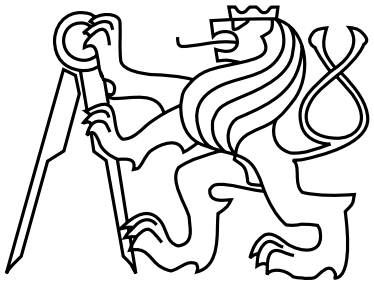
Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Automatizované testování telefonních aplikací

Bc. Ondřej Procházka

Vedoucí práce: Ing. Ondřej Guth

Studijní program: Elektrotechnika a informatika, strukturovaný, Navazující magisterský

Obor: Výpočetní technika

12. května 2014

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé bakalářské práce Ing. Ondřeji Guthovi za hodnotné rady a připomínky. Dále bych rád poděkoval mé manželce za podporu po celou dobu mého studia.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 12. 5. 2014 .....

Abstract

This diploma thesis deals with the design and implementation of an application system for automated testing of Cisco IP Phones applications. This thesis follows the bachelor thesis named „Cisco IP phone emulator“. The concept also includes the assessment of a suitable scripting language. Part of the thesis focuses on the implementation of a graphic tool designed to simply create and define testing scripts.

Abstrakt

Tato práce se zabývá návrhem a vytvořením systému k automatizovanému testování telefonních aplikací určených pro IP telefony společnosti Cisco Systems. Navazuje na bakalářskou práci „Emulátor IP telefonu“. Návrh se týká i výběru vhodného skriptovacího jazyka, ve kterém budou psány testovací skripty. Součástí práce je vytvoření grafického nástroje, pro jednoduché vytváření a definování testovacích skriptů.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

U každého vývoje počítačového programu je nutné výsledek řádně otestovat. Stejný požadavek je i u aplikací určených pro Cisco IP Telefony. V současné době však neexistuje žádný systém, který by dokázal testování telefonních aplikací automatizovat. Proto vznikl požadavek na vytvoření systému pro automatizované testování telefonních aplikací určených pro Cisco IP telefony, který jsem si zvolil jako téma své diplomové práce.

Diplomová práce navazuje na moji bakalářskou práci s názvem „Emulátor Cisco IP telefonu“. Uživatel si napíše testovací skript a pomocí testovacího programu je provedeno automatizované otestování telefonní aplikace. Testovaná aplikace je spuštěna pomocí emulátoru IP telefonu a chování testované aplikace je průběžně kontrolováno pomocí testovacího skriptu.

Kapitola 2

Popis problému, specifikace cíle

2.1 Vymezení cílů

Cílem diplomové práce je vytvoření systému, pomocí kterého lze automatizovaně testovat telefonní aplikace. Výsledný systém provádí funkční testování vyvíjené telefonní aplikace. Dalším požadavkem na výslednou aplikaci je možnost integrace výsledného programu do systému pro automatické sestavení aplikací - continuous build.

Testovací scénáře jsou uloženy ve formě skriptu. To umožní budoucí rozšíření testovacího systému a testovacích scénářů. Syntaxe a jazyk použitý ve skriptu není definován, proto je tématem předkládané práce také návrh nebo výběr vhodného skriptovacího jazyka.

Součástí projektu je i vytvoření editoru, pomocí kterého lze vytvářet a modifikovat testovací scripty.

2.2 Popis struktury diplomové práce ve vztahu k vytyčeným cílům

V kapitole 3 je popsáno aplikační rozhraní Cisco IP telefonu, analýza testování telefonních aplikací a návrh řešení s ohledem na vytyčené cíle. Popis implementace je uveden v kapitole 4. V kapitole **??** jsou popsány testy, kterými bylo ověřeno splnění vytyčených cílů tohoto projektu.

2.3 Existující řešení

Společnost Cisco Systems nabízí pro své partnery Software Developer Kit - SDK knihovnu s popisem aplikačního rozhraní telefonu. Bohužel v současné době neexistuje žádný speciální program pro vývoj telefonních aplikací a ani žádný systém pro jejich automatizované testování. Zatím je jedinou možností spustit aplikaci na Cisco IP telefonu a ručně projít všechny možnosti aplikace a také ručně zkontrolovat všechny dialogy, zda obsahují správné texty.

Protože aplikace pro Cisco IP telefony jsou založeny na XML dokumentech, je tak další možností jejich kontrola, kterou lze provádět několika různými způsoby. Využít lze například následující technologie:

- XSD – XML Schema Definition.
- DTD – Document Type Definition.
- Schematron – Validace dokumentu na základě pravidel.

Všechny výše uvedené způsoby jsou založeny na kontrole XML schématu. XSD i Schematron lze využít i pro kontrolu obsahu (hodnot), které dokument obsahuje. Kontrolu XML souborů lze pomocí těchto nástrojů provádět i automatizovaně. Vytvoření sady schémat pro kontrolu rozsáhlejší aplikace může být poměrně náročné a může zabrat více času. Další omezení se týká dynamických odkazů, které vznikají na základě vyplnění dat na telefonech, popřípadě různé parametry pro zajištění stejného sezení (SESSION ID). Tato data se v XML dokumentech mění a není je tak možné testovat pomocí standardních nástrojů.

U kontroly XML souborů a jejich obsahu je důležité vzít v úvahu i kontext. Stejná URL adresa může při různých parametrech nebo v jiný čas vrátit odlišný výsledek. Jedná se o případy, kdy má server uložené parametry, na základě kterých se rozhoduje, jakým způsobem zpracuje požadavek od telefonu. IP telefon může v těchto případech posílat požadavky se stejnou URL adresou. Ve výše uvedených způsobech kontroly ale nemusí být vždy možné rozhodnout, co je vlastně očekávaným výstupem telefonní aplikace a co se má tedy vlastně testovat. Mohu zmínit ještě poslední negativní vlastnost. Vazba mezi jednotlivými obrazovkami telefonní aplikace je pomocí URL adresy, proto by musel mít testovací systém uložené URL adresy jednotlivých obrazovek a k nim i očekávaný výstup. Pokud se ale URL adresa v aplikaci změní, bylo by nutné je změnit v celém systému. To je časově velice zdlouhavé.

Protože neexistuje hotový a uspokojivý systém pro automatizované testování telefonních aplikací, vznikla nutnost vytvořit aplikaci novou. Pro novou aplikaci jsem sestavil následující základní a nutné požadavky:

- Celé řešení je nezávislé na telefonní ústředně.
- Systém je schopen testovanou aplikaci testovat plně automaticky.
- Testování aplikace je integrovatelné do systému pro automatické sestavení výsledné aplikace - například CruiseControl.
- Testovací scénáře jsou opakovatelné i pro různé typy telefonů.
- Testovací scénáře jsou uložené v textovém souboru.
- Testovací scénáře jsou definovány ve skriptovacím jazyku.
- Simulace ovládání IP telefonu.

Kapitola 3

Analýza a návrh řešení

První část této kapitoly rekapituluje důležité části aplikačního rozhraní Cisco IP telefonů.

3.1 Cisco API

Aplikační rozhraní Cisco IP telefonů je detailně popsáno v bakalářské práci s názvem „Emulátor Cisco IP telefonu“, ze které pro lepší orientaci v dalším textu diplomové práce předkládám důležité části.

Cisco IP telefon si pomocí HTTP nebo HTTPS protokolu stahuje z definované adresy XML dokumenty. Tyto XML dokumenty obsahují definici objektů, na základě kterých je provedena příslušná činnost na telefon - například zobrazení textu nebo zobrazení menu. Seznam objektů, které podporují Cisco IP telefony:

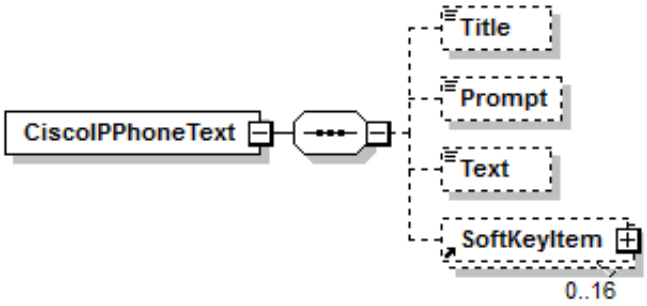
- CiscoIPPhoneMenu – zobrazení nabídky na IP telefonu.
- CiscoIPPhoneText – zobrazení textu.
- CiscoIPPhoneInput – zobrazení vstupního formuláře.
- CiscoIPPhoneDirectory – zobrazení adresáře.
- CiscoIPPhoneImageFile – zobrazení obrázku.
- CiscoIPPhoneGraphicFileMenu – zobrazení grafické nabídky.

3.1.1 CiscoIPPhoneText

Tento objekt zajistí zobrazení textové zprávy na IP telefonu. XSD schéma je na obrázku 3.1.

Objekt CiscoIPPPhoneText obsahuje následující informace:

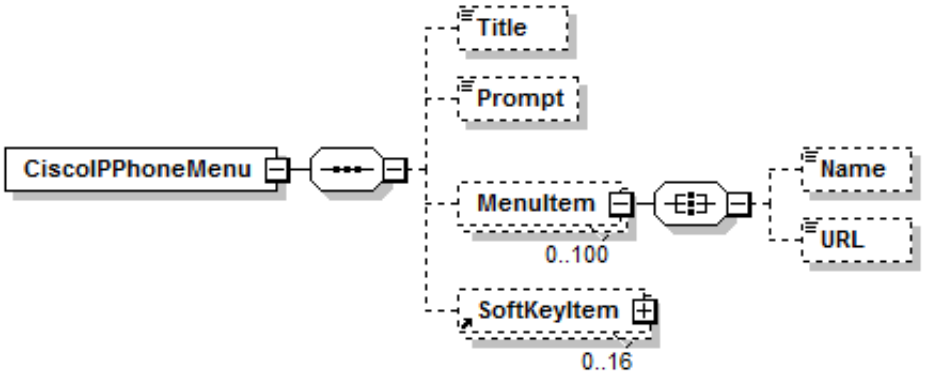
- Title – text, který se zobrazí v titulku okna.
- Prompt – text se zobrazí v předposledním řádku (označený jako řádek výzvy).
- Text – neformátovaný text, který se zobrazí v aplikační části displeje.



Obrázek 3.1: Schéma objektu CiscoIPPhoneText

3.1.2 CiscoIPPhoneMenu

Objekt zobrazí na IP telefonu nabídku, která může mít maximálně 100 položek. XSD schéma je uvedeno na obrázku 3.2.



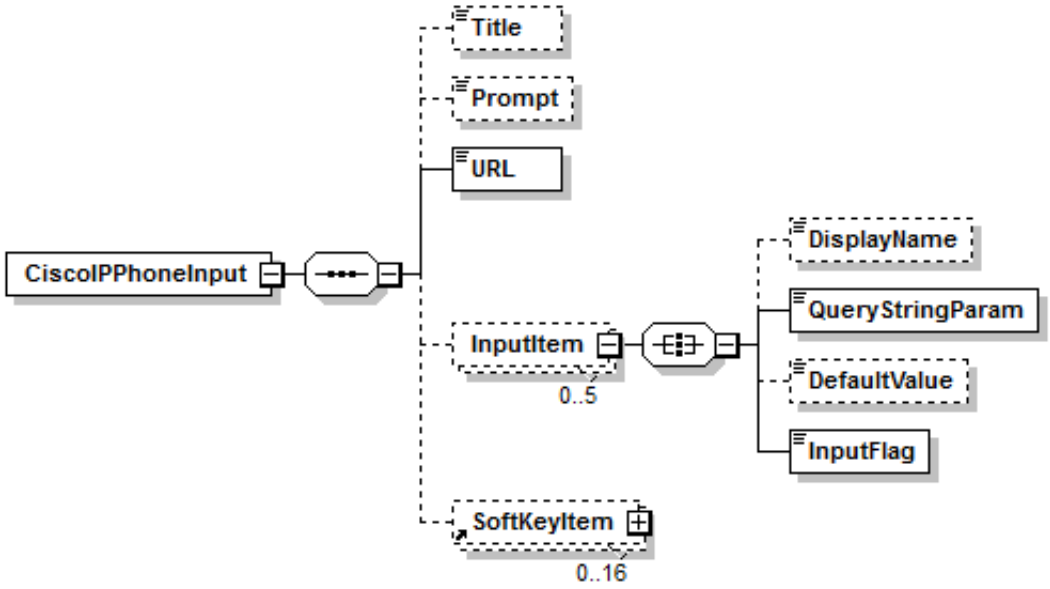
Obrázek 3.2: Schéma objektu CiscoIPPhoneMenu

3.1.3 CiscoIPPhoneInput

Pomocí tohoto objektu jsou vyžádána od uživatele potřebná data v předem definovaném formátu (ACSII text, telefonní číslo, číslo) pro další zpracování. Uživatel může data po dokončení zadávání odeslat na server pomocí programového tlačítka SoftKey. Vložená data jsou odeslána metodou GET na definovanou URL adresu. XSD schéma je na obrázku 3.3.

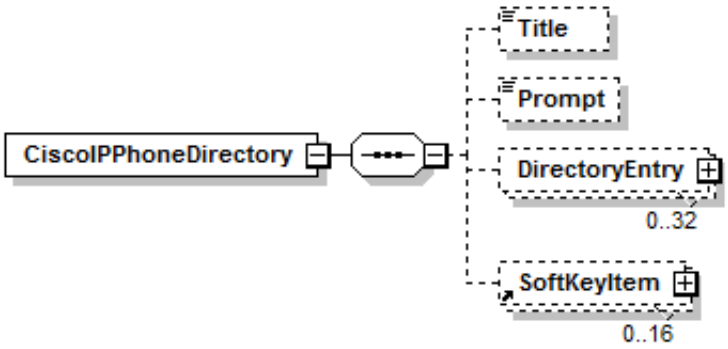
3.1.4 CiscoIPPhoneDirectory

Objekt zobrazí telefonní seznam na displeji IP telefonu. Telefonní seznam se zobrazuje velmi podobně jako nabídka. Hlavní rozdíl je v programových tlačítkách. V případě adresáře lze použít tlačítka pro vytočení vybraného telefonního čísla a tlačítko pro úpravu čísla před



Obrázek 3.3: Schéma objektu CiscoIPPhoneInput

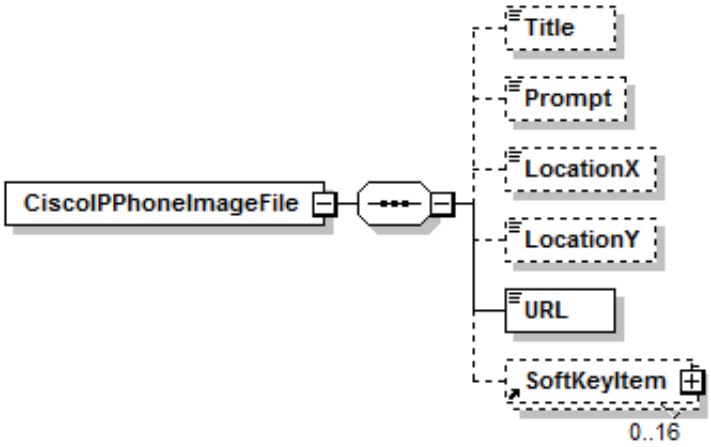
vytočením. Telefonní seznam je limitovaný maximálním počtem 32 položek. XSD schéma je na obrázku 3.4.



Obrázek 3.4: Schéma objektu CiscoIPPhoneDirectory

3.1.5 CiscoIPPhoneImageFile

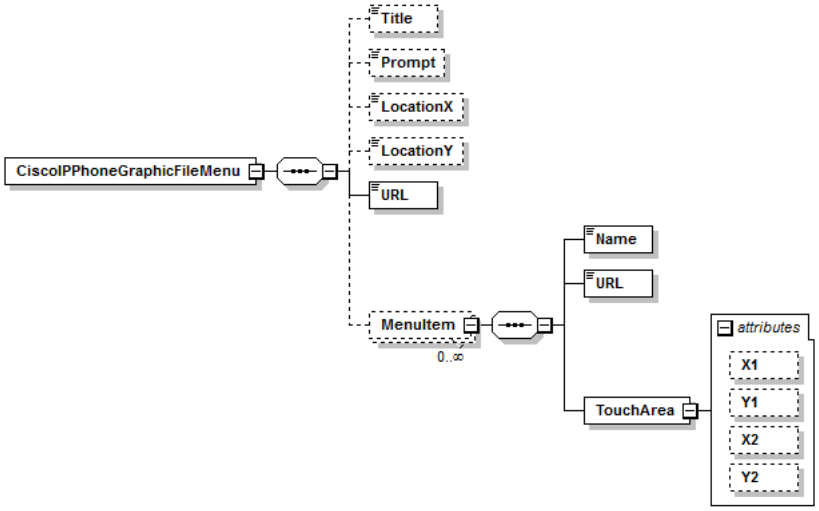
Cisco IP telefon umí zobrazit i obrázek ve formátu PNG, který může mít maximální rozměr 298x168 bodů a maximálně 12-ti bitovou barvu. Obrázek se stahuje z URL adresy a server musí vracet správnou MIME informaci – MIME Content-Type = image/png. XSD schéma je na obrázku 3.5.



Obrázek 3.5: Schéma objektu CiscoIPPhoneImage

3.1.6 CiscoIPPhoneGraphicFileMenu

Tento objekt zobrazí grafickou nabídku. To znamená, že se zobrazí PNG obrázek na displeji telefonu. K obrázku jsou definovány zóny a pokud uživatel zmáčkne displej v této zóně, zpracuje se daná akce. Tento objekt využívá dotykový displej telefonu Cisco 7970 a umožňuje plné využití tohoto telefonu. XSD schéma je na obrázku 3.6.

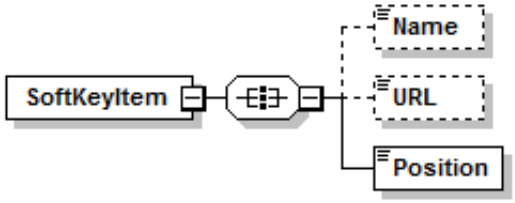


Obrázek 3.6: Schéma objektu CiscoIPPhoneGraphicMenu

3.1.7 SoftKey

U většiny objektů lze definovat uživatelská programová tlačítka, která se zobrazují ve spodní části displeje. Těchto programových tlačítek může být definováno až 16. Na fyzickém tele-

fonu je pod displejem 4 nebo 5 tlačítek, která odpovídají programovým tlačítkům na displeji. V případě, že je definováno více programových než fyzických tlačítek, jsou programová tlačítka rozdělena na stránky a pomocí posledního fyzického tlačítka lze v těchto stránkách listovat. XSD schéma je vyobrazeno na obrázku 3.7.



Obrázek 3.7: SoftKey schéma

3.1.7.1 SoftKey URI

Druhou skupinu interních URI tvoří adresy specifické pro určité aplikační objekty.

Seznam interních URI:

- Back – návrat k předchozí obrazovce.
- Cancel – zrušení zadávání dat.
- Exit – ukončení aplikace, což ve skutečnosti znamená zavření aktuálního okna.
- Next – přechod na další stránku.
- Select – výběr položky z nabídky.
- Submit – odeslání zadaných dat na server.
- Update – znovu načtení aplikace.
- Dial – vytočení daného kontaktu.
- EditDial – editace před vytočením.
- « – smazání jednoho znaku vlevo od kurzoru.

3.2 Analýza

Funkční testy u telefonních aplikací můžeme zúžit na testy grafického rozhraní. To je pro telefony definováno pomocí XML objektů - viz kapitola 3.1. Cílem práce je tedy vytvořit sadu funkcí, pomocí kterých je možné jednoduchým způsobem testovat grafické rozhraní zobrazované na telefonech.

Všechny funkce jsou rozděleny do dvou základních skupin. V první skupině jsou výkonné funkce, pomocí kterých se ovládá IP telefon. Druhou skupinu tvoří funkce, které ověřují, zda

jsou zobrazované informace na telefonu v pořádku. Ověřovací funkce jsou ještě rozděleny na dvě podskupiny, které se liší prefixem. Tento prefix definuje odlišné chování v případě, že provedený test nevyhovuje podmínce. Funkce s prefixem ***Validate*** provedou odpovídající kontrolu podle konkrétní funkce. Pokud dojde k neshodě s definovanou podmínkou, tato informace je zobrazena ve výsledku testu, ale testovací scénář pokračuje dál ve zpracování ostatních testovacích funkcí. Pokud ale podmínka u funkcí s prefixem ***Assert*** není splněna, je vykonávání testovacího skriptu ihned ukončeno. Všechny testovací funkce jsou navrženy s oběma prefixy.

Při testování telefonní aplikace je nutné specifikovat objekt nebo prvek, u kterého potřebujeme ověřit jeho existenci nebo hodnotu, kterou obsahuje. K řešení lze přistoupit dvěma způsoby. U prvního způsobu se jedná o vytvoření velké sady funkcí, které budou testovat jednotlivé prvky. Tím se uživatel odstíní od interní reprezentace dat, v mém případě reprezentace grafických objektů. Druhý přístup vychází právě ze znalosti interních dat, a proto není potřeba vytvářet velké množství specifických funkcí. Místo toho se pouze specifikuje, kterého prvku se test týká. Protože testovací skripty budou vytvářet lidé znalí interní reprezentace dat, zvolil jsem právě tento přístup. Navrhl jsem menší množství univerzálních funkcí. Jedním ze způsobů přístupu k datům v XML dokumentech je i dotazovací jazyk XPath, který je pro tuto aplikace zcela vyhovující. Jazyk XPath je implementovaný i v prostředí Microsoft .NET Framework. Verze 1.0, která je implementována od společnosti Microsoft, je naprosto dostačující a vyhovující. V případě potřeby je možné v budoucnu jazyk XPath nahradit novější verzí. Existují projekty třetích stran, které nabízejí verzi 2.0 nebo 3.0.

Požadavky na testování grafického rozhraní lze popsat těmito pravidly:

- Test, zda se zobrazuje správný typ obrazovky - text, obrázek, menu, apod.
- Testování zobrazovaných dat - například položky v menu.
- Test ovládacích tlačítek - SoftKey.

Obrazovka IP telefonu je definovaná konkrétním objektem v XML dokumentu. Tento objekt odpovídá názvu kořenového elementu. První funkce tedy slouží k otestování, zda XML dokument obsahuje definovaný element. Stejnou funkci lze využít například i pro testování jednotlivých položek v menu.

U testování obsahu je možné se omezit na práci s textovými řetězci. U nich můžeme testovat, zda jsou řetězce stejné a nebo zda je řetězec podřetězcem jiného. Stejné funkce jsem využil i pro definici testovacích funkcí. Funkce jsou navíc doplněny o parametr, zda porovnání textových řetězců bude rozlišovat mezi malými a velkými znaky.

Všechny funkce jsou ve variantě s prefixem ***Assert*** i ***Validate*** a navíc i v inverzní variantě. Testuje se tedy, zda prvek neexistuje, není shodný s jiným řetězcem nebo prvek neobsahuje daný podřetězec.

Pro zjednodušení práce s ovládacími tlačítky ***SoftKey*** jsou navrženy další funkce. Funkce je sice možné pomocí výše navržených funkcí a jazyka XPath plně nahradit, jsou ale definované pro zjednodušení práce při definování testovacích scénářů. V praxi je například častý požadavek na ověření, že tlačítko pro ukončení aplikace je na telefonu vždy na stejné pozici. Proto by se musela velmi často opakovat definice XPath cesty nejen v různých částech testovacího scénáře, ale i v různých scénářích.

3.2.1 Skriptovací jazyk

Na začátku jsem se snažil zjistit, jaké existují skriptovací jazyky a jaké mají vlastnosti. V dnešní době existuje velké množství skriptovacích jazyků, a proto jsem musel vydefinovat požadavky, kterými lze buď některý jazyk vybrat, nebo vyloučit všechny již existující jazyky.

Požadavky pro výběr vhodného skriptovacího jazyka:

- Jazyk musí být integrovatelný do prostředí Microsoft .NET Framework.
- Možnost definice funkcí a procedur.
- Musí existovat interpretr.
- Výhodou je podpora objektového programování.
- Výhodou je podpora různých platforem.

Těmto definovaným požadavkům vyhovuje více skriptovacích jazyků, například Python, Ruby, PHP, Lua, Tcl nebo Perl. Vytvoření speciálního skriptovacího jazyka je proto zcela zbytečné a je výhodnější využít některý z existujících jazyků. Po dlouhém studování detailů u jednotlivých jazyků jsem se rozhodl, že využiji skriptovací jazyk Python. Open-Source projekt IronPython nabízí možnost integrovat jazyk Python do prostředí Microsoft .NET Framework a pomocí projektu Mono je vše přenositelné i do operačního systému Linux, což dává možnost velké přenositelnosti, ale také univerzálnost celého řešení.

3.2.1.1 Volání .NET kódu z Python skriptu

Kód z prostředí Microsoft .NET je přeložen do souborů, které se nazývají ***Assembly***. Tyto soubory mají příponu EXE nebo DLL. Soubory obsahují přeložený program v bajt-kódu (byte-code). IronPython dokáže assembly v přenositelném kódu plně integrovat. K integraci slouží knihovna ***clr*** a funkce ***AddReference***. Po použití následujících dvou příkazů lze volat funkce z připojené assembly, které jsou psané v prostředí Microsoft .NET - například v jazyce Microsoft C#:

```
import clr
clr.AddReference("System.Xml")
```

3.2.1.2 Volání Python skriptu z .NET aplikace

IronPython má integrovaný syntaktický analyzátor (parser) a překladač (compiler). IronPython k tomu využívá knihoven dodávaných společností Microsoft - Microsoft.Scripting.

Na začátku je nutné vytvořit instanci Python prostředí voláním statické metody ***Python.CreateEngine*** definované v knihovnách IronPython. Tím se získá instance třídy ***ScriptEngine***. Následně je nutné rozšířit seznam adresářů, ve kterých se mají hledat rozšiřující knihovny a assembly pomocí metod ***GetSearchPaths*** a ***SetSearchPaths***.

Spuštění Python skriptu je možné provést dvěma způsoby. Prvním způsobem je volání metody ***Execute*** definované ve třídě ***ScriptEngine***. Metoda nejprve provede syntaktickou kontrolu následovanou překladem. Po úspěšném překladu je skript spuštěn. Opakované

spouštění stejného skripty provádí vždy opakovaně i syntaktickou kontrolu a překlad. Proto existuje i druhý způsob. Načtení Python skriptu a jeho překlad. Výsledek překladu je uložen ve speciální třídě ***CompiledCode***. Takto připravený kód lze kdykoliv spustit voláním metody ***Execute***.

Velmi důležitou částí je také sdílení proměnných mezi volajícím a volaným prostředím. To je důležité pro zajištění předávání vstupních parametrů do spouštěného Python skriptu, ale také pro předávání výsledku ze skriptu zpět do volajícího programu. Všechny vstupní i běhové proměnné Python skriptu jsou uloženy v instanci objektu třídy ***ScriptScope***. Jedná se vlastně o kolekci, ve které jsou uložené všechny proměnné skriptu. Jazyk Python nedefinuje typ pro jednotlivé proměnné, takže z pohledu prostředí Microsoft .NET jsou všechny proměnné typu ***object***. Při spouštění skriptu se předává metodě ***Execute*** i odkaz na instanci třídy ***ScriptScope*** se všemi vstupními proměnnými. Tato instance může být sdílena mezi různými spouštěnými skripty, což nabízí velkou univerzálnost celému řešení. Pro nastavení a čtení proměnných se využívá metod ***GetVariable*** a ***SetVariable*** třídy ***ScriptScope***.

Příklad volání Python skriptu:

```
scriptEngine = Python.CreateEngine();

paths = scriptEngine.GetSearchPaths();
paths.Add("c:\\test");
scriptEngine.SetSearchPaths(paths);

scriptScope = scriptEngine.CreateScope();
scriptScope.SetVariable("Variable", value);

compiledCode = scriptEngine.CreateScriptSourceFromString(script, SourceCodeKind.Statements).Com
compiledCode.Execute(scriptScope)

results = scriptScope.GetVariable("Variable");
```

3.2.2 Testovací funkce

Na základě analýzy jsem sestavil sadu testovacích funkcí. V tabulce 3.1 je výčet funkcí, které provádějí kontrolu obsahu zobrazovaných dat. Hned v následující tabulce 3.2 jsou funkce pro kontrolu ovládacích tlačítek SoftKey.

3.2.3 Funkce pro ovládání telefonu

Testovanou aplikaci je nutné také ovládat. V případě telefonních aplikací to znamená mít možnost zmáčknout ovládací tlačítka, nebo v případě dotykového displeje i vytvoření akce dotykem displeje. Dále se jedná i o možnost zadávání vstupních dat do formulářů.

V případě ovládacích tlačítek jsem navrhl tři samostatné funkce:

1. Funkce umožňující zmáčknutí kurzorových tlačítek.
2. Funkce ovládající tlačítka pod displejem telefonu - SoftKey.

AssertItemExists	Testuje, zda v XML existuje element s názvem. V případě chyby je test ihned ukončen.
ValidateItemExists	Testuje, zda v XML existuje element s názvem.
AssertItemEquals	Testuje, zda je hodnota elementu shodná s parametrem. V případě chyby je test ihned ukončen.
ValidateItemEquals	Testuje, zda je hodnota elementu shodná s parametrem.
AssertItemContains	Testuje, zda je parametr podřetězec hodnoty v elementu. V případě chyby je test ihned ukončen.
ValidateItemContains	Testuje, zda je parametr podřetězec hodnoty v elementu.
AssertItemNotExists	Testuje, zda v XML neexistuje element s názvem. V případě chyby je test ihned ukončen.
ValidateItemNotExists	Testuje, zda v XML neexistuje element s názvem.
AssertItemNotEquals	Testuje, zda hodnota elementu není shodná s parametrem. Element musí v dokumentu existovat. V případě chyby je test ihned ukončen.
ValidateItemNotEquals	Testuje, zda hodnota elementu není shodná s parametrem. Element musí v dokumentu existovat.
AssertItemNotContains	Testuje, zda hodnota v elementu neobsahuje parametr jako podřetězec. Element musí v dokumentu existovat. V případě chyby je test ihned ukončen.
ValidateItemNotContains	Testuje, zda hodnota v elementu neobsahuje parametr jako podřetězec. Element musí v dokumentu existovat.

Tabulka 3.1: Seznam testovacích funkcí

3. Funkce ovládající tlačítka číselníku, tedy tlačítka 0-9 a také tlačítko s hvězdičkou a křížkem.

3.2.4 Detailní popis navržených funkcí a proměnných

Tato kapitola popisuje detailně všechny navržené funkce a jejich parametry.

3.2.4.1 Výsledek testovacího skriptu

Každá testovací funkce uloží svůj výsledek do kolekce. Zde jsou postupně doplněny výsledky všech funkcí. Tato kolekce je typu ***List***. Obsahuje prvky třídy ***StepStatus***. Každý prvek tak obsahuje dvě informace - stav kroku/testu a zprávu. Zpráva je uživatelský text, který může být definován jako parametr u všech testovacích funkcí. Slouží primárně k popisu konkrétního testu a k identifikaci chyby v případě, že je testovací podmínka vyhodnocena jako ***nepravda***. Informace o výsledku testu je definovaná výčtovým typem ***Status***. Ten definuje tři možné výsledky - Success, Error, Failed. Stav ***Success*** znamená, že testovaná podmínka vyhověla. Další stav ***Error*** odpovídá stavu, ve kterém byla podmínka vyhodnocena jako nepravda a testovací funkce začínala prefixem ***Validate***. Testování telefonní aplikace ale pokračuje dále bez ohledu na stav vyhodnocení podmínky. Oproti tomu stav ***Failed*** nastane

AssertSoftKeyExitOnPosition	Testuje, zda je tlačítko ***Exit*** na definované pozici. V případě chyby je test ihned ukončen.
ValidateSoftKeyExitOnPosition	Testuje, zda je tlačítko ***Exit*** na definované pozici.
AssertSoftKeySubmitOnPosition	Testuje, zda je tlačítko ***Submit*** na definované pozici. V případě chyby je test ihned ukončen.
ValidateSoftKeySubmitOnPosition	Testuje, zda je tlačítko ***Submit*** na definované pozici.
AssertSoftKeySelectOnPosition	Testuje, zda je tlačítko ***Select*** na definované pozici. V případě chyby je test ihned ukončen.
ValidateSoftKeySelectOnPosition	Testuje, zda je tlačítko ***Select*** na definované pozici.
AssertSoftKeyCancelOnPosition	Testuje, zda je tlačítko ***Cancel*** na definované pozici. V případě chyby je test ihned ukončen.
ValidateSoftKeyCancelOnPosition	Testuje, zda je tlačítko ***Cancel*** na definované pozici.
AssertSoftKeyDialOnPosition	Testuje, zda je tlačítko ***Dial*** na definované pozici. V případě chyby je test ihned ukončen.
ValidateSoftKeyDialOnPosition	Testuje, zda je tlačítko ***Dial*** na definované pozici.
AssertSoftKeyDialEditOnPosition	Testuje, zda je tlačítko ***DialExit*** na definované pozici. V případě chyby je test ihned ukončen.
ValidateSoftKeyDialEditOnPosition	Testuje, zda je tlačítko ***DialExit*** na definované pozici.

Tabulka 3.2: Seznam funkcí pro testování SoftKey tlačítek

PressSoftKey	Zmáčknutí SoftKey tlačítek.
PressNumKey	Zmáčknutí tlačítek číselníku.
PressCursorKey	Zmáčknutí kurzorových kláves.
PressTouchScreen	Zmáčknutí dotykového displeje.
EnterData	Vloží příslušnou hodnotu do vstupního formuláře.

Tabulka 3.3: Seznam funkcí pro ovládání telefonu

také v případě vyhodnocení podmínky jako nepravda, testovací funkce však začíná prefixem ***Assert***. Testování je v tomto případě ukončeno.

Druhý parametr funkce je nepovinný. Pokud není uveden, je automaticky doplněna přednastavená hodnota. Parametr se nazývá ***Message*** a definuje zprávu, která se automaticky vloží do kolekce stavů a to pro všechny jejich typy. Text zprávy není nijak omezen, ale účelem je krátký a jednoznačně identifikující krok (testovací podmínka). Delší text je možné vložit do testovacího skriptu pomocí poznámky, která může být definovaná na stejném řádku za vlastním příkazem, nebo může být definována na samostatném řádku, popřípadě i jako víceřádková poznámka. Tyto poznámky jsou určeny pouze pro popis uvnitř testovacího skriptu a při spuštění testu nejsou nijak zpracovávány.

3.2.4.2 Funkce Assert a Validate

Funkce Assert a Validate jsou obecné funkce. Neprovádí aktivně žádnou kontrolu zobrazovaných dat v emulátoru telefonu. Primárně jsou využívány ostatními navrženými funkcemi. Lze je ale volat i z Python skriptu pro vyhodnocení libovolné jiné podmínky. Obě funkce mají jeden povinný parametr typu boolean. Pokud má parametr hodnotu ***TRUE*** (pravda), je i tento test vyhodnocen jako úspěšný a do výstupní kolekce s výsledkem je uložen tento stav. V případě, že parametr má hodnotu ***FALSE*** (nepravda), je test vyhodnocen jako neúspěšný. V případe funkce ***Validate*** je tato informace uložena do výstupní kolekce a testování pokračuje dalšími testy. U funkce ***Assert*** je informace opět uložena do výstupní kolekce. Do výstupní kolekce je ale uložena hodnota kroku jako ***FAILED*** a systém vygeneruje výjimku ***AssertException***. Tím je testovací skript ihned ukončen.

3.2.4.3 Funkce typu ItemExists a ItemNotExists

Jako u většiny ostatních typů testovacích funkcí existují celkem čtyři signatury pro typ ***ItemExists*** a čtyři pro ***ItemNotExists***. Dvě varianty s prefixem ***Assert*** a dvě s prefixem ***Validate***. Dále dvě funkce očekávají parametr ***Message*** a dvě mají tento parametr přednastaven - viz. předchozí kapitola 3.2.4.1.

Sémantika navržených funkcí:

```
public static void AssertItemExists(String xpath);
public static void AssertItemExists(String xpath, String message);
public static void ValidateItemExists(String xpath);
public static void ValidateItemExists(String xpath, String message);
public static void AssertItemNotExists(String xpath);
public static void AssertItemNotExists(String xpath, String message);
public static void ValidateItemNotExists(String xpath);
public static void ValidateItemNotExists(String xpath, String message);
```

Parametr ***xpath*** definuje cestu k elementu, u kterého chceme ověřit existenci v XML dokumentu. Polovina funkcí, která obsahuje slovo ***Not***, vyhodnocuje testovací podmínku inverzně. Pokud element s cestou definovanou parametrem ***xpath*** neexistuje, je podmínka vyhodnocena jako hodnota ***TRUE***.

3.2.4.4 Funkce typu ItemEquals a ItemNotEquals

Tyto funkce ověřují, zda element nebo atribut definovaný ***XPath*** cestou, je shodný (nebo různý v případě funkcí s ***Not***) s řetězcem v parametru ***value***. Dalším parametrem je nepovinný parametr ***caseSensitive***. Ten určuje, zda porovnání řetězců bude rozlišovat mezi malými a velkými písmeny. Pokud tento parametr není uveden, je přednastavena hodnota ***TRUE***, což znamená, že systém rozlišuje malá a velká písmena. Systém vždy rozlišuje mezi písmeny s diakritikou a bez ní. Toto chování není možné parametrem ovlivnit. V praxi je totiž potřeba odhalit i možné překlepy, které vzniknou tím, že programátor píše texty bez diakritiky. Proto jsem neuvažoval o zavedení varianty nerozlišování znaků s diakritikou a bez ní.

Sémantika funkcí:

```
public static void AssertItemEquals(String xpath, String value,
        Boolean caseSensitive = true);
public static void AssertItemEquals(String xpath, String value,
        String message, Boolean caseSensitive = true);
public static void ValidateItemEquals(String xpath, String value,
        Boolean caseSensitive = true);
public static void ValidateItemEquals(String xpath, String value,
        String message, Boolean caseSensitive = true);
public static void AssertItemNotEquals(String xpath, String value,
        Boolean caseSensitive = true);
public static void AssertItemNotEquals(String xpath, String value,
        String message, Boolean caseSensitive = true);
public static void ValidateItemNotEquals(String xpath, String value,
        Boolean caseSensitive = true);
public static void ValidateItemNotEquals(String xpath, String value,
        String message, Boolean caseSensitive = true);
```

3.2.4.5 Funkce typu ItemContains a ItemNotContains

Rozdíl oproti funkcím definovaných v kapitole 3.2.4.4 je pouze v části porovnání textového řetězce. Zde se ověřuje, zda řetězec v parametru ***value*** je podřetězcem v elementu nebo atributu definovaným pomocí XPath cesty v parametru ***xpath***.

Sémantika funkcí:

```
public static void AssertItemContains(String xpath, String value,
        Boolean caseSensitive = true);
public static void AssertItemContains(String xpath, String value,
        String message, Boolean caseSensitive = true);
public static void ValidateItemContains(String xpath, String value,
        Boolean caseSensitive = true);
public static void ValidateItemContains(String xpath, String value,
        String message, Boolean caseSensitive = true);
public static void AssertItemNotContains(String xpath, String value,
        Boolean caseSensitive = true);
public static void AssertItemNotContains(String xpath, String value,
        String message, Boolean caseSensitive = true);
public static void ValidateItemNotContains(String xpath, String value,
        Boolean caseSensitive = true);
public static void ValidateItemNotContains(String xpath, String value,
        String message, Boolean caseSensitive = true);
```

3.2.4.6 Funkce typu SoftKeyExists a SoftKeyNotExists

Tato sada funkcí je navržena k testování, zda definice ovládacího tlačítka SoftKey odpovídá požadavku v telefonní aplikaci. Každá metoda má parametr ***index***, který odpovídá pořadí

tlačítka. Pořadí je počítáno od čísla 1 a odpovídá pořadí na displeji IP telefonu. Dalším parametrem je ***name***, který určuje zobrazovaný text tlačítka. A třetím povinným parametrem je ***url***. Tento parametr slouží k testování URL nebo URI adresy, které je k tlačítku přiřazeno. Test je vyhodnocen jako hodnota ***TRUE***, pokud existuje tlačítko zobrazené na definované pozici a zároveň má odpovídající název a URL. V případě inverzních funkcí musí být vždy splněna podmínka, že tlačítko na zobrazované pozici existuje a následně se vyhodnotí vlastní podmínka testující název a URL adresu.

Sémantika funkcí:

```
public static void AssertSoftKeyExists(int index, String name,
        String url, Boolean caseSensitive = true);
public static void AssertSoftKeyExists(int index, String name,
        String url, String message, Boolean caseSensitive = true);
public static void ValidateSoftKeyExists(int index, String name,
        String url, Boolean caseSensitive = true);
public static void ValidateSoftKeyExists(int index, String name,
        String url, String message, Boolean caseSensitive = true);
public static Boolean CheckSoftKeyNotExists(int index, String name,
        String url, Boolean caseSensitive);
public static void AssertSoftKeyNotExists(int index, String name,
        String url, Boolean caseSensitive = true);
public static void AssertSoftKeyNotExists(int index, String name,
        String url, String message, Boolean caseSensitive = true);
public static void ValidateSoftKeyNotExists(int index, String name,
        String url, Boolean caseSensitive = true);
public static void ValidateSoftKeyNotExists(int index, String name,
        String url, String message, Boolean caseSensitive = true);
```

3.2.4.7 Funkce testující pozici SoftKey tlačítek - SoftKeyOnPosition

Samostatně jsem vytvořil sadu funkcí, které testují pouze polohu tlačítek s konkrétní funkcí, což je vhodné, pokud se jedná o tlačítka s interní URI adresou a neprovádí se nastavení názvu tlačítka. IP telefon v tomto případě zobrazuje interní název tlačítka, který je dán jazykovým prostředím definovaným na telefonu. To znamená, že pokud je telefon nastaven do českého prostředí, zobrazuje tlačítka s českým textem. Pokud ale telefon přepneme do anglického prostředí, je text SoftKey tlačítka zobrazen v angličtině. Následující funkce tak nekontrolují zobrazovaný text, ale pouze zkontrolují, zda SoftKey tlačítko na pozici s indexem definovaným parametrem ***index*** má nastavenou URI adresu dle dané testovací funkce.

Sémantika funkcí:

```
public static void AssertSoftKeyExitOnPosition(int index);
public static void AssertSoftKeyExitOnPosition(int index, String message);
public static void ValidateSoftKeyExitOnPosition(int index);
public static void ValidateSoftKeyExitOnPosition(int index, String message);
public static void AssertSoftKeySubmitOnPosition(int index);
public static void AssertSoftKeySubmitOnPosition(int index, String message);
```

```
public static void ValidateSoftKeySubmitOnPosition(int index);
public static void ValidateSoftKeySubmitOnPosition(int index, String message);
public static void AssertSoftKeySelectOnPosition(int index);
public static void AssertSoftKeySelectOnPosition(int index, String message);
public static void ValidateSoftKeySelectOnPosition(int index);
public static void ValidateSoftKeySelectOnPosition(int index, String message);
public static void AssertSoftKeyCancelOnPosition(int index);
public static void AssertSoftKeyCancelOnPosition(int index, String message);
public static void ValidateSoftKeyCancelOnPosition(int index);
public static void ValidateSoftKeyCancelOnPosition(int index, String message);
public static void AssertSoftKeyDialOnPosition(int index);
public static void AssertSoftKeyDialOnPosition(int index, String message);
public static void ValidateSoftKeyDialOnPosition(int index);
public static void ValidateSoftKeyDialOnPosition(int index, String message);
public static void AssertSoftKeyDialEditOnPosition(int index);
public static void AssertSoftKeyDialEditOnPosition(int index, String message);
public static void ValidateSoftKeyDialEditOnPosition(int index);
public static void ValidateSoftKeyDialEditOnPosition(int index, String message);
```

3.2.4.8 Funkce pro ovládání telefonu - PressSoftKey

Funkce slouží k simulování zmáčknutí SoftKey tlačítka. Tím je následně spuštěna odpovídající akce v závislosti na aktuální hodnotě URL nebo URI hodnoty u tlačítka.

Sémantika funkcí:

```
public static void PressSoftKey(int index);
public static void PressSoftKey(int index, String message);
```

3.2.4.9 Funkce pro ovládání telefonu - PressNumKey

K ovládání číselníku na IP telefonu slouží funkce s názvem ***PressNumKey***. První parametr je výčtového typu a definuje klávesu, u které se má simulovat zmáčknutí.

Sémantika funkcí:

```
public static void PressNumKey(NumKeyType numKey);
public static void PressNumKey(NumKeyType numKey, String message);
```

3.2.4.10 Funkce pro ovládání telefonu - PressCursorKey

Kurzorové klávesy jsou ovládány pomocí funkce ***PressCursorKey***. První parametr opět definuje konkrétní klávesu.

Sémantika funkcí:

```
public static void PressCursorKey(CursorKeyType cursorKey);
public static void PressCursorKey(CursorKeyType cursorKey, String message);
```

3.2.4.11 Funkce pro ovládání telefonu - PressTouchScreen

Některé modely Cisco IP telefonů disponují i dotykovým displejem. Proto také testovací scénáře musí být schopny testovat tuto možnost ovládání aplikace. Dotyk na displej je identifikován svoji polohou. Pozice x=0, y=0 odpovídá levému hornímu rohu displeje. Souřadnice X je horizontální (vodorovná) a Y je vertikální (svislá).

Sémantika funkcí:

```
public static void PressTouchScreen(int x, int y);
public static void PressTouchScreen(int x, int y, String message);
```

3.2.4.12 Funkce pro ovládání telefonu - EnterData

U vstupních formulářů je nutné simulovat zadávání vstupní hodnoty. Emulátor IP telefonu zde neemuluje chování telefonu stejným způsobem, jak se chová IP telefon. Při zadávání hodnoty na Cisco IP telefonu se po zmáčknutí číselné klávesy zobrazí malá nabídka s výběrem písmen. Opakovaným zmáčknutím stejné klávesy se vybírá požadované písmeno z nabídky. Například pomocí číselné klávesy ***2*** lze napsat písmena A, B a C. Pokud má telefon nastaveno české prostředí, lze pomocí stejné klávesy napsat i písmena Á a Č. Pro napsání písmene ***C*** se tak musí třikrát rychle po sobě zmáčknout numerická klávesa ***2***. Emulátor tento způsob zadávání písmen nepodporuje a vkládá výsledný text do příslušného pole přímo tak, jak se píše na klávesnici. Proto je funkce pro vložení hodnoty do vstupních formulářů díky tomu jednodušší. Parametr ***index*** určuje pořadí položky v menu a parametr ***value*** obsahuje hodnotu, která se do položky vloží.

Toto zjednodušené vkládání hodnot do vstupních formulářů žádným způsobem neovlivňuje funkční testy telefonních aplikací. Není totiž důležité, jak se daná hodnota do formuláře dostala, ale důležité je, jak na danou hodnotu telefonní aplikace zareaguje. Reakci můžeme pomocí emulátoru IP telefonu a pomocí tohoto projektu řádně otestovat.

Sémantika funkce:

```
public static void EnterData(int index, String value);
```

3.3 Návrh aplikace

Navržená aplikace je rozdělena na tři samostatné části:

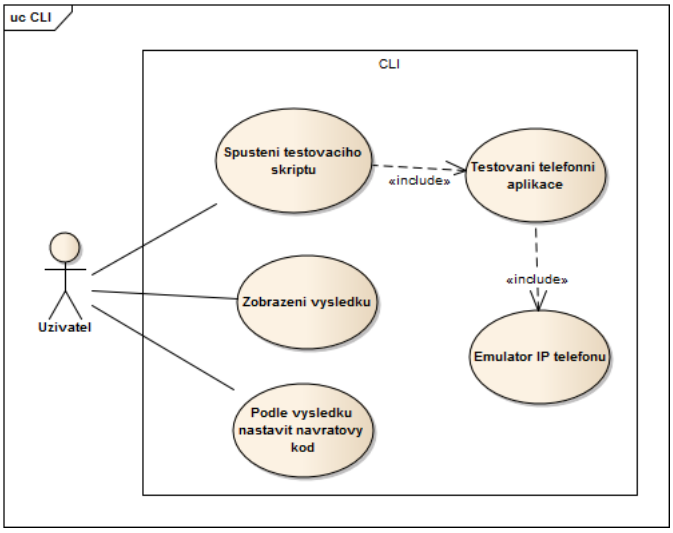
- Testovací aplikace s rozhraním CLI – aplikace spouštěná v příkazovém řádku.
- Testovací aplikace s rozhraním GUI – grafická verze aplikace s integrovaným jednoduchým textovým editorem.
- Grafický editor testovacího skriptu – grafický editor testovacích skriptů s integrovaným emulátorem IP telefonu.

Testovací aplikace s rozhraním CLI je primárně určena pro automatizované testování, které je integrovatelné do jiných systémů. Po dokončení testu aplikace vrátí tzv. návratový kód, který umožňuje volajícím aplikacím zjistit, zda testovací scénář doběhl v pořádku až do konce, nebo zda se vyskytl nějaký problém.

Návratové hodnoty:

- Hodnota 0 – testovací skript proběhl úspěšně a všechny kroky byly vyhodnoceny s hodnotou ***TRUE***.
- Hodnota 1 – jeden nebo více kroků typu ***Validate*** bylo vyhodnoceno jako ***FALSE***.
- Hodnota 2 – jeden krok typu ***Assert*** byl vyhodnocen jako ***FALSE***, a testovací skript byl předčasně ukončen.
- Hodnota 3 – při běhu aplikace došlo k neočekávané výjimce.

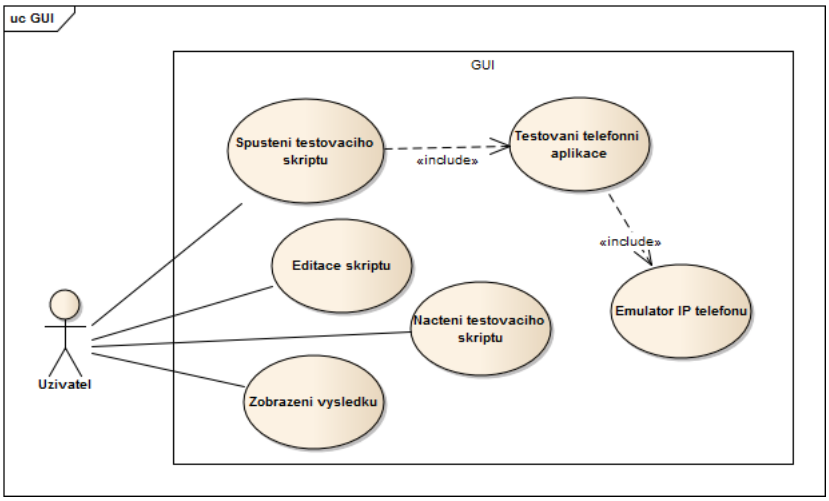
Definice požadavků je zobrazena v USE-CASE diagramu na obrázku 3.8.



Obrázek 3.8: Use-Case pro CLI verzi aplikace

Druhou aplikací je testovací program s grafickým rozhraním. Obsahuje jednoduchý textový editor a integrované zobrazení výsledku. Textový editor slouží k úpravě vstupního testovacího skriptu. Výstup z testování je zobrazen v pravé části okna. Každý krok je barevně označen podle svého stavu. Definice požadavků je opět ve formě USE-CASE diagramu na obrázku 3.9.

Třetí aplikací je grafický editor skriptů. Kumuluje vlastně tři aplikace - Phone Emulator, grafický editor skriptů a testovací program. Uživatel si může v aplikaci povolit automatické generování testovacího skriptu. Potom se při ovládání integrovaného emulátoru automaticky generují všechny testovací a ovládací kroky. Uživatel tak nemusí dlouze přemýšlet, které okno v emulátoru je právě zobrazeno a co je potřeba testovat. Uživatel se může soustředit na vlastní telefonní aplikaci a editor provede vše potřebné. Editor automaticky vytvoří pravidla



Obrázek 3.9: Use-Case pro grafickou verzi testovací aplikace

pro všechny ovládací prvky a uživatel může později tato pravidla a podmínky upřesnit. To výrazně zrychluje práci při generování testovacích scénářů. USE-CASE diagram s požadavky na editor je na obrázku 3.10.

3.3.1 Návrh tříd

Pro zdárné vytvoření aplikace je velmi důležitý správný návrh objektových tříd. V této kapitole popíši návrh pro ty nejdůležitější a klíčové třídy.

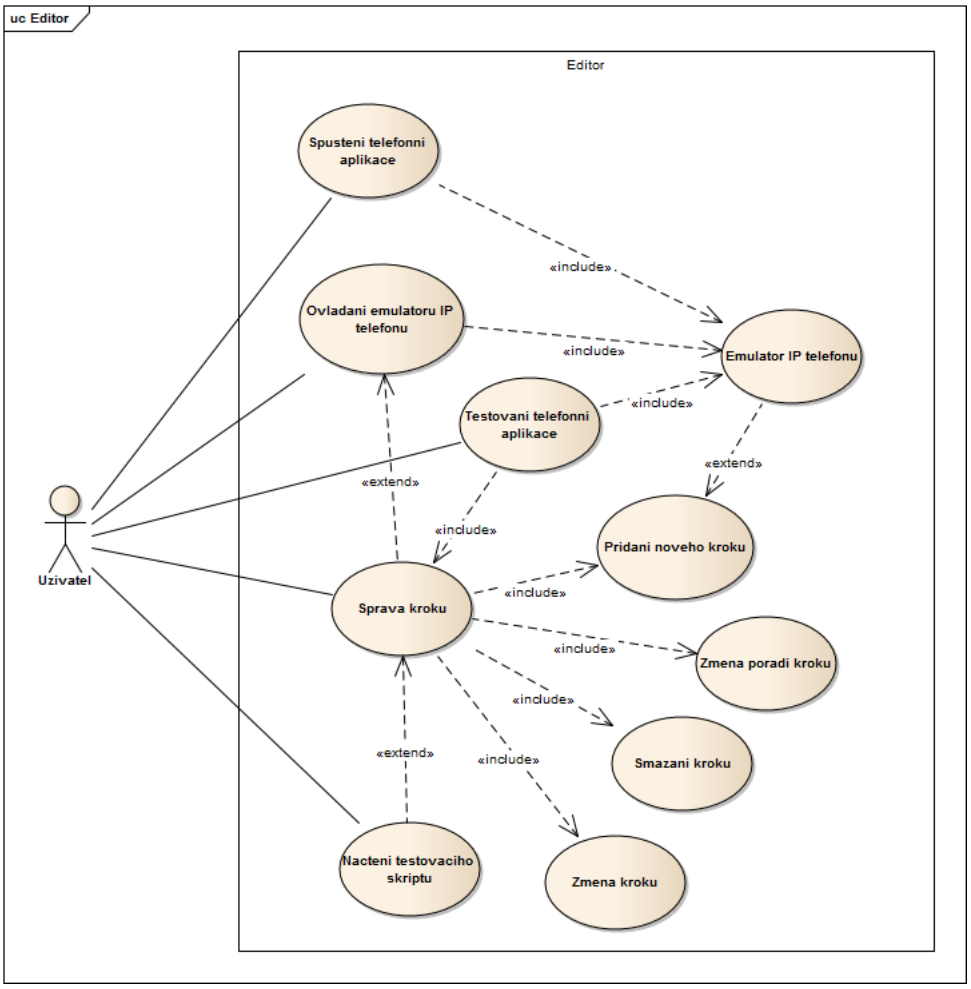
3.3.1.1 Třída s rozšiřujícími funkcemi pro Python skript

Jak již bylo ukázáno v kapitole 3.2.1.1, existuje možnost rozšíření Python skriptu o volání .NET metod. Aby toto volání bylo ještě zjednodušeno, jsou všechny metody třídy ***PhoneTestFunctions*** statické. To umožňuje metody této třídy volat pouze názvem metody a nemusí se opakovaně uvádět i název třídy nebo název instance třídy. Toto zkrácené volání je provedeno pomocí následujícího příkazu v Python skriptu:

```
from AutomatedTests.PhoneTestFunctions import *
```

Návrh třídy se všemi funkcemi je na obrázku 3.11. Výsledek jednotlivých testů (kroků) je postupně ukládán do kolekce typu ***List***. Každý krok vloží do kolekce svůj výsledek v objektu typu ***StepStatus***, ve kterém jsou definovány pouze dvě vlastnosti - jedná se o popis kroku ***Message*** a jeho stav ***Status***.

Ve stejném jmenném prostoru (NameSpace) je ještě jedna třída - ***TestEngine***, která provádí inicializaci knihovny ***IronPython***, překlad Python skriptů, jejich spuštění a převzetí výsledku testování z Python skriptu zpět do prostředí Microsoft .NET.

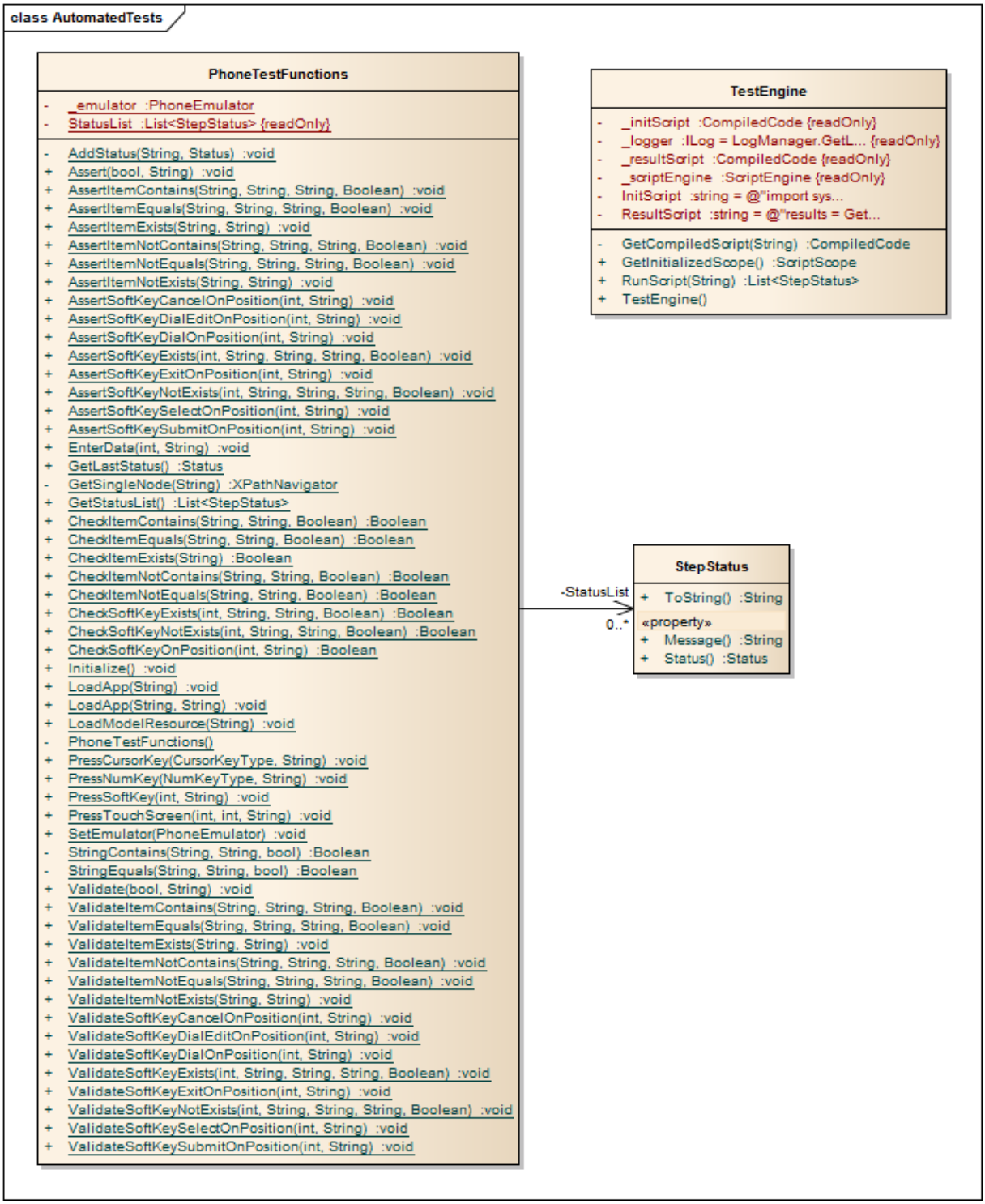


Obrázek 3.10: Use-Case pro editor testovacích skriptů

3.3.1.2 Třídy zajišťující grafické zobrazení

Pro grafický editor je navržena skupina tříd a grafických komponent, které zajišťují odpovídající zobrazení na obrazovce. Každý typ kroku má výsledně jednu svoji třídu. Ta zajišťuje zobrazení všech potřebných grafických komponent v okně editoru, pokud se daný krok přidává nebo se modifikuje. Zároveň se instance těchto tříd vkládají do kolekce kroků a jsou tedy využity i pro uschování všech potřebných informací daného kroku.

Třída reprezentuje rodiče pro všechny konkrétní grafické komponenty. Definuje pouze jednu vlastnost a to poznámku. Je zde definovaná virtuální metoda ***GetPythonCammand***, která vrací příkaz v syntaxi jazyka Python se všemi vyplněnými potřebnými parametry. Tuto metodu musí všechny podtřídy přetížit a vrátit správný formát příkazu. Další metodou je ***GetPythonCommandWithComment***. Tato metoda vrací stejně jako metoda ***GetPythonCammand*** příkaz v syntaxi jazyka Python a navíc je doplněna i řádkovou poznámkou. Poznámka je vložena pouze pokud ji má daný krok vyplněn. Poslední důležitou metodou je



Obrázek 3.11: Návrh tříd pro testování aplikací

Runtask. Pokud se totiž provádí spouštění testovacího skriptu přímo z editoru, nedochází k vygenerování Python kódu a jeho spuštění, ale místo toho je volaná metoda ***Runtask*** od každého kroku. To umožňuje krokování skriptu a průběžné zobrazování prováděných akcí a výsledku.

Základní třídou je tedy ***TaskControl***. Prvním potomkem je ***TaskValidateControl***. Zde je přidána vlastnost ***Message*** a z této třídy vychází všechny grafické objekty, které mohou přerušit testovací skript. Každá funkce má totiž popis - zprávu, která by měla jednoznačně identifikovat krok, který zapříčinil ukončení skriptu. Všechny testovací funkce mají celkem čtyři varianty. Liší se různým prefixem - viz. kapitola 3, a zda je očekávaný výsledek ***pravda*** nebo ***nepravda***. Protože jsou tyto funkce jinak totožné, je pro ně vytvořena pouze jedna grafická reprezentace. Z toho důvodu je navržen nový potomek s názvem ***TaskItemControl***, u které jsou dvě nové a důležité vlastnosti - ***TaskType*** a ***Inverse***. Tyto vlastnosti rozlišují mezi různými variantami testovací funkce.

Výše zmíněné třídy jsou pouze pomocné a abstraktní. Dále jsou popsány konkrétní třídy reprezentující konkrétní kroky/testy v testovacím skriptu. Pro funkce testující existenci elementu nebo atributu je to třída ***TaskItemExistsControl***. V ní je definovaná pouze vlastnost ***xpath***, která definuje cestu k hledanému prvku. Z ní je odvozená další třída - ***TaskItemCompareControl***. Ta je použita pro funkce, které testují obsah. Vlastní testovací podmínka je vybrána pomocí vlastnosti ***CompareType***. Jedná se o výčtový typ a může obsahovat dvě hodnoty: ***Equals*** a ***Contains***. Poslední dvě třídy slouží k definici kroků testujících programové tlačítko SoftKey. Jedná se o ***TaskCheckSoftKeyControl*** a ***TaskSoftKeyPositionControl***. Celé schéma je na obrázku 3.12.

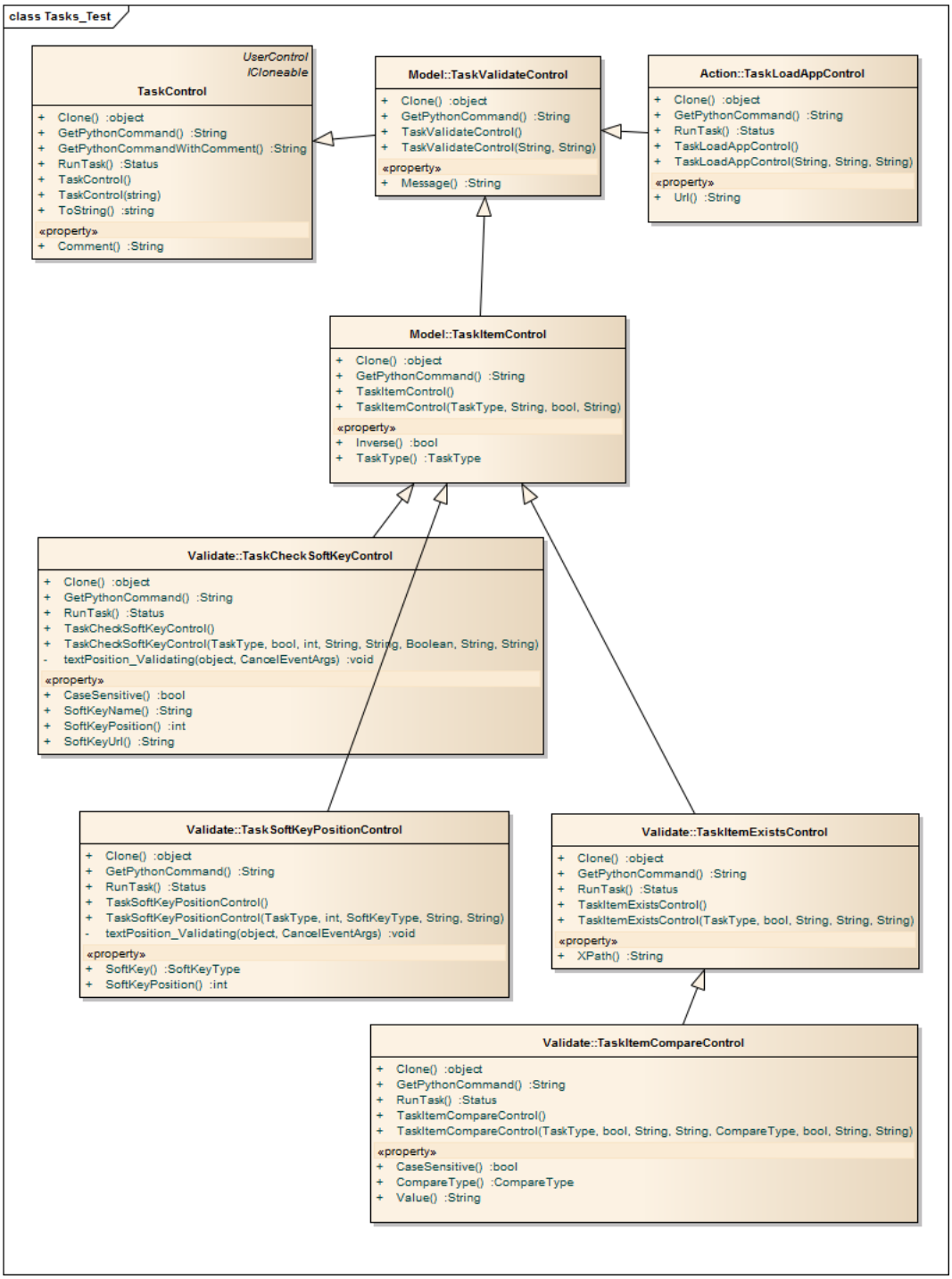
Další skupinou objektů, která vychází z třídy ***TaskControl***, jsou kroky, pomocí kterých se ovládá emulátor IP telefonu. Celkem je jich navrženo pět. Zajišťují ovládání pomocí číselníku telefonu a kurzorových kláves. Dále potom ovládání SoftKey tlačítek a simulování ovládání pomocí dotyku displeje. Poslední objekt provádí změnu hodnot ve vstupním formuláři.

3.3.2 Grafické rozhraní

3.3.2.1 Grafické rozhraní editoru

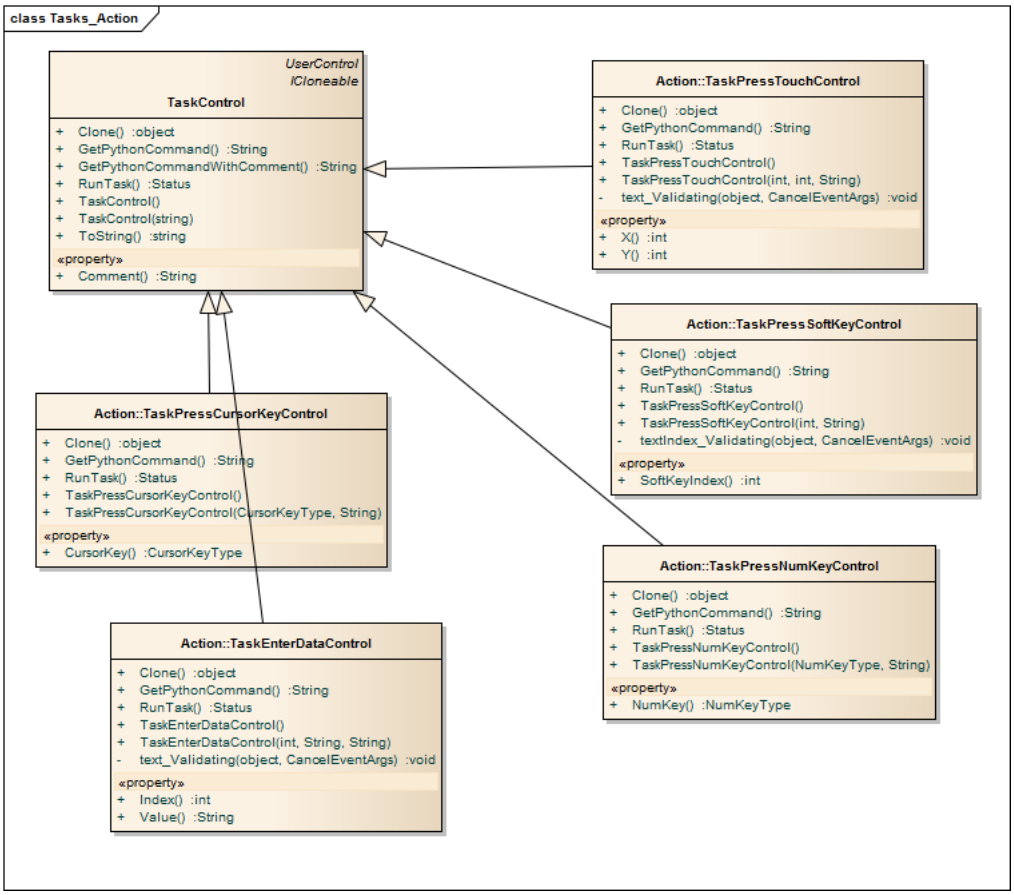
Grafické rozhraní editoru lze rozdělit na pět logických částí - viz. obrázek 3.14. Na obrázku je každá logická část označena číslem a obsah je detailně popsán v následujících odstavcích.

1. Emulátor IP telefonu se všemi potřebnými ovládacími prvky - SoftKey tlačítka, číselník a kurzorové klávesy.
2. Oblast pojmenovaná ***Options*** obsahuje více různých položek:
   - URL – URL adresa spouštěné telefonní aplikace. Jedná se vlastně o vstupní bod celého testu.
   - Tlačítko RUN – spustí telefonní aplikaci. URL adresa se předá emulátoru telefonu a provede se její načtení a spuštění.
   - Auto Create Tasks – po zaškrtnutí této volby bude editor při ovládání emulátoru automaticky generovat kroky. Generují se jak testovací kroky, tak i kroky ovládající emulátor telefonu.



Obrázek 3.12: Návrh tříd pro grafické zobrazení testů

- Tlačítko GENERATE – slouží k vygenerování testovacího Python skriptu z kroků, které jsou vidět v prostřední části obrazovky.



Obrázek 3.13: Návrh tříd pro grafické zobrazení akcí

- Tlačítko LOAD – načte existující skript do editoru. Pokud se provedly pokročilé úpravy v testovacím Python skriptu, nemusí se již podařit tento skript načíst do editoru. Editor totiž nedokáže zpracovat všechny příkazy jazyka Python.

3. Aktuální seznam kroků. V dolní části jsou také tlačítka pro vložení nového kroku, pro smazání vybraného kroku a dvě tlačítka pro změnu pořadí.

4. V této oblasti se dynamicky zobrazují prvky pro změnu obsahu vybraného kroku. Každý typ kroku má jiné ovládací prvky.

5. Pomocí tlačítka DEBUG a STOP lze spustit případně zastavit výsledný testovací skript.

V prostřední část aplikačního okna je zobrazen seznam kroků, které jsou aktuálně vloženy do testovacího skriptu. Každý krok je zobrazen na dvou řádcích. Na prvním řádku se zobrazuje text odpovídající vygenerovanému příkazu a na druhém řádku se zobrazuje poznámka, pokud ji má daný krok vyplněnou. Kroky jsou také barevně rozlišeny:

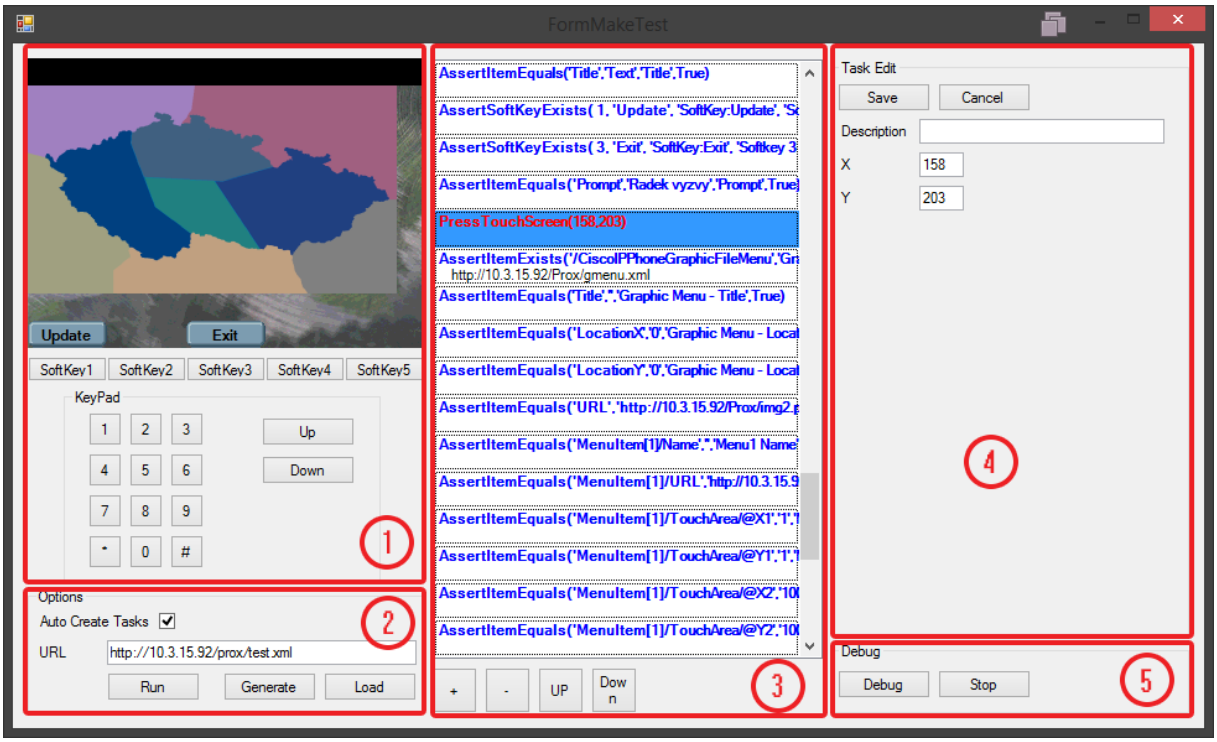
- Modrou barvu písma mají všechny kroky, které odpovídají testovacím funkcím.

- Červenou barvu písma mají kroky ovládací.
- Černou barvou se zobrazuje krok pro změnu hodnot ve vstupním formuláři.

Pod seznamem kroků jsou čtyři ovládací tlačítka, kterými lze přidat nový krok, smazat aktuálně vybraný krok a přesunout aktuálně vybraný krok nahoru nebo dolů v pořadí.

V oblasti označené číslicí 4 se zobrazují ovládací prvky pro změnu právě vybraného kroku. Pro každý typ kroku jsou zobrazeny odpovídající prvky, kterými lze měnit všechny jeho vlastnosti. Jsou zde i dvě fixní tlačítka. Tlačítkem ***Save*** lze uložit provedené změny u vybraného kroku a tlačítko ***Cancel*** se využije pro vrácení změn zpět na původně uložené hodnoty.

Editor umožňuje spouštět testovací skript. Po zmáčknutí tlačítka ***DEBUG*** začne editor provádět jednotlivé kroky testovacího skriptu. Provádění skriptu je uměle zpomaleno, což umožňuje sledovat chování testovacího skriptu přímo v Emulátoru IP telefonu. Tlačítkem ***STOP*** lze kdykoliv zastavit spouštěný skript. Spuštění aplikace je vždy od prvního kroku a není umožněno vracet provedené kroky zpět. Důvodem je, že jedna z částí telefonní aplikace běží na webovém serveru a na kterém nelze vzdáleně vrátit provedené kroky zpět.



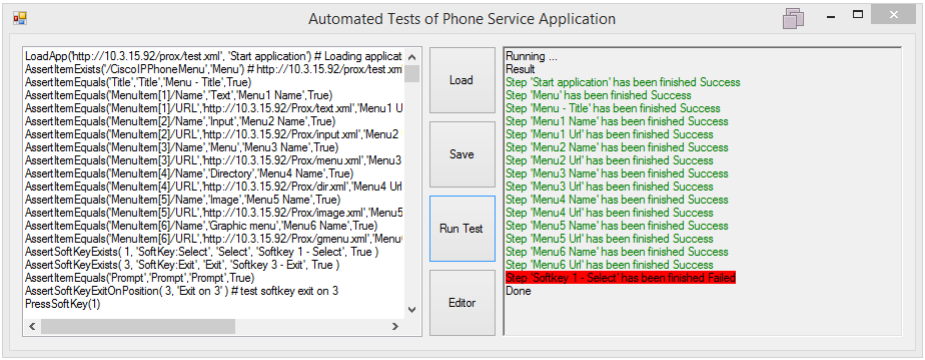
Obrázek 3.14: Obrazovka editoru

3.3.2.2 Grafické rozhraní testovací aplikace

Aplikace slouží k jednoduchému editování testovacích skriptů a také k jejích spuštění. Proto dialog obsahuje pouze nezbytné komponenty pro tyto činnosti. Dialog obsahuje jednoduchý

textový editor pro úpravu testovacího skriptu. Po spuštění skriptu je výsledek zobrazen v pravé části okna. Výsledky každého kroku jsou opět barevně označeny:

- Zelená barva textu znamená úspěšně provedený krok.
- Červená barva textu signalizuje, že pravidlo typu ***Validate*** bylo vyhodnoceno jako ***nepravda***.
- Kombinace černého textu s červenou barvou podkladu označuje neúspěšně provedený krok typu ***Assert*** a také to znamená, že skript byl ukončen.



Obrázek 3.15: Obrazovka testovací části aplikace

3.3.2.3 Textové rozhraní testovací aplikace

Výsledek testovacího skriptu je vypsán v textové podobě na standardní výstup. Výsledek není žádným způsobem formátován a není ani barevně zvýrazněn. Pomocí parametru ***silent*** je možné výstup vypnout. V tomto případě je podle výsledku nastaven pouze návratový kód.

3.3.3 Automaticky generované kroky

IP telefon má omezený počet grafických obrazovek se kterými dokáže pracovat. Všechny jsou popsané v kapitole 3.1. Pro každý typ obrazovky jsou definované kroky, které se automaticky vygenerují v grafickém editoru testů.

3.3.3.1 CiscoIPPhoneText

Objekt zajišťuje zobrazení textu na IP telefonu. XML dokument obsahuje jen tři položky - titulek, zobrazovaný text a řádek výzvy. Ukázka automaticky vygenerovaných kroků:

```
AssertItemExists('/CiscoIPPhoneText', 'Text') # http://10.3.15.92/Prox/text.xml
AssertItemEquals('Title', 'Text', 'Title', True) # Comment 2
AssertItemEquals('Prompt', 'Radek vyzvy', 'Prompt', True)
```

3.3.3.2 CiscoIPPhoneMenu

Zobrazení menu je provedeno pomocí objektu ***CiscoIPPhoneMenu***, který obsahuje elementy pro zobrazení textu titulku a řádky výzvy. Dále obsahuje jednotlivé položky menu s elementy pro zobrazení názvu položky menu a URL adresou. Podle aktuálního počtu položek v nabídce je vygenerovaný odpovídající počet příkazů pro kontrolu položek menu. Ukázka vygenerovaného kódu:

```
AssertItemExists('/CiscoIPPhoneMenu','Menu') # http://10.3.15.92/prox/test.xml
AssertItemEquals('Title','Title','Menu - Title',True)
AssertItemEquals('MenuItem[1]/Name','Text','Menu1 Name',True)
AssertItemEquals('MenuItem[1]/URL','http://10.3.15.92/Prox/text.xml',
    'Menu1 Url',True)
AssertItemEquals('Prompt','Prompt','Prompt',True)
```

3.3.3.3 CiscoIPPhoneDirectory

Zobrazení adresáře je velmi podobné jako zobrazení menu. Pouze místo URL adresy je definované telefonní číslo. Ukázka automaticky vygenerovaných příkazů:

```
AssertItemExists('/CiscoIPPhoneDirectory','Menu')
    # http://10.3.15.92/Prox/dir.xml
AssertItemEquals('DirectoryEntry[1]/Name','Ondrej Prochazka','Menu1 Name',True)
AssertItemEquals('DirectoryEntry[1]/Telephone','12345','Menu1 Telephone',True)
```

3.3.3.4 CiscoIPPhoneInput

CiscoIPPPhoneInput se využívá pro získání uživatelských dat od uživatele. Element URL definuje adresu, na kterou se zadaná vstupní data odešlou. Každé vstupní pole má definované zobrazované jméno, přednastavenou hodnotu, typ vstupních dat a název URL parametru použitý při odesílání dat.

```
AssertItemExists('/CiscoIPPhoneInput','input')
    # http://10.3.15.92/Prox/input.xml
AssertItemEquals('Title','Vstupni formular','Input - Title',True)
AssertItemEquals('URL','http://10.3.15.92/prox/result.asp','Input - URL',True)
AssertItemEquals('InputItem[1]/DisplayName','Uzivatelske jmeno',
    'Input - Item1 - DisplayName',True)
AssertItemEquals('InputItem[1]/QueryStringParam','username',
    'Input - Item1 - QueryStringParam',True)
AssertItemEquals('InputItem[1]/DefaultValue','User1',
    'Input - Item1 - DefaultValue',True)
AssertItemEquals('InputItem[1]/InputFlags','A',
    'Input - Item1 - InputFlags',True)
```

3.3.3.5 CiscoIPPhoneImageFile

Tento objekt se využívá pro zobrazení obrázku ve formátu PNG. Obrázek definovaný URL adresou je zobrazen na displeji telefonu na souřadnicích X,Y. Obrázek může mít definovaný i titulek. Ukázka vygenerovaného kódu:

```
AssertItemExists('/CiscoIPPhoneImageFile','Image')
    # http://10.3.15.92/Prox/image.xml
AssertItemEquals('Title','65464','Image - Title',True)
AssertItemEquals('LocationX','0','Image - LocationX',True)
AssertItemEquals('LocationY','0','Image - LocationY',True)
AssertItemEquals('URL','http://10.3.15.92/Prox/img.png',
    'Image - URL',True)
```

3.3.3.6 CiscoIPPhoneGraphicFileMenu

Definice grafického menu vychází z definice objektu pro zobrazení obrázku. Navíc jsou přidány definice oblastí, které definují oblast dotyku a URL adresu následujícího objektu zobrazeného na IP telefonu. Ukázka vygenerovaného kódu s definicí jedné dotykové oblasti:

```
AssertItemExists('/CiscoIPPhoneGraphicFileMenu','Graphic Menu')
    # http://10.3.15.92/Prox/gmenu.xml
AssertItemEquals('Title','','Graphic Menu - Title',True)
AssertItemEquals('LocationX','0','Graphic Menu - Location X',True)
AssertItemEquals('LocationY','0','Graphic Menu - Location Y',True)
AssertItemEquals('URL','http://10.3.15.92/Prox/img2.png',
    'Graphic Menu - URL',True)
AssertItemEquals('MenuItem[1]/URL','http://10.3.15.92/Prox/text.xml',
    'Menu1 Url',True)
AssertItemEquals('MenuItem[1]/TouchArea/@X1','1',
    'Menu1 TouchArea X1',True)
AssertItemEquals('MenuItem[1]/TouchArea/@Y1','1',
    'Menu1 TouchArea Y1',True)
AssertItemEquals('MenuItem[1]/TouchArea/@X2','100',
    'Menu1 TouchArea X2',True)
AssertItemEquals('MenuItem[1]/TouchArea/@Y2','100',
    'Menu1 TouchArea Y1',True)
```

3.3.3.7 SoftKey

Ovládací tlačítka mohou být definována u všech zobrazovaných objektů. Automaticky generovaný kód neprovádí pouze kontrolu polohy tlačítka, ale kontroluje tlačítko jako celek. Pokud stačí kontrola pouze polohy, lze využít funkce definované v kapitole 3.2.4.7 jako například ***AssertSoftKeyExitOnPosition***. Automaticky generovaný kód pro kontrolu SoftKey tlačítek definovaných u objektu CiscoIPPhoneImageFile:

```
AssertSoftKeyExists( 1, 'Update', 'SoftKey:Update',
    'Softkey 1 - Update', True )
AssertSoftKeyExists( 3, 'Exit', 'SoftKey:Exit',
    'Softkey 3 - Exit', True )
```

3.3.4 Výběr programovacího jazyka

Navrhovaný systém potřebuje využít emulátor IP telefonu. Tato komponenta byla vytvořena v rámci bakalářské práce a byla napsaná v jazyce Microsoft C#. Proto i tento projekt musí být napsaný v programovacím jazyce v prostředí Microsoft .NET. Společnost Microsoft má pro toto prostředí dva programovací jazyky - Visual Basic a C#, ze kterých jsem se rozhodl pro programovací jazyk Microsoft C#.

Celá aplikace je psaná formou tlustého klienta. Grafické rozhraní je psáno pomocí knihoven WinForms.

3.3.5 Výběr implementačního prostředí

S ohledem na zvolený programovací jazyk bylo jednoznačně zvoleno také implementační prostředí – Microsoft Visual Studio ve verzi Premium 2013.

Kapitola 4

Realizace

V této kapitole je detailně popsána finální realizace projektu. Kapitola je rozdělena dle funkčních částí.

4.1 Testovací třída a funkce

V kapitole 3.3.1.1 již bylo zmíněno, že třída, která je integrována do Python skriptu, je tvořena statickými metodami. Je to z důvodu jednodušší syntaxe při volání metod z Python skriptu. Třída obsahuje i statický konstruktor, ve kterém je vytvořena instance IP telefonního emulátoru. V něm jsou spouštěny testovací aplikace pomocí testovacího skriptu. Zároveň je vytvořena instance kolekce, ve které se ukládají výsledky z jednotlivých kroků:

```
_emulator = new Phone7970(null);
StatusList = new List<StepStatus>();
```

Chování aplikace může být odlišné na různých typech telefonu, proto je nutné u emulátoru nastavit všechny potřebné vlastnosti. Emulátor IP telefonu má potřebné nastavení uloženo v XML souboru. Tento projekt nemění stávající funkcionalitu, a proto je potřeba zavolat metodu emulátoru, která si nastavení načte z externího souboru.

Ukázka implementace kódu:

```
public static void LoadModelResource(String phoneConfig)
{
    if (String.IsNullOrEmpty(phoneConfig))
        throw new PhoneRuntimeException("PhoneConfig is null or empty");
    if (!Path.IsPathRooted(phoneConfig))
        phoneConfig = Path.Combine(
            Path.GetDirectoryName(System.Reflection.Assembly
                .GetExecutingAssembly().GetName().CodeBase) ??
                String.Empty, "Resources\\"+phoneConfig);
    var configXml = new XPathDocument(phoneConfig);
    _emulator.LoadConfig(configXml.CreateNavigator());
}
```

Metoda je vytvořena jako statická a zároveň jako ***public***. To umožňuje volat tuto metodu i přímo z Python skriptu.

Pro zjednodušení implementace jednotlivých testovacích funkcí je v této třídě vytvořeno i několik pomocných funkcí. První dvě funkce provádějí operace s textovými řetězci. Obě mají tři parametry:

- Parametr ***str1*** – první textový řetězec k porovnání.
- Parametr ***str2*** – druhý textový řetězec k porovnání.
- parametr ***caseSensitive*** – příznak, zda se má rozlišovat mezi malými a velkými znaky.

Metoda ***StringEquals*** porovnává, zda textové řetězce ***str1*** a ***str2*** jsou stejné. Metoda ***StringContains*** zjišťuje, zda řetězec ***str1*** obsahuje ***str2*** jako svůj podřetězec.

Ukázka implementace kódu:

```
private static Boolean StringEquals(String str1, String str2,
    bool caseSensitive)
{
    return str1.Equals(str2,
        caseSensitive
            ? StringComparison.InvariantCulture
            : StringComparison.InvariantCultureIgnoreCase);
}

private static Boolean StringContains(String str1, String str2,
    bool caseSensitive)
{
    return str1.IndexOf(str2,
        caseSensitive
            ? StringComparison.InvariantCulture
            : StringComparison.InvariantCultureIgnoreCase) > 0;
}
```

Ve třídě jsou definované i dvě obecné metody ***Assert*** a ***Validate***. Podle hodnoty v prvním parametru typu boolean vloží příslušný stav do kolekce s výsledky. Metoda ***Assert*** navíc vygeneruje výjimku ***AssertException***.

Ukázka implementace kódu:

```
public static void Assert(bool b, String message = @"Assert boolean")
{
    AddStatus(message, b ? Status.Success : Status.Failed);
    if (!b)
        throw new AssertException(message);
}

public static void Validate(bool b, String message = @"Validate boolean")
```

```
{
    AddStatus(message, b ? Status.Success : Status.Error);
}
```

Pro stejnou skupinu testovacích funkcí je vytvořena vždy jedna společná metoda. Ta provádí vlastní test a vrací výsledek typu boolean. Ostatní funkce využívají společné metody. Tím je veškerý výkonný kód soustředěn do malého množství metod, což zajišťuje větší přehlednost a čitelnost kódu a také minimalizaci možných chyb.

Příklad funkce pro testování obsahu:

```
public static Boolean CheckItemNotContains(String xpath, String value,
    Boolean caseSensitive)
{
    var node = GetSingleNode(xpath);
    if (node == null)
        return false;
    return !StringContains(node.Value, value, caseSensitive);
}
```

Funkce určené k ovládání telefonu jsou velmi jednoduché a krátké. Provede se pouze příslušná úprava parametrů a zavolá se odpovídající metoda v integrovaném emulátoru IP telefonu. Pokud dojde v emulátoru k jakékoliv chybě, vygeneruje se speciální výjimka ***PhoneRuntimeException*** s odkazem na původní výjimku.

Ukázka implementace funkce:

```
public static void PressTouchScreen(int x, int y, String message)
{
    try
    {
        _emulator.MouseClick(x, y);
    }
    catch (Exception exception)
    {
        AddStatus(message, Status.Failed);
        throw new PhoneRuntimeException(
    "Exception when pressing TouchScreen", exception);
    }
    AddStatus(message, Status.Success);
}
```

4.2 Testovací jádro

Třída zajišťující provedení vlastního testu se nazývá ***TestEngine***. Aby uživatel sestavující testovací skript nemusel neustále pamatovat na potřebnou inicializaci prostředí a referenci na .NET třídy, je interně vytvořen systémový skript, který provádí všechny potřebné inicializace

a přidává všechny potřebné reference. Tento inicializační skript je konstruktorem zkompilován a připraven pro spuštění před každým testovacím skriptem.

Inicializační skript v jazyce Python:

```
import sys
import clr
clr.AddReference('System')
clr.AddReference('AutomatedTests')
clr.AddReferenceToFile('AutomatedTests.dll')
from AutomatedTests.PhoneTestFunctions import *
LoadModelResource('C7970.xml')
Initialize()";
```

Pomocí metody ***Initialize*** se provede i vymazání vnitřní kolekce, ve které jsou ukládány výsledky jednotlivých kroků.

Přestože jazyk Python má podpory výčtových typů (enum), rozhodl jsem se pro zjednodušení syntaxe příkazů tento typ nevyužívat. Raději jsem si nadefinoval proměnné s přednastavenou hodnotou. Tyto proměnné jsou náhradou konstant, které jazyk Python nepodporuje. Inicializace těchto proměnných probíhá opět zcela automaticky před každým voláním testovacího skriptu.

Ukázka kódu, který provede nastavení proměnných a spustí inicializační skript předkompilovaný a uložený v proměnné ***_initScript***:

```
ScriptScope scriptScope = _scriptEngine.CreateScope();

scriptScope.SetVariable("Down", CursorKeyType.Down);
scriptScope.SetVariable("Up", CursorKeyType.Up);
scriptScope.SetVariable("Number0", NumKeyType.Number0);
scriptScope.SetVariable("Number1", NumKeyType.Number1);
scriptScope.SetVariable("Number2", NumKeyType.Number2);
scriptScope.SetVariable("Number3", NumKeyType.Number3);
scriptScope.SetVariable("Number4", NumKeyType.Number4);
scriptScope.SetVariable("Number5", NumKeyType.Number5);
scriptScope.SetVariable("Number6", NumKeyType.Number6);
scriptScope.SetVariable("Number7", NumKeyType.Number7);
scriptScope.SetVariable("Number8", NumKeyType.Number8);
scriptScope.SetVariable("Number9", NumKeyType.Number9);
scriptScope.SetVariable("Hash", NumKeyType.Hash);
scriptScope.SetVariable("Star", NumKeyType.Star);

_initScript.Execute(scriptScope);
```

Po ukončení testovacího skriptu je výsledná kolekce se stavem jednotlivých kroků překopírována z Python kódu do prostředí aplikace pomocí následujícího kódu:

```
List<StepStatus> results = scriptScope.GetVariable("results");
```

4.3 Formulář editoru

Jednou z komponent v okně editoru je i emulátor IP telefonu. Jedná se o komponentu vytvořenou v rámci bakalářské práce. Pro potřeby tohoto projektu jsem provedl drobné úpravy v komponentě emulátoru IP telefonu. Jednalo se však opravdu pouze o drobné opravy týkající se změny oprávnění k přístupu u některých vlastností nebo metod.

Emulátor IP telefonu při jakékoliv změně generuje události - ***events***. Formulář editoru se registruje pro odběr těchto událostí, na základě kterých lze automaticky generovat testovací kroky. Editor se tak dozví, pokud se v emulátoru změní aplikace, pokud je provedeno zmáčknutí libovolné klávesy nebo pokud dojde k ovládání pomocí dotyku displeje.

Příjem událostí zajistí následující kód:

```
phoneControl.Emulator.OnAppChange += Emulator_OnAppChange;
phoneControl.Emulator.OnPhoneKeyPress += Emulator_OnPhoneKeyPress;
phoneControl.Emulator.OnPhoneTouchPress += Emulator_OnPhoneTouchPress;
```

4.3.1 Seznam kroků

Pro zobrazení seznamu testovacích kroků je využito systémové komponenty ***ListBox***. Komponenta slouží k zobrazení seznamu položek a k výběru právě jedné položky. Pro zobrazení testovacích skriptů je upravena metoda „DRAW“. V nové metodě se provede vykreslení každého kroku na dva řádky a provede se také výběr barvy podle typu kroku.

Ukázka metody zajišťující zobrazení položky v seznamu:

```
private void listTasks_DrawItem(object sender, DrawItemEventArgs e)
{
    if (e.Index < 0)
        return;

    e.DrawBackground();
    using (var pen = new Pen(Color.Black))
    {
        pen.DashStyle = DashStyle.Dot;
        e.Graphics.DrawRectangle(pen, e.Bounds.Left + 1, e.Bounds.Top + 1,
            e.Bounds.Width - 3, e.Bounds.Height - 3);
    }

    var item = listTasks.Items[e.Index] as TaskControl;
    if (item == null)
        return;
    var commandFont = new Font(e.Font, FontStyle.Bold);
    e.Graphics.DrawString(item.ToString(), commandFont, GetItemColor(item),
        e.Bounds.Left + MarginItem, e.Bounds.Top + MarginItem);
    e.Graphics.DrawString(item.Comment, e.Font, Brushes.Black,
        e.Bounds.Left + MarginItem + 10,
        e.Bounds.Top + MarginItem + e.Font.Height);
```

```
}
```

Kolekce kroků obsahuje již připravené grafické reprezentace pro potřeby editování uživatelem. Po výběru se tak krok z kolekce vloží i do formuláře a zobrazí se všechny jeho nastavitelné vlastnosti. Aby bylo možné vrátit se zpět k uloženým hodnotám, je objekt před vložením do formuláře zkopírován pomocí metody ***Clone***.

Ukázka kódu zajišťující zobrazení kroku pro editaci:

```
var control = _tasks[listTasks.SelectedIndex];
if (control == null)
    return;
panelTask.Controls.Clear();
panelTask.Controls.Add((TaskControl) control.Clone());
```

4.3.2 Ladění skriptu

Při spuštění testovacího skriptu se v grafickém editoru zobrazují aktuálně běžící kroky. Toto není možné spuštěním celého vygenerovaného Python skriptu. Proto každý krok definuje metodu ***Runtask***, která provede patřičný krok a vrátí výsledek tohoto kroku. Aby bylo vůbec možné sledovat kroky na obrazovce, je mezi každým krokem provedeno zpomalení pomocí pauzy 100ms a také zpracováním všech nezpracovaných událostí.

Kód pro zpomalení kroků:

```
Thread.Sleep(100);
Application.DoEvents();
```

4.3.3 Čtení skriptu

Editor umožňuje uložení vygenerovaného Python skriptu a případně i načtení skriptu ze souboru. Editor umožňuje načtení skriptu, který byl uložen editorem a neumí načíst skript, který je ručně upraven a obsahuje další kód v jazyce Python. Takto upravený skript je ale možné normálně spouštět v testovacím okně nebo pomocí CLI verze aplikace. Při čtení uloženého skriptu je celý skript načten a výsledek je uložen v instanci třídy ***SkriptSource***. Následně je pro každý řádek provedena syntaktická kontrola a kompilace. Pomocí metody ***GetAstForScript*** je řádek přeložen a je také získána struktura ***PythonAst***. Obecná struktura AST (Abstract Syntax Trees) je definovaná společností Microsoft. Parser jazyka Python ji rozšiřuje o své specifické objekty. Editor prochází strukturu AST pro daný řádek, vytvoří konkrétní grafický krok a vloží jej do kolekce kroků.

Následující kód provede pro vstupní řádek v proměnné ***script*** kompilaci a vrátí strukturu ***PythonAST***:

```
var scriptSource = scriptEngine.CreateScriptSourceFromString
    (script, SourceCodeKind.Statements);
var sourceUnit = HostingHelpers.GetSourceUnit(scriptSource);
var compilerContext = new CompilerContext(sourceUnit, compilerOptions,
    ErrorSink.Default);
```

```
var parser = Parser.CreateParser(compilerContext, new PythonOptions());
return parser.ParseFile(true);
```

Následně se prochází AST strom a najde se název funkce a její parametry:

```
var ast = GetAstForScript(src, scriptEngine, compilerOptions);
var body = (SuiteStatement) ast.Body;
var expression = (ExpressionStatement) body.Statements[0];
var call = (CallExpression) expression.Expression;
var proc = (NameExpression) call.Target;
var args = call.Args;
```

4.4 Testovací okno a CLI

Grafické testovací okno i verze pro příkazový řádek využívají funkce, které jsou již popsané výše. Jedná se vlastně pouze o načtení externího Python skriptu, kompilaci a spuštění skriptu. Po úspěšném spuštění je zobrazen výsledek testovacího skriptu.

Kapitola 5

Testování

Kapitola popisuje způsob a výsledek testování výsledné aplikace. Můžeme je rozdělit na tyto části:

- Unit Testy – kontrola jednotlivých funkcí a metod.
- Test vygenerovaného scriptu.
- Test uživatelského grafického rozhraní.

5.1 Unit testy

Již v průběhu vytváření aplikace byly paralelně vytvářeny tzv. Unit testy. Ty slouží k testování jednotlivých tříd a jejich metod, zároveň také k ověření, zda chování metody na různé kombinace vstupů je dle očekávání a zda metoda generuje správný výsledek. Postupně bylo vytvořeno více jak 150 různých testů.

5.2 Testování vygenerovaného kódu

Editor generuje kód, který obsahuje pouze navržené funkce. Proto stačí ověřit, že vygenerovaný kód odpovídá syntaxi jazyka Python.

Pro otestování výsledného kódu jsem vytvořil jednoduchou telefonní aplikaci, která obsahuje všechny grafické objekty, které jsou podporované emulátorem IP telefonu. Procházením této aplikace v editoru se automaticky vygenerují všechny testovací kroky. Výsledný skript je spuštěn v testovací aplikaci a je zkontrolován výsledek, zda všechny kroky byly vyhodnoceny úspěšně.

5.3 Testování Python skriptu

Aplikace využívá pro kontrolu syntaxe a překladu Python skriptu komponenty IronPython. Proto není nutné provádět testy, které ověří vstupní text na správnou syntaxi, popřípadě i na správný překlad. Stačí se pouze zaměřit na rozšiřující funkce, které jsou systémem přidány

do testovací aplikace. Pro tuto potřebu byla vytvořena sada skriptů, pomocí kterých se ověřuje testovací aplikace. Jeden ze skriptů je například vytvořen tak, že všechny kroky jsou vyhodnoceny jako ***FALSE***. Dalším příkladem je speciálně upravený skript, který využívá možnosti Python jazyka, například cykly a definice funkcí.

Ukázka testovacího skriptu, který opakovaně testuje dvě obrazovky a pomocí ovládacích tlačítek mezi nimi přepíná:

```
def test_main_screen():
    AssertItemExists('/CiscoIPPhoneMenu','Menu')
        # http://10.3.15.92/prox/test.xml
    AssertItemEquals('Title','Title','Menu - Title',True)
    AssertItemEquals('MenuItem[1]/Name','Text','Menu1 Name',True)
    AssertItemEquals('MenuItem[1]/URL','http://10.3.15.92/Prox/text.xml',
        'Menu1 Url',True)
    AssertItemEquals('MenuItem[2]/Name','Input','Menu2 Name',True)
    AssertItemEquals('MenuItem[2]/URL','http://10.3.15.92/Prox/input.xml',
        'Menu2 Url',True)
    AssertItemEquals('MenuItem[3]/Name','Menu','Menu3 Name',True)
    AssertItemEquals('MenuItem[3]/URL','http://10.3.15.92/Prox/menu.xml',
        'Menu3 Url',True)
    AssertItemEquals('MenuItem[4]/Name','Directory','Menu4 Name',True)
    AssertItemEquals('MenuItem[4]/URL','http://10.3.15.92/Prox/dir.xml',
        'Menu4 Url',True)
    AssertItemEquals('MenuItem[5]/Name','Image','Menu5 Name',True)
    AssertItemEquals('MenuItem[5]/URL','http://10.3.15.92/Prox/image.xml',
        'Menu5 Url',True)
    AssertItemEquals('MenuItem[6]/Name','Graphic menu','Menu6 Name',True)
    AssertItemEquals('MenuItem[6]/URL','http://10.3.15.92/Prox/gmenu.xml',
        'Menu6 Url',True)
    AssertSoftKeyExists( 1, 'Select', 'SoftKey:Select', 'Softkey 1 - Select', True )
    AssertSoftKeyExists( 3, 'Exit', 'SoftKey:Exit', 'Softkey 3 - Exit', True )
    AssertItemEquals('Prompt','Prompt','Prompt',True)

def test_text_screen():
    AssertItemExists('/CiscoIPPhoneText','Text') # http://10.3.15.92/Prox/text.xml
    AssertItemEquals('Title','Text','Title',True) # Comment 2
    AssertSoftKeyExists( 1, 'Update', 'SoftKey:Update', 'Softkey 1 - Update', True )
    AssertSoftKeyExists( 3, 'Exit', 'SoftKey:Exit', 'Softkey 3 - Exit', True )
    AssertItemEquals('Prompt','Radek vyzvy','Prompt',True)

LoadApp('http://10.3.15.92/prox/test.xml', 'Start application') # Loading application
for x in range(0, 4):
    test_main_screen()
    PressSoftKey(1) # menu 1 - text
    test_text_screen()
    PressSoftKey(3) # back to main
```

5.4 Testování grafického rozhraní

Pro tuto aplikace je důležitá její snadná ovladatelnost. Proto jsem aplikaci otestoval i pomocí testů na uživatelské rozhraní.

5.4.1 Cílová skupina

Testeři byli vybráni přímo z řad uživatelů u zadavatelské firmy. Všichni tito testeři jsou znalí problematiky psaní telefonních aplikací. Jedná se o systémové inženýry v oblasti Cisco IP telefonie anebo o programátory telefonních aplikací. Celkem se testování zúčastnili dva systémový inženýři a dva programátoři.

5.4.2 Testovací scénář A

Ukázka testovacího scénáře, pomocí kterého testeři zkoušeli novou aplikaci. U tohoto testu dostal tester detailní popis činností, které je potřeba vykonat:

1. Spusťte testovanou telefonní aplikaci – URL adresa http://xyz/test/menu1.xml.
2. Nastavte automatické generování kroků.
3. Pomocí numerické klávesy ***1*** u emulátoru IP telefonu zvolte první menu.
4. Upravte krok testující titulek telefonní aplikace – z typu ***Assert*** změňte na typ ***Validate***.
5. Zmáčkněte SoftKey tlačítko ***UPDATE***.
6. Pomocí kurzorových kláves zvolte šesté menu s názvem – Grafické menu.
7. Na zobrazené mapě klikněte na oblast Severních Čech.
8. Upravte krok definující souřadnice dotyku – nastavte hodnoty X=100 a Y=50.
9. V telefonní aplikaci se vraťte zpět na zobrazení mapy.
10. Vytvořte ručně nový krok - ***PressTouchScreen*** a nastavte souřadnice X==150 a Y=100.
11. Změňte pořadí posledních dvou kroků.
12. Výsledný skript uložte do souboru „test.py“.

5.4.2.1 Výsledek testovacího scénáře A

1. Spuštění telefonní aplikace proběhlo u všech testerů bez obtíží.
2. Požadovanou akci našli všichni velmi rychle.
3. Tento krok byl proveden bez obtíží všemi testery.

4. Dva testeři chvilku hledali editační pole pro změnu titulku. Jejich zdržení však nebylo nijak zásadní a zatím tak není potřeba aplikaci upravovat.
5. Celkem tři testeři využili dotykového displeje u emulátoru a nezmáčkli odpovídající tlačítko SoftKey.
6. Tento krok byl proveden bez obtíží všemi testery.
7. Tento krok byl proveden bez obtíží všemi testery.
8. U tohoto kroku již neměli testeři problém s orientací na formuláři a požadované položky našli ihned.
9. Tento krok byl proveden bez obtíží všemi testery.
10. Všichni testeři chvilku hledali, kde se nový krok přidává. Následné činnosti provedli bez obtíží.
11. Všichni testeři se ihned snažili změnit pořadí položek pomocí myši – Drag and Drop. Tuto vlastnost aplikace ale neobsahuje. Hned následně si všimli tlačítek pro změnu pořadí a provedení úkolu tak bylo úspěšně dokončeno všemi testery.
12. Tento úkol nečinil žádné problémy.

5.4.3 Testovací scénář B

Druhým testem je obecné zadání úkolu bez specifikování jednotlivých kroků.

Úkol: U testované telefonní aplikace proveďte následující:

- Pomocí menu 3 se pokuste do aplikace přihlásit.
- První pokus zadejte se špatným heslem.
- U druhého pokusu zadejte správně jméno i heslo: username/password.

5.4.3.1 Výsledek testovacího scénáře B

Tři testeři využili u tohoto úkolu automaticky generované kroky. Díky tomu jim úkol nedělal obtíže a dokončili ho velmi rychle. Jeden tester si ale zvolil způsob, ve kterém generoval všechny kroky sám. Protože aplikace nezobrazuje XML dokument, který je vstupem do emulátoru IP telefonu, tester měl poměrně velké problémy s dokončením úkolu. Nevěděl, jak vypadá XML dokument pro vstupní formulář a jak se jmenují parametry předávané v GET požadavku. Vysvětlil jsem mu, že pokud nezná detailně testovanou aplikaci, je výhodnější využít automaticky generované kroky. Následně tester dokončil úkol bez obtíží.

5.4.4 Doplňující otázky

Po dokončení všech testů byly testerům položeny ještě následující otázky:

1. Co se vám na aplikaci nelíbí a co byste změnili?
2. Je aplikace použitelná pro definici testovacích skriptů pro reálné telefonní aplikace?

5.4.4.1 Požadavky na změny v aplikaci

Od testovacích uživatelů jsem získal následující připomínky a požadavky na úpravu aplikace.

Prvním požadavkem je změna grafického designu ovládacích tlačítek, aby byly podobné tlačítkům na IP telefonu. To více zpřehlední, která tlačítka jsou ještě součástí emulátoru IP telefonu, a které již slouží k ovládání testovacích kroků.

Druhým požadavkem je zobrazení XML dokumentu, který je právě zpracován v emulátoru. Pokud totiž uživatel nezná obsah XML dokumentu, nemusí být vždy schopen sestavit testovací kroky ručně. Nyní si tak musí jinou cestou (například pomocí internetového prohlížeče) zjistit obsah XML dokumentu a následně vytvořit odpovídající testovací krok.

5.4.4.2 Použitelnost aplikace

Všichni uživatelé se shodli, že aplikace je použitelná i pro definici testů reálných telefonních aplikací.

Dva programátoři, kteří byli vybráni do testovací skupiny, se také shodli, že vygenerovaný testovací skript editorem je poměrně dlouhý a u rozsáhlejších aplikací se stane nepřehledný. Pozitivně hodnotí, že editor vkládá do výsledného skriptu i poznámky s URL adresou právě testované obrazovky. Vygenerovaný skript je tak vhodné upravit a testování jednotlivých obrazovek telefonní aplikace uspořádat do samostatných funkcí. Tím lze využít již jednou napsané testy konkrétní obrazovky vícekrát. To je výhodné zejména v případech, kdy se v testované aplikaci několikrát vracíme do centrálního menu.

Při první diskuzi jsme ale nenašli cestu, jak jednoznačně detekovat stejné obrazovky přímo editorem a tím i generovat čitelnější a kratší kód. Proto se aplikace zatím ponechá se stávající logikou a tato optimalizace proběhne, až bude nalezen způsob jak logiku zoptimalizovat.

5.4.5 Závěr z testování grafického rozhraní

Provedené testy ukázaly, že aplikaci je možné využít i pro reálné aplikace a že ovládání aplikace je pro uživatele jednoduché a použitelné. Všechny požadavky na úpravy byly vyhodnoceny jako rozšiřující, které aplikaci zlepšují. Nejsou ale žádným způsobem kritické pro první nasazení do provozu.

Kapitola 6

Závěr

Cílem této diplomové práce bylo vytvořit aplikaci pro automatizované testování telefonních aplikací. Součástí je i návrh nebo výběr skriptovacího jazyka a editor testovacích skriptů.

Všechny vytyčené cíle byly splněny. Grafický editor skriptů byl vytvořen a pomocí testovacích skriptů lze provádět funkční testy telefonních aplikací. Vytvořený skript lze libovolně upravovat a lze využít i všech možností, které má k dispozici jazyk Python.

Vývoj aplikace probíhal v několika fázích. Na začátku byla provedena analýza. V této části byla navržena sada potřebných testovacích tříd, pomocí který bude provedeno testování telefonních aplikací. Následně byla provedena analýza, zda se využije existujícího skriptovacího jazyka nebo bude výhodnější navrhnout vlastní speciální jazyk. Nakonec byl zvolen jazyk Python.

Po dokončení analýzy proběhla implementace testovacích funkcí a ověření funkcionality na ručně vytvořeném skriptu. Následně byl vytvořen i grafický editor testovacích skriptu, pomocí kterého lze jednoduše a automaticky vygenerovat testovací skript.

Dalším krokem bylo testování aplikace. Testování ověřilo, že výsledný testovací skript je správně vytvořen a také, že tento skript je správně vyhodnocen. Rovněž proběhl test grafického rozhraní. Při něm byly od uživatelů získány hodnotné připomínky a požadavky na úpravu.

Vytyčený cíl práce byl splněn a testovanou aplikaci je tedy možné pro první nasazení do provozu s tím, že získané požadavky na úpravu pomohou aplikaci dále zdokonalit.

Literatura

[1] BRADLEY, N. *XML Kompletní průvodce*. 1. Grada Publishing, 2000.

[2] CHRISTIAN NAGEL, J. G. M. S. K. W. B. E. *C# 2008 Programujeme profesionálně*. 1. Computer Press, 2009.

[3] CHRISTIAN NAGEL, J. G. M. S. K. W. B. E. *C# 2008 Programujeme profesionálně*. 2. Computer Press, 2009.

[4] CISCO SYSTEMS, I. *Cisco Developer portál* [online]. 2012. [cit. 12. 5. 2014]. Dostupné z: <http://developer.cisco.com/>.

[5] CISCO SYSTEMS, I. Cisco Unified IP Phone Services Application Development Notes. Release 8.0, Text Part Number: OL-20949-01.

[6] FOUNDATION, P. S. *Python webová prezentace* [online]. 2014. [cit. 12. 5. 2014]. Dostupné z: <https://www.python.org/>.

[7] Členové IronPython komunity . *IronPython webová prezentace* [online]. 2014. [cit. 12. 5. 2014]. Dostupné z: <http://ironpython.net/>.

[8] MATCHAL, B. *XML v příkladech*. 1. Computer Press, 2000.

[9] Přispěvatelé Wikipedie. *LaTeX — online manuál* [online]. 2012. [cit. 12. 5. 2014]. Dostupné z: <http://en.wikibooks.org/wiki/LaTeX/>.

[10] TROELSEN, A. *C# a .NET 2.0 profesionálně*. 1. Zoner Press, 2006.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

ASCII American Standard Code for Information Interchange

CIPC Cisco IP Communicator

CUCM Cisco Unified Communication Manager

DOM Document Object Model

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IP Internet Protocol

IPPS IP Phone Services

JPEG Joint Photographic Experts Group

MIME Multipurpose Internet Mail Extensions

PNG Portable Network Graphics

SAX Simple API for XML

SIP Session Initiation Protocol

SDK Software Developer Kit

TFTP Trivial File Transfer Protocol

URI Uniform Resource Identifier

URL Uniform Resource Locator

XML eXtensible Markup Language

XPath XML Path Language

XSD XML Schema Definition

Příloha B

Instalační a uživatelská příručka

B.1 Instalace

Aplikace je vytvořená v prostředí Microsoft .NET Framework 4. Pro její spuštění je proto nutné mít na počítači nainstalované potřebné knihovny, které je možné stáhnout ze stránek výrobce – společnosti Microsoft. Konkrétně lze potřebný software stáhnout z adresy <http://www.microsoft.com/en-us/download/details.aspx?id=17851> nebo <http://www.microsoft.com/en-us/download/>.

Vlastní aplikaci není nutné instalovat, stačí ji pouze nakopírovat na počítač spolu se všemi konfiguračními soubory a obrázky.

B.2 Použití

Po spuštění aplikace se uživateli otevře okno editoru, ve kterém je zobrazen emulátor telefonu, ovládací klávesnice a textové pole pro vložení URL adresy, ve které se nachází telefonní aplikace. Dále je zde seznam testovacích kroků a ovládací tlačítka pro vytvoření nebo smazání aktuálního kroku a pro změnu pořadí kroků. Ukázka obrazovky je na obrázku B.1.

B.2.1 Parametry aplikace

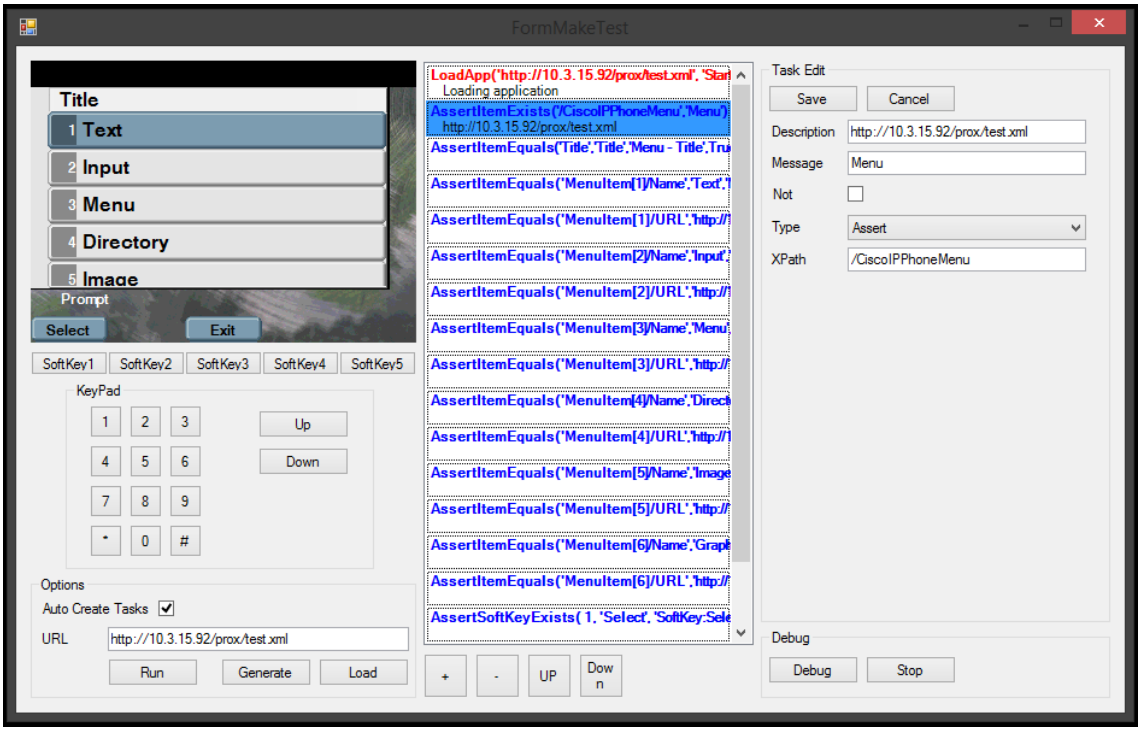
Aplikaci lze také spustit i s parametry, které ovlivňují, která část aplikace je spuštěna:

- Parametr „-editor“ – spustí se grafický editor.
- Parametr „-tester“ – spustí se grafická verze testovací aplikace.
- Parametr „-cli“ – spustí se pouze textová verze testovací aplikace.

B.2.2 Ovládání aplikace

Aplikace obsahuje následující ovládací prvky:

- Emulátor IP telefonu.



Obrázek B.1: Obrazovka grafického editoru

- Ovládací prvky emulátoru – číselník telefonu, ovládací tlačítka SoftKey.
- Application URL – odkaz na telefonní aplikaci.
- Tlačítko ***RUN*** – spustí telefonní aplikaci v emulátoru.
- Tlačítko ***GENERATE*** – generuje testovací skript z aktuálně vytvořených kroků.
- Tlačítko ***LOAD*** – načtení testovacího skriptu do editoru.
- Tlačítko + – vytvoří nový krok a vloží ho do seznamu.
- Tlačítko **-** – smaže aktuálně vybraný krok.
- Tlačítko ***UP*** – posune aktuálně vybraný krok o jednu pozici nahoru.
- Tlačítko ***DOWN*** – posune aktuálně vybraný krok o jednu pozici dolů.
- Tlačítko ***SAVE*** – uloží změny u aktuálního kroku.
- Tlačítko ***CANCEL*** – vrátí všechny provedené změny u aktuálního kroku.
- Tlačítko ***DEBUG*** – spustí ladění testovacího skriptu.
- Tlačítko ***STOP*** – zastaví aktuálně probíhající test.

Příloha C

Obsah přiloženého DVD

- app – spustitelná verze aplikace pro operační systém Microsoft Windows
- data – doplňující soubory (XSD šablony, testovací soubory, skin pro telefon Cisco 7970)
- src – soubory se zdrojovými kódy aplikace
- text – diplomová práce ve formátu pdf a tex včetně obrázků
- install.txt – postup instalace a spuštění aplikace
- readme.txt – popis jednotlivých adresářů