Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Automatická aktualizace aplikace na počítači uživatele

Plzeň 2018 Vít Mazín

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 2. května 2018

Vít Mazín

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Petru Přibylovi za cenné rady, věcné připomínky, ochotu a trpělivost, které mi pomohly tuto práci zkompletovat.

Abstract

Web applications saves big amount of capital because we do not have to spend it on managing of user's PCs. Users only need web browser to run this applications. On the other hand applications which are installed and run on user's PCs are more user friendly, faster and they do not have such big impact on network traffic. These applications are often also run because, users are accustomed to them and they do not want to learn new technologies. The aim of this bachelor thesis is to design and to implement solution for automatic updating applications installed on user's computer. The program will check for the latest versions of selected applications at specified time period as well as offer update to the latest versions. Last but not least the program will be capable of updating the applications.

Abstrakt

Webové aplikace přinášejí mimo jiné výraznou úsporu nákladů na správu stanic uživatelů. K provozu stačí nainstalovaný běžný internetový prohlížeč. Na druhou stranu aplikace, které jsou nainstalovány a provozovány přímo na počítači uživatele přinášejí větší komfort obsluhy, rychlost a úsporu síťových přenosů. Tyto aplikace jsou často provozovány také proto, že uživatelé jsou na ně zvyklí a nechtějí se učit nové technologie. Cílem práce bude navrhnout a implementovat řešení pro automatickou aktualizaci aplikací, nainstalovaných v prostředí počítače uživatele. Program bude ve stanovené době kontrolovat aktuální verze sledovaných programů a bude nabízet jejich povýšení na aktuální verzi. Program následně bude schopen toto povýšení provést.

Obsah

1 Úvod

Práce vzniká pro společnost CCA Group a.s. se sídlem v Praze. Cílem této práce je vytvořit knihovnu, jež dodá do programů tvořených společností CCA funkčnost automatické aktualizace. Knihovna bude primárně určená pro programy, jež rozšiřují funkcionalitu profesionálního programu Enterprise Architect pro systémovou analýzu a návrh.

První kapitola je zaměřená na seznámení se s procesem automatické aktualizace. Toto zahrnuje historický vývoj aktualizování software. Probrány budou způsoby, jimiž se aktualizování software řešilo a jaké problémy tento proces řeší. Dále bude nastíněn proces aktualizace jako takový. To znamená, objasnit, jakým způsobem jsou obecně řešeny aktualizace. Posléze budou nastíněny aspekty automatické aktualizace, jež ovlivňují funkčnost samotného programu, který má být aktualizován. V poslední části kapitoly budou rozebrány motivace k použití automatické aktualizace.

V druhé kapitole budou probrány volně dostupné způsoby, jimiž lze implementovat automatickou aktualizaci do programu. Popsáno bude, jak lze tento proces implementovat do programů tvořených technologií Java a programů vytvořených v prostředí Microsoft .NET. Obě platformy mají svá specifická řešení pro automatickou aktualizaci, která se významně odlišují implementací.

Další kapitola bude věnována popisu všech programů, jež bylo nutné pro splnění práce vytvořit. Rozebráno bude, proč bylo zvoleno dané řešení problému při tvorbě a jak budou programy rámcově fungovat. Nastíněn bude také protokol, jenž byl třeba implementovat pro komunikaci aktualizační knihovny se serverem, z něhož jsou aktualizace dodávány.

V předposlední kapitole vysvětlím, jak jsem implementoval řešení práce. Popsána bude implementace každého programu, který pro práci vznikl, a také jak jsem řešení integroval do programů vytvořených zadavatelem.

Poslední kapitola bude zaměřena na testování výsledného vytvořeného software. Bude vysvětleno, jaké testy byly použity a proč byly použity.

2 Automatická aktualizace

2.1 Vývoj aktualizací software

Vlivem nepozornosti nebo špatné úvahy lze do programu snadno zanést chybu, která je odhalena až poté, co je program u koncového uživatele. Chyba může být nezávažná, ale také může být příčinou vzniku kritické trhliny v bezpečnosti programu. Z tohoto důvodu je třeba program aktualizovat a chyby opravit.

V minulosti byly aktualizace programů realizovány například tak, že se děrný štítek, na kterém byl program, musel rozdělit a část programu byla nahrazena opravenou částí programu na děrném štítku [5]. S příchodem přenositelných médií typu magnetických pásků a disket bylo aktualizace možno šířit pohodlněji díky přenositelnosti těchto médií.

S masovým rozšířením internetové sítě přišla možnost aktualizace šířit mnohem snáze. Každý, kdo je připojen k internetu, má možnost rychle a pohodlně aktualizaci stáhnout a aplikovat. Aktualizace bylo třeba instalovat ručně, ale s velkým rozvojem všestrannosti softwaru je dnes tato činnost mnohdy plně automatizována, což zjednodušuje celý proces a zvyšuje komfort pro koncového uživatele.

Automatické aktualizace lze využít téměř pro každé programy, které jsou aktivně vyvíjeny nebo jsou natolik důležité, že je potřeba případnou nalezenou chybu v nich opravit. Tento proces se tedy stal velmi využívaným pro potřeby operačních systémů. Firma Microsoft implementovala automatické aktualizace již ve Windows ME a firma Apple tohoto řešení využila poprvé v MAC OS 9 [11, 15].

2.2 Proces aktualizace

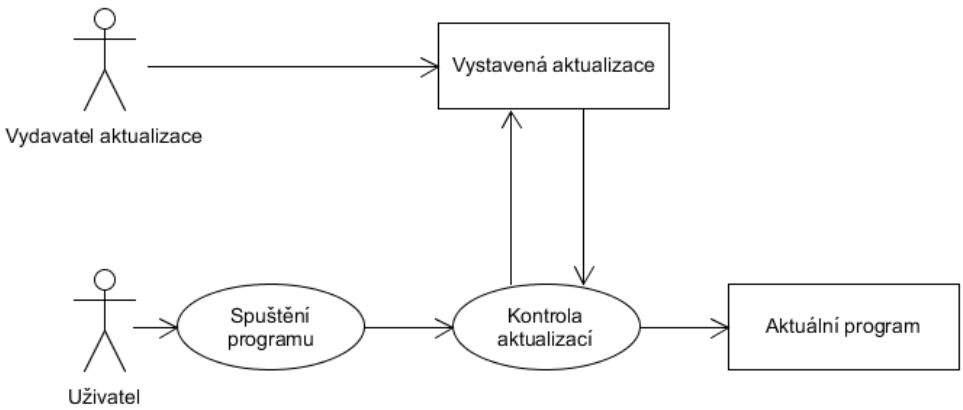
Dnešní programy běžně využívají aktualizací. Obvykle po spuštění program nabídne povýšení na novou verzi. Uživatel může přejít na novější verzi nebo nemusí. Není zcela běžné, aby program nedovolil uživateli spustit starší verze. Pokud je však z bezpečnostních důvodů důležité mít nainstalovanou nejnovější verzi, je nutné pravidelně kontrolovat, zda nevyšla. To ale znamená, že pro spuštění programu musí být počítač uživatele pravidelně připojen k internetu.

Obvykle programy využívají svých programových prostředků, např. Windows používá Windows Update, Java používá Java Auto Updater atd. Velikost aktualizace se odvíjí v závislosti na velikosti daného programu. Běžně je velikost aktualizace v řádech od jednotek megabytů po stovky. Jedná-li se o rozsáhlý program, může být velikost aktualizace v řádech jednotek až desítek gigabytů. Programy mají obvykle své vlastní programové vybavení na to, aby nabízenou aktualizaci stáhly do počítače a samy provedly povýšení verze. Ovšem časté řešení také bývá, že si nabízenou novou verzi musí uživatel ručně stáhnout a povýšení verze probíhá tak, že se odinstaluje stará verze programu a nová verze se ihned poté nainstaluje. Toto řešení je však pro koncového uživatele značně nekomfortní, protože vyžaduje jeho zapojení do procesu aktualizace více, než pokud je tohoto procesu program schopen sám.

2.3 Aspekty automatické aktualizace

Prvním z důležitých aspektů aktualizací je, kdy se vlastně bude zjišťovat, zda je k dispozici nová verze. Obvykle je nová verze vyhledávána při spuštění programu a následně je uživateli nabídnuta. Aktualizace mohou být vyhledávány i během běhu programu a nabízeny uživateli, což ale může uživatele obtěžovat, protože například má rozpracovanou práci atd. Tento proces je znázorněn na obrázku 2.1.

Obrázek 2.1: Proces spuštění programu



Další věcí je rozhodnout o tom, zda půjde spustit neaktuální verze. Pokud program nekomunikuje dále, například se serverem, není teoreticky nutná okamžitá aktualizace. Na druhou stranu, pokud se jedná o důležitou úpravu

jako například bezpečnostní záplata, změnilo se rozhraní serveru nebo by mohla vzniknout nekonzistence ve vyprodukovaných datech, je důležité aktualizaci vynutit tím, že program nepůjde spustit za kontrolu aktualizací.

Instalátor aktualizace musí také řešit další důležité úlohy. Pokud je to třeba, musí přenastavovat údaje v registrech, změnit cesty k souborům nebo systémová nastavení atd.

V neposlední řadě je také třeba určit, jak přesně bude program aktualizován. Může být například kompletně nainstalován znovu, aby se zajistilo, že proběhnou veškeré změny, které jsou potřeba. Toto řešení je spolehlivé, ale může být časově náročnější a náročnější vzhledem k rychlosti připojení uživatele. Pokud je program velký a „zbytečně“ by se stahovala data, která nebyla změněna a jsou objemná, není pro uživatele takováto aktualizace komfortní, vzhledem k časové náročnosti u pomalejšího připojení. Proto je možné instalovat pouze části, které je třeba aktualizovat. V tomto případě je ale nutné, aby bylo zajištěno, že po instalaci bude program konzistentní. Pokud by se přeskočila aktualizace, která mění něco, co ta následující již nemění, je tedy třeba postupně nainstalovat všechny ve správném pořadí.

Z hlediska bezpečnosti je důležité, aby vydavatel podepsal aktualizační soubory nebo aby se data stahovala z bezpečného serveru. Toto lze zařídit elektronickými certifikáty. Uživatel je tak chráněn před stažením škodlivého programu.

2.4 Aktualizace ve videoherním průmyslu

Proces automatických aktualizací je dnes naprosto běžným ve videoherním průmyslu. Aktualizace jsou často vydávány kvůli problému s výkonem či pro rozšíření podpory hardware v daném programu.

Tento proces ale dříve nebyl standardní kvůli tomu, že ne každý uživatel měl přístup k internetové síti. Pro starší konzole byly vydávány pouze na pevných nosičích jako kompaktní disk atd. Programy tedy byly prodávány s možností, že ukrývají chybu. Toho poté někteří uživatelé využívali a dokázali například nalézt tzv. epxloit, což je chyba, která vedla mnohdy i k napadení firmware herního zařízení, a to mohlo být využito ke spouštění kradeného software. Například na konzoli Playstation Vita byly vydány hry, které obsahovaly exploit. Díky této chybě bylo možné instalovat neoficiální hry a mnoho dalšího. Firma Sony se proti útokům bránila tak, že byly hry staženy z internetového obchodu, byla vydána nová verze firmware a po nějaké době byla hra opět vrácena [2].

2.5 Motivace CCA k použití automatické aktualizace

Firma CCA vyvíjí programy a aplikace pro mnoho platforem. Na některých z nich lze programy udržovat v aktuální verzi mnohem snadněji než jinde.

Jedná-li se o mobilní aplikace, je proces aktualizace vlastně vyřešený přímo v rámci platformy. Pro platformu Android provozuje firma Google online distribuční službu Google Play. Firma Apple na platformě iOS dovoluje instalovat aplikace pouze přes vlastní distribuční službu App Store. Tyto služby se přímo starají o instalaci a aktualizaci aplikací.

Pokud se jedná o webové aplikace, je proces aktualizace poměrně jednoduchý. Stará se o něj pověřená osoba tak, že na server, kde aplikace běží, nahraje upravené soubory nebo celou novou aplikaci. Takováto údržba bývá prováděna přes noc, kdy uživatelé nejsou téměř aktivní, a při dalším spuštění pracují automaticky s nejnovější verzí.

V případě tzv. „desktopových“ programů na systém Windows je proces aktualizace obtížnější. Firma Microsoft vyvinula vlastní distribuční službu Windows Marketplace, která je ovšem již od roku 2012 nahrazena službou Windows Store. Pokud je program vydán pomocí této distribuční služby, je možné aktualizace instalovat snadno díky této službě jako například v iOS App Store. Pokud je však program vydán jinou cestou, musí se o aktualizace postarat vydavatel.

Firma CCA vyvíjí například webovou aplikaci, která pracuje s dokumenty. Dokumenty je třeba pro jisté potřeby digitálně podepsat. Proces podpisu musí proběhnout v počítači, do kterého je připojeno například hardwarové zařízení obsahující privátní klíč. Z důvodu bezpečnosti nelze takovéto citlivé informace předat webové aplikaci, protože by tato informace mohla být například odposlechnuta. Tento problém lze řešit aplikací, která je naprogramována jako Java applet, který je spuštěn z internetu v internetovém prohlížeči uživatele. Ovšem technologie Java applet přestala být z důvodů bezpečnosti podporována většinou internetových prohlížečů. Proto firma CCA od appletů upouští a vyvíjí programy, které jsou nainstalovány v počítači uživatele, a proto vyvstala nutnost k vyřešení aktualizací těchto programů.

Firma CCA pro modelování projektů používá program Enterprise Architect od společnosti Sparx Systems, jenž umožňuje modelovat projekt v modelovacím jazyce UML. Protože si firma CCA vyvíjí vlastní metodiku a modelovací jazyk UML tedy značně upravuje, je nutné, aby pro danou metodiku program Enterprise Architect obsahoval funkčnost, která by metodiku

podporovala. Tato funkčnost je tedy dodávána addiny, které si sama firma vyvíjí. Program Enterprise Architekt jako takový nabízí velikou škálu funkcí pro práci s obecnou metodikou, ale ne všechny jsou využitelné pro metodiku vyvíjenou firmou CCA. V případě, že se metodika změní, je nutné změnit i chování addinů. Proto je třeba řešit aktualizace těchto programů.

3 Dostupná řešení pro automatické aktualizace

3.1 Java Web Start

Java Web Start je řešení, které umožňuje stažení a spuštění programu vytvořeného v jazyce Java z internetu. Program lze takto spustit jedním kliknutím na odkaz, což je komfortní pro koncového uživatele.

Java Web Start je součástí Java Runtime Environment (JRE) od vydání Java verze 5.0. To znamená, že Java Web Start je automaticky nainstalována s prostředím JRE, které je potřeba pro běh programů vytvořených v tomto jazyce [7].

3.1.1 Vytvoření instalace Java Web Start

Program, který má být uživateli dodán pomocí Java Web Start, je napsán jako běžný program v jazyce Java. Není třeba do kódu programu přidávat žádné příkazy, které by souvisely s Java Web Start. Program pouze musí být vytvořen jako `JAR` archiv. Instalace, spuštění a další chování je popsáno v `JNLP` (Java Network Launch Protocol) souboru, který je nutno vystavit s programem a který program spouští. Informace pro tvorbu instalace čerpám především ze zdrojů [3] a [13].

Programy stažené z internetu jsou potencionálně nebezpečné. Proto je nutné `JAR` archiv programu elektronicky podepsat, pokud program využívá operace, na které je třeba vyšší úroveň práv, než které stačí pro spuštění programu. Při vyžádání vyšších práv jsou zobrazeny informace z certifikátu [10].

Certifikát lze vytvořit použitím programu `keytool`, jenž je součástí instalace prostředí Java. Vytvořit certifikát lze z příkazové řádky například takto:

```
keytool -genkey -keystore testkey -alias jws
```

Poté se vyplní vše, co program vyžaduje, a certifikát je vytvořen. Podepsat `JAR` archiv vytvořeným certifikátem lze programem `jarsigner`, který je taktéž součástí instalace prostředí Java. Program poté lze podepsat certifikátem takto:

```
jarsigner -keystore testkey TestJnlp.jar jws
```

Pokud je archiv podepsaný, je program připravený k distribuci. Je tedy nutné vytvořit `JNLP` soubor. Jedná se o XML soubor, tudíž se skládá ze zanořených tagů. Nejdříve je doporučeno uvést, o jakou verzi XML se jedná a jaké používá kódování.

```
<?xml version="1.0" encoding="utf-8"?>
```

Dále jsou všechny tagy uzavřeny v tagu `<jnlp>`. Ten nám říká, jaká verze Java Network Launch Protocol je použita, odkud má být brán `.jar` soubor a jméno tohoto souboru, který by měl být umístěn ve stejném adresáři jako program.

```
<jnlp spec="1.0+" codebase="http://localhost:8080/"
   href="Test.jnlp">
```

Tag `<information>` uzavírá informace a nastavení, zobrazení a chování programu po tom, co je nainstalován. Tagy `<title>, <vendor>, <homepage>` a `<description>` obsahují informace o programu a tvůrci. Dobrovolný tag `<offline-allowed/>` říká instalaci, že program může být používán, i když není PC připojené k internetu. Pro nastavení ikon slouží tag `<shortcut>`, který obaluje tag `<desktop/>` pro instalaci ikony na plochu, `<menu>` pro přidání programu do menu. První tag `<menu>` zastupuje hlavní menu systému, vnořené tagy jsou vnořené složky a poslední je zástupce programu v menu.

```
<information>
   <title>Test Jnlp</title>
   <vendor>Vit Mazin</vendor>
   <homepage href="http://localhost:8080/" />
   <description>Testovaci program</description>
   <offline-allowed/>
   <shortcut online="false">
      <desktop/>
      <menu submenu="Mazin">
         <menu submenu="Test Jnlp"/>
      </menu>
   </shortcut>
</information>
```

Pro nastavení oprávnění programu slouží `<security>`, do kterého je vnořen tag s příslušnou úrovní práv jako například `<all-permissions/>`.

```
<security>
   <all-permissions/>
</security>
```

V `<resources>` je určeno, jaká verze Java runtime má být nainstalována a jméno archivu s programem.

```
<resources>
   <j2se version="1.6+" />
   <jar href="TestJnlp.jar" />
</resources>
```

Pro určení hlavní třídy je použit tag `<application-desc/>`

```
<application-desc main-class="TestJnlp" />
```

V tagu `<update/>` je určeno, kdy má program vyhledávat aktualizace a kdy je má instalovat.

```
<update check="always" policy="always"/>
```

Celý `JNLP` souboru, který program nainstaluje, vytvoří mu zástupce na ploše a v menu, nastaví spouštěcí třídu a zajistí sledování aktualizací, vypadá tedy takto:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://localhost:8080/"
    href="Test.jnlp">
   <information>
      <title>Test Jnlp</title>
      <vendor>Vit Mazin</vendor>
      <homepage href="http://localhost:8080/" />
      <description>Testovaci program</description>
      <offline-allowed/>
      <shortcut online="false">
         <desktop/>
         <menu submenu="Mazin">
            <menu submenu="Test Jnlp"/>
         </menu>
      </shortcut>
   </information>
   <security>
      <all-permissions/>
```

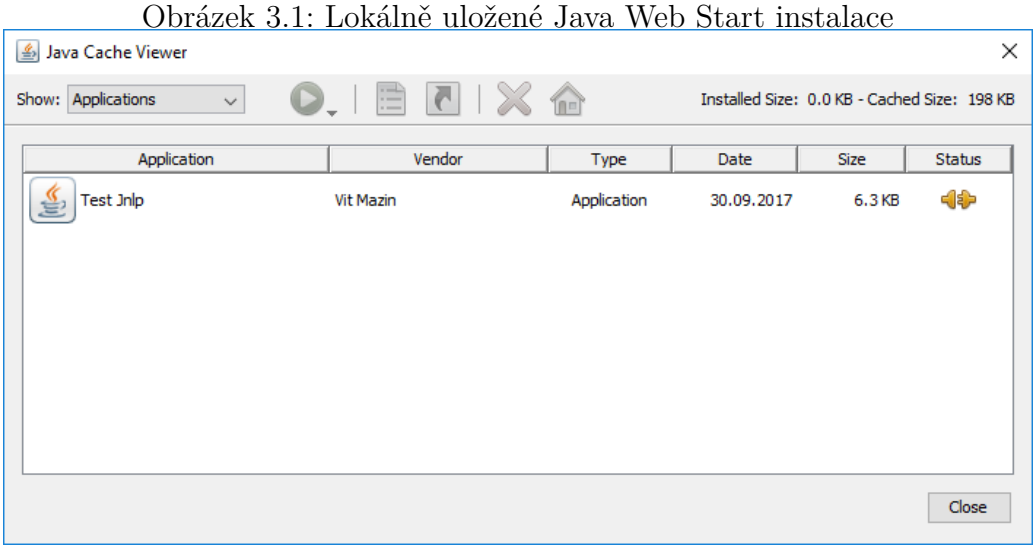
```
    </security>
    <resources>
        <j2se version="1.6+" />
        <jar href="TestJnlp.jar" />
    </resources>
    <application-desc main-class="TestJnlp" />
    <update check="always" policy="always"/>
</jnlp>
```

3.1.2 Instalace, běh a aktualizace programu

Instalace běžně probíhá tak, že se program poprvé spustí tak, že uživatel otevře odkaz, který vede na `JNLP` soubor daného programu. Ten po prvním spuštění zařídí sám vše potřebné.

Java Web Start celý program uloží do počítače, tudíž jsou další spuštění programu téměř okamžitá. Dle nastavení v `JNLP` souboru jsou vyhledány aktualizace. Pokud je nová verze programu k dispozici, je zajištěno její stažení a instalace bez toho, aniž by musel uživatel jakkoliv do tohoto procesu zasahovat.

Obrázek 3.1: Lokálně uložené Java Web Start instalace



3.2 ClickOnce

S vydáním Microsoft .NET Framework 2.0 byla vydána technologie ClickOnce pro instalaci programů vytvořených v tomto frameworku. ClickOnce

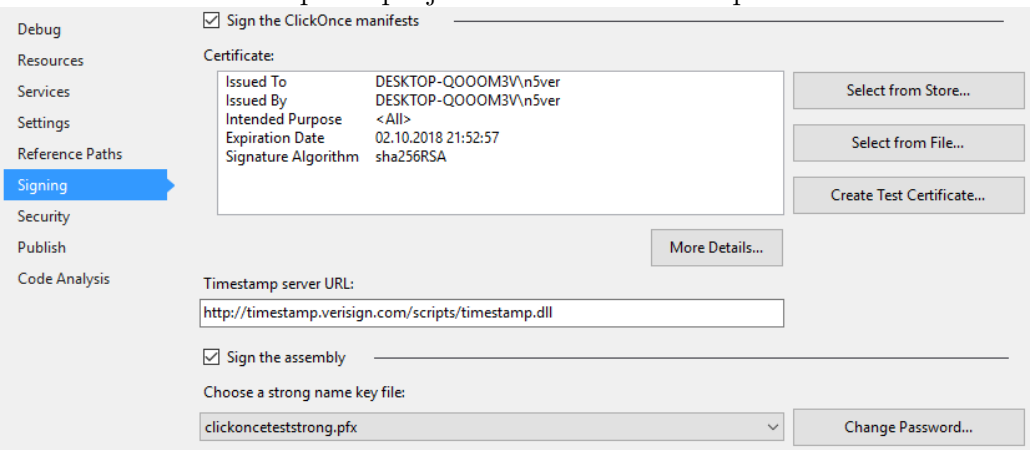
je součást .NET frameworku, proto není třeba pro vytvoření instalace ClickOnce instalovat separátně. Tato technologie byla vyvinuta, aby uživatele ušetřila dvou „uživatelsky nepříjemných“ činností a to instalace a aktualizace programu. Pro vydavatele programu ClickOnce přináší velké zjednodušení v podobě automatické aktualizace. Poznatky o technologii ClickOnce a tvorbě instalátoru pomocí ClickOnce čerpám především ze zdrojů [9] a [6].

3.2.1 Vytvoření instalace ClickOnce

Po naprogramování programu je třeba vytvořit soubory finální verze. Je tedy nutné z tzv. „debug“ módu přepnout ve Visual Studiu na „release“ mód, aby se připravily soubory, pro finální verzi programu.

Poté, co je program přeložený, je třeba ve vlastnostech projektu nastavit digitální podepisování programu a XML souboru, který obsahuje metadata, což je tzv. manifest. Vytvoření certifikátu na podepsání manifestu lze vytvořit snadno přímo ve Visual Studiu v nastavení projektu na záložce „Signing“ viz obrázek 3.2. Pro podepsání sestavení projektu je třeba vytvořit tzv. strong name key soubor, který obsahuje klíče a další potřebné informace pro podepsání. Tento podpis slouží k tomu, aby program získal unikátní identifikaci, aby byla zajištěna unikátnost a jiný software se za něj nemohl „vydávat“. Dále je třeba zadat internetovou službu věrné autority, která poskytne časovou značku, jež zajistí průkaznost data vzniku programu. K hashi vypočítaného z dat programu se u autority připojí tato značka a zašifruje se privátním klíčem. Nový hash je s časovou značkou vrácen od této autority. Poté lze jednoznačně prokázat čas vzniku použitím hashe, časové značky a veřejného klíče autority.

Obrázek 3.2: Podepsání projektu distribuovaného pomocí ClickOnce

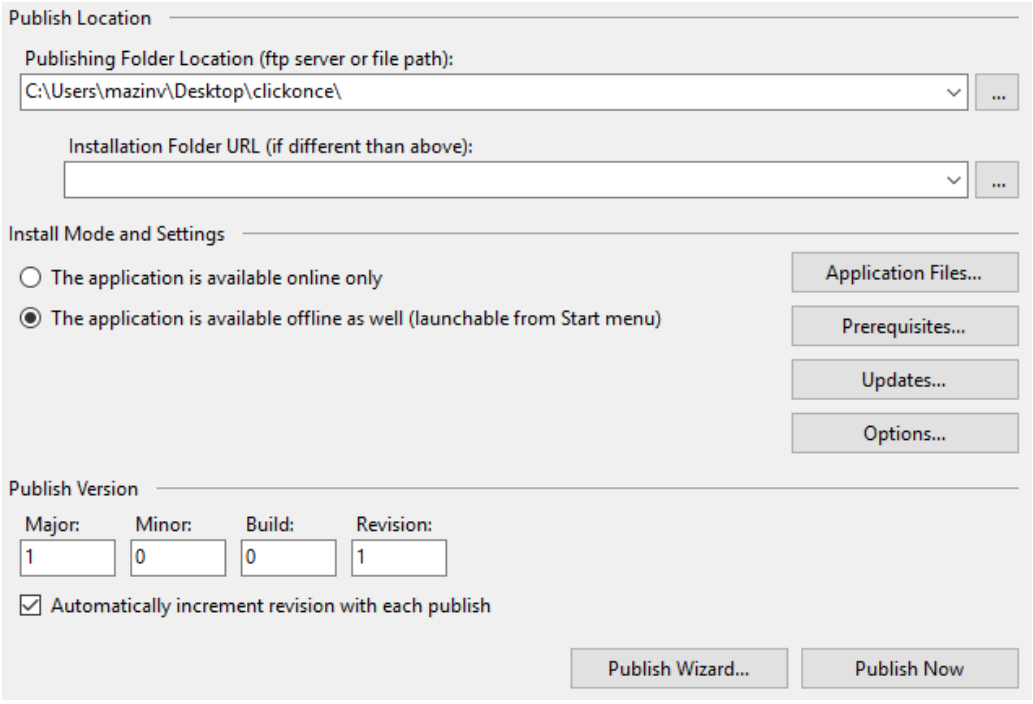


Pokud je podepisování připraveno, lze na záložce „Security“ jednoduše na-

stavit, jaký stupeň oprávnění program vyžaduje. Po vybrání „Enable ClickOnce security settings“ lze vybrat ze dvou stupňů oprávnění.

Když jsou nastavené body, které jsou popsané výše, zbývá nastavit, jak bude program vydán. Toto nastavení je na záložce „Publish“ viz obrázek 3.3. Zde lze v první části okna určit, kam se mají uložit soubory instalace. V další části se nachází možnosti, které určují, zda má být program přístupný pouze online nebo i offline, jaké soubory má obsahovat instalace, co musí mít uživatel nainstalováno v PC, zda má program vyhledávat aktualizace a případně jak. Poslední možnost v této části okna je nastavení pod tlačítkem „Options“. Zde jsou čtyři kategorie nastavení. V kategorii „Description“ lze vyplnit informace o tom, kdo program vydal atd. Pod kategorií „Deployment“ se skrývá nastavení, zda bude program instalován online z webového serveru, nebo je instalace určena pro distribuce například kompaktním diskem. V kategorii „Manifest“ lze specifikovat například, zda má být vytvořen zástupce na ploše atd. V poslední kategorii „File Associations“ lze nastavit, jaké typy souborů mají být s programem asociovány. V poslední části okna lze specifikovat verzi souboru, a zda se má verze při každém vydání inkrementovat. Pokud je vše výše specifikované, lze použít tlačítko „Publish Now“ a instalace bude vytvořena. Pro zjednodušené nastavení instalace lze stisknout tlačítko „Publish Wizard...“, které spustí průvodce nastavením instalace.

Obrázek 3.3: Publikování projektu distribuovaného pomocí ClickOnce



3.2.2 Instalace programu

Pokud byla vybrána online distribuce, stačí na vytvořené stránce kliknout na odkaz představující tlačítko pro spuštění instalace a instalace se spustí. Ta nejprve zkontroluje, zda je nainstalováno vše potřebné, a poté spustí jednoduché dialogové okno, kde se uživatel rozhodne, zda instalovat či ne. Stejným způsobem probíhá instalace z pevného média. Pouze pro spuštění není použita webová stránka, ale je použit soubor.

3.3 Výběr řešení

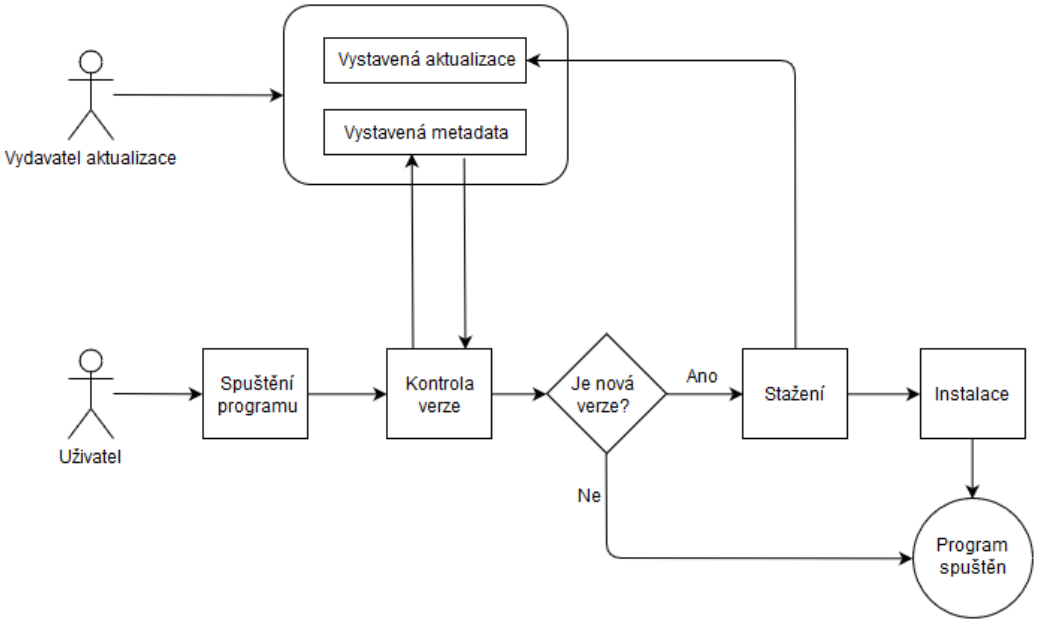
Výše popsaná řešení umožňují vytvořit program, který bude mít implementovaný proces automatické aktualizace. Nicméně z pohledu zadavatele se jeví jako nedostačující, protože jsou příliš přímočará a neposkytují příliš prostoru pro přizpůsobení. Z těchto důvodů byla po konzultaci se zadavatelem zvolena cesta tvorby vlastního řešení.

4 Návrh

4.1 Proces distribuce aktualizace

Pro vytvoření aktualizovatelného programu je třeba stanovit, jak bude proces probíhat. Proces je znázorněn na obrázku 4.1.

Obrázek 4.1: Proces distribuce aktualizace



1. Kontrola nové verze programu
2. Zastavení programu
3. Stažení nové verze
4. Spuštění instalace nové verze
5. Opakovaný start programu

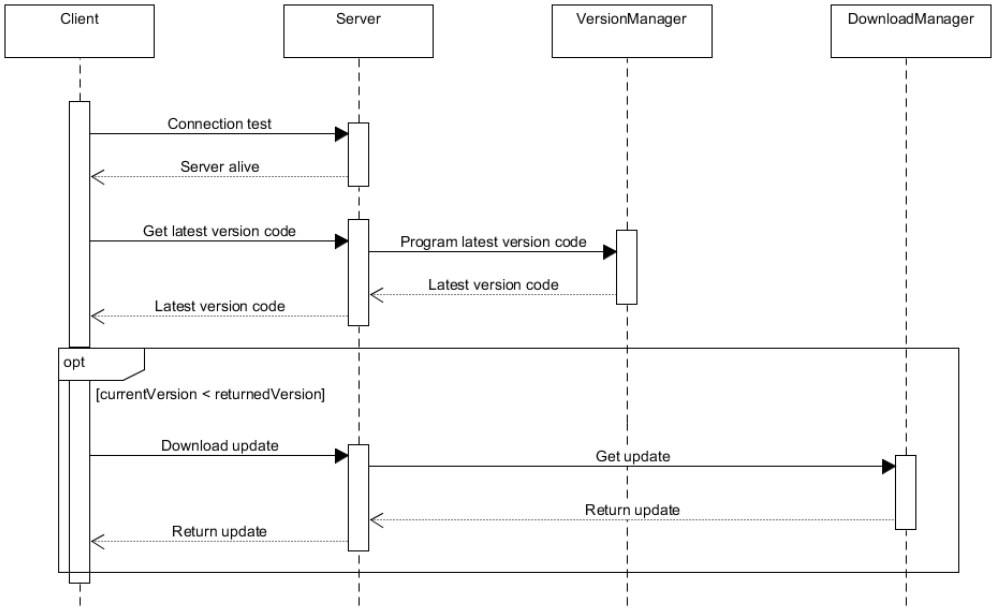
Program po spuštění zkontroluje dle vystavených metadat, zda je běžící verze aktuální. Pokud ano, program je spuštěn a uživatel může pracovat. Pokud je ovšem k dispozici nová verze, program je pozastaven, nová verze stažena a nainstalována a program je znovu spuštěn v aktuální verzi.

4.2 Komunikační protokol mezi serverem a klientem

Klient, který si bude kontrolovat, zda by neměl být aktualizovaný, musí komunikovat se serverem určitou danou pevnou formou. Proto je třeba navrhnout, jakým způsobem by spolu měli komunikovat.

Jako ideální forma komunikace mezi serverem a klientem se jeví komunikace na bázi REST API. Klient tedy bude zasílat na server HTTP požadavky a server mu na ně bude odpovídat. Využít protokolu HTTP je vhodné z toho důvodu, že má implementovaný přenos informací a také stahování souborů, čehož je vhodné využít. Sekvenční diagram komunikačního protokolu je znázorněn na obrázku 4.2.

Obrázek 4.2: Sekvenční diagram komunikačního protokolu



4.2.1 Test dosažitelnosti serveru

Klient po spuštění neví, zda je aktualizační server v provozu. Proto je potřeba implementovat volání, na které se vrátí jednoduchá odpověď, čímž se před další komunikací zjistí, zda je server dosažitelný. Pro toto volání se jeví jako dostačující HTTP metody GET. Pokud na server přijde požadavek GET na test spojení, server jednoduše zareaguje tak, že vrátí stavový kód „200 OK“.

4.2.2 Zjištění poslední vystavené verze

Základní operace, kterou musí aktualizační software provést, je zjištění poslední vystavené verze. Program bude muset odeslat požadavek, který bude říkat, na verzi jakého programu je server tázán. Tento požadavek je opět realizovatelný pomocí HTTP metody GET. Do URL se tedy doplní název programu a požadavek bude metodou GET na danou URL odeslán. Jako odpověď by měl server odeslat kódové označení verze programu, na jehož verzi je dotazován. Pokud server nebude mít záznam o programu daného jména, bylo by vhodné vrátit předem stanovenou hodnotu, dle které lze určit, že server neposkytuje aktualizace pro tento program nebo HTTP kód „404 Not Found“.

4.2.3 Stáhnutí aktualizace

Pokud program zjistí, že je na serveru vystavena aktualizace na novou verzi, musí aktualizační soubor stáhnout. Pro požadavek na stažení aktualizačního souboru lze opět využít HTTP metodu GET. V URL se předá jméno programu, který je třeba aktualizovat, a jako odezva přijde HTTP odpověď s aktualizačním souborem. Pokud bude server dotázán na stažení aktualizace pro program, jehož aktualizace server neposkytuje, lze v odpovědi předat příslušný HTTP stavový kód. Tento stavový kód může být například „404 Not Found“.

4.3 Klientská knihovna

Klientská knihovna byla v době psaní bakalářské práce určena primárně pro aktualizaci tzv. addinů do programu Enterprise Architect. Tento program je nástroj pro systémovou analýzu a návrh s využitím jazyka pro vizualizaci UML. Ačkoliv je program velmi komplexní a nabízí širokou paletu funkcí, je možné upravit si metodologii a s tím související funkce. Funkčnost, jež je dodávaná addinem, musí sedět vždy na danou verzi metodologie. Proto je nutné aktualizovat addiny. Když není uživatel na aktualizaci upozorňován nebo omezován v práci, a když nemá aktuální addin, může se snadno stát, že vznikne v projektu nekonzistence v datech. To může mít u rozsáhlého projektu velmi negativní dopad a způsobit velké zdržení kvůli opravě dat či dokonce jejich ztrátě.

Požadavek od zadavatele na knihovnu byl ten, aby byla knihovna snadno připojitelná a do kódu addinu přibylo jen nezbytné množství dalšího kódu, a aby nedošlo k znepřehlednění původního programu. Proto bylo nutné imple-

mentovat kontrolu spojení se serverem, poslední vystavené verze na serveru, stažení aktualizace a spuštění instalace přímo v knihovně a ne pouze využívat v programu addinu její funkce (mimo té která spustí kontrolu).

Pro zjištění aktuální verze bylo třeba prozkoumat možnosti, jak kódové označení verze získat. Jako ideální řešení se jeví úprava „Assembly version“, kterou lze za běhu programu zjistit a použít ji pro porovnání s verzí na serveru. Spolu s verzí je třeba dodat jméno programu. Pod dodaným jménem bude program hledán na serveru, proto slouží jako jakýsi identifikátor programu. To jsou jediné dvě věci, které musejí být knihovně dodány, aby byla schopna zkontrolovat aktualizace. Kontrola aktualizace addinu proběhne ihned po spuštění programu Enterprise Architect, protože je nutné případnou funkčnost addinů znepřístupnit, což je vhodné udělat, než se k použití addinů bude moci uživatel dostat.

Uživatel by měl být o průběhu procesu informován. Je tedy vhodné, aby byly použity informační boxy. Uživatel si může rozhodnout, zda v danou chvíli chce aktualizaci nainstalovat. K tomuto je využit dialogový box. Pokud se ovšem z nějakého důvodu rozhodne, že momentálně nechce addin aktualizovat, je mu addin znepřístupněn a uživatel je o tom náležitě informován.

Pokud je nutná instalace nové verze, je po odsouhlasení stažení a instalace aktualizace stažen aktualizační soubor. O toto stažení se stará knihovna. Ta po stažení informuje uživatele, že tento proces proběhl, a předá stažený soubor spouštěči instalace. Tímto činnost aktualizační knihovny končí.

Knihovna volá server, jehož adresa může být změněna. Z tohoto důvodu je adresu serveru nutné připojit k aktualizační knihovně (tedy i k addinu). Jakožto nejlepší způsob bylo zvoleno umístění do registrů systému Windows (přesné umístění je popsáno dále v části 5.4.1). Registry mají předem definovanou strukturu a hledání nastavení v nich je tudíž snazší než „parsování“ konfiguračního souboru.

4.4 Aktualizační server

Hlavním úkolem aktualizačního serveru je poskytovat informaci o poslední verzi daného programu a možnost stažení instalátoru aktualizace pro tento program. Server byl navržen tak, aby na základě filozofie REST odpovídal na HTTP požadavky klienta. Ačkoliv hlavním cílem práce bylo vytvořit knihovnu pro aktualizaci programu, bylo nutné ke knihovně navrhnout a vytvořit server, jenž by potřebná data knihovně poskytoval.

Protože server poskytuje informace o verzích a případně soubor ke stažení,

nebylo třeba hledat příliš komplexní a robustní řešení. Ačkoliv zvolená technologie ASP.NET Web API umožňuje vytvořit velmi robustní a výkonný server, lze s jejím využitím naprogramovat i menší projekt, jakým tento aktualizační server je.

Server byl navržen jakožto tzv. „self-host“ aplikace. Tento druh aplikací zajišťuje to, že pro vlastní běh nepotřebuje žádné další (nebo jen minimální) konfigurace a další programy. Tím se stává server snadno přenositelným a flexibilním, co se týče stroje, na kterém běží.

Server nevyžaduje propracované uživatelské rozhraní. Ve spuštěném programu serveru lze tedy program danými příkazy pouze vypnout nebo si vypsat nápovědu. Pro informace, co za požadavky server zpracovává, byly použity logy. Logy se ukládají do souboru, aby bylo možné odhalit případné chyby programu nebo zmapovat dotazy. Tyto logy jsou také ve zjednodušené formě vypsány uživateli na obrazovku. Uživatel tak má přehled o dotazech a úkonech, které server momentálně zpracovává, nebo které zpracovával.

Proces vystavení aktualizace na server je velmi důležitý a dá se řešit mnoha způsoby. Pro tento server byl zvolen co nejjednodušší, ale zároveň efektivní způsob vystavení aktualizací. Server má programy a aktualizace uložené v předem definované adresářové struktuře (popis vystavení aktualizací je popsán dále v části 5.3). Ukládání dat do adresářové struktury bylo zvoleno, protože nevyžaduje zastavení serveru a ani nijak jinak nezasahuje do jeho běhu při vystavování nové aktualizace. Server hledá data v adresářích na základě každého požadavku, a proto není struktura nijak uzamčená a dá se s ní pracovat i za běhu serveru.

4.5 Spuštění instalace

Kontrola aktualizací probíhá ihned po spuštění programu Enterprise Architect. To však znamená, že addiny, které jsou tvořeny jako knihovny („.ddl“ soubory), jsou již v paměti a nedá se s jejich soubory pracovat. Pro přeinstalování addinu je typicky nutné nahradit jeho soubory novými s upravenou funkčností. Knihovny addinů jsou po zavedení do paměti program Enterprise Architect zamčené. Nedají se tudíž nahradit novými. Proto bylo třeba vytvořit spouštěč instalace.

Spouštěč instalace informuje uživatele o tom, že čeká na vypnutí programu Enterprise Architect. Na vypnutí musí počkat, protože pouze po vypnutí jsou knihovny addinů uvolněny z paměti a lze je nahradit knihovnami nové verze. O spuštění spouštěče instalace aktualizace se stará knihovna pro automatickou aktualizaci. Ta stáhne instalaci aktualizace do dočasného adresáře

uživatele a předá cestu k instalaci aktualizace tomuto spouštěči.

Pouze předaná cesta k instalaci aktualizace by ovšem nestačila na to, aby mohl spouštěč vyčkávat na ukončení běhu programu Enterprise Architect. Po prozkoumání možností, jak zjistit, zda je spuštěn určitý proces, jsem dospěl k tomu, že aktualizační knihovna předá spouštěči tzv. „process ID“ programu Enterprise Architect. Veškeré procesy spuštěné v systému Windows mají přiřazeny tento jednoznačný identifikátor [14]. Protože je addin, do kterého je přidána knihovna pro aktualizaci, připojen přímo do programu Enterprise Architect, je identifikátor procesu, kterž si zjistí knihovna, stejný jako identifikátor procesu spuštěného programu Enterprise Architect. Se znalostí tohoto identifikátoru lze procházet všechny spuštěné procesy a kontrolovat jejich identifikátory. Pokud po průchodu celým seznamem všech spuštěných procesů není nalezen žádný s tímto zjištěným identifikátorem, lze s jistotou říci, že byl proces ukončen, a je tedy možné spustit instalaci aktualizace, aniž by byly knihovny addinů uzamčeny.

Spouštěč instalace aktualizace je okenní program, aby mohlo být uživateli zobrazeno okno o tom, že je vyčkáváno na ukončení Enterprise Architect. Kontrola, zda je stále spuštěn proces blokující knihovny, by byla tedy ve vlákně okna nevhodná, protože by se uživateli jevilo jako zaseknuté z důvodu neustálé kontroly seznamu procesů. Je tedy důležité, aby tato kontrola probíhala v jiném vlákně, které by neblokovalo vlákno okna, jež musí reagovat na případné uživatelské vstupy (kliknutí na okno atd.).

4.6 Instalace aktualizace

Pro instalaci aktualizací byl zvolen nástroj pro tvorbu instalátorů WiX. Tímto nástrojem tvoří firma CCA instalátory pro addiny. Při aktualizaci se addin přeinstaluje, čímž je stará verze plně nahrazena novou verzí. K tomuto způsobu instalace bylo přistoupeno po konzultaci s vedoucím práce, protože addiny jsou velmi malé programy. Celý instalátor takto malého programu je velikostně v řádech stovek kilobytů. Takto veliký soubor není při rychlosti ve firemní síti firmy CCA problém stáhnout ve zlomku sekundy, proto nebylo třeba vytvářet instalátor, jenž by speciálně aktualizoval pouze vybrané soubory.

Instalátor vytvořený nástrojem WiX je schopný detekovat, že jsou soubory, jež budou při instalaci ovlivněny, používány. Pokud toto detekuje, lze instalaci provést, ale je nutné restartovat počítač. Protože není žádoucí, aby byl počítač kvůli instalaci aktualizace addinu restartován, je tato instalace spouštěna přes spouštěč instalace (viz 4.5).

5 Realizace

5.1 Komunikační protokol

Komunikační protokol jsem navrhl tak, aby odpovídal dnes velmi rozšířenému REST API. REST API definuje, že pokud je na server zaslán požadavek na získání dat, měl by být požadavek zaslán HTTP metodou GET. Pokud je požadavek v pořádku a lze na něj vrátit data, jsou vrácena spolu s HTTP kódem „200 OK“. Při nesprávném požadavku na neexistující produkt by měl být vrácen HTTP kód „404 Not Found“ [12].

5.1.1 Test dosažitelnosti serveru

Na test dosažitelnosti serveru má server vyčleněnou jednu funkci. Požadavek je třeba odeslat HTTP metodou GET a funkce se volá takto (testováno na lokálním počítači, proto je použita adresa localhost):

```
http://127.0.0.1:9000/api/test
```

Pokud server odpoví, je vrácen HTTP kód „200 OK“.

5.1.2 Zjištění poslední vystavené verze

Pro zjištění poslední vystavené verze na serveru je zasílán požadavek HTTP metodou GET. Požadavek je zasílán na adresu v tomto tvaru:

```
http://127.0.0.1:9000/api/version/jmeno_programu
```

Server zkontroluje, zda má vystaven program se jménem, které bylo zasláno v URL. Při nalezení programu, server zašle odpověď, jež v těle obsahuje kódové označení poslední verze např.: `1.1.0.0`. HTTP kód odpovědi je „200 OK“. V případě, že server nemá žádné vystavené aktualizace pro dotazovaný program, je vrácena odpověď s HTTP kódem „404 Not Found“.

5.1.3 Stáhnutí aktualizace

Stažení aktualizačního souboru ze serveru je možné zasláním požadavku HTTP metodou GET na adresu v tomto tvaru:

```
http://127.0.0.1:9000/api/download/jmeno_programu
```

V případě, že server má vystavený aktualizační soubor pro program daného jména, je vrácena odpověď s kódem „200 OK“, která obsahuje v těle přílohu typu „`application/octet-stream`“. Tento typ přílohy je používán pro binární soubory a umožňuje za použití proudů daný soubor z těla odpovědi stáhnout. Tento požadavek stáhne aktualizační soubor pro poslední vystavenou verzi programu. Pro stažení specifické verze lze do předešlého požadavku doplnit označení specifické verze takto:

```
http://127.0.0.1:9000/api/download/jmeno_programu/w.x.y.z
```

Pro požadavek, jenž obsahuje chybné jméno či chybné označení specifické verze, je vrácena odpověď s HTTP kódem „404 Not Found“.

5.2 Aktualizační server

Aktualizační server je naprogramován jako Microsoft ASP.NET Web API aplikace. ASP.NET Web API aplikace jsou spuštěny obvykle jako internetové stránky a běží pod webovým serverem Microsoft IIS. Protože je tento způsob málo flexibilní a vyžaduje konfiguraci celého webového serveru, jsem se rozhodl použít pro tvorbu NuGet balíček `Microsoft.AspNet.WebApi.OwinSelfHost`, který program oprostí od IIS, ale zachová hlavní funkčnost ASP.NET Web API [8]. Server přijímá HTTP požadavky na předem dané URL (viz 5.1), které zpracuje a vrátí definovanou odpověď.

5.2.1 Připojení a konfigurace Web API

Balíček se instaluje zadáním následujícího příkazu do Package Manager Console ve Visual Studio:

```
Install-Package Microsoft.AspNet.WebApi.OwinSelfHost
```

S tímto balíčkem se doinstaluje do projektu vše potřebné pro správný běh Web API. Pouze stažením této knihovny je projekt založený jako konzolová aplikace schopný fungovat jako server, který odpovídá na HTTP požadavky, a již není potřeba instalovat žádný webový server [8]. Pro správný běh je pouze třeba mít nainstalovaný .NET Framework (ve verzi min. 4.5.2), bez kterého by ovšem nefungovaly ani programy vytvořené v jazyce C#.

Pro nakonfigurování Web API je vytvořena třída `Startup`, v níž je metoda `Configuration`, která je volána při spouštění serveru. V této metodě je vytvořena konfigurace, jež definuje, jakým způsobem jsou tvořené URL, a je v ní povoleno mapování atributů z URL. Toto mapování dovoluje dyna-

mičtější přizpůsobení URL, která nemusí být podle jednoho pevného vzoru a kterou lze snáze tvořit tak, aby byla aplikace tzv. RESTful. Konfigurační metoda tedy vypadá takto:

```
public void Configuration(IAppBuilder appBuilder)
{
   HttpConfiguration config = new HttpConfiguration();

    config.MapHttpAttributeRoutes();

    config.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}"
    );
    appBuilder.UseWebApi(config);
}
```

V URL je pevně nastaveno pouze to, že za adresou serveru musí následovat „`/api/jmeno_controlleru`“, kde `jmeno_controlleru` je název třídy, jež obsahuje naprogramovanou obsluhu daných HTTP požadavků.

Web API je poté spuštěno zavoláním metody `Start` třídy `WebApp`. Této metodě je předána třída s vytvořenou konfigurací a adresa s portem, na níž server bude spuštěn. Volání metody je zapouzdřené do příkazu `using` (volání vrací instanci třídy `WebApp`, v jehož těle je nekonečná smyčka pro stálý běh serveru). Tento příkaz zajistí, že po skončení kódu v jeho těle bude spuštěné Web API správně ukončené. Nekonečná smyčka je ukončena pouze po zadání daného příkazu (viz 5.1). Tato smyčka je zde z toho důvodu, aby server nebyl ukončen dříve, než si uživatel přeje, protože by byl ukončen hned po vykonání kódu v těle příkazu `using`. Start Web API je tedy volán takto:

```
using (WebApp.Start<Startup>(url: baseAddress))
{
   do
   {
      ...
   } while (!(command =
       Console.ReadLine().ToLower().Trim()).Equals("quit"));
}
```

5.2.2 Zpracování HTTP požadavků

URL, na kterou je požadavek zasílán, musí vždy obsahovat jméno třídy, jež obsluhuje požadavek. Tyto třídy definuje Wep API tak, že musejí mít za svým názvem slovo „Controller“ a musejí dědit od třídy `ApiController`. Jak vypadají URL pro jednotlivé požadavky je popsáno v 5.1.

Pro test, zda je server dostupný, je zaslán požadavek, jenž je zpracován třídou `TestController`. Ve třídě je metoda `Get` obsluhující HTTP metodu GET, jež na dotaz vrátí instanci třídy `HttpResponseMessage`, která představuje HTTP odpověď. Instanci se v konstruktoru přiřadí kód odpovědi „200 OK“ a ta je poté vrácena metodou `Get`.

Třída `VersionController` zpracovává požadavky na zjištění aktuální verze. Zde jsou opět vytvářeny instance třídy `HttpResponseMessage`, jež představují odpovědi na požadavky, a do jejich těla je vkládán kód poslední verze programu. Verze je zapsána v těle jako řetězec. Ten se přidá tak, že se upraví proměnná `Content` HTTP odpovědi. Do této proměnné je uložena instance třídy `StringContent`, která nese daný řetězec. V případě, že server daný produkt nemá vystavený, jsou vytvářeny odpovědi s kódem „404 Not Found“. `VersionController` využívá metody třídy `VersionManager`. Tyto metody procházejí složky s aktualizacemi a hledají nejnovější verzi. Vyhledávání funguje tak, že se nejdříve v hlavní složce „updates“ (jež je umístěna ve složce se serverem, viz 5.3) hledá složka s názvem programu, který byl předán v požadavku. Při nalezení takové složky je složka otevřena a vyberou se z ní všechny její podsložky. Ty jsou pojmenovány kódovým označením verze. Poté je seznam s podsložkami seřazen sestupně dle verze a je vrácen název první složky.

Požadavky na stažení aktualizace obsluhuje třída `DownloadController`. Zde je v metodě `Download` vytvářena odpověď na požadavky. Třída volá metodu `GetFileStreamAndName` třídy `DownloadManager`. Metoda vrací dvojici proměnných. První je proud otevřený nad souborem aktualizace a druhá je název tohoto souboru. Tato metoda pro získání proudu a jména využívá dalších privátních metod. Ty mají za úkol nalézt ve složkách s programy (umístěných ve složce „updates“) ten, který odpovídá dotazovanému programu. Ve složce programu poté naleznou složku s požadovanou verzí a vrátí proud vytvořený nad souborem v nalezené složce požadované verze a jméno souboru. Server může vrátit buď poslední verzi nebo specifickou verzi. Stažení specifické verze je momentálně nevyužité, ale je připraveno pro případnou potřebu stáhnout ze serveru jednu specifickou verzi. Do těla odpovědi je podobně jako v případě vracení řetězce verze uložen soubor a jeho jméno. Pro uložení souboru je do proměnné `Content` uložena instance třídy

`StreamContent`. Dále je v hlavičce odpovědi nutné nastavit typ dat, který je přenášen, a jméno souboru. Typ dat je nastaven na „application/octet-stream“. Takto vytvořená odpověď je poté odeslána zpět klientovi. Pokud není daný program nebo verze nalezena, je odpovědi nastaven kód „404 Not Found“ a je vrácena tato odpověď.

5.2.3 Uživatelské rozhraní a ovládání serveru

Činnost serveru je logována knihovnou NLog. Hlášky jsou vypisované do konzole a lze z nich tak vyčíst, co server zpracovává. Podrobnější informace jsou ukládány do souboru „logs.txt“, který je umístěn do složky spolu se serverem. Jak vypadá výpis do konzole, je znázorněno na obrázku 5.1.

Pro spuštění serveru je potřeba mu v argumentech předat číslo portu, na kterém má server běžet. Port se předá tak, že je program spuštěn s argumenty `-p cislo_portu`. Pokud jsou argumenty zadány špatně, program informuje uživatele o chybě s radou, jak zobrazit návod. Ten se vypíše tak, že je program spuštěn s argumentem `-h`.

Pokud server běží, lze do konzole napsat `quit` pro ukončení serveru nebo `help` pro vypsání nápovědy. Pokud je zadáno něco jiného, je uživatel informován o možnosti vypsání nápovědy.

Obrázek 5.1: Výpis informací ze serveru do konzole

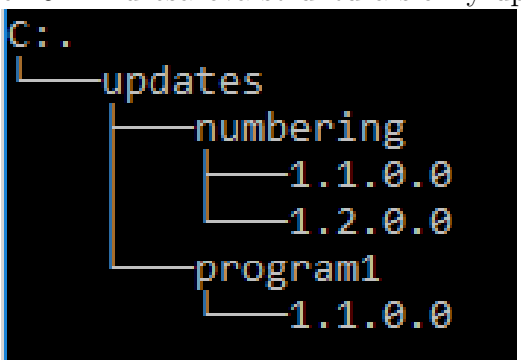
```
22:16:44 - Debug: Creating WebApp
22:16:45 - Info: Server started on port 9000...
22:17:33 - Trace: Version request for numbering
22:17:36 - Trace: Returning version for numbering - 1.2.0.0
22:18:02 - Info: Download request
22:18:08 - Trace: Request for numbering
22:18:08 - Trace: Returning latest update (1.2.0.0) for numbering
```

5.3 Vystavení aktualizací na server

Program „`UpdateServer.exe`“, který slouží jako aktualizační server, má ve své složce, kde je umístěný spolu s dalšími knihovnami potřebnými pro běh, vytvořenou složku `updates`. V této složce server po dotazu na verzi programu nebo požadavku na stažení aktualizačního souboru začíná vyhledávat. Ve složce jsou založené další složky, které mají jméno podle programu, jehož aktualizační soubory obsahuje. Je-li tedy server dotázán na poslední verzi programu „program1“, podívá se do složky `updates` a v ní hledá složku

se jménem stejným, jako bylo jméno dotazovaného programu. Složka daného programu obsahuje další složky s aktualizačními soubory. Jednotlivé složky mají název dle verze aktualizačního souboru, který obsahují. Aktualizační soubor je jediná věc, již složky označené verzí obsahují. Adresářový strom složky `updates` pro server, který má vystavené aktualizace pro programy `numbering` a `program1` je znázorněn na obrázku B.1

Obrázek 5.2: Adresářová struktura složky updates



5.4 Klientská knihovna

Klientská knihovna byla v době psaní bakalářské práce určena primárně pro aktualizaci tzv. addinů do programu Enterprise Architect. Protože addiny, které si zadavatel (tedy firma CCA) interně vyvíjí, jsou naprogramované v jazyce C#, rozhodl jsem se v tomto jazyce vyvíjet klientskou knihovnu.

Projekt byl založen v Microsoft Visual Studio jako Class Library, aby byl výsledek jedna přenosná .dll knihovna. Jmenný prostor knihovny je `UpdateManager`, ve kterém je umístěna hlavní veřejně přístupná třída `Manager`, která obsahuje veřejné metody. Další třídy této knihovny jsou neveřejné.

5.4.1 Vytvoření instance třídy `Manager` a kontrola verze

Pro vytvoření instance třídy `Manager` je nutné do konstruktoru předat informaci o názvu programu, podle kterého jsou aktualizace vyhledávány, a aktuální verze programu. Program si při vytváření instance zjistí adresu serveru, která je uložena v registrech systému Windows. Klíč v registrech obsahující základní adresu serveru je uložen pod touto cestou:

```
Computer\HKEY_CURRENT_USER\Software\CCA\jmeno_programu\BaseUrl
```

Místo části cesty označené jako „jmeno_programu“ je vždy jméno programu, které bylo předáno v konstruktoru třídy `Manager`. Hodnota, jež je uložena jako „Default“ pod klíčem „BaseUrl“ je základní adresa serveru, která vypadá například takto:

```
http://127.0.0.1:9000/api/
```

Adresa aktualizačního serveru je takto v registrech uložena pro každý program, protože je teoreticky možné mít pro různé skupiny programů různé aktualizační servery, které běží na jiné adrese. Registry byly zvoleny místo konfiguračního souboru, protože C# nabízí rozhraní pro práci s nimi. Lze je tedy dobře číst a zapisovat do nich. Díky tomu, že operace čtení z registrů je poskytnuta samotným jazykem, jevila se jako vhodnější než načítat informace ze souboru, který by bylo nutné parsovat. Pokud se programu nepodaří v registrech požadovanou informaci nalézt, je o tom uživatel informován informačním boxem.

Nad vytvořenou instancí lze zavolat metodu `StartCheck`, která vykoná proces kontroly a případně stáhne aktualizaci. Nejdříve je zjištěno, zda se programu povedlo z registrů získat adresu aktualizačního serveru. Pokud ano, je spuštěn test dosažitelnosti serveru. V případě, že nebyla nalezena adresa serveru, nebo se nepodařilo k serveru připojit, je i přes to Enterprise Architect spuštěn a addin je přístupný. K tomuto rozhodnutí bylo přistoupeno na základě domluvy se zadavatelem.

Program dále postupuje tak, že odešle na server dotaz, jaká poslední verze programu se na serveru nachází. Ze serveru přijde v odpovědi kódové označení poslední verze. Z něj se vyhodnotí, zda je třeba aktualizace. V případě, že přijde odpověď s HTTP kódem „404 Not Found“, znamená to, že server program nezná a uživatel je na to upozorněn. Pokud přijde chybná odpověď serveru, je addin přístupný. Přístupný je také v případě, že ho server nezná, protože nelze s jistotou určit, zda není chyba na straně serveru a program nemá na serveru žádné aktualizace. Pokud přišlo kódové označení verze, které je vyšší než aktuální verze, je uživatel dotázán, zda chce aktualizační soubor stáhnout a spustit instalaci. Uživatel je upozorněn, že nestažení a nenainstalování dostupné aktualizace znamená, že Enterprise Architect půjde používat, ale addin nebude přístupný.

Po odsouhlasení stažení a nainstalování aktualizace je uživatel upozorněn, že program začne aktualizační soubor stahovat. V případě, že se nepodaří stáhnout aktualizaci, je na to uživatel upozorněn a addin nelze používat.

Po stažení aktualizace je na dokončení stahování uživatel upozorněn a program Enterprise Architect je spuštěn. Spolu s ním je spuštěn i mnou vy-

tvořený program `UpdateSetupStarter.exe`, jehož úkol je čekat na vypnutí programu Enterprise Architect. Program bude dále vysvětlen (viz kapitola 5.5). Jakmile je Enterprise Architect vypnut, spustí se instalace aktualizace.

5.4.2 Zasílání dotazů na server a zpracování odpovědí

Dotazy na server jsou zasílané metodami HTTP GET. Jejich přesná forma je popsána v kapitole 5.1.

Pro vytvoření HTTP požadavku byla použita třída `HttpWebRequest`. Požadavek je vytvořen pomocí statické metody `Create` této třídy, které se jako argument předá adresa, na kterou má být požadavek zaslán. Dále je třeba upravit proměnnou `Method` vytvořené instance. Touto proměnnou je specifikováno jakou HTTP metodou má být požadavek zaslán. V tomto případě je tedy hodnota vždy nastavena na „GET“.

Nad vytvořenou a inicializovanou instancí třídy `HttpWebRequest` je poté zavolána metoda `GetResponse`, která vrátí instanci třídy `HttpWebResponse`. Z objektu odpovědi lze zjistit kontrolou hodnoty proměnné `StatusCode` HTTP stavový kód, čehož je využíváno, aby se dále zpracovávaly pouze odpovědi, které přijdou s kódem „200 OK“.

Přečíst data z těla odpovědi lze získáním proudu, který vrátí metoda `GetResponseStream`. V případě, že je zpracovávána odpověď na aktuální verzi programu, je kódové označení verze získáno tak, že se přečte první řádek těla odpovědi a ten se dále zpracovává jako řetězec. Pro čtení je použita metoda `ReadLine` třídy `StreamReader`, jež je zavolána nad instancí této třídy. Instance je vytvořena z původního získaného proudu.

Ovšem při zpracovávání odpovědi na stažení aktualizačního souboru je třeba zjistit nejdříve jméno původního souboru. To lze získat z proměnné `Headers` objektu odpovědi pod indexem „Content-Disposition“. Jakmile je jméno získáno, je před něj připojena časová značka, aby byly soubory odlišitelné. Pod vytvořeným názvem je aktualizační soubor uložen do uživatelovy dočasné složky. Soubor je z proudu přečten metodou `CopyTo` třídy `Stream`, která přečte všechna vstupní data a zapíše je do výstupního proudu.

Práce s dotazy na server a zpracovávání odpovědí jsou naprogramované v třídách `VersionHandler` a `DownloadHandler`. Metody těchto tříd jsou však volány pouze interně a nejsou přístupné jinak než uvnitř knihovny.

5.5 Spuštění aktualizace - `UpdateSetupStarter`

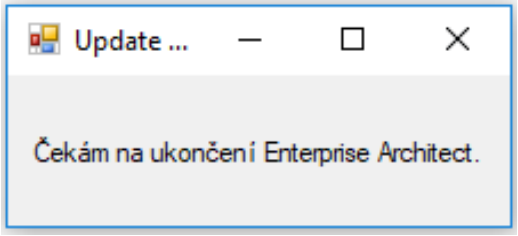
Úkol tohoto programu je vyčkat na vypnutí aktualizovaného programu, v našem případě tedy Enterprise Architect, a spustit instalaci aktualizace.

Na vypnutí Enterprise Architect je nutné čekat, protože addin, který má být aktualizován, je vytvořen jako knihovna, jež je připojena do Enterprise Architect. Když je knihovna využívána, jsou její soubory zamčené a nedá se je nahradit. Proto je třeba vyčkat, až budou soubory uvolněny z paměti. V tomto případě jsou uvolněny, když není Enterprise Architect spuštěný.

Pro spuštění programu je třeba mu předat dva argumenty. První z nich je process ID programu, na jehož ukončení je nutné čekat, a druhý je cesta k instalátoru aktualizace.

Program je vytvořený jako Windows Forms Application. V hlavní metodě ve třídě `Program` je vytvořena instance třídy `MainForm`, nad kterou je zavolána metoda `StartCheck` této třídy, které se předají argumenty, s nimiž byl program spuštěn. V metodě je zkontrolováno, zda je předaný PID opravdu číslo a zda instalace na předané cestě existuje. Poté se spustí okno programu, které informuje uživatele o tom, že program čeká na vypnutí Enterprise Architect (viz obrázek 5.3).

Obrázek 5.3: Okno programu UpdateSetupStarter



V případě že nejsou podmínky splněny, je zobrazen uživateli informační box a program se ukončí. V opačném případě je spuštěno další vlákno, jež vykonává metodu `CheckThread` [1]. V této metodě je nekonečný cyklus, ve kterém se kontroluje, zda je daný PID mezi PIDy všech procesů. Kontrola probíhá po 500 milisekundách.

Jakmile program zjistí, že se již mezi běžícími procesy nenachází proces s daným PID, je přerušen nekonečný cyklus a je spuštěna instalace z předané cesty a program `UpdateSetupStarter` se ukončí.

5.6 Připojení klientské knihovny k addinu a jeho instalaci

Vytvořenou klientskou knihovnu spolu s programem pro spuštění je nutné integrovat do daného programu a do jeho instalace, která při vydání nové verze představuje aktualizační soubor.

5.6.1 Připojení knihovny do projektu addinu

Do referencí daného C# projektu je třeba přidat soubor `UpdateManager.dll` a direktivou `using` přidat ve třídě, kde se bude knihovna využívat, referenci na jmenný prostor `UpdateManager`. S nastavenými referencemi je zpřístupněna třída `Manager`.

Programy, které jsou programovány jako addiny do programu Enterprise Architect, musejí implementovat osm metod, které zpracovávají události vytvářené programem Enterprise Architect [4]. Po prostudování událostí jsem se rozhodl implementovat kontrolu aktualizace do metody `EA_Connect`. Tato metoda je volána, poté co je program Enterprise Architect spuštěn.

V těle metody je tedy třeba vytvořit instanci třídy `Manager`. Do konstruktoru je předán název programu, pod kterým jsou vystaveny aktualizace na aktualizačním serveru a pod kterým je uložena v registrech informace o adrese serveru. Dále je nutné předat konstruktoru třídy informaci o aktuální verzi addinu. Před každým vydáním nové verze addinu by měla být zvýšena Assembly Version. Assembly Version je verze, která je používána .NET Frameworkem za běhu programu. Díky Assembly Version může program za běhu snadno zjistit svoji verzi například takto:

```
string version =
    Assembly.GetExecutingAssembly().GetName().Version.ToString();
```

Takto získané označení verze se poté použije pro zjištění, zda neexistuje nová verze.

Programátor tedy při vydání nové verze zvýší označení stávající verze v souboru `AssemblyInfo.cs` a vytvořenou instalaci s novou verzí programu nahraje na server pod označením této nové verze. Při dodržení tohoto postupu aktualizační knihovna vždy získá aktuální verzi běžícího addinu a pokud je na serveru nová verze, bude správně vyhodnocena potřeba aktualizace.

Nad vytvořenou instancí poté stačí následovně zavolat metodu `StartCheck`:

```
Manager manager = new Manager(programName, version);
enable_addin = manager.StartCheck();
```

Metoda provede proces zjištění a případného stažení s instalací aktualizace. Návratová hodnota metody je booleovská hodnota, která indikuje, zda by měl být addin přístupný nebo ne.

Tuto hodnotu je poté třeba uložit jako privátní proměnnou. Proměnná je poté použita v podmínce, která se umístí na začátek těla metody `EA_GetMenuState`. `EA_GetMenuState` je metoda, která zpřístupňuje položky

menu daného addinu. Přes tyto položky je využívána funkčnost addinu. Pokud by tedy měl být nepřístupný, lze tyto položky „zakázat“. Toho lze tedy docílit přidáním této podmínky na začátek metody (proměnná `enable_addin` je uložená návratová hodnota metody `StartCheck`):

```
if (!enable_addin)
{
   IsEnabled = false;
   return;
}
```

Proměnná `IsEnabled` indikuje, zda mají být položky v menu použitelné. Pokud je nastavena a booleovskou hodnotu `false`, je položka šedivá a uživatel na ni nemůže kliknout.

5.6.2 Připojení knihovny do instalace

Firma CCA pro instalaci svých addinů využívá volně dostupnou sadu nástrojů WiX. Nástroj se musí nainstalovat a existuje k němu rozšíření pro Microsoft Vistual studio, díky kterému lze instalaci vyvíjet v tomto prostředí a překlad funguje jako překlad programů. Primárně je projekt instalátoru tvořen dvěma soubory s příponou .wxs, které jsou ovšem známé .xml jen s jinou příponou. Já jsem provedl úpravu pouze v souboru `Files.wxs`, jenž definuje, jaké soubory se mají kam nainstalovat.

Byl mi poskytnut testovací projekt s kostrou addinu, na kterém jsem testoval integraci s knihovnou a projekt pro tvorbu instalace. Do souboru `Files.wxs` byl doplněn soubor s klientskou knihovnou `UpdateManager.dll` a program pro spuštění instalace aktualizace `UpdateSetupStarter.exe`. Program musí být nainstalován spolu s knihovnou ve stejné složce. Definice souborů byla doplněna do skupiny, která sdružuje kopírované soubory. Doplněné řádky vypadají takto:

```
<File Source="..\libs\UpdateManager.dll" KeyPath="no"
    Name="UpdateManager.dll" />
<File Source="..\libs\UpdateSetupStarter.exe" KeyPath="no"
    Name="UpdateSetupStarter.exe" />
```

Dále byla doplněna do komponenty, která sdružuje vytváření klíčů a hodnot v registrech, tvorba klíče a hodnoty pro zjištění adresy serveru. Kde a jakým způsobem musí být tato informace v registrech uložena je vysvětleno v kapitole 5.4.1. Doplněný kód vypadá takto:

```
<RegistryKey Root="HKCU" Key="Software\CCA\numbering\BaseUrl"
    ForceCreateOnInstall="yes" ForceDeleteOnUninstall="yes">
   <RegistryValue Type="string" Value="http://127.0.0.1:9000/api/"
      />
</RegistryKey>
```

6 Testování

6.1 Testování serveru

Pro ověření funkčnosti serveru byl vytvořen projekt `ServerTest`, který byl založen jako „Visual C# Unit Test Project“. V tomto projektu jsou vytvořené třídy, které obsahují metody samotných testů. Každá třída testuje tzv. kontrolér serveru. Kontroléry jsou třídy, v nichž je naprogramována obsluha HTTP metod. V projektu serveru jsou tři takovéto kontroléry, proto obsahuje testovací projekt také tři třídy.

Testy jsou koncipovány jakožto Unit testy. Vždy před testem se vytvoří objekt daného kontroléru, jehož metody jsou poté volány s příslušnými parametry. Zkoumány jsou instance třídy `HttpResponseMessage`, které jsou z metod vráceny. Jsou testovány hlavně stavové HTTP kódy odpovědí a případný obsah v těle odpovědí (pokud nemá být vrácen pouze stavový kód).

6.2 Testování klientské knihovny a spouštěče aktualizací

Pro testování klientské knihovny a spouštěče instalace aktualizací byly vytvořeny testovací scénáře pro manuální funkční testy. Manuální testování klientské knihovny bylo zvoleno, protože proces aktualizace je řízen dialogovými boxy, které čekají na vstup od uživatele. Spouštěč instalace aktualizace je otestován v rámci testování klientské knihovny. Jinak než manuálními testy v rámci testování knihovny testován nebyl, protože čeká na vypnutí procesu, jehož PID mu bylo předáno při startu.

6.2.1 Testovací scénáře

Scénáře předpokládají nainstalovaný program Enterprise Architect s upraveným addinem, .NET Framework 4.5.2 a možnost spuštění aktualizačního serveru. Po úspěšné aktualizaci je třeba opět nainstalovat addin verze 1.1.0.0.

Úspěšné aktualizování

Tento test ověřuje správnou funkčnost celého procesu aktualizace.

Kroky:

1. Spusťte server.
2. Spusťte Enterprise Architect.
3. Při dotazu na povolení aktualizace stiskněte „Ano“.
4. Stiskněte „OK“ u informativního boxu o následovném stahování.
5. Stiskněte „OK“ u informativního boxu o nutnosti vypnutí Enterprise Architect.
6. Potvrďte spuštění programu UpdateSetupStarter.exe.
7. Vypněte Enterprise Architect.
8. Vyčkejte na spuštění instalace aktualizace.
9. Nainstalujte aktualizaci.

Očekávaný výsledek:

1. Addin bude aktualizován.
2. Po opětovném spuštění Enterprise Architect a spuštění libovolného projektu bude funkčnost addinu CCA Číslování v záložce Extend přístupná.

Odmítnutí stažení instalace aktualizace

Test ověřuje nepřístupnost addinu po odmítnutí stažení aktualizace.

Kroky:

1. Spusťte server.
2. Spusťte Enterprise Architect.
3. Při dotazu na povolení aktualizace stiskněte „Ne“.

Očekávaný výsledek:

1. Po otevření libovolného projektu nebude funkčnost addinu CCA Číslování v záložce Extend přístupná.

Vypnutí spouštěče instalace

Test ověřuje opětovný dotaz na aktualizaci po tom, co uživatel vypne spouštěč aktualizace před tím, než bude instalace aktualizace spuštěna.

Kroky:

1. Spusťte server.
2. Spusťte Enterprise Architect.
3. Při dotazu na povolení aktualizace stiskněte „Ano“.
4. Stiskněte „OK“ u informativního boxu o následovném stahování.
5. Stiskněte „OK“ u informativního boxu o nutnosti vypnutí Enterprise Architect.
6. Potvrďte spuštění programu UpdateSetupStarter.exe.
7. Vypněte program UpdateSetupStarter.exe.

Očekávaný výsledek:

1. Po otevření libovolného projektu nebude funkčnost addinu CCA Číslování v záložce Extend přístupná.
2. Po opětovném spuštění Enterprise Architect bude opět zobrazena výzva k aktualizování addinu.

Zamítnutí spuštění spouštěče instalace

Test ověřuje opětovný dotaz na aktualizaci po tom, co uživatel nepovolí spuštění spouštěče aktualizace.

Kroky:

1. Spusťte server.
2. Spusťte Enterprise Architect.
3. Při dotazu na povolení aktualizace stiskněte „Ano“.
4. Stiskněte „OK“ u informativního boxu o následovném stahování.
5. Stiskněte „OK“ u informativního boxu o nutnosti vypnutí Enterprise Architect.
6. Zamítněte spuštění programu UpdateSetupStarter.exe.

Očekávaný výsledek:

1. Bude zobrazeno dialogové okno informující o tom, že UpdateSetupStarter.exe se nepodařilo spustit.
2. Po otevření libovolného projektu nebude funkčnost addinu CCA Číslování v záložce Extend přístupná.
3. Po opětovném spuštění Enterprise Architect bude opět zobrazena výzva k aktualizování addinu.

Server je nedostupný

Test ověřuje reakci na nedostupnost serveru.

Kroky:

1. Nezapínejte server.
2. Spusťte Enterprise Architect.

Očekávaný výsledek:

1. Bude zobrazen informativní box o tom, že se nepodařilo navázat spojení se serverem.
2. Po otevření libovolného projektu bude funkčnost addinu CCA Číslování v záložce Extend přístupná.

7 Závěr

Cílem této bakalářské práce bylo navrhnout a vytvořit aktualizační software pro doplňkové programy tvořené společností CCA, jež rozšiřují funkcionalitu programu Enterprise Architect tak, aby v programu bylo možné použít vlastní metodiku společnosti CCA.

Na počátku práce jsem se seznámil s historií a vývojem procesu automatické aktualizace. Dále jsem prozkoumal aspekty procesu, a jakými způsoby je aktualizování obecně řešeno. Na konci první části jsem také popsal, jaká je motivace společnosti CCA k tomu, aby své programy obohatili o funkčnost automatického aktualizování.

Po seznámení se s obecnými principy a postupy týkajících se automatických aktualizací jsem prozkoumal volně dostupná řešení pro přístupné implementování tohoto procesu. Řešení jsou vázána na programovací jazyk, v němž je program, který vyžaduje aktualizaci, naprogramován. Tyto řešení se vztahují na programovací jazyk Java a jazyky zastřešené platformou Microsoft .NET. Řešení jsou popsána formou návodu, jak je implementovat do vlastního programu. Řešení ClickOnce vytvořené firmou Microsoft pokrývá software vytvořený speciálně pro platformu Microsoftu a řešení Java Web Start pokrývá veškeré platformy, na nichž je možné programy napsané v jazyce Java provozovat.

Dále jsem popsal mnou navržené řešení, které je vytvořené dle požadavků společnosti CCA. V návrhu byly popsány všechny programy, které jsem vytvořil, a související komunikační protokol. Součástí tohoto návrhu je také diagram, jenž znázorňuje obecně proces aktualizace, kterým jsem se při práci řídil a diagram komunikačního protokolu.

V předposlední části jsem popsal jakým způsobem, jsou programy implementovány. Spolu s tím bylo také popsáno výsledné řešení komunikačního protokolu mezi serverem a aktualizační knihovnou. Aktualizační knihovna byla vytvořena s důrazem na to, aby pro její připojení bylo potřeba co nejmenších úprav kódu aktualizovaných programů. Server vystavující aktualizace, jenž byl v rámci této práce vytvořen, nebyl primárním zadání práce. Proto na serveru není řešena otázka bezpečnosti jako je například autentizace a šifrování protokolem SSL. Momenty, které vedou k znepřístupnění funkčnosti aktualizovaného programu například z důvodu nenainstalování nové verze nebo nedostupnost serveru, byly probrány se zadavatelem a dle požadavků bylo nastaveno za jakých podmínek bude funkčnost přístupná a za jakých ne. Tato funkčnost je nicméně dobře upravitelná a dále záleží na

zadavateli, jak se při aktivním využívání aktualizační knihovny rozhodne, za jakých podmínek mají být funkce programu přístupné.

Nakonec jsem popsal, jakým způsobem byly programy otestovány. Pro manuální testy byly vytvořeny a popsány testovací scénáře. Práce byla testována funkčními testy z důvodu nutnosti běhu programu Enterprise Architect, ve kterém není možnost krokovat doinstalované programy. Touto poslední částí bylo splněno zadání práce a veškeré požadavky zadavatele na funkčnost.

Přehled zkratek

- API - Application Programming Interface
- ASP - Active Server Pages
- DLL - Dynamic-link library
- HTTP - Hypertext Transfer Protocol
- JAR - Java Archive
- JNLP - Java Network Launch Protocol
- JRE - Java Runtime Environment
- JSP - JavaServer Pages
- PC - Personal Computer
- PID - Process identifier
- REST - Representational State Transfer
- UML - Unified Modeling Language
- URL - Uniform Resource Locator
- WXS - WiX Source File
- XML - Extensible Markup Language

Literatura

[1] Agafanov, E. *Multithreading in C# 5.0 Cookbook*. Packt Publishing Ltd., 2013. ISBN 978-1-84969-764-4.

[2] Bernedo, A. *All You Need to Know About Hacking Your PS Vita: VHBL, TN-V, Exploits, Emulators Explained* [online]. [cit. 2017/10/19]. Dostupné z: https://www.guidingtech.com/33224/hacking-ps-vita-emulators/.

[3] Ing. Pavel Herout, P. *Java Web Start - tvorba instalátoru* [online]. doc. Ing. Pavel HEROUT, Ph.D. [cit. 2017/10/20]. Výzkum doc. Ing. Pavel HEROUT, Ph.D. Dostupné z: https://www.kiv.zcu.cz/~herout/pruzkumy/JWS/htmls/pridani.html.

[4] *Enterprise Architect Guide* [online]. Sparx Systems. [cit. 2018/03/28]. Dostupné z: https://www.sparxsystems.com/enterprise_architect_user_guide/13.5/automation/creatingaddins.html.

[5] Fisk, D. *Programming with Punched Cards* [online]. Columbia University. [cit. 2017/10/19]. Dostupné z: http://www.columbia.edu/cu/computinghistory/fisk.pdf.

[6] Hashimi, S. *Deploying .NET Applications: Learning MSBuild and ClickOnce (Expert's Voice in .NET)*. Apress, 2006. ISBN 978-1590596524.

[7] *Java Web Start* [online]. Oracle. [cit. 2017/10/20]. Java. Dostupné z: https://www.java.com/en/download/faq/java_webstart.xml.

[8] Mehrotra, K. *Use OWIN to Self-Host ASP.NET Web API 2* [online]. Microsoft, 2013. [cit. 2018/04/10]. Dostupné z: https://docs.microsoft.com/en-gb/aspnet/web-api/overview/hosting-aspnet-web-api/use-owin-to-self-host-web-api.

[9] *ClickOnce – zabezpečení a nasazení* [online]. Microsoft. [cit. 2017/10/20]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/t71a733d.aspx.

[10] *Signing JAR Files Used in Java Web Start* [online]. Oracle, 2010. [cit. 2017/10/20]. Java Documentation. Dostupné z: https://docs.oracle.com/cd/E19501-01/819-3659/gcjlg/index.html.

[11] Pogue, D. *Mac OS 9: The Missing Manual*. O'Reilly Media, Inc., 2000. ISBN 1565928571.

[12] *REST API Tutorial* [online]. [cit. 2018/03/19]. Dostupné z: http://www.restapitutorial.com/lessons/httpmethods.html.

[13] Rogers Cadenhead, L. L. *Sams Teach Yourself Java 6 in 21 Days, 5th Edition*. SAMS Publishing, 2007. ISBN 978-0-672-32943-2.

[14] Stone, D. *What Is the PID in Task Manager?* [online]. Leaf Group Ltd. Leaf Group Media, 2015. [cit. 2018/04/03]. Dostupné z: https://www.techwalla.com/articles/what-is-the-pid-in-task-manager.

[15] *Description of Automatic Updates in Windows Millennium Edition (Me)* [online]. Microsoft. [cit. 2017/10/19]. Windows ME update. Dostupné z: https://support.microsoft.com/en-us/help/268331/description-of-automatic-updates-in-windows-millennium-edition-me.

Seznam obrázků

A Integrace knihovny do addinu Numbering

Výsledný projekt s integrovanou knihovnou je přiložen na CD k bakalářské práci. Dále je přiložen návod, jak připojit knihovnu k programu a poté k instalaci (úprava byla provedena ve vývojovém prostředí SharpDevelop 5.1).

A.1 Připojení knihovny do programu

Uvedený návod vysvětluje, jak do existujícího programu zavést knihovnu pro kontrolu a provádění automatických aktualizací.

1. Do referencí projektu připojte referenci na `UpdateManager.dll`.
2. V hlavní třídě `Main` přidejte direktivu `using UpdateManager;`.
3. V metodě `EA_Connect` získejte aktuální verzi programu pomocí `Assembly.GetExecutingAssembly().GetName().Version.ToString();`.
4. V téže metodě vytvořte instanci třídy `Manager`, jejímuž konstruktoru je v parametrech předáno jméno programu (pod nímž je uložen i na serveru) a získaná verze.
5. Zavolejte metodu `StartCheck` vytvořené instance a uložte si do privátní booleovské proměnné její návratovou hodnotu.
6. Do metody `EA_GetMenuState` přidejte podmínku vyhodnocující návratovou hodnotu metody `StartCheck`; Pokud je hodnota nastavena na `false`, je nastavena proměnná `IsEnabled` také na `false` a je zavolán příkaz `return`.

Listing A.1: Výsledná úprava třídy `Main`

```
using UpdateManager;

namespace EnterpriseArchitect.Numbering
{
    public class Main
    {
```

```
        ...

        const string programName = "numbering";

        private bool enable_addin = false;

        public String EA_Connect(EA.Repository Repository)
        {
            string version =
                Assembly.GetExecutingAssembly().GetName().Version.ToString();

            Manager manager = new Manager(programName, version);
            enable_addin = manager.StartCheck();

            return " ";
        }

        ...

        public void EA_GetMenuState(EA.Repository Repository,
            string Location, string MenuName, string ItemName, ref
            bool IsEnabled, ref bool IsChecked)
        {
            if (!enable_addin)
            {
                IsEnabled = false;
                return;
            }

            ...

        }

        ...

    }
}
```

A.2 Úprava instalace

Uvedený návod ukazuje, jak upravit nastavení instalace programu, aby se spolu s ním instalovala knihovna pro automatickou aktualizaci.

1. V souboru `Files.wxs` přidejte k souborům addinu také `UpdateManager.dll` a `UpdateSetupStarter.exe`.
2. Do komponenty s registry přidejte tvorbu klíče `Computer\HKEY_CURRENT_USER\Software\CCA\jmeno_programu\BaseUrl` jehož hodnota je URL server, kam se posílají dotazy (např. `http://127.0.0.1:9000/api/`).

Listing A.2: Výsledný soubor `Files.wxs` instalace pro addin Numbering

```
<?xml version="1.0"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
   <Fragment>
      <DirectoryRef Id="TARGETDIR">
         <Directory Id="ProgramFilesFolder" Name="PFiles">
            <Directory Id="CCADIR" Name="CCA">
               <Directory Id="INSTALLDIR" Name="CislovaniElementu">
                  <Component Id="CislovaniElementuLib"
                      Guid="E0E693E1-713B-404A-BD60-300732AFE544">

                     ...

                     <File Source="..\libs\UpdateManager.dll"
                        KeyPath="no" Name="UpdateManager.dll" />
                     <File Source="..\libs\UpdateSetupStarter.exe"
                        KeyPath="no" Name="UpdateSetupStarter.exe"
                        />

                     ...

                  </Component>
                  <Component Id="CislovaniElementuReg"
                      Guid="9F4C1942-89EB-417F-A112-F8E7AE4D51BC">

                     ...

                     <RegistryKey Root="HKCU"
                        Key="Software\CCA\numbering\BaseUrl"
                        ForceCreateOnInstall="yes"
                        ForceDeleteOnUninstall="yes">
```

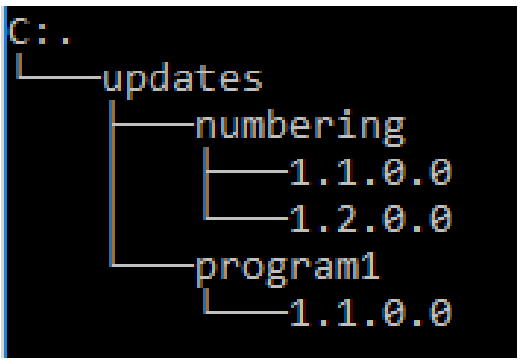
```
                    <RegistryValue Type="string"
                        Value="http://127.0.0.1:9000/api/" />
                </RegistryKey>
            </Component>
        </Directory>
    </Directory>
</Directory>
</DirectoryRef>
</Fragment>
</Wix>
```

B Vystavení programu a aktualizací

Programy a aktualizace lze dále popsaným postupem na server vystavovat za běhu serveru. Restart serveru nebo jeho stopnutí pro tyto úkony není nutný.

1. Otevřete složku `updates` umístěnou v adresáři serveru.
2. Ve složce založte pro program novou složku, která se bude jmenovat dle jména programu (např. numbering).
3. Pro každou verzi založte ve složce programu novou složku se jménem, které odpovídá kódovému označení verze (např. 1.1.0.0, 1.2.0.0, 1.2.0.1,...).
4. Do vytvořené složky pojmenované verzí umístěte instalátor aktualizace.

Obrázek B.1: Adresářová struktura složku updates s vystaveným programem `numbering` a `program1`



C Obsah přiloženého CD

- obsah.txt - Popis obsahu CD
- BP_Mazin_A15B0092P.pdf - PDF bakalářské práce
- zdrojove_kody/AutomaticUpdate/ - C# projekty vytvořené v rámci BP
- zdrojove_kody/enterprise-architect-cca-addins/ - testovací projekty Enterprise Architect addinu a jeho instalace dodané zadavatelem
- zdrojove_kody/latex/ - zdrojové kódy textu BP
- funkcni_ukazka/easetup.msi - instalace trial verze Enterprise Architect
- funkcni_ukazka/ukazkovy_addin_1.1.0.0.msi - instalace upraveného addinu do Enterprise Architect
- funkcni_ukazka/server/ - složka s aktualizačním serverem
- funkcni_ukazka/navod.txt - návod na zprovoznění Enterprise Architect s testovacím addinem a spuštění serveru