protocol`

písma, ale i dalších prvků. Toto škálování dobře funguje pro toolkity GTK3 a Qt5. Další úrovní může být problém, kdy máte připojené 2 obrazovky, jednu s HiDPI, druhou ne, a okno s aplikací mezi nimi přetahujete. Aplikaci je nutné v té chvíli přeškálovat – a to je podporováno pouze na Waylandu. X Window System tuto funkci neumí, umí pouze pohled přiblížit, důsledkem čehož dojde k znekvalitnění zobrazení. Toto škálování je možné na Waylandu nastavit a Qt si ho samostatně přečte. V Tk (a Tkinteru) je také možné nastavit škálování, není ho však možné během běhu změnit.[21]

Shrnutí Vzhledem k výše uvedenému vyplývá, že pro Tk neexistuje podpora, která by umožňovala ho používat na Waylandu, tak, aby naplnila požadavky plynoucí z čím dále běžnějšího HiDPI zobrazení. Cílem projektu je využít existující podpory pro Qt a tuto podporu nabídnout aplikacím pro Tkinter výměnou používané sady nástrojů.

Kapitola 3

Analýza

V kapitole analýzy blíže rozeberu důležité součásti knihovny Tkinter a balíčku PyQt, které zatím nebyly blíže popsány.

3.1 Tkinter

Jednotlivé části knihovny Tkinter jsem již popsal v předchozí kapitole (2.4). V této části se budu blíže zabývat správci rozložení, nastavení objektů a událostmi v Tkinteru. V této práci se nebudu věnovat widgetu Canvas, který umožňuje kreslení po obrazovce, jelikož by neúměrně rozšířil rozměr práce.

V této části budu používat následující pojmy:

- `window, okno` – Označuje obdélníkovou plochu někde na obrazovce.
- `top-level window, top-level` – Označuje okno existující nezávisle na obrazovce. Toto okno bude přizpůsobeno podobě a ovládání dle systémového správce obrazovky. S oknem může být pohybováno po obrazovce,
- `widget, grafická komponenta` – Základní stavební blok grafického uživatelského rozhraní. Widgetem může být tlačítko, přepínač, pole pro vyplnění uživatelského vstupu, zaškrtávátko a další.
- `frame, rám` – Základní organizační jednotka Tkinteru. Může obsahovat widgety ve vztahu rodič (rám) a potomci (widgety).

3.1.1 Správci rozložení

Tkinter disponuje třemi takzvanými geometrickými správci (správci rozložení). Vytvořený widget není vykreslen, dokud není zařazen pomocí některého z geometrických správců.

Prvním z geometrických správců je `grid`. Ten umožňuje rozložení do mřížky (tabulky), tedy widgety mohou být umístěny do konkrétního řádku a sloupce,

respektive buňky tabulky. Šířka řádku nebo sloupce tabulky je určena podle největší buňky daného řádku nebo sloupce. Pro widgety, které nezaplní celou buňku, je možné specifikovat, co s místem navíc – jestli má být widget rozšířený, nebo má být místo ponecháno prázdné. Widget je také možné roztáhnout po více buňkách.[22]

Druhým z geometrických správců je `pack`. Tento správce je výrazně složitější z hlediska algoritmu umístění widgetů, než správce `grid`. Správce `pack` umisťuje widgety na okraje volného prostoru daného rodiče potomků. Algoritmus správce `pack` na vstupu přijímá list potomků v pořadí, ve kterém byli vytvoření, společně s dalším nastavením (např. vlastnost `-side`, tedy strana, na kterou má být potomek umístěn), a spočívá zjednodušeně v těchto krocích:

1. Alokuj místo po stranách podle nastavení `-side`.

2. Rozhodni dimenze obsahu podle požadované velikosti, nastavení okrajů (`-padx, -pady, -ipadx, -ipady`) a nastavení roztáhnutí (`-fill`).

3. Pokud je obsah menší, než aplikovaný prostor, aplikuj nastavení vlastnosti `-anchor`, tedy kotvy k nějaké straně a nebo rohu.[23]

Možnostmi pro zvolení strany jsou: `TOP, LEFT, BOTTOM, RIGHT`, tedy postupně shora, zleva, zdola a zprava. Umístění shora a zleva se chová podle očekávání, tedy widgety jsou umístěny shora nebo zleva. V případě umístění zprava nebo zdola jsou widgety rovněž umístěny v očekávaném pořadí, ale v případě, že není dostatek prostoru, mohou být překryté. Obecně je možné použít dohromady v rámci jednoho rodiče umístění do různých stran, správce `pack` se však doporučuje používat bez kombinací umístění k různým stranám, jelikož výsledek nemusí být takový, jaký by vývojář očekával. Pro složitější rozmístění je doporučeno použít správce `grid`.[6]

Posledním správcem rozložení je `place`, který umožňuje widget potomka vložit na konkrétní (relativní) souřadnice rodiče.

3.1.2 Standardní atributy

Každý widget v Tkinteru má množinu vlastností, které je možné u něj nastavit. Tyto vlastnosti korespondují s vlastnostmi, které je možné nastavit widgetům z Tk. Je možné je nastavit již v konstruktoru widgetu, nebo po vytvoření pomocí `config` metody.

Mezi standardní atributy Tkinteru dle [24] patří:[3]

- **Dimenze** – nastavitelné pomocí různých měr – centimetry a milimetry, palce, body.

[3] Z tohoto seznamu byly vyřazeny části týkající se widgetu Canvas, dle úvodu sekce.

- **Geometrické řetězce** – zadávají velikosti a umístění dle formátu `wxh±x±y`, kde `w` zastupuje šířku, `h` výšku, společně oddělené písmenem „x“. Následuje část `x`, která může být kladná, nebo záporná – určuje posun okna zleva (plus) nebo zprava (mínus) po ose x (horizontální). To samé platí pro část `y` po vertikální ose shora dolů.
- **Barvy** – Nastavení barvy pozadí, popředí (text), a to pomocí čtyř až dvanácti místného RGB kódu, nebo výběrem z předdefinovaných barev.
- **Typ písma** – Obvyklé nastavení písma (rodina, velikost, tučnost, náklon, podtržení, proškrtnutí.
- **Umístění, kotvy** – Umístění do určitého rohu, na určitou stranu, nebo doprostřed.
- **Styl okraje, reliéfu** – Simulované 3D efekty okrajů widgetu.
- **Bitmapa** – Některé widgety umožňují nastavení bitmapy jako vlastní ikony.
- **Kurzor** – Vzhled kurzoru je možné měnit (například na základě určitých událostí – naznačení prokliku)
- **Obrázky** – V Tkinteru je možné zobrazit bitmapy a nebo barvené obrázky ve formátu `.git, .pgm, .ppm`. Další formáty je možné zobrazit v Tkinteru pomocí třídy `ImageTk` z balíčku `Pillow` pro Python.

3.1.3 Události

Widgety z Tkinteru mají několik vestavěných funkcí obsluhující přicházející události – například tlačítko reaguje na stisk pomocí zavolání funkce, která je přiřazena k jeho atributu příkaz (callback), pole pro zadání uživatelského vstupu po získání zaměření (focus) reaguje na stisknutí klávesy tím, že stisknutý znak vyplní do pole.

Tyto vestavěné funkce však Tkinter rozšiřuje možností svázání (bind) události a umožňuje tak přidat, změnit nebo odebrat chování aplikace po vzniknutí určité události. Tyto události jsou pak obsluhovány v metodách widgetů nebo v jiných funkcích. Svázání je realizováno spojením události a této obsluhující metody.

Svázání může proběhnout na několika úrovních:

- **instance** – Událost může být svázána s metodou z konkrétního widgetu. Toho je docíleno pomocí univerzální metody `bind` volané na instanci.
- **třída** – Událost může být svázána s metodou ze všech widgetů dané třídy. Toho je docíleno pomocí univerzální metody `bind_class` volané na jakémkoliv widgetu. Této funkce je možné využít, pokud máme několik

shodných objektů, které mají na nějakou událost reagovat jednotně. Danou událost obslouží widget, na kterém je aktuálně zaměření (focus).

- **aplikace** – Událost může být svázána přímo s celou aplikací – nezáleží tak, kde se aktuálně nachází kurzor. Svázání je docíleno pomocí univerzální metody `bind_all` volané na jakémkoliv widgetu.[25]

Tkinter také podporuje tzv. sekvence událostí, kdy je obsluhující funkce volána pouze tehdy, kdy je vzor sekvence splněn kompletně. Zjednodušeně se jedná o řetězec, který kombinuje různé modifikátory, typ události a detail ve formátu `<[modifikátor-]...typ události[-detail]>`. Například sekvence se vzorem `<Control-KeyPress-A>` vyvolá událost pouze tehdy, kdy bude současně zmáčknut (typ události `KeyPress`) modifikátor `Control` (klávesa Ctrl) a detail `A` (klávesa A). Modifikátory je možné kombinovat, místo některých typů událostí (stisknutí tlačítka na klávesnici nebo myši) je možné psát pouze část detail.

3.1.4 Modul ttk

Od Tk verze 8.5 je v Tkinteru dostupný modul `ttk`. Ten nahrazuje několik z widgetů dostupných v Tkinteru za nové, takzvané tématizované widgety (themed widgets). Tyto nové třídy vnáší do Tkinteru následující přínosy.

Vzhled specifický pro platformu Ve verzích před Tk 8.5 byla jedna z nejčastějších stížností na aplikace Tk ta, že nevyhovovaly stylu různých platforem. Modul `ttk` umožnil psát aplikace jedním způsobem pro všechny platformy. Aplikace může vypadat jako Windows aplikace v systému Windows, jako MacOS aplikace v systému MacOS atd., beze změny programu. Každý možný odlišný vzhled je reprezentován pojmenovaným tématem z `ttk`. Takto existuje například „klasické téma“, který Tkinter normálně používá.

Zjednodušení a zobecnění chování widgetů ve specifických stavech V klasickém Tkinteru je nutné nastavit několik vlastností widgetu, aby se widget choval nebo vypadal jina v souvislosti na různých podmínkách. Například u widgetu `Button` je možné nastavit atribut `foreground`, což atribut pro nastavení barvy nápisu tlačítka. V případě, že by bylo třeba manipulovat s touto barvou v době, kdy je tlačítko aktivní, nebo neaktivní, musela by být nastavena barva také v atributech `activeforeground` a `disabledforeground`. Pokud by se takto jednalo o více atributů, brzo začne být definice velice nepřehledná. Modul `ttk` toto dekomponuje na přehlednější systém stavů, kterých může widget nabývat, a mapy stylů, kde může být pro každý stav určený jiný styl.[26]

Kromě nahrazení widgetů `ttk` také doplňuje Tkinteru o pokročilé widgety. Jedním z příkladů je widget třídy `Combobox`, což je kombinace třídy `Entry` pro zadání vstupu uživatelem a výběru pomocí rozbalovacího menu.

3.1.5 Dialogy

Pro tvorbu dialogů existuje v Tkinteru několik modulů. První z nich je `tkMessageBox` – po jeho importování je možné rychle vytvořit dialogy pro potvrzení akce, opakování akce, zobrazení chybové hlášky, informace a nebo upozornění. V těchto dialozích je možné nastavit titulek, zprávu a další možnosti – výchozí tlačítko a ikonu okna.[27]

Pro práci se soubory existuje druhý modul jménem `tkFileDialog`, který poskytuje dva typy dialogů – dialog pro výběr a otevření souboru a dialog pro uložení do souboru. Při této akci je definovat povolené typy souborů, výchozí složku a výchozí jméno souboru. Jako u předchozích dialogů je možné nastavit titulek okna.[28]

Dalším užitečným modulem je dialog pro výběr barvy – `tkColorChooser`. Uvedené moduly je možné importovat přímo, nebo je možné importovat soubory s jejich implementací napřímo, dle odpovídajícího souboru viz 2.4.

3.2 PyQt

PyQt se skládá z několika různých komponent. PyQt verze 4 se částečně těmito komponentami liší, dále uvažuji jen PyQt verze 5. Existuje několik modulů pro Python, které jsou instalovány společně s balíčkem PyQt5:

- `QtCore` – Základní Qt třídy, které nejsou určeny pro práci s GUI.
- `QtGui` – Základní Qt třídy pro práci s GUI – rozšiřuje `QtCore`.
- `QtWidgets` – Třídy pro vytváření klasických desktopových prostředí, zastupují obvyklé widgety.
- ... a mnohé další, jak je možné naleznout v [29].

QtCore Modul QtCore obsahuje kromě základních tříd také smyčky událostí a Qt mechanismus signálů a slotů. Zahrnuje také abstrakce nezávislé na platformě pro animace, stavové automaty, vlákna, mapované soubory, sdílenou paměť, regulární výrazy a uživatelského nastavení.[30]

Qt namespace Tento jmenný prostor je důležitou součástí modulu `QtCore`. Obsahuje několik důležitých identifikátorů pro Qt knihovnu. Identifikátory mají konstanty pro práci s rozložením objektů v okně, podobu kurzoru, nastavení časové zóny, význam speciálních znaků klávesnice, formátování textu a mnoho dalšího. Každá konstanta má definovanou svojí číselnou hodnotu,

která odpovídá svému výčtovému typu (enum). Kompletní výčet pro PyQt5 je možné naleznout v [31].[4]

QtGui Modul QtGui obsahuje třídy pro integraci okenního systému, manipulace s událostmi, 2D grafiku, základní zobrazování, písmo a text. Obsahuje také kompletní sadu vazeb na OpenGL. Většinou se však používá rozhranní vyšší úrovně, jako je rozhranní obsažené v modulu `QtWidgets`.[33]

3.2.1 QWidget

`QWidget` je třída, která je součástí modulu `QtWidgets`. Z této třídy vychází resp. dědí všechny objekty související s grafickým uživatelským prostředím. Je také základním prvkem pro stavbu uživatelského rozhraní. Rovněž získává informace o událostech, které mohl vyvolat uživatel pomocí kliknutí či pohybem myší, nebo stisknutím nějakých kláves.

Widgety je možné skládat na sebe pomocí systému rodičů a potomků. Widgety jsou zobrazeny podle daného pořadí. Potomci jsou vložení do rodičů. Widget bez rodiče je nazýván oknem (window). Jedná se většinou o widgety, které mají vlastní okno, název a třeba i ikonu. Typicky je takto určen widget `QMainWindow`, včetně jeho specializace `QDialog`, který se používá pro zobrazení různých uživatelských dialogů. Potomci jsou v rodiči zobrazeny dle nastaveného rozložení (viz dále v 3.2.4).

Mnoho metod je definovaných přímo v třídě `QWidget`, další jsou definovány v jeho specializujících se třídách, které poskytují reálnou funkčnost, například:

- `QLabel` – Textový nadpis pole pro zadání vstupu.
- `QEntry` – Pole pro zadání uživatelského vstupu.
- `QPushButton` – Stisknutelné tlačítko.[34]

3.2.2 Události

Widgety odpovídají na události, které jsou typicky způsobeny uživatelskou akcí. Qt takto vzniklé události předává widgetu voláním jeho obsluhujících metod, s instancí třídy dědící z `QEvent`, s informacemi o dané události.

Základními obsluhujícími metodami (sloty) třídy `QWidget` jsou:

- `paintEvent()` – Volána při nutnosti překreslit widget.
- `resizeEvent()` – Volána při nutnosti změnit velikost widgetu.

[4] V dokumentaci pro PyQt5 chybí mnoho číselných hodnot – dále jí využívám hlavně z toho důvodu, že jsem použil PyQt5 při implementaci prototypu. V případě zájmu a nalezení kompletnější dokumentace shledávám dokumentaci pro PyQt4 úplnější - viz. [32].

- `mousePressEvent()` – Volána při kliknutí tlačítka myši, kdy je kurzor v oblasti widgetu.
- `mouseReleaseEvent()` – Volána, pokud je tlačítko myši uvolněno, musí následovat za předchozí událostí.
- `mouseDoubleClickEvent()` – Volána při dvojitém kliknutí tlačítka myši. Současně jsou však volány odpovídající obsluhující funkce pro jednotlivé kliknutí nebo uvolnění.

Tyto metody jsou pro widgety, umožňující práci s klávesnicí, rozšířeny o:

- `keyPressEvent()` – Volána kdykoliv je stisknutá klávesa na klávesnici. Při delším stisknutí je tato metoda volána opakovaně.
- `focusInEvent()` – Volána když widget nabyl zaměření (focus) klávesnice. Obsluha této události by měla obsahovat i uživatelsky viditelnou reakci widgetu.
- `focusOutEvent()` – Volána když widget pozbyl zaměření klávesnice.

Dalšími obsluhujícími metodami jsou:

- `mouseMoveEvent()` – Volána při pohybu kurzoru za doby stisknutí tlačítka myši.
- `keyReleaseEvent()` – Volána kdykoliv kdy je klávesa na klávesnici stisknutá po delší dobu opět uvolněná.
- `wheelEvent()` – Volána při pohybu kolečka myši.
- `enterEvent()` – Volána při vstupu kurzoru do oblasti widgetu.
- `leaveEvent()` – Volána při odchodu kurzoru z oblasti widgetu.
- `moveEvent()` – Volána při přesunu widgetu relativně k jeho rodiči.
- `closeEvent()` – Volána uzavření widgetu uživatelem.[34]

3.2.3 Možnosti zobrazení widgetů

Kromě standardních stylů widgetů pro každou platformu lze widgety také modifikovat podle pravidel uvedených v šabloně stylů. Šablona má podobnou strukturu, jako šablona stylů pro kaskádový jazyk CSS, jelikož jím byla inspirována.

Tato funkce umožňuje přizpůsobit vzhled konkrétních widgetů, aby uživateli poskytly vizuální podněty o jejich účelu a napověděly tak uživateli, jakým způsobem prvky použít. Šablonu je však možné také použít na celou aplikaci nebo všechny instance konkrétní třídy. Jaký má být aplikován styl je řešeno kaskádově, jako je tomu v CSS.[35]

3.2.4 Management rozložení widgetů

Widgety – potomci jsou v rodiči zobrazeny dle jeho nastaveného rozložení. Rozložení je automaticky přidáno na widget, pokud je widget umístěn do rodiče, stejně tak je vložen widget do rodiče tím, že je mu nastaveno rozložení rodiče.

PyQt5 obsahuje 4 základní typy rozložení:

- `QGridLayout` – Umožňuje rozložení formou mřížky (tabulky), kdy je možné definovat sloupec a řádek v tabulce, kde má být potomek vykreslen, společně s možností sloučení některých buněk a nastavení roztažení widgetu.
- `QBoxLayout` – Má 2 podtypy:
  - `QVBoxLayout` – Umožňuje widgety rozložit vertikálně. Je možné nastavit směr řazení widgetů.
  - `QHBoxLayout` – Umožňuje widgety rozložit horizontálně.
  - Oběma je možné nastavit směr přidávání widgetů (zprava doleva, shora dolů, a naopak).
- `QFormLayout` – Rozložení vhodné pro formuláře – sdružuje pole pro vstup s jejich popisky. Může být použit společně s kontejnerem `QButtonGroup`, který sdružuje tlačítka.
- `QStackedLayout` – Rozložení odpovídající zásobníku – v jeden okamžik je vidět jen jeden widget.

Algoritmus určení velikostí widgetů pro vykreslení postupuje následovně:

1. Všem widgetům je přiděleno místo v souladu s jejich pomocnými funkcemi pro určení potřebné velikosti.
2. Pokud má některý z widgetů nastaven faktor roztažení s hodnotou větší než nula, je jim přiděleno místo v poměru k jejich rozšiřujícím faktorům.
3. Pokud má některý z widgetů nastaven faktor roztažení na nulu, získá více místa, pouze pokud žádný jiný widget nebude vyžadovat prostor. Z nich je místo nejprve přiděleno widgetům se zásadou rozšiřující se velikosti.
4. Jakýmkoli widgetům, kterým je přiděleno méně místa, než je jejich minimální velikost (nebo nápověda minimální velikosti, pokud není uvedena minimální velikost), je přidělena tato minimální velikost, kterou vyžadují.
5. Jakýmkoli widgetům, kterým je přiděleno více místa, než je jejich maximální velikost, je přidělen prostor o této velikosti.[36]

3.2.5 Dialogy

Dialogy v PyQt jsou realizovány pomocí třídy `QDialog` z modulu `QtWidgets` a dalších specializovaných tříd. Tyto specializované třídy obsluhují například chybové a jiné hlášky, manipulaci se soubory a tisk.[37]

3.3 PySide

PySide se od PyQt příliš neliší, jak už bylo popsáno v 2.5.4. Pro část Návrhu a Realizace jsem zvolil balíček PyQt verze 5. Za určitých podmínek by za pomoci několika substitucí neměl být problém vyvinutý prototyp spustit i pro balíček PySide verze 2 (verze 2 je určená pro Qt5).

Kapitola 4

Návrh

V této kapitole proberu návrhy mapování a zhodnotím je podle navržených parametrů. V druhé části navrhnu kvantifikátory, dle kterých bude možné implementované mapování zhodnotit.

4.1 Mapování

V kapitolách 2.4 a 2.5 jsem popsal, jakým způsobem Tkinter mapuje své třídy na Tk komponenty a jakým způsobem PyQt využívá SIP, popřípadě jak PySide využívá Shiboken. Z těchto mapování je možné vyjít pro návrh vlastních mapování komponent a funkcí Tkinteru na Qt komponenty a PyQt funkce. Vzhledem k tomu, že PySide není příliš rozdílné od PyQt, jak jsem vysvětlil v 2.5.4, rozhodl jsem se dále zohledňovat hlavně PyQt.

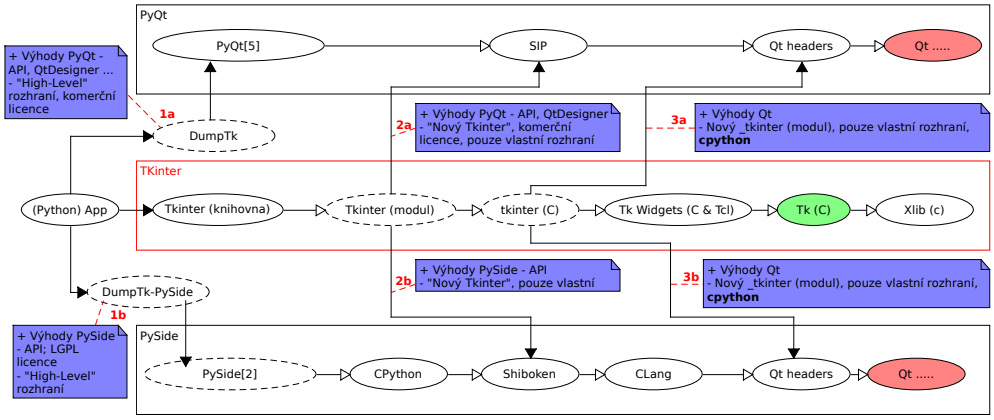
Pro návrh těchto mapování jsem navrhl následující body, kterými návrhy zhodnotím:

1. Vhodná náročnost přípravy mapování.
2. Jednoduchá možnost rozšiřitelnosti.
3. Maximalizace pokrytí možností Tkinteru.
4. Další přínosy návrhu.

4.1.1 High-level rozhraní

První možností, kterou jsem zvážil, je implementace určitého high-level rozhraní, tedy rozhraní, které kompletně odstíní Tkinter a jakékoliv jeho závislosti na Tk, a bude se napojovat přímo do rozhraní balíčku PyQt.

Znamenalo by to implementaci nové knihovny, která implementuje třídy a funkce, které se tváří jako z Tkinteru, reálně ale dědí z tříd nebo volají funkce PyQt



Obrázek 4.1: Mapování v nižších úrovních

Přednostní výhodou tohoto high-level rozhraní by jistě byla možnost implementovat každý objekt a funkci na míru potřebám Tkinteru. Rozšiřitelnost tohoto mapování by byla jednoduchá - kdykoliv by bylo možné doplnit novou třídu, s potřebnými atributy a metodami, aby podporovala funkčnost, kterou zamýšlí Tkinter. Další vnímaný přínos by mohla být možnost využívat tuto náhradu Tkinteru jako hybridní řešení, tudíž v části kódu programovat pro Tkinter, v další části pro PyQt. Tohoto by mohlo být využito v případě aplikace, která by byla přepisována z Tkinteru do Qt – bylo by jí možné přepisovat po částech.

Značnou nevýhodou této možnosti mapování je její pracnost přípravy, jelikož malý prototyp, který jsem vyzkoušel a který podporoval pár tříd, neúměrně rostl vzhledem k velikosti aplikace, která by řešení v budoucnu používala. Každá třída, funkce a další objekty z Tkinteru by musely být explicitně přeprogramovány, aby poskytly odpovídající rozhraní. Tkinter je však pouze rozhraní pro interpretaci Tcl příkazů pro Tk, kde je několik tříd podobných, několik atributů shodných, včetně shodných metod, problém tohoto návrhu musí být řešitelný, v nižší úrovni komunikace Tkinteru s Tk komponentami.

4.1.2 Mapování v nižších úrovních

Pro přehlednost nižších úrovní knihoven Tkinter, PyQt a PySide jsem připravil následující diagram 4.1. Každá šipka představuje určitou vrstvu knihoven, případně možnost, kde knihovny propojit na stejné úrovni.

Na 4.1 jsou možnosti namapování označeny následovně:

- **Vazby 1a a 1b** – Vytvoření high-level rozhraní – viz. 4.1.1

- **Vazby 2a a 2b** – Vytvoření nového modulu `_tkinter`.
- **Vazby 3a a 3b** – Přepsání tkinteru v C.

Vazby 3a a 3b jsem ihned vyloučil, jelikož by znamenaly, že modul Tkinter, spravovaný ve standardní knihovně Pythonu, by musel mít závislosti do Qt frameworku, což by bylo pravděpodobně nebylo přijato komunitou, a musel by tak být vytvořen nový Tkinter. Tyto vazby tedy nepřináší nic nového oproti vazbám 2a a 2b, které také vyžadují vytvoření nového Tkinteru. Navíc by změna byla provedena v tkinteru v C.

Vazby 2a a 2b mi tak přijdou smysluplnější oproti vazbám 3a a 3b. Realizace vazeb 2a či 2b by znamenala připojení k vazbám vygenerovaných pomocí SIP či Shibokenu. Tuto práci však již realizovaly knihovny PyQt a PySide.

4.1.3 Mapování v různých úrovních

Předchozí myšlenka vazeb 2a a 2b vede k tomu, že by se Tkinter stal závislý na PyQt či PySide. Tuto závislost je možné vyměnit za závislost na modulu `_tkinter`. Co by toto mapování přineslo?

Při vytvoření aplikace v Tkinteru je nutné instancovat třídu `Tk`, která ve svém konstruktoru vytváří instanci Tcl interpretu v podobě modulu `_tkinter`. Tato instance je uložena do proměnné `tk`. Instance třídy `Tk` může být jen jedna. Každý vytvořený widget v Tkinteru si přebírá referenci z této proměnné, buď napřímo, nebo za použití pomocné funkce. Každá třída Tkinteru, které odpovídá nějakému widgetu, obsahuje nebo dědí mechanizmy pro překlad vytvoření nebo úpravy instance na n-tici, kterou pak pomocí volání metody `tk.call` s n-ticí jako parametrem předává do Tcl interpretu v modulu `_tkinter`, jak už jsem částečně popsal v jak jsem popsal v 2.4.1. Metoda `tk.call` je volána v modulu `_tkinter` který je napsaný v CPythonu.

Pro lepší pochopení uvádím ukázku – kód typického programu „Hello world“[5] v Tkinteru dle 4.2 produkuje příkazy sestavené z n-tic slov v 4.3.

Problém high-level řešení dle 4.1.1 spočíval v tom, že by bylo nutné několik vlastností a funkcí objektů v Tkinteru implementovat opakovaně. Rovněž se potýkal s problémem několika možností práce s objekty, kterou Tkinter (nebo Python) podporuje. Dekompozice těchto akcí na jednoduché n-tice, které Tkinter samostatně dekomponuje, umožní tyto různé způsoby práce s objekty sjednotit. Tkinter navíc zajišťuje i další funkcionality samostatně, jako je pojmenování objektů a zavedení funkcí, které mají být volány na základě určité události. Toto je možné pozorovat na příkladu uvedeném v 4.3. Tím se výrazně sníží složitost zavedení mapování, které tyto n-tice přeloží pro rozhraní PyQt. Díky dekompozici příkazů na n-tice bude možné jednoznačně určit, jaké nastavení widgetu má být provedeno.

[5] Program „Hello world“ je jednoduchý program, který má za úkol vypsat jednoduchou zprávu „Hello world!“

```
super().__init__(master)
self.master = master
self.pack()

self.hi_there = tk.Button(self)
self.hi_there["text"] = "Hello World"
self.hi_there["command"] = self.say_hi
self.hi_there.pack(side="top")

self.quit = tk.Button(self, text="QUIT", fg="red")
self.quit.command=self.master.destroy
self.quit.pack(side="bottom")}
```

Obrázek 4.2: Hello world program v Tkinteru

```
('frame', '.!application')
('pack', 'configure', '.!application')
('button', '.!application.!button')
('.!application.!button', 'configure', '-text', 'Hello World')
('.!application.!button', 'configure', '-command', '140192520say\ hi')
('pack', 'configure', '.!application.!button', '-side', 'top')
('button', '.!application.!button2', '-text', 'QUIT', '-fg', 'red')
('.!application.!button2', 'configure', '-command', '140192493destroy')
('pack', 'configure', '.!application.!button2', '-side', 'bottom')
```

Obrázek 4.3: N-tice pro Tcl příkaz vygenerované programem Hello world 4.2

Rozšiřitelnost tohoto mapování může být jednoduchá, jelikož spočívá hlavně v jednotném chování widgetů Tkinteru – widgety mají společné atributy a funkce, které jsou doplněny o atributy a funkce specifické pro widget. V případě přidání nově podporovaného widgetu bude stačit přidat pouze obsluhu funkcí specifických pro widget. Pokrytí možností Tkinteru by mělo být prakticky neomezené.

Tento způsob mapování nazývám jako mapování „v různých úrovních“, jelikož mapování bude provedeno z poměrně nízké vrstvy (až u modulu `_tkinter`) na vysokou vrstvu – rozhraní PyQt či PySide.

4.1.4 Vybraný způsob mapování

Dle sekce 4.1.3 je mapování v různých úrovních vhodné řešení, jak mapovat Tkinter rozhraní do Qt komponent. Modul `_tkinter` je možné vyměnit přímo v Tkinteru, a to tak, že v místě vytváření instance Tcl interpretu bude namísto tohoto interpretu vytvořena obalující třída, která zabalí metody (hlavně metodu `call`) interpretu a umožní tak vlastní implementaci, která zpracuje n-tice a vytvoří prostředí pomocí Qt komponent.

Za těchto podmínek by však musela být vytvořena nová knihovna Tkinteru, která by toto umožňovala. Tím jsem se vrátil k již vyloučenému řešení, které by znamenalo Tkinteru poskytnout závislosti na Qt komponentách. Lepší způsob je podědit třídu `Tk` z Tkinteru a v tomto potomkovi implementovat konstruktor tak, že referenci uloženou v proměnné `tk` změní za obalující třídu, která bude volána namísto klasického Tcl interpretu.

Tato obalující třída, kterou můžeme pojmenovat `TkWrapper`, bude muset obsahovat minimálně metodu `call` pro zadání příkazů v podobě n-tic a metodu `createcommand`, obsluhující vytvoření příkazu jako reakce na nějakou událost. Kromě dalších pomocných metod bude muset implementovat metodu `mainloop`, kterou je Tkinter program spuštěn.

Vzhledem ke konstrukci příkazů bude nutné v rámci implementace metody `call` udržovat informace o instancích objektů (widgetů) a instancí příkazů k událostem, jak napovídá příklad n-tic v 4.3.

Díky úpravě popsané v předchozích odstavcích bude možné implementovat nový nástroj, který bude moct být importován místo knihovny Tkinter.

4.2 Návrh kvantifikátorů

Pro zhodnocení úspěšnosti navrženého mapování a jeho implementace je třeba navrhnout kvantifikátory. Dle následujícího seznamu uvádím navržené kvantifikátory s důvodem jejich navržení:

- **Implementované třídy (widgety)** – Pro účely nového rozhraní je třeba podporovat co nejvíce widgetů, které jsou podporovány v Tkinteru.
- **Výchozí parametry** – Widgety z Tkinteru sdílejí některé atributy a funkce, jelikož všechny dědí ze stejní třídy (`Misc`). Pro účely nového rozhraní bude vhodné podporovat co nevíce z těchto parametrů.
- **Speciální parametry** – Widgety z Tkinteru mají i své speciální parametry, které bude také nutné brát v potaz, tuto skupinu však rozlišuji od výchozích parametrů.
- **Funkcionality** – Tkinter podporuje několik funkcionalit, kromě samotného vytváření widgetů umožňuje jejich manipulaci pomocí správců geometrie, svázání události s určitými funkcemi, tvorbu dialogy a další funkce.
- **Rozšiřitelnost** – Tkinter jistě není zastaralý projekt a stále je v tomto projektu přispíváno několik vylepšení jeho tvůrci – komunitou. Implementace by tak měla být vhodně rozšiřitelná, aby umožnila rozšíření pro nově přidané třídy, parametry a další.

Kapitola 5

Realizace

Realizaci vybraného mapování jsem započal s Pythonem verze 3.7, nicméně pro testování s IDLE jsem přešel na Python 3.8, který byl v aktuální verzi IDLE vyžadován. Pro mapování do frameworku Qt jsem využil balíček PyQt5, jelikož v době implementace nebyly dostupné vazby na Qt6, nyní nejnovější verze Qt toolkitu.

Projekt implementace jsem pojmenoval jako „dumpTk“, nicméně samotný modul (nástroj) jsem nazval jako „Qtinter“, podobně, jako byl nazván Tkinter. Zdrojový kód byl zdokumentován v anglickém jazyce, formát zdrojového kódu byl zkontrolován pomocí `flake8`. Zdrojový kód byl umístěn do veřejného repozitáře na serveru GitHub: `https://github.com/SohajCZ/dumpTk` a je možné ho dále šířit na základě permisivní svobodné licence (MIT licence).

5.1 Struktura repozitáře

Struktura repozitáře je zjednodušeně popsána v přehledu 5.1.

5.2 Qtinter

Hlavní soubor nástroje Qtinter je soubor `__init__.py`. Na začátku tohoto souboru jsou importovány třídy z Tkinteru, které jsou používány v rámci aplikací pro Tkinter. Toto rozhraní také určuje, které třídy budou Qtinterem podporovány.

Tkinter (pro Python 3) se obvykle importuje s jedním z následujících způsobů:

- `import tkinter *`
- `import tkinter as tk`

Po importu musí být vytvořena instance třídy Tk. Třída `Tk` v tomto souboru zde dědí z původní třídy Tk v Tkinteru, což znamená, že většina

```
.
├── examples ..... Složka s implementovanými příklady
│   ├── all_user_input_widgets.py Používá většinu uživatelských widgetů
│   ├── events.py ..... Příklad práce s událostmi
│   ├── grid.py ..... Příklad práce s rozložením v mřížce
│   ├── hello_world.py ..... Typický Tkinter „Hello world“ příklad
│   ├── menu_and_dialog.py ..... Ukázka menu, akcí v menu a dialogu
│   └── pack.py ..... Příklad práce s rozložením pomocí balíčku (pack)
├── idlelib ..... Kopie zdrojového kódu IDLE z oficiálního repozitáře
├── LICENSE ..... Soubor licence (MIT)
├── qtinter ..... Hlavní složka implementovaného balíčku
│   ├── constants.py ..... Konstanty odpovídající hodnotám v Tk
│   ├── event_builder.py ..... Sestavení událostí – 5.4
│   ├── event_translate.py ..... Překlad hodnot událostí – 5.5
│   ├── filedialog.py ..... Funkce pro tvorbu souborových dialogů – 5.6
│   ├── implementer.py. ..... Třída implementující většinu logiky – 6.3
│   ├── __init__.py ..... Hlavní soubor implementovaného řešení – 5.2
│   ├── layouter.py ..... Třída Layouter – správce geometrie – 5.7
│   ├── parameters_mapping.py ..... Mapování parametrů a hodnot – 5.9
│   ├── tktoqt.py ..... Obsahuje funkce pro překlad mapování – 5.10
│   └── translate_qt_core.py ..... Podpůrné funkce pro Layouter – 5.8
├── README.md
├── requirements.txt ..... Soubor pro nastavení virtuálního prostředí
└── setup.py ..... Setup skript pro instalaci balíčku qtinter
```

Obrázek 5.1: Struktura repozitáře

metod zůstává stejná. Tkinter je tak využíván jako preprocesor požadavků. Pro použití vlastní logiky je implementována třída `TkWrapper`, podobně, jako v Tkinteru. Třída Tk z Tkinteru tímto wrapperem obaluje vazbu na Tk, ale ve skutečnosti zde TkWrapper obaluje vlastní třídu - `Implementer` - který implementuje hlavní logiku celého nástroje. Výsledkem je, že při volání Tkinter místo metod originálního Tcl interpretu volá metody implementované v tomto `TkWrapperu`.

Tyto vlastní metody jsou hlavně:

- `call` – Tato metoda původně volá Tcl interpret z modulu `_tkinter`, který předává argumenty do Tcl/Tk, který je zpracovává. Tyto argumenty mají pevnou strukturu pro Tcl, jsou proto snadno analyzovatelné v třídě `Implementer`. Přesněji, této metodě jsou předávány již dříve popsané n-tice dle 4.3.

- `createcommand` – tato metoda také volá modul `_tkinter`, se záměrem vytvořit příkaz v Tcl/Tk. Implementovaná třída `QtCallWrapper`, po-

dobně jako původní třída `CallWrapper` v Tkinteru, zapouzdřuje tento příkaz (po rozkladu ze zapouzdření z instance třídy `CallWrapper`) a předá jej také instanci třídy `Implementor`, aby tento příkaz uložila, pro pozdější vyzvednutí při svázání s událostí.

Třída StringVar zde prototypuje použití proměnných z Tk. Protože proměnné Tk jsou původně uloženy v Tcl, není snadné získat hodnotu proměnné. Proto zde Qtinter dědí původní proměnnou a nahradí ji pro vlastní třídou, jejíž hodnota není uložena v Tcl.

5.3 Implementer

Tato třída implementuje většinu logiky Qtinteru. Třída `Implementer` dědí z PyQt třídy `QApplication`. Hlavní metodou třídy `Implemetor` je metoda `call`. Tato metoda přebírá jako argument n-tice pro příkaz do Tcl a analyzuje tuto n-tici v následujících krocích:

1. Přeskoč nebo vynech všechny příkazy, které nejsou užitečné pro Qt.
2. Příkaz obvykle obsahuje některé další parametry, ty nejprve extrahuj. Tyto parametry jsou pak dále přeloženy pomocí překladových slovníků do formátů pro PyQt.
3. Pokud se hlavní příkaz týká svázání událostí, proběhne zpracování těchto svázání a metoda je ukončena. Během tohoto se používá volání funkce `_if_binding_holds`, protože Tkinter umožňuje vázat příkazy na speciální zkratky, ale Qt má své sloty na widgetech a speciální zkratky jsou obvykle řešeny přímo až v tomto slotu. Vzhledem k tomu, že slot je zde realizován již existující funkcí očekávající chování svázání událostí z Tkinteru, je nutné tuto logiku řešit samostatně. Pro události se používají dva další soubory: `event_translate.py` a `event_builder.py`.
4. Pokud se hlavní příkaz týká rozložení, správy geometrie a podobně, předej příkaz metodě `_add_widget`, a metodu poté ukonči.
5. Pokud se hlavní příkaz týká konfigurace následujícího seznamu widgetů – `ListWidget, ComboBox, Menu` a `MenuAction` – vyřeš (založ, nastav) tyto widgety a metodu poté ukonči. Tyto widgety vyžadují speciální logiku jejich obsluhy.
6. Vytvoř základní widgety. Během tohoto je použit `translate_class_dict`, který obsahuje překlad z názvu objektu Tk (nebo `ttk`) do třídy PyQt. Dále nastav widget dle dalších parametrů a metodu poté ukonči.

Další metody třídy `Implementer` jsou:

- `add_to_namer` – Malá funkce, která řeší problém s nabídkami (viz 5.3).

- `create_menu` – Funkce, která spouští lištu nabídek v aplikaci.
- `show` – Řešení zobrazující správný widget jako toplevel, což je problematické ve chvíli, kdy je použito menu, nebo není přímo vytvořen hlavní snímek (viz 5.3).
- `call_method` – Konfiguruje widget pomocí jeho metod s danými parametry – použité při kroku 2. Název metody a parametry je třeba nejprve přeložit z názvosloví a formátu Tcl/Tk pomocí funkce `translate_parameters_for_class`.
- `createcommand` – Protože je příkaz již vytvořen při dřívějším volání metody `createcommand` v třídě `TkWrapper`, tato metoda pouze uloží příkaz do „mezipaměti“ instance třídy `Implementer`.
- `_add_widget` – Přidá widget do rozložení rodiče.
- `mainloop` – Spustí PyQt aplikaci.

Během tohoto používá třída `Implementer` vlastních „pamětí“, případně „cachí“, jak bylo predikováno v návrhu:

- `namer` – Ukládá widget pod svým názvem používaný v Tkinteru. Toho je využito pro vyhledání rodiče widgetu při vytváření widgetu, při přiřazování příkazů a při dalších situacích.
- `commands` – Ukládá příkazy vytvořené metodou createcommand. Používá se když je příkaz přiřazován widgetu (resp. je přiřazen na nějaký slot widgetu).
- `layouter` – Ukládá rozložení, přesněji třídu `Layouter`, pod názvem rodičovského widgetu, který toto rozložení používá.
- `masters` – Ukládá rodiče widgetu pod názvem widgetu.
- `bindings` – Ukládá vazby událostí pod názvem widgetu. Tyto vazby tedy jsou používány se ve funkci `call_if_binding_holds`.

„Cache“ `layouter` a `masters` se používají společně – každý widget, na kterém je použitá jakákoli správa rozložení, je umístěn do rozložení rodiče widgetu. Pokud toto rozložení respektive instance třídy `Layouter` zatím neexistuje, je vytvořeno. Toto funguje rekurzivně. Klíče těchto pamětí, jak bylo řečeno, jsou názvy widgetů, jelikož v Tcl/Tk jsou objekty identifikovány pomocí jejich jména.

Druhou třídou v souboru `implementer.py`, ve kterém je třída `Implementer` umístěna, je třída `Menu`. Tato třída dědí z třídy `QMainWindow`, což umožňuje aplikaci mít nabídku. V Tkinteru má třída `Application` možnost přiřazení nabídky, třída `QApplication` toto však nepodporuje. Pokud aplikace nemá

žádné menu, třída `Implementer` si buď vytvoří vlastní widget nebo dostane vlastní widget jako hlavní okno, které je zobrazeno metodou `show`. Pokud má aplikace mít nabídku, tento výchozí widget je vyměněn za třídu `Menu`. Třída sama o sobě není opravdová nabídka, je to jen toplevel widget, který má možnost přiřazení nabídky (díky dědění z `QMainWindow`). Nabídka je vytvořena uvnitř třídy metodou `add_menu`. Tkinter někdy přiřadí označení akce nabídky před samotnou akcí, proto je implementována metoda `remeber_label`. Nabídka v Tkinteru je zcela odlišná od nabídky v PyQt, je ji tak třeba aplikaci předat rozdílně, což je vyřešeno při volání metody `add_to_namer` v třídě `Implementer`. Detailně je možné toto najít v překladech metod tříd `Menu` a `Implementer` v mapování parametrů 5.9. Jelikož třída `Application` z Tkinteru může získat nabídku, příkaz `-menu` je přeložen do metody `create_menu` v třídě `Implementer`, která ve skutečnosti pouze nastaví příznak, který je poté použit k rozhodnutí, zda má být instance nabídky deklarována jako centrální widget v metodě `show`.

5.4 Překlad událostí

Mechanismus svázání události s určitým příkazem či funkcí je zastřešen nejdříve dekompozicí v metodě `createcommand` třídy `TkWrapper` (5.2) a poté svázáním v rámci implementace metody `call` třídy `Implementer` (6.3).

Protože je aplikaci vytvořena pomocí Qt toolkitu, dochází k událostem v Qt, ale původní program v Tkinteru mohl použít událost ve svých funkcích nebo metodách, které byly svázány s určitou událostí. Proto je třeba událost v Qt přeložit zpět na událost v Tk.

K tomuto překladu dochází v třídě `EventBuilder` v souboru jménem `event_builder.py`. Tkinter implementuje podobnou logiku s událostmi z Tk. Když Tkinter váže funkci k nějaké události, neváže jen samotnou událost, ale váže také logiku, která funguje nejen jako slot, ale také ukládá parametry události v předpřipraveném řetězci.

Detaily tohoto řetězce je možné nalézt v [6]. Je vhodné poznamenat, že chování některých typů událostí z Tkinteru není stejné jako v PyQt, například `QKeyEvent` nemá možnost pracovat s pozicí události.

5.5 Podpora událostí

Pro podporu překladů událostí dle 5.4 vznikl soubor `event_translate.py`. Tento soubor obsahuje podpůrné funkce k překladu atributů a jmen používaných se při práci s událostmi. Funkce z tohoto souboru jsou použité při přiřazování vazby na události, při nastání události v Qt, při kontrole svázání a při překladu události z Qt na událost Tk, má tak několik využití.

- `key_translator` – Přeloží znak z Qt na znak z `TkKey`. Tento překlad je proveden pouze porovnáním ascii, které by mělo být přesné pro většinu obvyklých kláves, dle [38] a [32] [6]. Je možné si všimnout, že události PyQt nerozlišují, jestli byl znak velké písmeno, nebo ne.
- `mouse_button_translator` – Přeloží `QtMouseButton` na `TkMouseButton` nebo naopak, jelikož číslování tlačítek myši je realizováno v Tk a Qt rozdílně.
- `sequence_parser` – Analyzuje sekvenci (vzor pro vyvolání události) z formátu pro Tkinter na čitelnější a jednotnější formát.
- `tk_modifier_to_qt` – Přeloží modifikátor z Tk a přeloží jej do Qt modifikátoru. Ne každý modifikátor z Tk má svojí odpovídající modifikátor v Qt.
- `get_method_for_type` – Název typu události pro Tkinter přeloží na metodu resp. slot pro widget z PyQt. Je možné pozorovat, že mnoho událostí z Tk nemá odpovídající slot v Qt widgetech, nebo naopak.

5.6 Dialog pro nahrání souboru

Qtinter podporuje použití dialogů pro práci se soubory. Zdrojový soubor `filedialog` jako náhrada za modul `tkinter.filedialog`. Aktuálně implementuje následující dialogy (nalevo je akce `filedialogu` z Tkinteru, napravo akce z PyQt):

- `askopenfilename` – pomocí `getOpenFileName`,
- `asksaveasfilename` – pomocí `getSaveFileName`,
- `askopenfilenames` – pomocí `getOpenFileNames`,
- `askdirectory` – pomocí `getExistingDirectory`.

Ostatní souborové dialogy nemají přímou podporu v PyQt.

5.7 Layouter

Soubor `layouter.py` obsahuje třídu `Layouter`. Tato třída řeší správu rozložení (geometrie) pomocí správců rozložení `QGridLayout` nebo `QBoxLayout` z Qt, co nejpodobněji správci geometrie v Tkinteru. Třída `Layouter` nepodporuje správce geometrie `place` z Tkinteru.

Layouter je vytvořen jako neinicializovaný. Je to proto, že instance třídy `Layouter` je vytvořena okamžitě při vytváření widgetu, jak bylo popsáno v 5.3,

[6] Odkaz na dokumentaci PyQt verze 4 je zde uveden schválně z důvodu uvedeného v 4

kdy widget nemusí být rodičem žádného dalšího widgetu, rozložení potomků tak nemusí být potřeba. Rozložení pro potomky je vytvořeno až ve chvíli, kdy je vytvořen další widget s původním widgetem jako rodičem.

Potomek může být vložen do rodičovského rozložení metodou `add_widget`. Pokud instance třídy `Layouter` není inicializována, je zavolána vnitřní metoda `_manual_init`. Metoda vložení prvně vloženého potomku rozhodne, jaká strategie inicializace je využita.

Pokud je použit správce geometrie mřížky (grid), je inicializována instance třídy `QGridLayout`. Pro tohoto správce třída `Layouter` aktuálně podporuje určení řádků a sloupců, rozpětí (sloučení) řádků a sloupců a nastavení parametru `sticky`, které rozhoduje, jak je využito volné místo v mřížce.

Pokud je použit správce geometrie balení (pack), inicializace se se liší dle zabalené strany potomka (parametr side):

- **Pokud je potomek zabalen k levé straně**, je inicializována instance třídy `QHBoxLayout` (horizontální) a postupně naplněna:
  1. `QHBoxLayout` (pro potomky zabalené k levé straně),
  2. `QHBoxLayout` s `QVBoxLayout` (pro potomky zabalené k horní straně – shora) a `QVBoxLayout` s opačným řazením (pro potomky zabalené k dolní straně – zdola),
  3. `QHBoxLayout` s opačným řazením (pro potomky zabalené k pravé straně).

  – Toto odpovídá třem sloupcům, kde prostřední sloupec umožňuje balení ze shora a ze spoda.

- **Pokud je potomek zabalen k jiné než levé straně**, je inicializována instance třídy `QVBoxLayout` (vertikální) a postupně naplněna instancemi tříd:
  1. `QVBoxLayout` (pro potomky zabalené k horní straně – shora),
  2. `QVBoxLayout` s `QHBoxLayout` (pro potomky zabalené k levé straně) a `QHBoxLayout` s opačným řazením (pro potomky zabalené k pravé straně),
  3. `QVBoxLayout` s opačným řazením (pro potomky zabalené k dolní straně – zdola).

  – Toto odpovídá třem řádkům, kde prostřední řádek umožňuje balení zleva a zprava.

Tyto dvě strategie balení se ukázaly jako nejpodobnější řešení správce geometrie balení (pack) z Tkinteru a také nejlépe vypadající, jelikož zabalení prvního widgetu ke spodní nebo pravé straně nevytvářelo graficky vhodné

výsledky. Možnosti třídy `Layouter` tedy omezují použití správce geometrie pack.

Po inicializaci třída `Layouter` používá samostatně metody `pack_widget` nebo `grid_widget`. V případě balení (pack) má třída `Layouter` odkazy na každé rozložení (vlevo, vpravo, dole, nahoře), všechny jsou vytvářeny pokaždé.

Třída `Layouter` nepodporuje míchání rozložení, jakmile je inicializována, druh rozložení (balíček, mřížka) se již nemůže změnit. Pokud je přidán potomek vyžadující vložení do rozdílného rozložení, je vyvolána výjimka. Toto chování sice Tkinter podporuje, nicméně opět dochází k příliš nepředpovídatelným výsledkům.

`Layouter` je podle své definice rekurzivní struktura. Druh rozvržení nelze kombinovat v jednom rodičovském widgetu, ale rodičovský widget může být umístěn v jiném druhu rozložení než jeho vlastní druh rozvržení, což také platí pro jeho potomky a tak dále.

5.8 QtCore konstanty

Soubor `translate_qt_core` obsahuje podpůrné funkce pro třídu `Layouter`. V současné době obsahuje 1 metodu:

- `translate_align` – Přeloží parametr `sticky` z Tkinteru do hodnoty konstanty `QtAlignment` z Qt. K tomu se používají se bitové operace.

Soubor je vhodným kandidátem pro rozšíření o další překlady konstant z Tk na konstanty Qt a naopak.

5.9 Mapování parametrů

Myšlenka vybraného mapování dle 4.1.3 spočívá v tom, že jednotlivá nastavení widgetů bude možné jednoznačně přeložit do PyQt. Zdrojový soubor `parameters_mapping.py` obsahuje mapování, které využívá k překladu parametrů metoda `call_method` z třídy `Implementer`. Toto mapování je využito pro překlad názvů parametrů na určití metody určitých widgetů z PyQt a pro překlad hodnot těchto parametrů. Implementovány jsou následující struktury:

- `parameters_names_mapping` – Tato struktura mapuje atributy příkazu Tcl/Tk metodám Qt a vlastních objektů. Vlastní objekty jsou zahrnuty, kvůli podrobnostem implementace, jako je problém s nabídkou (viz popis třídy Implementer v 6.3).

- `parameters_values_mapping` – Tato struktura mapuje hodnoty atributů Tcl/Tk na platné hodnoty pro parametry metod PyQt a vlastních tříd. Použitá je rovněž pro hodnoty, které se týkají stylistické stránky.

- `_get_mapping` – Mapování získané na základě dané třídy z Tkinteru.

Celé mapování pro `call_method` z třídy `Implementer` dle 6.3 je získáno funkcí `get_parameters_names_mapping` a `get_parameters_values_mapping` s pomocí privátní funkce `_get_mapping`.

Tato struktura mapování byla takto implementována, aby bylo možné ji jednoduše rozšířit o nové vlastnosti, které Qtinter bude podporovat, a to pouze jednoduchým zásahem do této struktury. V první úrovni struktury je vždy klíčem název třídy widgetu v Qt, vlastní třída Qtinteru, nebo klíč `all`. Klíč `all` slouží pro standarní atributy všech widgetů z Tkinteru. V druhé úrovni dle klíče z první úrovně této struktury je slovník, který má jako klíč vlastnosti pro Tcl (dle vygenerované n-tice) a hodnotou klíče je v případě:

- `parameters_names_mapping` – funkce, kterou je možné volat v PyQt (nebo Qtinteru) pro danou třídu dle první úrovně (nebo všechny dle klíče `all`).
- `parameters_values_mapping` – cílový formát hodnoty.

V ukázce 5.2 je naznačeno, jak je možné pomocí těchto struktur:

- Nastavit mapování Tk vlastnosti `-text` na metodu `setText` pro všechny objekty z PyQt.
- Nastavení popředí vlastností `-fg`, která je překládána na funkci nastavení šablony `setStyleSheet` z PyQt, jež vyžaduje také zformátování vstupní hodnoty pomocí `parameters_values_mapping`.
- Nastavení svázání události po kliknutí na slot třídy `QPushButton`.

5.10 Překlad dle mapování - tktoqt

Samotný překlad podle struktur z 5.9 je prováděn v souboru `tktoqt.py`. Tento soubor obsahuje funkce, které řeší překlad z příkazu Tcl/Tk na metody a parametry validní pro PyQt třídy. Tohoto docílí pomocí následujících funkcí:

- `translate_parameters_values` – Přeloží hodnoty parametrů do formátu pro metody PyQt tříd.
- `translate_parameters_names` – Přeloží názvy parametrů do názvů metod PyQt tříd.
- `translate_parameters_for_class` – Přeloží názvy parametrů a hodnot parametrů do formátu PyQt pro daný název třídy z PyQt / vlastní třídy.

```
parameters_names_mapping = {
    "all": {
        '-text': 'setText',
        '-fg': 'setStyleSheet',
    },
    "QPushButton": {
        '-command': 'clicked.connect',
    },
}

parameters_values_mapping = {
    "all": {
        '-fg': 'color: {};',
    },
    "QWidget": {
        '-background': 'background-color: {};',
    },
}
```

Obrázek 5.2: Ukázka mapovacích struktur

Kapitola 6

Testování a zhodnocení

Během implementace probíhalo k otestování implementovaného mapování převážně pomocí manuálního testování pohledů jednoduchých aplikací, doplněné o automatizované testy některých funkcionálních částí nástroje Qtinter.

6.1 Regresní testování

Jelikož jsem implementaci Qtinteru vícekrát v průběhu tvorby přepisoval, bylo vhodné zvážit možnosti automatizace regresního testování pro pohledy aplikací.

Tato automatizace testování pohledů může probíhat pomocí nástrojů obsahující technologii OCR (z anglického Optical Character Recognition), která umí rozpoznat písmo na obrázku. Tento obrázek je možné vytvořit pomocí balíčku `pyautogui` pro Python. Mimo jeho další funkce umí udělat snímek obrazovky. Po tom, co je obrázek pořízen, je třeba z obrázku analyzovat text. K tomu je možné využít Python balíček `tesserocr`, který integruje Tesseract rozhraní z C++ pro rozpoznání textu. Ve složce `ocr` ve zdrojových kódech jsem přidal menší příklad použití tohoto nástroje. Bohužel jsem nenašel vhodné nastavení, při kterém by byla detekce textu spolehlivá, navíc tento způsob neměl možnost kontrolovat tvary nebo styly tlačítek, ohraničení a dalších důležitých částí pohledů.

Silnější nástroj pro tento typ testování je nástroj openQA. Tento nástroj umožňuje automatizované testování celého operačního systému, od jeho instalačního procesu po jeho nastavení. Tento nástroj je produktem projektu openSUSE. Síla tohoto nástroje spočívá v možnosti definovat takzvané „jehly“[7], což jsou zjednoduše pohledy správného výsledku určitého příkazu. OpenQA je schopné tyto pohledy porovnat a zajistit kontrolu grafického výstupu výsledku. Jehlou nemusí být celá oblast pohledu, může to být jen jeho část. V případě částí je nutné počítat s tím, že openQA stačí najít jehlu kdekoliv v pohledu pro označení testu jako úspěšného. Jehly je možné vytvářet v nástroji Python

[7] Vychází z fráze „hledat jehlu v kupce sena“, respektive „find needle in haystack“.

Needle Editor. Nástroj openQA je nativně dostupný pro openSUSE distribuci, v dalších distribucích je možné ho používat v nástroji docker. Funguje tak, že ve virtualizovaném prostředí spustí zadaný obraz operačního systému a provede příkazy zadané v programovacím jazyce Perl.

Pro vlastní účely testování je nástroj openQA často využíván tak, že je rozšířen nějaký existující případ testování z openQA o část, kde bude kromě spuštění obrazu operačního systému také spuštěna aplikace pracující s balíčkem Qtinter, a podoba aplikace bude testována vůči připraveným jehlám, které jsou přidané v jednotlivých testovacích případech. Tuto možnost jsem zkoušel, bohužel se mi ale nepodařilo úspěšně spustit instanci ve virtualizovaném prostředí v dockeru.[39]

6.2 Otestování s aplikací IDLE

Po implementaci provedené podle předchozí kapitoly jsem vyzkoušel program IDLE spustit se závislostí na balíčku Qtinter. Jelikož spuštění hlásilo několik chyb, rozšířil jsem implementované části balíčku Qtinter o několik částí, které nebyly plně implementovány. Avšak ani po několikerém opakování tohoto procesu jsem nebyl schopný IDLE korektně spustit. Vytvořil jsem proto sadu jednodušších příkladů, které jsem zanesl do složky `examples` ve zdrojových kódech, a na kterých je možné pozorovat implementované funkcionality. Těmito příklady jsou:

- `all_user_input_widgets.py` – Příklad vykreslující většinu uživatelských widgetů. Kromě možností vyplnění widgetů, tlačítek pro ukončení programu a rozložení do mřížky také tento příklad prezentuje práci s menu a otevíráním souborových dialogů.

- `events.py` – Příklad práce s událostmi, reaguje na kliknutí myši nebo zmáčknutí klávesy. Rozlišuje, které tlačítko myši bylo stisknuto a do konzole tiskne hodnotu odpovídající pro Tkinter.

- `grid.py` – Příklad práce s rozložením v mřížce. Kliknutím na středové tlačíko je změněno jeho rozšíření a ukotvení v prostředních (volných) buňkách tabulky.

- `hello_world.py` – Typický Tkinter „Hello world!“ příklad.

- `menu_and_dialog.py` – Ukázka menu, akcí v menu, podmenu a dialogu pro výběr souboru, včetně vytištění jména zadaného souboru a cesty k němu.

- `pack.py` – Příklad práce s rozložením pomocí balíčku (pack), vycházející z „Hello world!“ příkladu.

6.3 Zhodnocení dle kvantifikátorů

Pro zhodnocení úspěšnosti navrženého mapování a jeho implementace jsem navrhl kvantifikátory dle sekce 4.2. Otestování s aplikací IDLE se nezdařilo, ovšem tyto kvantifikátory je stále možné zhodnotit:

- **Implementované třídy (widgety)** – Implementoval jsem většinu tříd z Tkinteru, včetně několika z `ttk`. Pokud by bylo třeba nějakou přidat, je možné podporované třídy přidat do slovníku `translate_class_dict` jednoduchým způsovbem v rámci zdrojového souboru `Implementer` dle 6.3.
- **Výchozí parametry** – Z výchozích parametrů widgetů z Tkinteru jsem implementoval jen některé. Tyto parametry je ale možné velice jednoduše rozšířit dle 5.9.
- **Speciální parametry** – Ze speciálních parametrů jsem implementoval také jen některé. Rozšíření je opět možné dle 5.9.
- **Funkcionality** – Z hlavních funkcionalit Tkinteru jsem implementoval několik z nich:
  - Zavedl jsem podporu pro několik widgetů, včetně podpory nabídky.
  - Qtinter umožňuje práci s událostmi, na úrovni spojení s instancí a aplikací, včetně navázání a překladu mezi událostmi z Qt do Tk pro jejich další obsluhu v zavedených funkcích.
  - Díky třídě `Layouter` jsem dle 5.7 umožnil takovou správu rozložení, která vychází ze správců geometrie z Tkinteru.
  - Umožnil jsem používat souborové dialogy.
  - Mezi základními widgety je i podpora pro widget pro zobrazení grafiky.
  - Vytvořil jsem prototyp pro práci s proměnnými z Tkinteru – konkrétně proměnnou řetězce. Ostatní by mohly být implementovány podobně.
  - Kromě překladu základních a speciálních parametrů jsem navrhl možnost zformátování hodnot těchto parametrů. Vytvořil jsem tak možnost překladu stylizace Tk widgetů na stylizaci pomocí šablon v Qt.
  - Mezi význačné funkcionality, které **nebyly zpracovány**, patří podpora dialogů z Tkinter modulu `tkMessageBox` a podpora práce s písmeny, z části `tkinter.font`, jelikož jsem jim při implementaci nedal dostatečnou váhu. Například chybějící podpora `tkinter.font` určitě může za problémové spuštění nastroje IDLE.

- **Rozšiřitelnost** – Celý nástroj Qtinter jsem se snažil navrhnout tak, aby ho bylo možné jednoduše rozšířit, protože jsem nepředpokládál, že v rámci implementace budu schopný zaručit plnohodnotnou podporu. V rámci implementace jsem využil nebo doporučil mechanismy, které by rozšiřitelnosti měly docílit. Hlavní sílu rozšiřitelnosti své implementace vidím v množství struktur a slovníků, které umožňují jednoduché rozšíření. Hlavní část programu, třída `Implementer` je až na vyjímky vyjmenované v algoritmu třídy dle napsána tak, aby nebrala v potaz konkrétní třídy nebo nastavované atributy těchto tříd, zpracovává čistě jen vstupní n-tice, pomocí překladů ze souborů s mapováním dle 5.9. Možnosti rozšíření nástroje Qtinter jsem popsal v daných místech zdrojového kódu.

Závěr

V teoretické části této práce jsem provedl rešerši k tématu grafického uživatelského rozhraní, zahrnující popis sad nástrojů Tk, Qt a jejich možnostech využití v Pythonu v knihovně Tkinter nebo v balíčcích PyQt a PySide. Teoretickou část jsem zakončil shrnutím motivace ke zpracování tématu, které spočívá umožnění provozu Tkinter aplikací jako klientů nejnovějších správců obrazovky na moderních displayích.

V rámci praktické částí své práce jsem analyzoval knihovnu Tkinter a balíčky PyQt a PySide podrobněji, popsal jsem jejich důležité mechanismy a vzájemně je srovnal. Na základě provedené analýzy jsem navrhl možnosti mapování a navrhl pro realizaci to, které splňovalo definovaná kritéria. Abych mohl zhodnotit implementovaný prototyp zvoleného mapování, navrhl jsem hodnotící kvantifikátory.

V rámci realizace jsem zvolené pro zvolené mapování implementoval prototyp podporující nejdůležitější funkce rozhraní Tkinteru. Při konstrukci tohoto prototypu jsem dbal na to, aby bylo možné ho jednoduše rozšířit, což se osvědčilo při testování s reálnými aplikacemi. Způsob rozšíření je v práci náležitě popsán včetně ukázek.

Implementovaný prototyp obsahuje podporu většiny základních prvků Tkinteru s jejich nastaveními, podporu událostí, různých způsobů tvorby rozložení a dialogů. Avšak ani s takto pokročilým prototypem nebylo možné využít implementovaného nástroje pro aplikaci IDLE. Pro účely prezentace funkcí implementovaného nástroje jsou v práci uvedeny funkční příklady použití, který vychází z jednoduchý aplikací a pro využití implementovaného prototypu potřebují pouze změnit importovaný modul.

Přínos své práce vnímám v návrhu vhodného mapování, které umožnilo jednoduchým aplikacím v Tkinteru využívat Qt komponenty, a které je možné jednoduše rozšířit úpravou, která nevyžaduje znalost vnitřní implementace nástroje. Prototyp byl vydán pod svobodnou permisivní licencí a může tak být volně šířen a upravován k plnohodnotnému nástroji.

Literatura

[1] Daryl Harms, K. M.: *Začínáme programovat v jazyce Python*. Brno: Computer Press, a.s., druhé vydání, 2008, ISBN 978-80-251-2161-0, 3-14, 27 s.

[2] Martinez, W. L.: Graphical user interfaces. *WIREs Computational Statistics*, ročník 3, č. 2, 2011: s. 119–133, doi:https://doi.org/10.1002/wics.150, [Online; cit. 2021-02-14], https://onlinelibrary.wiley.com/doi/pdf/10.1002/wics.150. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/wics.150

[3] Wikipedia contributors: Graphical user interface. 2004, [Online; cit. 2021-02-14]. Dostupné z: https://en.wikipedia.org/wiki/Graphical_user_interface

[4] Wikipedia contributors: Widget toolkit. 2005, [Online; cit. 2021-02-14]. Dostupné z: https://en.wikipedia.org/wiki/Widget_toolkit

[5] Brad Myers, R. P., Scott E. Hudson: Past, Present and Future of User Interface Software Tools. *Carnegie Mellon University*, 2000, [Online; cit. 2021-02-14]. Dostupné z: https://www.cs.cmu.edu/~amulet/papers/futureofhciACM.pdf

[6] Python Software Foundation: tkinter — Python interface to Tcl/Tk. [online], 2001, [Online; cit. 2021-02-14]. Dostupné z: https://docs.python.org/3/library/tkinter.html

[7] Tcl Developer Xchange: Language. [Online; cit. 2021-05-04]. Dostupné z: http://www.tcl.tk/about/language.html

[8] Tcl Developer Xchange: Uses for Tcl/Tk. [Online; cit. 2021-05-04]. Dostupné z: http://www.tcl.tk/about/uses.html

[9] The Qt Company: Legal — Licensing - Qt. [Online; cit. 2021-05-05]. Dostupné z: https://www.qt.io/licensing/

[10] Thelin, J.: *Foundations of Qt Development*. Apress, 2007, ISBN 978-1-59059-831-3.

[11] Riverbank Computing Limited: What is PyQt? [online], [Online; cit. 2021-02-14]. Dostupné z: https://riverbankcomputing.com/software/pyqt/intro

[12] Riverbank Computing Limited: What is SIP? [online], [Online; cit. 2021-02-14]. Dostupné z: https://www.riverbankcomputing.com/software/sip/intro

[13] PySide Team: PySide. [online], [Online; cit. 2021-02-14]. Dostupné z: https://pypi.org/project/PySide/

[14] The Qt Company Ltd.: Shiboken. [online], [Online; cit. 2021-02-14]. Dostupné z: https://doc.qt.io/qtforpython/shiboken6/

[15] Qt Wiki: Differences Between PySide and PyQt. [Online; cit. 2021-05-04]. Dostupné z: https://wiki.qt.io/Differences_Between_PySide_and_PyQt

[16] The Spyder Development Team: QtPy: Abstraction layer for PyQt5/PyQt4/PySide2/PySide. [Online; cit. 2021-05-04]. Dostupné z: https://github.com/spyder-ide/qtpy

[17] Tcl Developer Xchange: Tcl/Tk Licensing Terms. [online], [Online; cit. 2021-05-05]. Dostupné z: http://www.tcl.tk/software/tcltk/license.html

[18] Johan Helsing: Creating devices with multiple UI processes using Wayland. [Online; cit. 2021-05-05]. Dostupné z: https://www.qt.io/blog/2017/01/23/creating-devices-with-multiple-ui-processes-using-wayland

[19] Wikipedia contributors: Wayland (display server protocol) — Wikipedia, The Free Encyclopedia. 2021, [Online; cit. 2021-05-05]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Wayland_(display_server_protocol)&oldid=1018059209

[20] Tcl Developer Xchange: Binary Distributions. [Online; cit. 2021-05-05]. Dostupné z: https://wiki.tcl-lang.org/page/Binary+Distributions

[21] Jiří Eischmann: HiDPI v Linuxu: podpora se zlepšuje, kazí ji staré aplikace a frameworky. [Online; cit. 2021-05-06]. Dostupné z: https://www.root.cz/clanky/hidpi-v-linuxu-podpora-se-zlepsuje-kazi-ji-stare-aplikace-a-frameworky/

[22] John W. Shipman: Layout management. [Online; cit. 2021-05-05]. Dostupné z: https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/layout-mgt.html

[23] Sun Microsystems, Inc.: Tcl8.6.11/Tk8.6.11 Documentation. [Online; cit. 2021-05-05]. Dostupné z: http://www.tcl.tk/man/tcl8.6/TkCmd/pack.htm

[24] John W. Shipman: Tkinter 8.5 reference: a GUI for Python. [online], [Online; cit. 2021-05-05]. Dostupné z: https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html

[25] John W. Shipman: Tkinter 8.5 reference: a GUI for Python. [Online; cit. 2021-05-05]. Dostupné z: https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/events.html

[26] John W. Shipman: Themed widgets. [Online; cit. 2021-05-05]. Dostupné z: https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/ttk.html

[27] John W. Shipman: The tkMessageBox dialogs module. [Online; cit. 2021-05-05]. Dostupné z: https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/tkMessageBox.html

[28] John W. Shipman: The tkMessageBox dialogs module. [Online; cit. 2021-05-05]. Dostupné z: https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/tkFileDialog.html

[29] Riverbank Computing Limited: Modules. [online], [Online; cit. 2021-05-05]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/module_index.html

[30] Riverbank Computing Limited: QtCore. [Online; cit. 2021-05-05]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtcore/qtcore-module.html

[31] Riverbank Computing Limited: Qt Class Reference. [Online; cit. 2021-05-05]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtcore/qt.html

[32] Riverbank Computing Limited: PyQt Reference Guide. [Online; cit. 2021-05-05]. Dostupné z: https://doc.bccnsoft.com/docs/PyQt4/qt.html

[33] Riverbank Computing Limited: QtGui. [Online; cit. 2021-05-05]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtgui/qtgui-module.html

[34] Riverbank Computing Limited: QWidget. [Online; cit. 2021-05-05]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtcore/qtcore-module.html

[35] The Qt Company Ltd.: Qt Style Sheets. [Online; cit. 2021-05-05]. Dostupné z: https://doc.qt.io/qt-5/stylesheet.html

[36] The Qt Company Ltd.: Layout Management. [Online; cit. 2021-05-05]. Dostupné z: https://doc.qt.io/qt-5/layout.html

[37] Riverbank Computing Limited: QDialog. [Online; cit. 2021-05-05]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtwidgets/qdialog.html

[38] Roger E. Critchlow Jr.: Tk Built-In Commands - keysyms. [Online; cit. 2021-05-06]. Dostupné z: http://www.tcl.tk/man/tcl8.4/TkCmd/keysyms.htm

[39] openSUSE: openQA Tutorial. [Youtube; cit. 2021-05-06]. Dostupné z: https://www.youtube.com/watch?v=2zwU9_bV_zI

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

CLI Command line interface

HID Human Interface Device

PSFL Python Software Foundation Licence

IDLE Integrated DeveLopment Environment

MFC The Microsoft Foundation Classes

WTL The Windows Template Library

OWL The Object Windows Library

VCL The Visual Component Library

WYSIWYG What You See Is What You Get

OSS Open Source Software

LGPL GNU Lesser General Public License

GPL GNU General Public License

I/O Input/Output – Vstupní a výstupní operace

TDBC Tcl Database Connectivity Interface

CSS Cascading Style Sheets

RGB Red, Green, Blue – definice barvy pomocí aditivního modelu

SDL2 Simple DirectMedia Layer

GSoC Google Summer of Code

API Application Programming

QWS Qt Window System

OCR Optical Character Recognition

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Schuh_Matěj_2021.tex` ..... text práce ve formátu PDF