ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Návrh a prototypová implementace nástroje pro instalace a aktualizace webových Java aplikací
Student:	Petr Gondek
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je prototyp nástroje pro zjednodušení procesu instalace a aktualizace webových aplikací v prostředí podniku. Jde zejména o řešení konfliktů ve verzích požadovaných dílčích systémů/komponent a jejich konfigurací. Nástroj bude pracovat v těchto krocích:
1)Porovná aktuální konfiguraci v místě instalace (v případě upgrade) s požadavky a konfigurací nové verze aplikace, která se má instalovat.
2)Zobrazí uživateli případné konflikty v konfiguraci (řešení konfliktů není součástí prototypu).
3)Provede samotnou instalaci resp. upgrade a zobrazí protokol o výsledku.

Zadání řešte na konkrétní aplikaci-Manta Tools, při analýze a návrhu však pracujte pokud možno tak, aby výsledek mohl být zobecněn i pro jiné aplikace.
1)Proveďte rešerši podobných systémů.
2)Zpracujte požadavky na systém s ohledem na specifika SW ekosystému projektu Manta Tools.
3)Navrhněte řešení.
4)Zvolte vhodné nástroje a proveďte prototypovou implementaci.
5)Otestujte na vhodných případech a zhodnoťte přínosy vašeho řešení.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 13. listopadu 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového inženýrství

Bakalářská práce

Návrh a prototypová implementace nástroje pro instalace a aktualizace webových Java aplikací

Petr Gondek

Vedoucí práce: Ing. Michal Valenta, Ph.D.

15. května 2017

Poděkování

Chtěl bych poděkovat firmě Profinit EU s.r.o. a Manta Tools s.r.o., že mi umožnili vytvořit bakalářskou práci na jednom z jejich projektů.

Také bych chtěl poděkovat vedoucímu bakalářské práce Ing. Michalu Valentovi, Ph.D za poskytnutí zkušených rad a za jeho pomoc při tvorbě mé práce.

Dále bych chtěl poděkovat RNDr. Lukáši Hermannovi za nápady při tvorbě návrhu aplikace a za pomoc při řešení dílčích problému, které během projektu nastaly.

V neposlední řadě chci poděkovat mé rodině a přátelům za pomoc a podporu, kterou mi během studia ochotně poskytovali.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů.

V souladu s ust. § 2373 občanského zákoníku tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům), vč. možnosti Dílo upravit či měnit, spojit jej s jiným dílem a/nebo zařadit jej do díla souborného. Toto oprávnění je časově, teritoriálně i množstevně neomezené a uděluji jej bezúplatně.

V Praze dne 15. května 2017 .....

Odkaz na tuto práci

Gondek, Petr. *Návrh a prototypová implementace nástroje pro instalace a aktualizace webových Java aplikací.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Práce popisuje návrh a prototypovou implementaci nástroje, který zjednodušuje proces instalace a aktualizace webových aplikací. Nástroj je navržen jako Java webová aplikace. Instalace a aktualizace, které tento nástroj podporuje, se provádí na jiných aplikacích běžících na stejném serveru.

Součástí práce je návrh slučování konfiguračních textových souborů a řešení jejich kolize pomocí knihovny Java Diff Utils.

Prototyp nástroje byl vytvořen ve spolupráci s firmou Manta Tools s.r.o. a byl otestována na jejich projektu Manta Flow. Je napsaný v programovacím jazyku Java za použití standardních knihoven pro operace se soubory a knihovny Java Diff Utils, která se používá při slučování textových souborů. Webová část je navržená s pomocí frameworků Spring MVC, Bootstrap a FreeMarker.

Klíčová slova Instalátor, Aktualizátor, Enterprise aplikace, Java diff utils, Java, Manta Tools s.r.o., Slučování, Spring MVC, FreeMaker, Bootstrap, Webové aplikace

Abstract

The bachaleor thesis describes design and prototype of a tool which simplifies installation and update process of web applications. The tool is designed as a Java web application. Installations and updates are being realized on different applications at a same server.

Merging of configuration files and resolving their collisions are also part of this thesis, using the Java Diff Utils library.

The prototype was created in cooperation with Manta Tools Ltd. and was tested on their project Manta Flow. It is implemented in Java programing language while using standard libraries for files operations and Java Diff Utils library, which is used for merging of text files. Web user interface is designed with the help of Spring MVC, Bootstrap and FreeMarker.

Keywords Installer, Updater, Enterprise applications, Java diff utils, Java, Manta Tools Ltd., Merge, Spring MVC, FreeMaker, Bootstrap, Web applications

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Úvod

Instalace a aktualizace aplikace je celkem komplikovaný proces. Obvykle vyžaduje mnoho ručních úprav, a proto je náchylný na chyby způsobené tím, kdo jej provádí. Zejména pokud se nejedná o programátora či řádně proškoleného profesionála. Tímto problémem se musí zabývat každá firma, která vyvíjí a dodává software běžící na prostředích zákazníka. Každý klient přichází s jiným zařízením, jinou konfigurací, jiným operačním systémem, jiným stylem spuštění aplikace, ale také s jinými schopnostmi a zkušenostmi ohledně počítačů.

Tato práce zmíněnou problematiku analyzuje a porovnává ji s jinými již existujícími řešeními. Ukazuje, jak se tyto řešení dají použít a popisuje, v čem je nové řešení lepší a kdy se nevyplatí.

Analýza myslí i na to, že jiné běžící aplikace je pro provedení aktualizace potřeba vypnout a po dokončení zapnout. Zmíněné aplikace mají své konfigurační soubory. Tyto soubory nelze bezmyšlenkovitě mazat nebo přepsat, protože v nich uživatel může mít nastavené speciální hodnoty závislé na jeho prostředí.

Projekt se zaměřuje na aktualizace webových služeb a aplikací, ke kterým může uživatel tento produkt připojit. Přesněji řečeno pustit ho paralelně jako další webovou službu. Jedním z nefunkčních požadavků je, aby aktualizaci zvládl byznys uživatel. To je takový uživatel, který počítač denně používá k běžným činnostem jako je e-mailová komunikace, prohlížení webových stránek a podobné aktivity. Proto je webový instalační průvodce vhodným řešením.

Cíl práce

Cílem práce je provést analýzu nástroje, který přináší řešení a usnadnění instalace a následné aktualizace webových Java enterprise aplikací. Dále je úkolem

vytvořit návrh takového nástroje, implementovat jeho prototyp, který bude následně otestován.

Výsledným produktem projektu je webová aplikace, která zvládne nainstalovat nebo aktualizovat jinou webovou Java aplikaci běžící na stejném zařízení. Nástroj je nezávislý na operačním systému. Prototyp je implementovaný s ohledem na specifikace softwarového ekosystému projektu Manta Tools.

Práce bere v potaz možnost slučovat konfigurační soubory[1] a přináší řešení kolizí vznikajících slučováním.

Členění textu práce

Text této práce je rozložen do následujících kapitol:

- **Úvod** Popis problému a motivace k jeho vyřešení.
- **Byznys analýza** V této kapitole se pojednává o návrhu aplikace z pohledu byznys procesů. Řeší se požadavky na aplikaci, použití aplikace, návrh webového rozhraní. Analýza se zabývá i existujícími řešeními.
- **Technický design** V technickém designu se řeší návrh aplikace pro její následnou implementaci. Tento návrh je už závislý na použití programovacího jazyka a frameworků. Ukazuje se zde výčet jednotlivých použitelných frameworků a důvody jejich použití či nepoužití. Lze zde nalézt popis návrhu tříd, způsob nasazení nebo návrh struktury aplikace.
- **Prototyp** Kapitola prototyp se zabývá částečnou implementací výsledného produktu. Popisuje, jak jsou jednotlivé vrstvy a třídy implementovány. Druhou částí kapitoly je způsob testování prototypové implementace.
- **Závěr** Poslední kapitola shrnuje celou práci. Rozebírá se zde naplnění cílů a návrhy do budoucna.

[1] Jedná se o textové soubory. Například ve formátu XML, CSV, TXT a další.

Kapitola 1

Byznys analýza

Následující kapitola pojednává o byznys analýze projektu. Lze zde nalézt byznys doménový model, funkční a nefunkční požadavky, existující řešení, diagram použití, stavové diagramy a diagramy aktivit.

1.1 Použité pojmy

V analýze a dalších kapitolách se budou používat následující nejdůležitější pojmy. Tyto pojmy je nutné detailněji vysvětlit pro lepší porozumění textu.

Pojmem „Aktualizace“ se myslí jak aktualizace, ale i čistá instalace. Rozdíl mezi těmito dvěma procesy je pouze v tom, že jeden musí reagovat na to, že v cílovém adresáři se nachází nějaká data. Proces aktualizace je tedy z pohledu vývoje složitější. Práce se bude více zaměřovat na proces aktualizace, který je zajímavější a podstatnější.

Pojmem „Aktualizátor“ se rozumí produkt (aplikace) tohoto projektu, který bude sloužit především za účelem aktualizace jiných aplikací podle smluvených dohod popsaných v následujících kapitolách. Nejedná se o produkt (prototyp) této bakalářské práce, ale hotovou konečnou aplikaci.

Pojmem „Zadavatel“ se rozumí firma Manta Tools s.r.o., kterou zastupoval RNDr. Lukáš Hermann. Doktor Hermann byl oprávněn klást požadavky na projekt, řešit a schvalovat změny a následně projekt akceptovat jako hotový.

Pojmem „Klient“ se rozumí fyzická nebo právnická osoba (může být organizace), která legálně nabyla kopii aktualizátoru a využívá ji především pro účely aktualizací jiných aplikací.

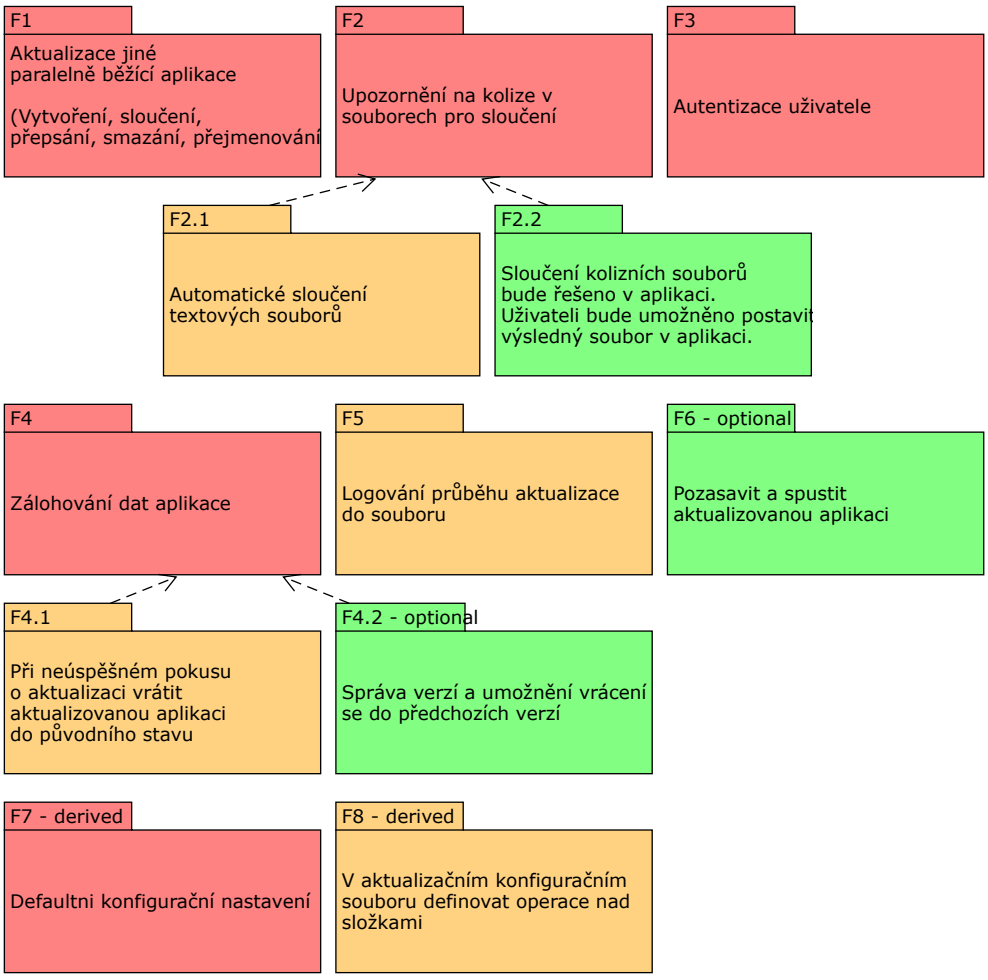
Pojmem „Uživatel“ se rozumí fyzická osoba, která přistupuje a ovládá aktualizátor a to výhradně pomocí uživatelského rozhraní.

1.2 Funkční požadavky

Funkční požadavky 1.1 vyřčené zadavatelem jsou zachycené a popsané v následující tabulce 1.1. Jsou barevně označené podle priorit zadavatele. Červená značí nejvyšší prioritu, oranžová střední a zelená nejnižší.

Funkční požadavky označené klíčovým slovem „optional“ (zkráceně „opt“), jsou pro zadavatele postradatelné. Nemusí být vyřešené do prvního vydání aplikace a můžou být odložené do některého z dalších vydání.

Funkční požadavky označené klíčovým slovem „derived“ (zkráceně „der“), vznikly až posléze při předání návrhu aplikace zadavateli. Jedná se o požadavky zpřesňující realizaci.



Obrázek 1.1: Funkční požadavky podle priorit

Tabulka 1.1: Popis funkčních požadavků

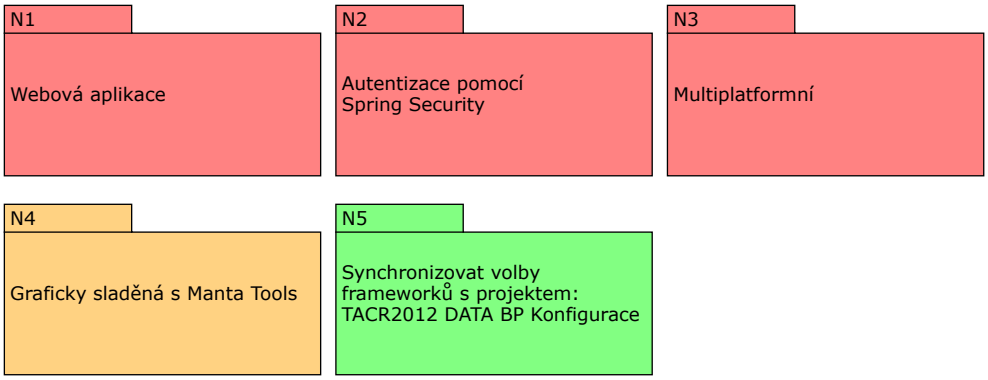
Kód	**Název**	**Popis**
F1	Aktualizace jiné paralelně běžící aplikace	Provést aktualizaci jiné aplikace nacházející se na stejném zařízení jako aktualizátor.
F2	Upozornění na kolize v souborech pro sloučení	Upozorní uživatele na soubory ke sloučení, které nelze jednoduše nahradit, protože jsou editované a může dojít ke ztrátě dat.
F2.1	Automatické sloučení textových souborů	Vylepšené řešení slučování, kde soubory, které neobsahují kolize, aplikace automaticky sloučí. Na kolizní soubory upozorní.
F2.2	Řešení kolizí v aplikaci	Lepší řešení pro kolizní soubory. Soubory budou sloučeny „side by side“ pomocí aplikace podobně, jako nabízí jiné nástroje pro slučování.
F3	Autentizace uživatele	Aktualizace bude umožněna pouze uživatelům, kteří se přihlásí a splňují politiku klienta.
F4	Zálohování dat aplikace	Aplikace bude umět zálohovat důležitá data aktualizovaných klientovo aplikací.
F4.1	Vrácení změn při neúspěchu	Aktualizátor provede vrácení změn, pokud detekuje chybu během aktualizace.
F4.2 (opt)	Správa a obnova do předchozích verzí	Vzhledem k zálohování aktualizované aplikace bude umožněno zobrazit jednotlivé zálohy a provádět nad nimi správu jako je například jejich mazání. Také bude umožněno nahrát vybranou zálohu.
F5	Logování průběhu aktualizace do souborů	Aktualizátor bude vytvářet log do souboru.
F6 (opt)	Pozastavit a spustit aktualizovanou aplikaci	Aktualizátor bude moci pozastavit a spustit aktualizovanou aplikaci a při neúspěchu na to upozorní.
F7 (der)	Defaultní konfigurační nastavení	Aktualizátor bude při spuštění nahrávat konfigurační soubor, který ho uvede do požadovaného nastavení.

Kód	Název	Popis
F8 (der)	V aktualizačním konfiguračním souboru definovat operace nad složkami	Aktualizační konfigurační soubor bude umožňovat definovat operace nad složkami. Definované pravidlo se rekurzivně aplikuje na její obsah. Pokud některá část obsahu bude mít vlastní pravidlo, bude aplikované toto pravidlo.

1.3 Nefunkční požadavky

Nefunkční požadavky 1.2 jsou zachycené a popsané v následující tabulce 1.2.

Vzhledem k vlastnostem a způsobu použití aktualizátoru bylo rozhodnuto a schváleno, že aktualizátor bude vytvořen za pomoci frameworků používaných na již existujících aplikacích Manta Tools. Tyto frameworky umožňují „server side“ přístup tvoření webových aplikací, a proto dojde k nenaplnění nefunkčního požadavku „N5“, který využívá frameworky pro „client side“ přístup.



Obrázek 1.2: Nefunkční požadavky podle priorit

Tabulka 1.2: Popis nefunkčních požadavků

Kód	Název	Popis
N1	Webová aplikace	K aktualizátoru se bude přistupovat přes webové rozhraní.
N2	Autentizace pomocí Spring Security	Pro autentizaci se použije funkční celek s Spring Security, který zadavatel má naimplementovaný a dodá ho.

Kód	Název	Popis
N3	Multiplatformní	Aplikace bude schopna běžet na operačních systémech z rodiny Windows, Linux a MacOS.
N4	Graficky sladěná s Manta Tools	Grafické zpracování uživatelského rozhraní bude sladěné podle stávajících aplikací Manta Tools. Grafiku zadavatel dodá.
N5	Synchronizace volby frameworků s projektem TACR2012 DATA BP Konfigurace	Vzhledem k vznikajícímu dalšímu projektu je požadavek na použití stejných frameworků mezi těmito projekty. Důvodem je předejít příliš velkým rozlišnostem jednotlivých projektů společnosti Manta Tools a nutnosti rozsáhlých znalostí pro navazující vývojáře.

1.4 Diagram použití

Diagram použití 1.3 zachycuje činnosti, ke kterým bude uživatel systém využívat. Aktualizátor má jednu primární činnost, nazývající se `Aktualizovat aplikaci` a jednu sekundární `Obnovení do předchozí verze`. Pro přehlednost byly činnosti rozpadlé na menší celky. Obě dvě činnosti se tedy skládají z dalších menších činností, kde některé jsou označené jako povinné (`include`) a některé nepovinné/podpůrné (`extends`).

V tabulce 1.3, která popisuje diagram 1.3, je sloupec kód, který mapuje činnosti na funkční požadavky.

Tabulka 1.3: Popis diagramu použití

Název	Popis	Kód
Aktualizovat aplikaci	Provést aktualizaci jiné aplikace nainstalované na stejném zařízení. Provést nezbytné úkony pro nahrazení souborů.	F1, F7, F8
Slučování textových souborů	Sloučit po řádcích textové soubory jako jsou txt, xml a další. Musí se řešit jejich různá kódování.	F2, F2.1, F2.2
Zálohování aktualizované aplikace	Provést zálohu aktualizované aplikace zkopírováním její souborové struktury do archivu.	F4
Run/Stop aktualizované aplikace	Aktualizátor potřebuje výhradní právo na operaci se souborovým systémem. Pokud aktualizovaná aplikace běží, musí se vypnout, aby nedržela zámky nad jejími soubory. Po aktualizaci se musí vrátit aplikace do stavu běžícího.	F6

Název	Popis	Kód
Autentizování uživatele	Ověřit uživatele, že je oprávněn provádět aktualizace. Zadavatel dodá jako hotovou funkčnost.	F3
Obnovení předchozí verze	Prohlížet zálohy a případně mít možnost se do nich vrátit. Součástí je i smazání předchozí verze.	F4.1, F4.2
Logování akcí	Aplikace bude vytvářet log operací které provádí. Log bude uložen v souboru.	F5

1.5 Byznys doménový model

Diagram 1.4 je modelovaný z pohledu `Aplikace` a zachycuje, jak vidí `Aplikace` svůj okolní svět. Objektů `Aplikace` může existovat více. Byznys doménový model vzešel z vzájemné diskuze se zadavatelem. Popis jednotlivých objektů diagramu zachycuje tabulka 1.4.

1.5.1 Popis byznys doménového diagramu

Základním kamenem celého projektu je `Soubor`, který tvoří obsah téměř všech zachycených objektů. Jedná se o reprezentaci souboru ze souborového systému. Identifikátorem je `cesta` v souborovém systému. Soubor je ve většině případů ve vztahu $1 : N$, kdy musí vždy existovat alespoň jednou, neboť vztah $0 : N$ nedává pro existenci ostatních objektů smysl.

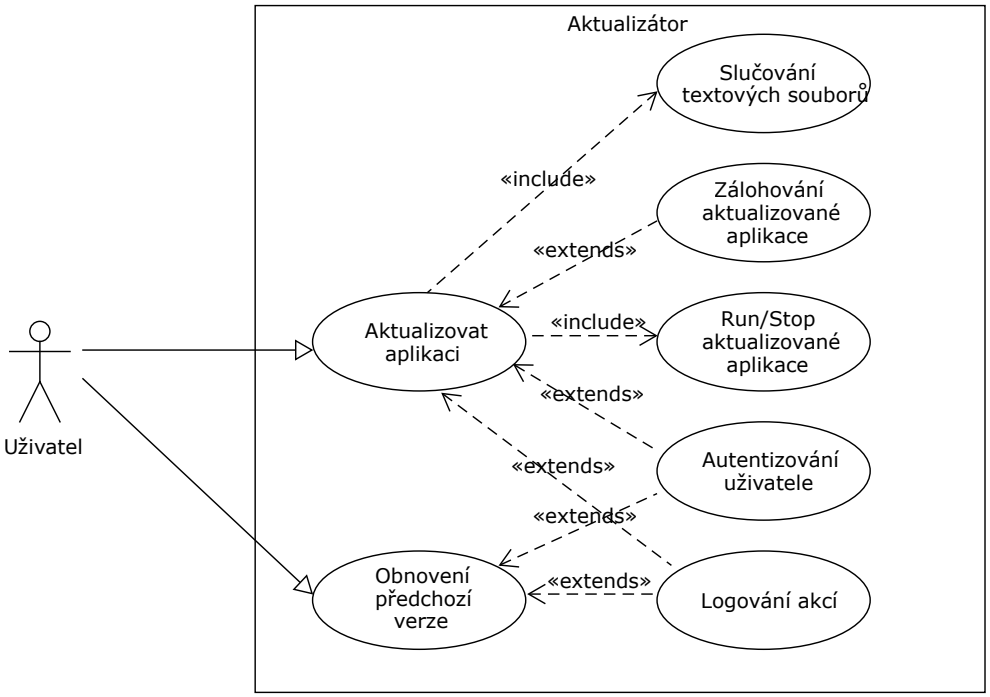
Nejpodstatnějšími objekty pro aktualizátor jsou `Aplikace` a `Aktualizace`. Tyto dva objekty jsou ve vztahu $1 : N$, kdy aplikaci může aktualizovat více `Aktualizací`. Aktualizátor musí evidovat i aplikace, pro které nezná žádné aktualizace.

Objekt `Aplikace` je tvořen trojicí `název`, `verze` a `umístění`. Dostatečným identifikátorem je `umístění` v souborovém systému, který ukazuje na kořenovou složku. Pro lepší přehlednost bude vyžadována i unikátnost podle jména.

Z `Aplikace` musí jít vytvořit zálohu, která se použije pro obnovu dat při nezdařené aktualizaci. Dostačující identifikátorem zálohy je zdroj (ze které aplikace vzešla) a její `verze`. Záloha obsahuje `Soubory` z verze, ze které byla vytvořena.

Objekt `Aktualizace` je složen z objektu `Soubor` a vnitřního atributu `verze`, který označuje verzi aplikace po aktualizaci. Dalším atributem je `konfigurace`, který nese informace o průběhu aktualizace. Identifikátorem objektu je dvojce cílová `Aplikace` a `verze`.

`Balíček sloučení` může vzniknout z `Aktualizace`, pokud budou nějaké soubory v `konfiguraci` označeny ke sloučení. Objekt obsahuje odkaz na tyto



Obrázek 1.3: Diagram použití

dva soubory a atribut `id` pro lepší identifikaci. Počet balíčků sloučení nemůže být větší než počet souborů v aktualizaci.

Pokud soubory z objektu `Balíček sloučení` kolidují, tedy obsahují řádky, které nelze triviálně sloučit, vznikne `Kolize`. Tento objekt zná `pozici` kolize v souboru, `obsah` kolize (řádky kolidujícího textu) a svůj identifikátor. Kolidující řádky v jednom souboru musí mít prázdný průnik, tedy se nemůžou překrývat a počet kolidujících řádků nemůže být větší než obsah souboru.

Tabulka 1.4: Popis byznys doménového modelu

Název	**Popis**
Aplikace	Reprezentuje to, co se bude aktualizovat. Hlavní parametry, které je potřeba evidovat, jsou název, umístění v souborovém systému a aktuální verze.
Aktualizace	Reprezentuje balíček souborů, které jsou součástí aktualizace. Důležité parametry jsou cílová verze aplikace a konfigurace aktualizace, která popisuje, jak aktualizace bude provedena.
Záloha	Tvoří se z aplikace a obsahuje soubory aplikace z určité verze.

Název	Popis
Balíček sloučení	Balíček sloučení obaluje soubory, které je třeba sloučit. Jsou právě dva: revidovaný a aktualizovaný soubor.
Kolize	Pokud data v balíčku sloučení kolidují, je zapotřebí tuto kolizi nějak označit a evidovat.
Soubor	Reprezentuje jednotlivá data potřebná pro fungování celého systému. Jedná se o jednotlivé soubory v souborovém systému.

1.6 Diagramy aktivit

Následující kapitola se zabývá modelováním byznys procesů pomocí diagramů aktivit.

1.6.1 Proces aktualizace

Aplikace je testovaná na aplikacích zadavatele, proto následující diagramy budou popisovat změny na jeho aplikaci s názvem Manta Flow.

Manta Flow slouží k zobrazení datových toků na SQL scriptech. Jedná se o dvě aplikace: klientská, která analyzuje SQL scripty a serverová, která zobrazuje datové toky.

1.6.1.1 Stávající stav

Ve stávajícím stavu mají aktualizace obou aplikací Manta Flow (klientská i serverová) podobný průběh. První krok, který je třeba vykonat, je vypnutí běžící aplikace a provedení zálohy dat, kde u serverové části proběhnou exporty z webového rozhraní. V dalším kroku se smaže celá aplikace a místo ní nahraje nová verze z aktualizačního archivu. Nakonec se ručně slučují konfigurační soubory a pro serverovou část importují exporty. Podrobněji stávající stav zachycuje diagram 1.5, který je popsán v tabulce 1.5.

Tabulka 1.5: Popis diagramu procesu aktualizace aplikace Manta Flow - klientské části (původní)

Název	Popis
Vypnout aplikaci	Zkontrolovat, jestli Manta Flow - klient není spuštěná a případně ji vypnout.
Provést zálohu	Provést zálohu dat aplikace. Jedná se o generované soubory a změněné konfigurační soubory s nastavením. Vzniká objekt zálohované soubory.

Název	Popis
Smazat aplikaci	Smazat celý obsah kořenového adresáře aktualizované aplikace.
Rozbalit novou aplikaci	Místo smazané aplikace se nahraje nová verze z aktualizačního archivu.
Sloučit zálohu	Nahrát zpět zálohovaná data a sloučit konfigurační soubory s pozměněným nastavením. Vstupuje objekt zálohované soubory.
Spustit aplikaci	Spustit aktualizovanou aplikaci.
Zkontrolovat log	Zkontrolovat log aplikace, jestli aktualizace proběhla v pořádku a aplikace běží, jak má.

Pro přínos lepšího řešení, se musí pochopit problém předchozího aktualizačního postupu. Z předchozího diagramu a popisu lze vyčíst minimálně dva hlavní problémy, při kterých se může aktualizace nezdařit.

Prvním je nedodržení postupu nebo přeskočení nějakého kroku. Například uživatel, který bude provádět aktualizaci, může říct, že nebude mazat aplikaci a jenom nahradí soubory novými. To ve většině případů dopadne dobře. Ovšem pokud se některé soubory mají odstranit, tak tímto způsobem se odstranění přeskočí a aktualizace se nezdaří. Dalším příkladem je nevypnutí aktualizované aplikace. Poté může dojít k výjimce při přepisování souboru, že soubor je otevřen jiným programem a smazání se nezdaří.

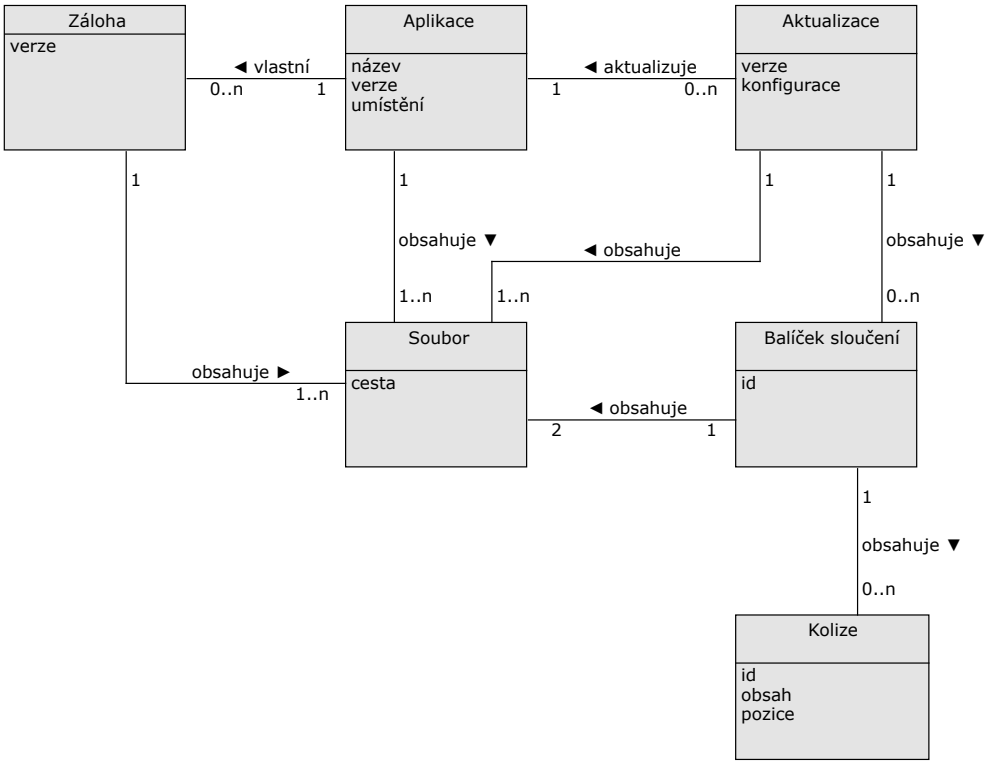
Druhým problémem je slučování souborů. Tato fáze se zdá být poněkud kritičtější. Pro představu může mít uživatel za úkol sloučit pro něj nic neříkající, stořádkový XML dokument s jiným stořádkovým, nic neříkajícím XML dokumentem, ve kterém před půl rokem změnil jednu hodnotu. S největší pravděpodobností to dopadne tak, že se nic nesloučí a původní upravený konfigurační soubor se nahraje novým neupraveným, a tedy i nefunkčním.

Zadavatel se také pokusil využívat jiný způsob aktualizací. Použil nástroj BitRock InstallBuilder, který slouží k tvorbě instalačních balíčků. Bohužel se také projevil jako nevyhovující. Tento aktualizační mechanizmus bude více rozebrán v kapitole o existujících řešení.

1.6.1.2 Nově navržený stav

Nově navržený způsob, jak ho zachycuje diagram v příloze C.1, je na první pohled složitější. Jeho velikost a složitost je také odůvodněna tím, že zadavatel je seznámený s UML diagramy. Jedná se o softwarového inženýra, který je obeznámen s možnostmi instalací a aktualizacemi aplikací. Aby se dosáhlo jeho plného pochopení a kontroly, je diagram rozepsán detailněji.

Hlavní výhodu, kterou proces přináší, je, že neumožní uživateli vynechat žádný krok aktualizace, provede ho jednotlivými kroky a pomůže mu s proce-



Obrázek 1.4: Byznys doménový model

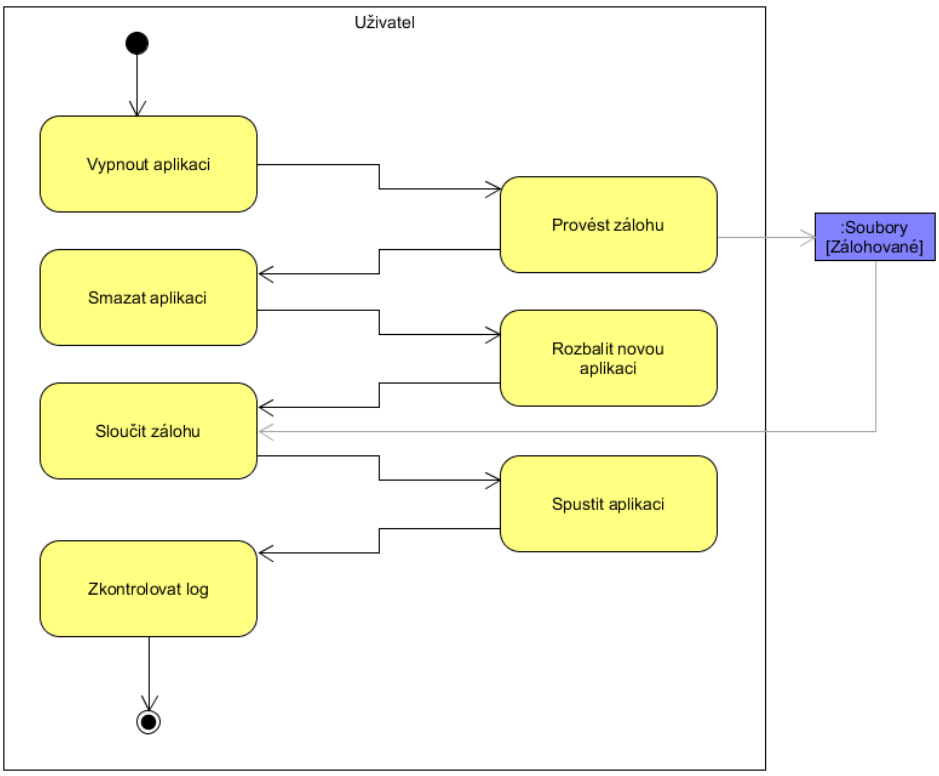
sem slučování. Navíc proces aktualizace je validovaný, tedy neumožní uživateli použít aktualizační balíček na špatnou aplikaci, a tudíž je použitelný i na jiné aplikace než jenom pro Manta Flow. Popis diagramu se nachází v tabulce 1.6. Tabulku lze číst postupně, jednotlivé kroky jsou psané v chronologickém pořadí.

Pro porovnání s předchozím procesem aktualizace je zde nastíněno, jaké úkony bude uživatel muset provést pro aktualizaci aplikace podle nově navrženého průchodu.

Prvním krokem bude ověření uživatele. Ne všichni uživatelé ve firmě klienta můžou být oprávněni provádět aktualizace aplikací. Přináší to výhodu možnosti evidování, kdo a kdy provedl jakou operaci. Tuto funkčnost dodá zadavatel.

V následujícím úkonu bude muset uživatel zadat do aplikace cestu v lokálním adresáři k aktualizačnímu balíčku. Tento balíček se nahraje do aplikace na server a budou se nad ním vykonávat další operace.

Zadání cesty k aktualizaci se rozumí nejenom zadání exaktní cesty ke kořenovému adresáři aplikace. Aktualizátor si může pamatovat už dříve zadané cesty k aplikacím a „zadáním cesty“ se může rozumět pouhé vybrání ze seznamu známých aplikací. Seznam lze také získat z konfiguračního souboru



Obrázek 1.5: Proces aktualizace aplikace Manta Flow – klientská část

aktualizátoru, který může být dodán s aktualizátorem nebo ze seznamu nainstalovaných aplikací v operačním systému.

Následně přichází schválení aktualizace. Aktualizátor má vše potřebné pro provedení čisté instalace nebo aktualizace. Toto je poslední záchytný bod, než začne aktualizátor provádět jakékoliv zásahy a změny do souborového systému aktualizované aplikace.

Aktualizátor se pokusí automaticky vypnout aktualizovanou aplikaci. Pokud se mu to nepodaří, bude klient upozorněn a vyzván k provedení manuálního vypnutí. Klient schválí, že aplikace je vypnutá, a že přebírá zodpovědnost za případné následky, pokud tomu tak nebude. Může zde proběhnout kontrola běžících procesů.

Dále aktualizátor začne provádět vlastní aktualizaci. Prvními kroky jsou validace aktualizace a aktualizované aplikace a přípravy na sloučení textových souborů, které jsou označené podle aktualizačního balíčku ke sloučení. Automatické sloučení proběhne obvyklým způsobem. Pokud je některý řádek u klienta změněný a v aktualizaci není, tedy vypadal jako před zásahem klienta, bude použit změněný řádek. Opačný způsob, kdy změna je v aktualizaci

a u klienta není, proběhne stejným způsobem. Pokud během automatického sloučení nastane kolize, bude uživatel vyzván k vyřešení této kolize. Pro řešení kolize se zobrazí obsah kolidujících souborů, kde budou označené kolidující řádky a uživatel kliknutím vybere řádek, který se použije nebo napíše řádek vlastní.

Pokud aktualizace doběhne v pořádku, bude provedeno automatické spuštění aktualizované aplikace. Když se automatické spuštění nezdaří, bude po uživateli vyžadované spuštění manuální.

V posledních krocích bude uživatel dotázán, jestli je spokojen se změnami a zdali nechce vrátit aktualizaci zpět nahráním zálohy. Tento finální krok proběhne vždycky, i když aktualizace skončí chybně. Zda se nahraje záloha nebo ne, už nemá vliv na konečné zobrazení výsledku aktualizace, které uzavírá aktualizační proces.

Tabulka 1.6: Popis diagramu procesu aktualizace aplikace Manta Flow - klientské části (nový)

Název	**Popis**
Zadání přihlašovacích údajů	Uživatel se musí přihlásit do aplikace.
Ověření uživatele	Kontrola, zda uživatel disponuje právy provádět aktualizace.
Zadání cesty k aktualizačnímu balíčku	Aktualizátor potřebuje znát, kde se nachází aktualizace. Vzhledem k tomu, že je aktualizátor navržený jako webová aplikace, bude touto aktivitou chápané nahrání aktualizace na server přes webové rozhraní.
Zadání cesty k aktualizované aplikaci	Aktualizátor potřebuje vědět, jakou aplikaci je třeba aktualizovat. Proto potřebuje znát cestu do jejího kořenového adresáře. Získají se tím soubory aplikace
Rozbalení	Aktualizátor rozbalí archiv aktualizace a získá tím soubory pro aktualizaci, které jsou zatím nezpracované.
Validace aktualizace	Zde se provede kontrola souborové struktury aplikace oproti aktualizaci a validace verzí získaných z konfiguračních souborů aktualizace a aplikace. Pokud proces validace dopadne neúspěchem, zobrazí se chybový výsledek a aktualizace je ukončena.
Potvrzení aktualizace	Zobrazení informací o aktualizaci a vyzvání uživatele o potvrzení aktualizace. Pokud aktualizaci zamítne, je proces aktualizace ukončen.
Vypnout běžící aplikaci	Aktualizátor se pokusí vypnout aktualizovanou aplikaci.

Název	**Popis**
Provedení manuálního vypnutí aplikace	Pokud automatické vypnutí nedopadlo dobře, bude uživatel informován o neúspěchu a vyzván k provedení manuálního vypnutí. Uživatel bude muset potvrdit, že aplikaci vypnul. Pokud aplikace nevypne proces, aktualizace se ukončí.
Provést zálohu	Aktualizátor vytvoří zálohu aktualizované aplikace. Vznikají zálohované soubory. Záloha vznikne v podadresáři kořenového adresáře aplikace, ve složce s časem a datem provedení aktualizace. Záloha bude komprimovaná pro ušetření místa na disku.
Spustit předaktualizační script	Pro některé aktualizace a instalace bude potřeba zasáhnout do systémových adresářů nebo registrů, inicializovat systémové proměnné, či přidat aplikace do proměnné `PATH`. Toto všechno bude řešené scriptem v aktualizačním balíčku, který bude spuštěn v tento moment. Pokud script skončí chybou, je provedena obnova aplikace do stavu před aktualizací, oznámena chyba a proces aktualizace je ukončen.
Provedení aktualizace	Provedení kopírování, mazání a slučování. Po skončení vznikají zpracované soubory, které jsou novými soubory aktualizované aplikace. Pokud aktivita dopadne neúspěchem, je provedena obnova aplikace do stavu před aktualizací, oznámena chyba a proces aktualizace je ukončen. Tento proces je detailněji popsaný v dalších kapitolách.
Spustit poaktualizační script	Totožný případ jako předaktualizační script. Tento script se pouští po operacích se soubory. Pokud script skončí chybou, je provedena obnova aplikace do stavu před aktualizací, oznámena chyba a proces aktualizace je ukončen.
Kontrola existence poaktualizačního souboru	Vyhledá se poaktualizační soubor, který po sobě aktualizátor zanechává v adresářích aplikací.
Vytvořit poaktualizační soubor	Pokud soubor v aplikaci neexistuje, tak se vytvoří.
Zapsání informací po poslední aktualizaci	Zapíší se informace z poslední aktualizace. Zejména verze aktualizované aplikace, ale i další.
Automatické spuštění aplikace	Aktualizátor se pokusí automaticky spustit aktualizovanou aplikaci.

Název	Popis
Manuální spuštění aplikace	Pokud se nepodařilo automaticky spustit aplikaci, bude o tom uživatel informován a bude vyzván k manuálnímu zapnutí. Poté potvrdí, že zapnutí provedl.
Zobrazit pozitivní výsledek	Pokud se zapnutí zdaří, tak aplikace zobrazí informace o aktualizaci a proces aktualizace se ukončí.
Rozhodnutí o vrácení změn.	Pokud se zapnutí nezdaří nebo uživatel zapnutí odmítne, bude dotázán, zda chce vrátit aktualizovanou aplikaci do původního stavu před aktualizací. Odmítne-li akci, aktualizační proces se ukončí.
Vrácení provedených změn	Pokud uživatel výzvu přijal a rozhodl se vrátit aplikaci do stavu před aktualizací, nahradí se aplikace zálohou a spustí se reversní script. Poté se zobrazí výsledek operace a proces aktualizace je ukončen.

1.6.2 Proces aktualizace – operace se soubory

Následující kapitola podrobněji rozebírá část aktualizace, kde se provádí operace se soubory 1.6. Diagram je popsán v tabulce 1.7. Operace jsou evidované v konfiguračním souboru, který je detailněji rozebrán v následující kapitole.

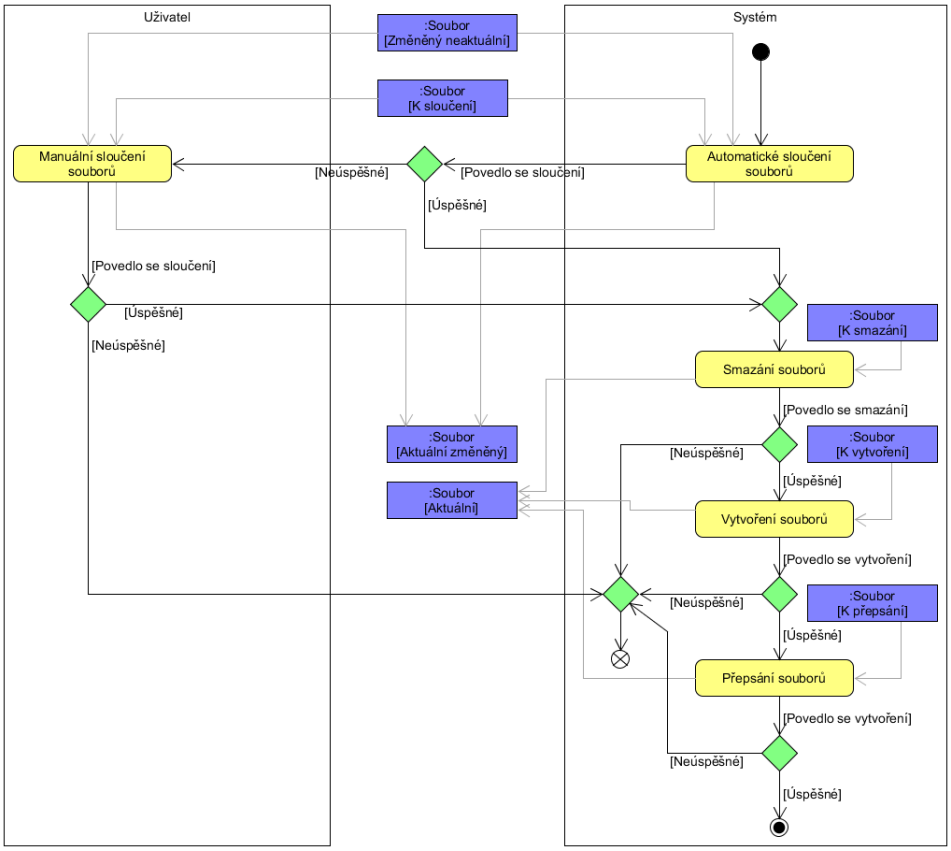
Operace se soubory z pohledu operačního systému lze rozdělit na tři základní: vytvoření, zkopírování a smazání. Aktualizátor rozlišuje operací pět: sloučení, smazání, vytvoření, přepsání a přejmenování.

Operací sloučení se rozumí spojení dvou dokumentů v dokument jeden s použitím změn z obou dokumentů. Tento proces je více rozepsán v následující kapitole o slučování souborů.

Vytvořením a přepsáním se může rozumět jedna a tatáž operace. Aktualizátor je rozděluje kvůli validaci souborové struktury. Pokud se soubor vytváří, nesmí v původní aplikaci existovat a pokud se přepisuje, tak v aplikaci existovat musí. Je to jedna z validací, pomocí které se aktualizátor snaží předejít tomu, aby se aktualizace neaplikovala na špatnou aplikaci nebo špatnou verzi aplikace. Aktualizátor bude podporovat i nedefinovanou operaci „vlož nebo přepiš“. Více se lze dozvědět v kapitole o konfiguračním souboru.

Operace smazání je triviální. Jedná se o obyčejné odstranění souboru. Důležité je zmínit, že operace smazání musí být vykonaná dřív než vytváření a přepisování souborů. Je to z důvodu mapování souborů z konfiguračního souboru na souborovou strukturu aplikace a aktualizačního balíčku. Více o tom, proč tomu tak je, bude popsáno v technickém designu.

Operace přejmenování by byla náročnější na zaznamenávání v konfiguračním souboru, proto bylo z úsporných důvodů zvolené prozatímní řešení, kde se starý soubor smaže a vloží se soubor nový s novým jménem.



Obrázek 1.6: Proces aktualizace – operace se soubory

Tabulka 1.7: Popis diagramu operací se soubory

Název	**Popis**
Automatické sloučení souborů	Aktualizátor se pokusí automaticky vyřešit sloučení. Jak to provede, je popsáno v kapitole slučování souborů. Do aktivity vstupuje `změněný neaktuální` soubor z aplikace a soubor určený `ke sloučení` z aktualizačního balíčku. Vzniká soubor `změněný aktuální`.

Název	Popis
Manuální sloučení souborů	Pokud se automatické sloučení nezdaří, bude uživatel vyzván, aby vyřešil kolize a vytvořil nový soubor vybráním řádků z kolize, které se mají aplikovat. Pokud se jedná o složku, aplikuje se rekurzivně pravidlo na její obsah. Soubory ve složce, na které má být použito jiné pravidlo, se přeskočí. Vstupují objekty `neaktuální` soubor z aplikace a soubor `ke sloučení` z aktualizačního balíčku. Vzniká objekt `změněný aktuální` soubor. Změněný proto, že obsahuje změny provedené uživatelem již z dřívější verze.
Smazání souborů	Smažou se všechny soubory určené k odstranění. Pokud se jedná o složku, aplikuje se rekurzivně pravidlo na její obsah. Soubory ve složce, na které má být použito jiné pravidlo, se přeskočí. Vstupuje objekt soubor určený `ke smazání`. Vzniká objekt `aktuální`.
Vytvoření souborů	Vytvoří se všechny soubory určené k vytvoření. Pokud se jedná o složku, aplikuje se rekurzivně pravidlo na její obsah. Soubory ve složce, na které má být použito jiné pravidlo, se přeskočí. Vstupuje objekt soubor určen `k~vložení`. Vzniká objekt `aktuální`.
Přepsání souborů	Všechny existující soubory aplikace se přepíší soubory z aktualizace. Na soubory z aktualizace, které nemají záznam v konfiguračním souboru, bude aplikované pravidlo vlož nebo přepiš, které bude vyhodnoceno právě teď. Pokud se jedná o složku, aplikuje se rekurzivně pravidlo na její obsah. Soubory ve složce, na které má být použito jiné pravidlo, se přeskočí. Vstupuje objekt soubor určen `k~přepsání`. Vzniká objekt `aktuální`.

1.6.3 Slučování

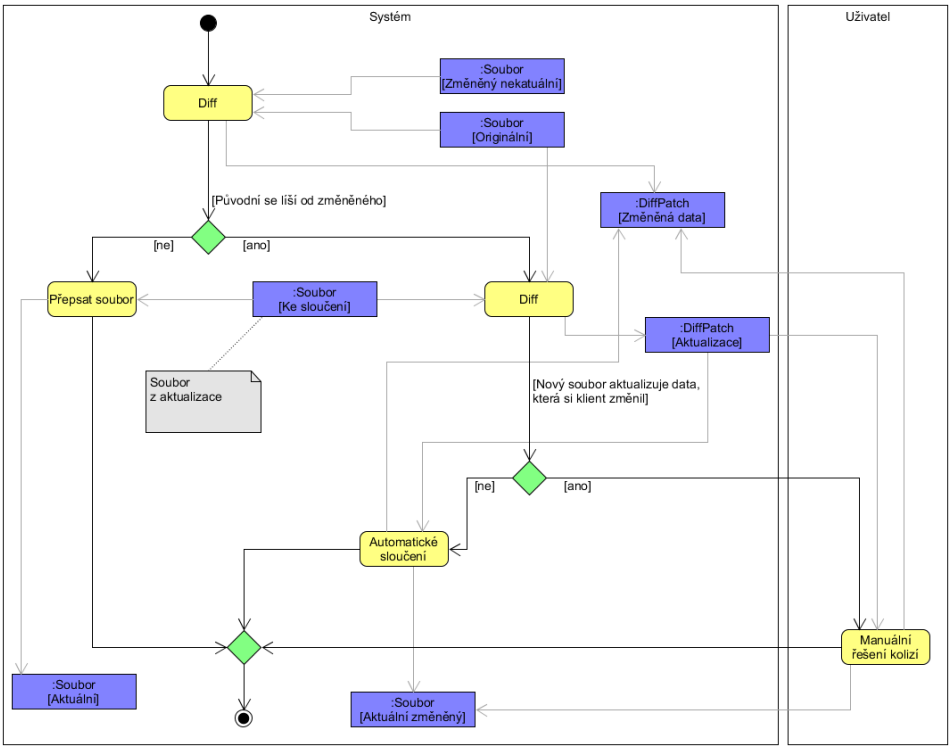
Při slučování dvou textových souborů po řádcích je důležité si uvědomit, že se neslučují dva textové soubory, ale slučují se změny, které byly na těchto souborech provedeny. Slučují se tedy celkem tři soubory: originální, který obsahuje původní nezměněný text, revidovaný, který obsahuje změny provedené uživatelem a aktualizovaný, který obsahuje změny provedené tvůrcem aktualizace (zadavatelem). Po domluvě se zadavatelem bylo rozhodnuto, že původní originální soubor bude dodáván s aktualizací.

Pro získání změn, které se budou slučovat, je třeba získat rozdílné řádky mezi originálním souborem a revidovaným, jakožto i mezi souborem originál-

ním a aktualizovaným.

Jakmile jsou rozdíly získány[2], je snadné zjistit o jaké řádky se jedná. Pokud se jedná o změnu nad stejnými řádky v originálním souboru, tak se jedná o kolizi. Kolizi musí vyřešit uživatel, protože aktualizátor se nemá podle čeho rozhodnout, jaká změna se má aplikovat. Většinou bude třeba aplikovat změny obě, akorát poupravené.

Tento proces je zachycen následujícím diagramem 1.7, který je popsán v tabulce 1.8.



Obrázek 1.7: Proces slučování souborů

Tabulka 1.8: Popis diagramu procesu slučování souborů

Název	**Popis**
Diff (1)	V prvním kroku se musí získat rozdíly mezi `originálním` a `změněným` `neaktuálním` souborem z aplikace. Vzniká nový objekt `DiffPatch[Změněná data]`.

[2]Oficiálně používaným označením je Patch, který se získává z operace DIFF, tato označení se můžou v práci vyskytovat.

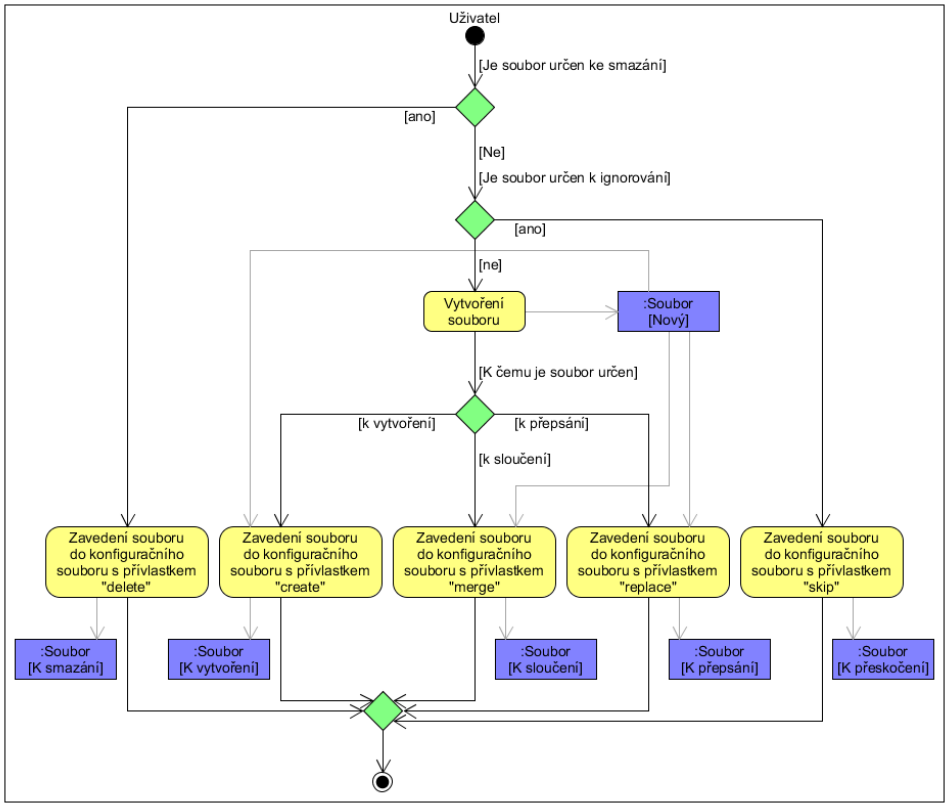
Název	Popis
Přepsat soubor	Pokud se zjistí, že `změněný neaktuální` a `originální` jsou stejné soubory, tak k žádnému slučování docházet nemusí a soubory se můžou přepsat soubory z aktualizace a proces slučování je možné ukončit.
Diff (2)	Pokud `změněný neaktuální` a `originální` jsou rozdílné bude muset slučování proběhnout. V tomto kroku se získá `DiffPatch[Aktualizace]` z originálního a aktualizovaného souboru.
Automatické sloučení	Pokud rozdíly neupravují stejné řádky a nevzniká kolize, tak může proběhnout automatické sloučení, kdy se vytvoří nový soubor z originálního a řádky v něm se přepíšou změněnými řádky ze souboru z aplikace a souboru z aktualizace. Poté může proces aktualizace skončit. Vzniká soubor typu `aktuální změněný`.
Manuální sloučení	Pokud rozdíly upravují stejné řádky a vzniká kolize, tak musí proběhnout manuální sloučení. Nekolizní řádky můžou být sloučeny automaticky jako v aktivitě `Automatické sloučení`, ale kolizní řádky budou muset být sloučeny uživatelem. Poté může proces aktualizace skončit. Vzniká soubor, který je `aktuální změněný`.

1.6.4 Tvorba aktualizačního balíčku

Aktualizační balíček je zkomprimovaný archiv obsahující aktualizované soubory aplikace. Balíček je navržený tak, že bude soubory obsahovat ve stejné relativní souborové struktuře, jako kdyby byly v aplikaci. Na základě toho si musí konfigurační soubor, který je v balíčku též obsažen, pamatovat pouze to, jaké soubory bude jakou operací aktualizovat. Nemusí si pamatovat na jaké místo v aplikaci je bude vkládat. Zjištěním relativní cesty od kořenového adresáře aktualizace lze tyto soubory namapovat na kořenový adresář aplikace.

Operace `DELETE` a `SKIP` jsou speciální a nemají svůj soubor v aktualizačním konfiguračním souboru, ale pouze záznam, jaký soubor v aplikaci bude mazán.

V předchozí kapitole bylo zmíněno, že pro slučování je potřeba originální nezměněný soubor. Tento soubor bude také obsažen v aktualizačním balíčku v relativní podobě, akorát jeho kořenový adresář se posouvá o úroveň níž do složky `.original`.



Obrázek 1.8: Proces vytváření aktualizačního balíčku

Tabulka 1.9: Popis diagramu pro vytváření aktualizačního balíčku

Název	**Popis**
Zavedení souboru do konfiguračního souboru s přívlastkem „delete“	Pokud je soubor určen ke smazání, tak se nemusí vytvářet a jedinou operací je zapsání cesty k souboru s přívlastkem `DELETE` nebo zkráceně `D`.
Zavedení souboru do konfiguračního souboru s přívlastkem „skip“	Pokud je soubor určen k přeskočení, tak se nemusí vytvářet a jedinou operací je zapsání cesty k souboru s přívlastkem `SKIP` nebo zkráceně `S`. Soubor určený k přeskočení bude ignorován, pokud bude na jeho rodičovskou složku aplikované nějaké pravidlo.
Vytvoření souboru	Vzhledem k tomu, že k následujícím operacím jsou potřeba data, musí se tato data nejdřív vytvořit.

Název	Popis
Zavedení souboru do konfiguračního souboru s přívlastkem „create“	Soubory, které je potřeba vytvořit, stačí už jenom zapsat do konfiguračního souboru s přívlastkem `CREATE` nebo zkráceně `C`.
Zavedení souboru do konfiguračního souboru s přívlastkem „merge“	Soubory, které je potřeba sloučit, stačí už jenom zapsat do konfiguračního souboru s přívlastkem `MERGE` nebo zkráceně `M`.
Zavedení souboru do konfiguračního souboru s přívlastkem „replace“	Soubory, které je potřeba nahradit, stačí už jenom zapsat do konfiguračního souboru s přívlastkem `REPLACE` nebo zkráceně `R`.

Konfigurační balíček bude také obsahovat scripty. Scripty bude třeba zavést do aktualizačního konfiguračního souboru. Jedná se o `SHUTDOWNSCRIPT` (zkráceně `SHS`), script pro vypnutí aktualizované aplikace, `BEFORESCRIPT` (zkráceně `BES`), script pro přípravu před aktualizací, `AFTERSCRIPT` (zkráceně `AFS`), script, který se spustí po dokončení souborových operací a `STARTUPSCRIPT` (zkráceně `STS`), script, který zajistí spuštění aktualizované aplikace.

Pokud bude v aktualizačním balíčku soubor, který nemá záznam v konfiguračním souboru, bude na něj aplikované pravidlo rodičovské složky nebo pravidlo „vlož nebo přepiš“. Rodičovské pravidlo se použije, pokud je jedna z rodičovských složek zaznamenaná v konfiguračním souboru s jakýmkoliv pravidlem. Ve druhém případě, kdy žádný rodičovský záznam nebyl nalezen, se soubor vloží do aktualizované aplikace nebo se v ní přepíše již existující soubor. Tímto se přeskočí validace kontrolující existence a neexistence souborů v aktualizované aplikaci.

Aktualizační konfigurační soubor ponese jméno `updateconfig.cfg` a pod tímto jménem bude po rozbalení balíčku vyhledán. Pokud nebude nalezen, aktualizace skončí chybou. Následující ukázka 1.1 je obsah nápovědy v konfiguračním souboru a popisuje strukturu jazyka konfiguračního souboru. Jakýkoliv text, který se nachází za symbolem `#` je považován za komentář.

Zdrojový kód 1.1: Návod pro tvorbu updateconfig.cfg

```
# Name of this file must be "updateconfig.cfg"
#
# Config structure: "<OPERATION_IDENTIFIER> <RELATIVE_PATH>"
#
```

```
# Example:
# i ./dir/file.css
# delete ./dirB
# BES dirC/file.sh
#
# Symbol '#' starts comment. Everything after this symbol is
    ↪ ignored.
# Relative path must be same in update package as in
    ↪ installed application.
# File can be directory. If it is directory then operation is
    ↪  recursively used for its subfiles.
# If file is not written in this config then Insert or
    ↪ Replace operation is used. This file is not validated
    ↪ for its existence in installed application.
#
# Use this operation identifiers for files
# i,c,insert,create: File for insertion, must not exist.
# d,delete: File for deletion, must exist.
# r,replace: File for replace, must exist.
# m,merge: File for merging, must exist.
# s,skip: File is not skipped if parent directory is targeted
    ↪  to an operation.
# sts,startup: Startup script, must be unique in package.
# shs,shutdown: Shutdown script, must be unique in package.
# bes,before: Before script, must be unique in package.
# afs,after: After script, must be unique in package.
```

1.7 Stavové diagramy

V této kapitole jsou zaznamenány stavy nejpodstatnějších objektů aktualizátoru. Jedná se o objekty s větším počtem stavů a složitějšími přechody mezi nimi.

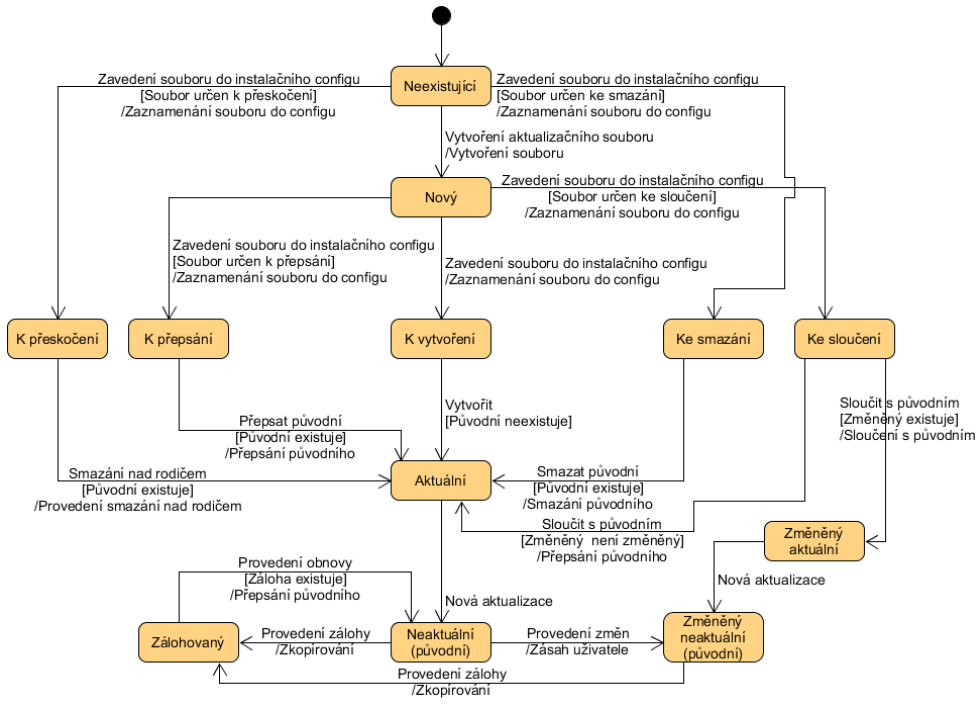
1.7.1 Soubor jako byznys objekt

Následující kapitola zachycuje složitější vnímání souborů jako byznys objektů z pohledu aktualizátoru. Stavový diagram 1.9 se snaží objasnit jednotlivé přechody mezi stavy. O detailnější vysvětlení jednotlivých stavů se stará přiložená tabulka 1.10.

Tabulka 1.10: Popis stavů souboru jako byznys objektu

Název	**Popis**
Neexistující	Soubor ještě není nijak evidovatelný systémem. Tvůrce aktualizačního balíčku si zatím rozmýšlí existenci takového souboru.

Název	**Popis**
K přeskočení	Soubor se nemusí vytvářet. Je určen pouze k ignorování aktualizátorem. Je to z důvodu, kdyby nad jeho rodičovskou složkou byla definovaná nějaká operace (například `DELETE`), tak aby se na soubor neaplikovala.
Ke smazání	Soubor neexistující v aktualizačním balíčku, ale je zavedený v aktualizačním konfiguračním souboru s označením ke smazání.
Nový	Soubor existující v aktualizačním balíčku, ale není zavedený v konfiguračním souboru aktualizace. Na takovýto soubor bude aplikované pravidlo vlož nebo přepiš.
K přepsání	Soubor existuje v aktualizačním balíčku a je zavedený v aktualizačním konfiguračním souboru s označením pro přepsání.
K vytvoření	Soubor existuje v aktualizačním balíčku a je zavedený v aktualizačním konfiguračním souboru s označením k vytvoření.
Aktuální	Pokud aktualizátor soubor zpracoval, tedy aktualizoval aplikaci, tak tento soubor z aktualizačního balíčku se stává souborem aplikace, a tedy i aktuálním souborem.
Ke sloučení	Soubor je sloučen se souborem neaktuálním, tedy původním souborem aplikace, podle pravidel uvedených v předchozích kapitolách. Původní soubor je fyzicky jiný soubor v jiném stavu.
Změněný aktuální	Po sloučení se z objektu `Ke sloučení` stane aktuální, ale změněný soubor aplikace.
Neaktuální původní	Pokud vznikne nová aktualizace pro aplikaci, tak všechny její soubory se stávají neaktuálními. Jedná se o soubory, které nejsou editované uživatelem.
Změněný neaktuální	Pokud vznikne nová aktualizace pro aplikaci, tak všechny její soubory se stávají neaktuálními. Jedná se o soubory, které jsou editované uživatelem.
Zálohovaný	Před začátkem aktualizace se neaktuální soubory zálohují pro případné vrácení změn.



Obrázek 1.9: Stavový diagram souboru jako byznys objektu

1.7.2 Stavy aktualizátoru

Aktualizátor může nabývat až šesti rozlišných stavů. Jejich přechody zachycuje stavový diagram 1.10 a jejich popis lze nalézt v tabulce 1.11[3].

Tabulka 1.11: Popis stavů aktualizátoru

Název	**Popis**
NOT RUNNING	„Neběžící“ – Po zapnutí aplikace aktualizátor nezačne okamžitě provádět aktualizace, musí se mu to explicitně říct. Aktualizátor přijímá aktualizace, ale nezpracovává je.
RUNNING WAITING	„Běžící, čekající“ – Po aktivaci aktualizátoru se dostane do stavu čekajícího na příchozí aktualizace, které může začít zpracovávat. V tomto stavu se nachází, pokud neexistuje nic, co by mohl aktualizovat.

[3]Stavy jsou v angličtině, protože odpovídají názvům konstant použitých v prototypu. Jejich překlad lze nalézt v tabulce.

Název	Popis
RUNNING UPDATING	„Běžící, aktualizující“ – V tomto stavu se aktualizátor nachází, pokud právě provádí aktualizaci jiné aplikace.
RUNNING STOPING	„Běžící, vypínající se“ – Do vypínajícího se stavu se dostane, pokud přijme signál pro vypnutí. Aktualizátor nechá doběhnout právě probíhající aktualizaci, aby předešel ztrátě dat.
CRASHED STOPPING	„Nefunkční, vypínající se“ – Pokud nastane kritická chyba, aktualizátor se přepne do stavu, kdy není schopen zpracovávat další aktualizace. Kvůli tomu, aby se předešlo ztrátě dat, je vyžadován zásah proškolené osoby. Aktualizátor se pokusí dokončit běžící aktualizace.
CRASHED	„Nefunkční“ – Z tohoto stavu nelze aplikaci dostat do běžícího stavu. Je to kvůli tomu, že pokud nastala kritická chyba, je potřeba oprava a zásah proškolené osoby. Aktualizátor se tím pokouší předejít dalším možným způsobeným problémům. Z tohoto stavu se lze dostat pouze restartováním aplikace.

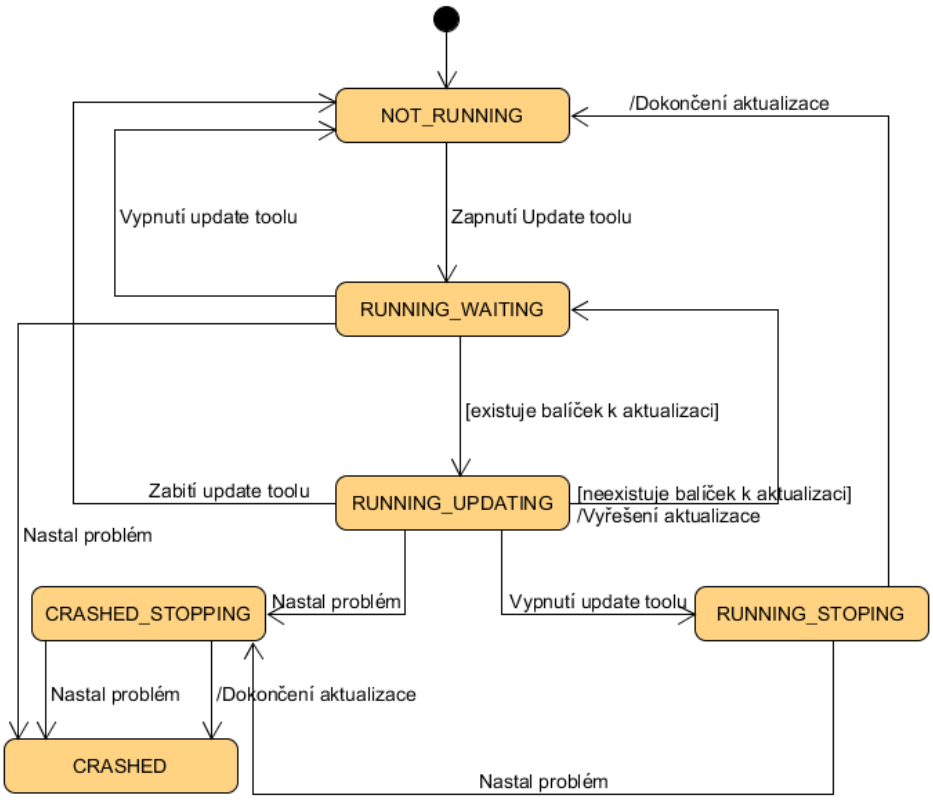
1.8 Návrh webu – průchod aktualizací

Uživatelským rozhraním musí (podle nefunkčního požadavku N1) být webová aplikace. Při návrhu UI je kladen zřetel na to, že aplikaci bude obsluhovat uživatel, který nemusí být příliš zdatný v ovládání počítače. Proto je volen jednoduchý web s jednoznačnými možnostmi pro pokračování. Je dbáno na informovanost, tedy každá obrazovka se bude pokoušet o to, aby jednoduchým způsobem dala na první pohled vědět, co se děje, a poskytla maximum informací. Toho bude docíleno kombinací designu, výstižného nadpisu a případně doplňujícím popisem.

Přihlašovací obrazovka D.1 Nejdříve se klient musí připojit vyplněním přihlašovacího hesla a jména. Bude také řešeno pomocí externí správy uživatelů (Spring Security). Zadavatel dodá funkcionalitu ověřování.

Kliknutím na logo se nic nestane.

Hlavní obrazovka 1.11 Na hlavní obrazovce se nachází informace o stavu serveru. Zda běží, je vypnut a další stavy, které jsou popsané v předchozích kapitolách. Nalezneme zde tlačítko pro zapnutí a vypnutí. Bude zobrazeno pouze to, které je aktuálně možné použít.

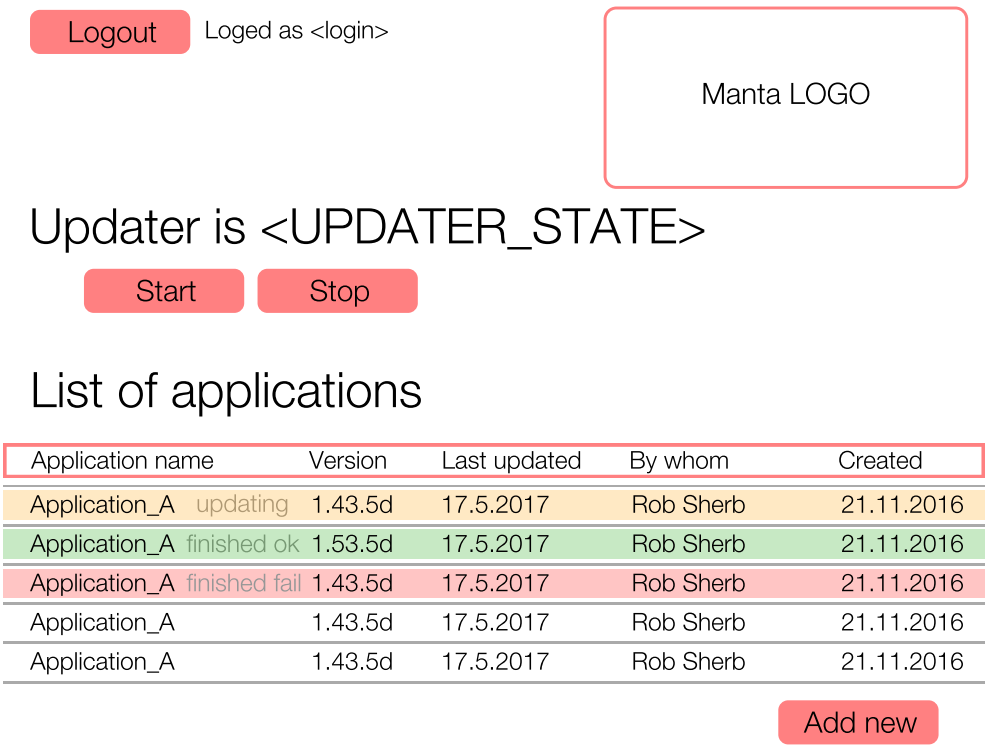


Obrázek 1.10: Stavový diagram aktualizátoru

Kromě informací o serveru zde budou zobrazené známé aplikace. Bude to jednoduchý seznam. Eviduje se hlavně název aplikace, který slouží pro její rozlišení. Pomocí podbarvení bude rozlišováno, jestli aplikace je aktuálně aktualizována (oranžová barva) nebo jestli aktualizace doběhla a je zapotřebí se podívat na výsledek aktualizace. Podbarvení informuje i o výsledku. Červené podbarvení se použije pro chybné aktualizace a zelené pro aktualizace, které doběhly v pořádku.

V záhlaví lze nalézt tlačítko pro odhlášení s informací, kdo je aktuálně přihlášen. Pod seznamem aplikací se nachází tlačítko na přidání další aplikace. Kliknutím na logo se na této obrazovce nic nestane. Na všech ostatních obrazovkách (pokud nebude řečeno jinak) bude kliknutím uživatel přesměrován na tuto obrazovku.

Přidání další aplikace D.3 Po kliknutí na tlačítko pro přidání další aplikace se zobrazí následující obrazovka. Obrazovka slouží k přidání kořenové složky a názvu aplikace do aktualizátoru. Název musí být jedinečný.



Obrázek 1.11: Wireframe – hlavní obrazovka s přehledem

Takto přidaná aplikace se zobrazí na hlavní obrazovce.

Po kliknutí na tlačítko `Browse` se zobrazí průzkumník serverového souborového systému. Cestu lze zadat i přímo.

Tlačítkem `Back` se dostane uživatel zpět na hlavní obrazovku a tlačítkem `Continue` potvrdí výběr.

Vybrání aktualizačního balíčku D.4 Obrazovka slouží k nahrání aktualizačního balíčku do aktualizátoru. Po kliknutí na tlačítko `Browse` se zobrazí průzkumník lokálního souborového systému. Cestu lze zadat i přímo.

Tlačítkem `Back` se dostane uživatel zpět na hlavní obrazovku a tlačítkem `Continue` potvrdí výběr, který ho přesune v lepším případě na slučování souborů. V opačném případě bude zobrazena obrazovka problému.

Problém D.5 Pokud nastane problém, který nesouvisí s chybou během aktualizace, zobrazí se unifikovaná obrazovka oznamující tento problém. Jednou z příčin může být nahrávání aktualizačního balíčku a jeho následné mapování na aplikaci.

Tlačítkem zpět se dostaneme na předchozí obrazovku, kde chyba nastala.

Slučování D.6 Po nahrání se vyřeší kolize a uživatel bude postupně vyzván ke sloučení všech souborů, které obsahují kolizi.

V levém sloupečku uživatel vidí, jak soubor vypadá aktuálně na serveru, a v pravém vidí, jak vypadá v aktualizaci. V prostředím sloupečku vytváří nový sloučený soubor. Nekolizní řádky budou vyřešeny automaticky a budou pouze zvýrazněné. Kolizní řádky budou obsahovat zobáčková tlačítka pro aplikování změny. V prostředním sloupečku bude možné provádět ruční editace.

Tlačítkem `Continue` se uživatel dostane na další soubor nebo na obrazovku potvrzení aktualizace. Tlačítkem `Back` se dostane na předchozí soubor nebo až na výběr aktualizačního balíčku.

Potvrzení aktualizace D.7 Jedná se o unifikovanou obrazovku, která se zobrazí, pokud bude cokoliv potřeba potvrdit.

Obrazovka obsahuje checkbox, kterým uživatel dává najevo, že si přečetl zobrazené informace, je obeznámen s následky jeho jednání a je rozhodnut pokračovat dál. Po zaškrtnutí a kliknutí na tlačítko `Continue` bude uživatel přesměrován na další obrazovku. Dále obsahuje tlačítko `Back`, které přesune uživatele na obrazovku, ze které se sem dostal.

Manuální vypnutí aplikace D.8 Pokud nebude možné vypnout aplikaci automaticky, bude uživatel vyzván k vypnutí manuálnímu.

Jestliže vypnutí bude možné, ale nepodaří se, tak aktualizace havaruje a zobrazí se průběhová obrazovka s neúspěšně ukončenou aktualizací.

Jedná se také o unifikovanou potvrzovací obrazovku.

Průběh aktualizace D.9 Na této obrazovce bude zobrazován průběh aktualizace prováděné aktualizátorem. Tato obrazovka bude také využita, pokud bude uživatel zjišťovat informace o aktualizaci prováděné jiným uživatelem nebo bude vyzvedávat výsledek aktualizace.

Tlačítkem `Continue` bude uživatel přesměrován buď na zapnutí aplikace, nebo na hlavní obrazovku.

Pokud aktualizace zhavaruje, bude zobrazené tlačítko `Revert`, které spustí automatický proces vrácení aplikace do stavu před aktualizací. Jestliže i proces obnovy zhavaruje, tak bude zobrazena tato obrazovka s tlačítkem `Continue`, které přesměruje uživatele na hlavní obrazovku. Aplikace bude ve stavu nepoužitelná a bude vyžadován zásah proškolené osoby.

Manuální zapnutí aplikace D.10 Když nebude možné zapnout aplikaci automaticky, bude uživatel vyzván k zapnutí manuálnímu.

V případě, že zapnutí bude možné, ale nepodaří se, tak aktualizace havaruje a zobrazí se průběhová obrazovka.

Jedná se také o unifikovanou potvrzovací obrazovku.

Tlačítkem `Continue` bude uživatel přesměrován na hlavní obrazovku.

Pokud uživatel přeruší aktualizační průběh v kterékoliv části, bude kliknutím na požadovanou aplikaci přesměrován do pozice, ve které skončil. Přerušit ho může kliknutím na logo, které většinou přesměrovává na hlavní obrazovku, nebo vypnutím okna prohlížeče. Toto pokračování se mapuje na uživatele, tedy jiný uživatel nemůže pokračovat v rozpracované aktualizaci. Takto rozpracovaná aktualizace podbarví aplikaci na hlavní obrazovce oranžovou barvou pro všechny uživatele. Oranžové aplikace jsou blokované a ostatní uživatelé na nich nemůžou provádět jiné aktualizace. Blokování je časově omezené a po definovaném čase vyprší. Pokud jiný uživatel klikne na oranžovou aplikaci, je mu zobrazena informace o stavu aktualizace.

Další aktualizaci nelze začít, dokud jakýkoliv uživatel nevyzvedne zprávu o proběhlé aktualizaci. Proběhlá aktualizace se signalizuje podbarvením aplikace zelenou nebo červenou barvou na hlavní obrazovce.

1.9 Existující řešení

Tato podkapitola se zaměřuje na existující řešení, jejich možné použití, výhody a nevýhody. Nerozebírají se zde aktualizační komponenty, které jsou součástí produktů, ale nástroje třetích stran určené pro distribuci. Webovou aplikaci, která by alespoň částečně splňovala požadavky zadavatele a umožňovala instalace a aktualizace softwaru, se nepodařilo najít.

1.9.1 Podle návodu

Nejedná se přímo o produkt třetí strany, ale jde o nejtriviálnější přístup k instalaci a aktualizaci, který přichází v úvahu. Jde o to, dodávat se softwarem instalační příručku, ve které je krok po kroku napsané, co má uživatel, který program instaluje, udělat. Součástí tohoto řešení můžou být nějaké automatické scripty, které můžou produkt, pokud budou mít dostatečné oprávnění, propsat do systémových registrů a na další místa.

Nevýhodou tohoto řešení je, že uživatel může kterýkoliv krok přeskočit. Tím může aplikaci dostat do stavu, ze kterého ji ani autoři softwaru nemusí dostat zpět. V lepším případě to skončí udělením vzdáleného přístupu ke klientovi a firma, která produkt dodává, provede instalaci sama. V horším případě to může skončit dlouhými hovory s uživatelem a slovním navigováním, kde se má co opravit a napsat.

Tento přístup lze aplikovat u malých firem s malým počtem zákazníků. Výhodou je cena a čas strávený implementací tohoto řešení, kde se obě hodnoty pohybují kolem nuly. Tento způsob je právě používán zadavatelem.

1.9.2 BitRock InstallBuilder

InstallBuilder je vývojářský nástroj na vytváření multiplatformních instalátorů pro stolní počítače a servery. Primárně je cílen na operační systémy Linux, Mac OS, Windows, Solaris, ale i další. Také podporuje automatické aktualizace. Pokud je aplikace už jednou nainstalovaná, tak další aktualizace jsou snadnější.[1][4]

Bohužel InstallBuilder nepodporuje slučování souborů, takové jako ho známe dnes. Slučování lze implementovat vlastním scriptem, případně zakázat přepsání souborů s časovou známkou starší, než byla provedena instalace (aktualizace).

Nástroj je velice vhodný pro prvotní nasazení. Bohužel, vzhledem k nemožnosti slučování, se nehodí pro projekty, kde se často mění konfigurační soubory a obsah těchto souborů je složitý. Od toho se odvíjí, že instalaci nezvládne kdokoliv. Na takovouto instalaci, kde je potřeba provést něco, čím uživatele neprovede instalační průvodce, je potřeba technicky zaměřený člověk s pokročilými znalostmi počítačů. Cena produktu začíná na $995.[2]

Instalační nástroj byl použitý na stejném projektu[5], jako byla testovaná tato práce. Instalátor se ukázal jako nevyhovující. Firma se vrátila k aktualizaci pomocí textového návodu a balíčku ZIP.

[4] Volný překlad autora.

[5] Manta Flow od firmy Manta Tools s.r.o.

Kapitola 2

Technický design

Na základě předchozích poznatků získaných z byznys analýzy lze vytvořit návrh aplikace, která by mohla sloužit nejenom zadavateli, ale i jiným společnostem, které řeší stejný problém jako Manta Tool s.r.o.

Následující kapitoly se zaměřují na návrh aplikace z technického pohledu. Popisuje se zde struktura aplikace, jednotlivé třídy aplikace, jejich vazby a vzájemná komunikace. Dále zde lze nalézt diagram nasazení a jaké jsou doporučené frameworky na vytvoření plnohodnotné aplikace.

2.1 Technologie a frameworky

V této kapitole jsou rozepsané některé aktuálně používané frameworky a řešení, které jsou aplikovatelné pro implementaci práce.

2.1.1 Webová část

Pro webovou část lze zvolit spoustu různých řešení a o žádném nelze říci, že je vyloženě špatné. Pro roli zprostředkovatele obsahu uživateli se může použít každý ze známých používaných frameworků. Mezi komunitami se vedou ostré diskuze a nelze jednoznačně říct, který je nejlepší.

Cílem této práce není porovnávat jednotlivé frameworky, a proto se zde pouze krátce představí dva zástupci z frameworků, které se používají pro generování obsahu na straně klienta (client-side) a dva zástupci z frameworků serverových (server-side). Dále jsou zde zmíněny dva podpůrné frameworky. Jeden na dynamické generování textových souborů a jeden rozšiřující jazyky, které se používají při tvorbě webových stránek.

2.1.1.1 AngularJS

„AngularJS je strukturární framework, který se používá pro tvorbu dynamických webových aplikací. Umožňuje použití HTML jako šablonovacího jazyka

a poskytne jeho rozšíření, aby šlo jasně a stručně vytvářet aplikační komponenty. Jeho datová vazba a injektování závislostí eliminuje spoustu kódu, které by bylo jinak nutné napsat. Všechno tohle se děje uvnitř prohlížeče. To dělá z Angular JS ideálního partnera pro jakoukoliv serverovou technologií.“[3][6]

„AngularJS není pouze jedním dílkem při vytváření „client-side“ webových aplikací. Obsluhuje veškerou spojovací DOM a AJAX část, která byla jednou ručně napsaná a vkládá jí do dobře navržené struktury. Tím AngularJS říká, jak by měly CRUD aplikace vypadat. Přestože říká, jakou cestou by se mělo jít, tak tato možnost je pouze výchozím bodem, který lze snadno změnit. [. . .]“[3][7]

2.1.1.2 React

„React je JavaScriptová knihovna pro vytváření webových komponent. V pomyslném MVC představuje *V* neboli view vrstvu a dal by se tak přirovnat například k Latte v Nette. React je však daleko více. Přináší totiž zásadní změnu paradigmatu. S Reactem už nepíšeme kód, který něco mění, ale kód, který popisuje, jak má vypadat výsledek, což je řádově snažší úloha. Dvojnásob to pak platí, pokud tím výsledkem je „těžká váha“ v podobě DOMu.“[4]

React byl původně vyvíjen firmou Facebook Inc., která ho roku 2013 věnovala komunitě. První odezva byla špatná. React se nedočkal pochopení. Dnes je situace jiná a tato knihovna má na GitHubu 80 601 commitů, 1 002 přispěvatelů.[4][5] „Je tak jedním z nejoblíbenějších a nejaktivnějších repozitářů na GitHubu.“[4]

Jedná se o nástroj, který slouží k tvoření „client-side“ dynamických webů pomocí technologie AJAX.

2.1.1.3 Spring MVC

„Spring Web model-view-controller (MVC) framework je designován jako `DispatcherServlet`, který vysílá žádosti konfigurovatelným obsluhovačům. Základní obsluhovač je postaven na anotacích `@Controller` a `@RequestMapping`, nabízející široké a flexibilní množství řídících metod. S příchodem Spring 3.0

[6]Překlad autora, originál: *AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.*

[7]Překlad autora, originál: *AngularJS is not a single piece in the overall puzzle of building the client-side of a web application. It handles all of the DOM and AJAX glue code you once wrote by hand and puts it in a well-defined structure. This makes AngularJS opinionated about how a CRUD (Create, Read, Update, Delete) application should be built. But while it is opinionated, it also tries to make sure that its opinion is just a starting point you can easily change. [. . .]*

kontroler začíná nabízet možnost tvořit RESTful webové aplikace pomocí `@PathVariable` anotací a další novinky.“[6][8]

Jedná se o framework, který se používá pro tvorbu server-side aplikací.

2.1.1.4 JavaServer Faces

„JavaServer Faces je standardně komponentově orientovaný framework pro Java Enterprise Edici. Používá se pro tvorbu uživatelského rozhraní. Označuje se také jako „Java webový framework“. Framework je již přiložen v Java EE platformě, takže může být použit pro tvorbu aplikací, aniž by bylo potřeba přidávat další knihovny. [...]“[7][9]

„JSF má dvě hlavní funkce. První je generování uživatelského rozhraní (typicky HTML) jako odpověď, která je předložená prohlížeči a zobrazena jako webová stránka. [...] Druhou funkcí je odpovídání na uživatelem generované události pomocí serverových metod označených jako „listeners“. Na základě těchto událostí dochází ke generování nového UI jinému uživateli nebo za použití AJAX technologie k aktualizaci již zobrazeného. [...]“[8][10]

Tento framework se používá pro tvorbu „server-side“ aplikací.

2.1.1.5 Apache FreeMarker

„Apache FreeMarker je šablonovací engine. Jedná se o Java knihovnu, která umožňuje generovat textový výstup založený na šabloně a změně dat. Šablony jsou napsané v FTL což je jednoduchý, specializovaný jazyk. Data pro zobrazení je třeba připravit za použití některého z běžně používaných programovacích jazyků. Můžeme tedy například vytvořit dotaz do databáze, provést výpočet a takto upravená data zobrazit v šabloně. Šablonovací engine se soustředí na zobrazení dat a ostatní kód pak na přípravu dat k prezentaci.“[9][11]

[8] Překlad autora, originál: *„The Spring Web model-view-controller (MVC) framework is designed around a `DispatcherServlet` that dispatches requests to handlers, with configurable handler mappings, view resolution, locale, time zone and theme resolution as well as support for uploading files. The default handler is based on the `@Controller` and `@RequestMapping` annotations, offering a wide range of flexible handling methods. With the introduction of Spring 3.0, the `@Controller` mechanism also allows you to create RESTful Web sites and applications, through the `@PathVariable` annotation and other features.“*

[9] Překlad autora: „JavaServer Faces (JSF) is the standard component-oriented user interface (UI) framework for the Java EE platform. In terms which may sound more familiar, it's a Java-based web framework. JSF is included in the Java EE platform, so you can create applications that use JSF without adding any extra libraries in your project. [...]“

[10] Překlad autora, originál: *„JSF has two main functions. The first is to generate a user interface, typically an HTML response that is served to a browser and viewed as a web page. [...] The second function of JSF is to respond to user-generated events in the page by invoking server-side listeners, followed by the generation of another user interface (e.g., web page) or an update to the user interface already displayed (possibly through Ajax). [...]“*

[11] Překlad autora, originál: *„Apache FreeMarker is a template engine: a Java library to generate text output (HTML web pages, e-mails, configuration files, source code, etc.) based on templates and changing data. Templates are written in the FreeMarker Template Language*

2.1.1.6 Bootstrap

„Bootstrap je HTML, CSS a Javascriptový framework určený na tvorbu responzivního webového projektu. Framework oplývá předpřipravenými UI komponentami.“[10][12] „Původně byl navrhnut designéry a vývojáři z Twitteru a stal se jedním z nejvíce populárních front-end frameworků a open-source projektů na světě.“[11][13] Bootstrap je stále aktivně vyvíjen a právě teď je na oficiálních webových stránkách dostupná alpha verze 4[14].

2.1.1.7 Volba

Frameworky byly rozděleny do hlavních dvou skupin – serverových a klientských. Aktualizátor je serverová aplikace. Veškeré logické funkčnosti musí probíhat na serveru a být synchronizované s ostatními uživateli, kteří aplikaci používají. U každé operace musí být zkontrolováno, zda je proveditelná. Tedy jestli jiný uživatel neprovedl něco, co by bránilo v provedení akce. Proto neexistuje moc funkčností, které by se obešly bez komunikace se serverem. Vhodnějším řešením je tedy server-side aplikace.

Rozhodnout, jaký serverový framework použít, usnadňuje zadavatel, který na svých ostatních projektech používá Spring MVC. Vývojáři pracující ve firmě Manta Tool s.r.o. jsou již s tímto frameworkem obeznámeni. Je tedy nežádoucí zavádět nové technologie, když to není vyloženě nutné.

Použitím předpřipravených komponent z Bootstrap se usnadní tvorba webu a generování šablon bude usnadněno enginem FreeMarker. Tato řešení jsou již rovněž používaná na projektech Manta Tool.

2.1.2 Slučování

Tato podkapitola se zabývá možnostmi slučování textových souborů. Popisuje známé technologie, knihovny a frameworky a jejich možné využití pro tuto práci.

(FTL), which is a simple, specialized language (not a full-blown programming language like PHP). You meant to prepare the data to display in a real programming language, like issue database queries and do business calculations, and then the template displays that already prepared data. In the template you are focusing on how to present the data, and outside the template you are focusing on what data to present.“

[12]Překlad autora, originál: *„Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.“*

[13]Překlad autora, originál: *„Originally created by a designer and a developer at Twitter, Bootstrap has become one of the most popular front-end frameworks and open source projects in the world.“*

[14]15. 5. 2017 – Během vzniku této práce došlo k aktualizaci a právě aktuální verzí je Alpha 6.

2.1.2.1 Javers

„JaVers je odlehčená a všestranná Java knihovna určená pro kontrolu změn v datech. [...] Nedělá předpoklady o datovém modelu nebo bean kontajneru. [...] Protože používá JSON pro serializaci, nepotřebuje detailní mapování jako jiné ORM nástroje. Je ale zapotřebí dodat JaVers high-level fakta o datovém modelu. [...]“[12][15]

JaVers je napsaný pro Java 7[16], nabízí podporu pro Spring a je zveřejněný pod licencí Apache License verze 2.0.[12]

2.1.2.2 Google Diff Match Patch

„Knihovny Diff Match and Patch nabízí robustní algoritmy pro synchronizaci textu. [...] V knihovně je použit Myerův diff algoritmus, který je považován za nejlepší diff algoritmus pro všeobecné použití. Knihovna implementuje Bitap matching algoritmus, který tvoří základ snadno přizpůsobitelným „matching a patching“ postupům.“[13][17]

Knihovna je ve verzích pro jazyky Java, JavaScript, Dart, C++, C#, Objective C, Lua a Python. Chrání ji licence Apache License 2.0. Bohužel má ukončenou podporu a nedostupné zdrojové kódy na SVN, tedy není veřejně k dispozici[18].[13]

2.1.2.3 Java Diff Util

„Diff Utils je open source knihovna, která provádí porovnávací operace mezi texty. Mezi operace se řadí vypočítávání změnových záznamů a jejich aplikování jako patche, generování unifikovaných změnových záznamů a jejich parsování, vytváření změnových záznamů pro jejich budoucí jednoduší zobrazení jako „side-by-side“.“[14][19]

[15] Překlad autora, originál: „JaVers is a lightweight java library for auditing changes in your data. [...] We don't make any assumptions about your data model, bean container or underlying data storage. [...] Since we use JSON for object serialization, we don't want you to provide detailed ORM-like mapping. JaVers only needs to know some high-level facts about your data model. [...]“

[16] 15. 5. 2017 – Během vzniku této práce došlo k aktualizaci a nejnovější verze podporuje pouze Javu 8.

[17] Překlad autora, originál: *„The Diff Match and Patch libraries offer robust algorithms to perform the operations required for synchronizing plain text. [...] This library implements Myer's diff algorithm which is generally considered to be the best general-purpose diff. This library also implements a Bitap matching algorithm at the heart of a flexible matching and patching strategy.“*

[18] 4. 5. 2017 – Během vzniku této práce došlo k aktualizaci a zdrojové kódy Google Diff Match Patch jsou zase k dispozici.

[19] Překlad autora, originál: „Diff Utils library is an OpenSource library for performing the comparison operations between texts: computing diffs, applying patches, generating unified diffs or parsing them, generating diff output for easy future displaying (like side-by-side view) and so on.“

„Knihovna používá pro svoje účely Myer's diff algoritmus, který je ale snadno nahraditelný.“[14][20] Tento algoritmus je používaný ve známém verzovacím systému Git.[15]

Diff Util je chráněna licencí The Apache Software License, Version 1.1.

2.1.2.4 Mergely

Mergely je webová aplikace určená pro prohlížení a slučování dvou dokumentů online. Jedná se o knihovnu napsanou čistě v jazyce Javascript.[16]

„Jádro knihovny je založené na Longest Common Subsequence diff algoritmu a uzpůsobitelným značkovacím engine. Mergely nabízí široké API, které umožňuje integraci do jiných projektů. Může být použité jako nástroj určený pro zobrazení rozdílů nebo i pro sloučení textových dokumentů. [...]“[17][21]

Mergely je Open source projekt založený na Closed Distribution License a pro svůj běh vyžaduje jQuery a CodeMirror.[17][18]

2.1.2.5 Volba

První vhodnou volbou se může zdát být knihovna Javers, se kterou se původně počítalo do prototypu. Bohužel její účel není úplně vhodný a typický pro potřeby aktualizátoru. Knihovna umožňuje verzování a zjišťování změn, ale pouze nad celými objekty. Speciální implementací by bylo možné dosáhnout kýženého cíle.

Protože Google Diff Match Patch nemá dostupné zdrojové kódy a Mergely je chráněn nevhodnou licencí, tak jako další možnost se naskytuje Java Diff Utils. Hlavní výhodou této knihovny je její jednoduchost a možnost se přizpůsobit potřebám projektu. Seznámení se s Diff Utils je díky její jednoduchosti mnohonásobně snazší než s jejím mohutným sokem Javers. Proto se stala favoritem a součástí prototypové implementace. Díky ní se úspěšně podařilo naimplementovat manuální i automatické slučování.

2.2 Architektura aplikace

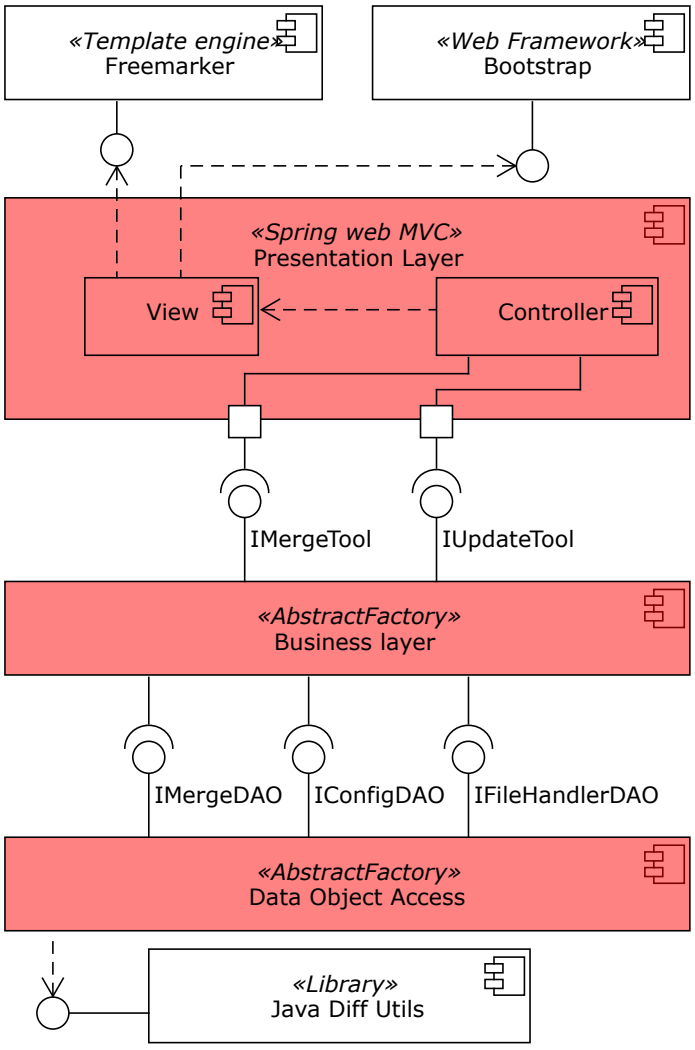
Pro vytvoření aplikace byla zvolená striktně třívrstvá architektura, která přináší bezespornou výhodu v možnosti oddělení jednotlivých vrstev a jejich případné nahrazení jinou implementací. Tento návrh například umožní snadnou změnu UI, bez zbytečných zásahů do ostatních vrstev.[19] Tato volba je

[20] Překlad autora, originál: This library implements Myer's diff algorithm. But it can easily replaced by any other which is better for handing your texts.

[21] Překlad autora, originál: *„The core of mergely is a javascript-based Longest Common Subsequence diff algorithm (LCS) and customizable markup engine. Mergely provides a rich API that enables integration into your own application. It can be used as a diff tool (read-only) or as both a diff and merge tool for plain text [...]“*

opodstatněna tím, že někteří uživatelé mohou v budoucnu upřednostnit desktopovou nebo konzolovou aplikaci před webovou aplikací, ke které je ještě zapotřebí webový server.

Přístup k datové a byznys vrstvě je zajištěn pomocí rozhraní a návrhového vzoru abstract factory. Tento vzor umožňuje za běhu aplikace zvolit implementaci daného rozhraní. [20]



Obrázek 2.1: Architektura aplikace

2.3 Diagramy tříd

Pro zachycení struktury aplikace podle jednotlivých tříd byl také zvolen jazyk UML. V této části budou rozepsané pouze nejpodstatnější komponenty.

Nepopsané části je možné vyčíst z diagramů nebo případně z JavaDoc.

2.3.1 Datová vrstva

Datová vrstva se stará o správu souborového systému, čtení aktualizačního konfiguračního obsahu a získávání rozdílů mezi dvěma textovými obsahy. Je rozdělená do třech funkčních celků.

FileHandlerDAO E.1 se zabývá veškerými operacemi se souborovým systémem.

Umožňuje rozbalení archivu, smazání nebo zkopírování souboru. Dále předepisuje implementaci čtení a zápisu do souboru. Nabízí možnost získání obsahu adresáře rekurzivně bez složek nebo pouze aktuálního včetně složek. Vytváří zálohu aplikace. Pro aktualizační balíček umí nalézt konfigurační soubor a získat originální soubor k jinému souboru.

Také řeší problematiku s dočasnými adresáři a soubory, které jsou použity během procesu aktualizace. Tato část musí být řešena dvěma způsoby. Jeden využívá dočasný adresář operačního systému a druhý umožní nastavení adresáře manuálně.

`FileHandlerDAO` využívá strukturu `FileDataObject`, která obaluje objekt `File` a `Charset`. Je to z toho důvodu, aby třídy vyšších vrstev nepoužívali příliš specifický prvek jako je `File` a kódování. Tyto třídy pro ně totiž nepřináší nic procesně důležitého. `FileDataObject` umožňuje přidání relativní cesty na konec své cesty a odříznutí absolutní části, čímž se vytvoří relativní cesta. Odřezávání a připojování se používá při mapování aktualizačního balíčku na data aktualizované aplikace.

ConfigDAO E.2 slouží jako překladač z textového formátu konfiguračního souboru do objektového modelu. Využívá `UpdateConfig` jako uložiště.

MergeDAO E.3 se používá k řešení rozdílů mezi dvěma seznamy řetězců. Umožňuje získat jejich rozdíly, vytvořit z toho rozdílovou strukturu a tu případně na jeden z nich aplikovat.

Struktura rozdílů je tvořená obalem `DifferenceCover`, který si pamatuje originální tvar, změněný tvar a jejich rozdíly. Jeden rozdíl (objekt `Difference`) se dělí na tři typy: smazání, změna a vložení. Dále obsahuje kontexty originálního a upraveného obsahu. Kontext `DifferenceContext` je už pouze odkaz na obsah. Jedná se o jednotlivé řádky rozdílu, na kterém řádku v obsahu je lze nalézt a z toho jde vypočítat, jak je kontext velký.

Předchozí struktura byla inspirována strukturou z knihovny Java Diff Utils a pro jejich propojení byly vytvořené třídy `PatchDifferenceCover`, `DeltaDifference` a `ChunkDifferenceContext`. Tyto třídy zastiňují gettery a settery a data přesměrovávají do objektů knihovny.

Protože se jedná o struktury bez funkčností, tak v datové vrstvě je zadefinovaný i `MergeResult`, který se používá jako návratová hodnota v Byznys vrstvě. Obsahuje částečně nebo úplně sloučený obsah a kolize (`Collision`), tedy rozdíly, které nešlo sloučit.

2.3.2 Byznys vrstva

Byznys vrstva obsahuje logickou část aplikace. Stará se o server, jednotlivé aktualizace a slučování změn získaných z datové vrstvy. Dělí se na dvě části.

MergeTool E.5 slouží ke slučování obsahů na základě rozdílů, které byly získány z `MergeDAO`. Využívá `MergeBundle` jako strukturu pro pamatování dat.

UpdateTool E.6 Komplexnější `UpdateTool` řídí celý proces aktualizace. Zajišťuje možnost spustit jednotlivé procesy aktualizace a také je vypnout. Jako byznys objekt pro aktualizaci využívá `UpdateBundleBO`. Tyto objekty přijímá pomocí `beginUpdate` a pokud je ve stavu, kdy může aktualizovat, provádí na těchto objektech aktualizace.

`UpdateBundleBO` může nabývat různých stavů, které je potřeba zmínit. Prvním stavem je `Created`, který je při úpravě objektu změněn na `Preparing`. Dále se spouští validace `validateUpdate`, která zjišťuje, jestli vůbec dává smysl aktualizaci spouštět. Jestli není už předem rozhodnuto, že dopadne chybně. Pokud validace projde, nastaví se stav `Waiting_for_update` a na základě kontroly tohoto stavu si může být `UpdateWorker` jistý, že může začít s aktualizací.

Pro aktualizace se využívá `UpdateWorker`, doporučená implementace bude producent jako `UpdateTool` a `UpdateWorker` jako konzument. Při mazání celých složek, se zjišťuje jejich obsah, který se následně kontroluje v aktualizační konfiguraci. Předchází se tím smazáním souborů ve složce, kterému byla určena jiná operace. Toto mapování se musí provést před vkládáním nových souborů. Pokud by soubory byly vloženy na základě operace, kterou podědily od rodičovské složky, tak by mohlo dojít k jejich smazání.

Nejjednodušším řešením vypínání a zapínání aplikací je napsat scripty, které se budou dodávat s aktualizací. Pro tento případ je navržen spouštěč scriptů s názvem `ScriptExecutorBO`, který si tyto scripty drží a umí je spouštět. Jsou zde i scripty, které se spouští před a po aktualizaci. Dají se využít pro případné nastavení proměnných prostředí a zásahů do registrů.

Jaká aplikace je aktualizovaná je uloženo v `UpdateApplication`.

2.3.3 Prezentační vrstva

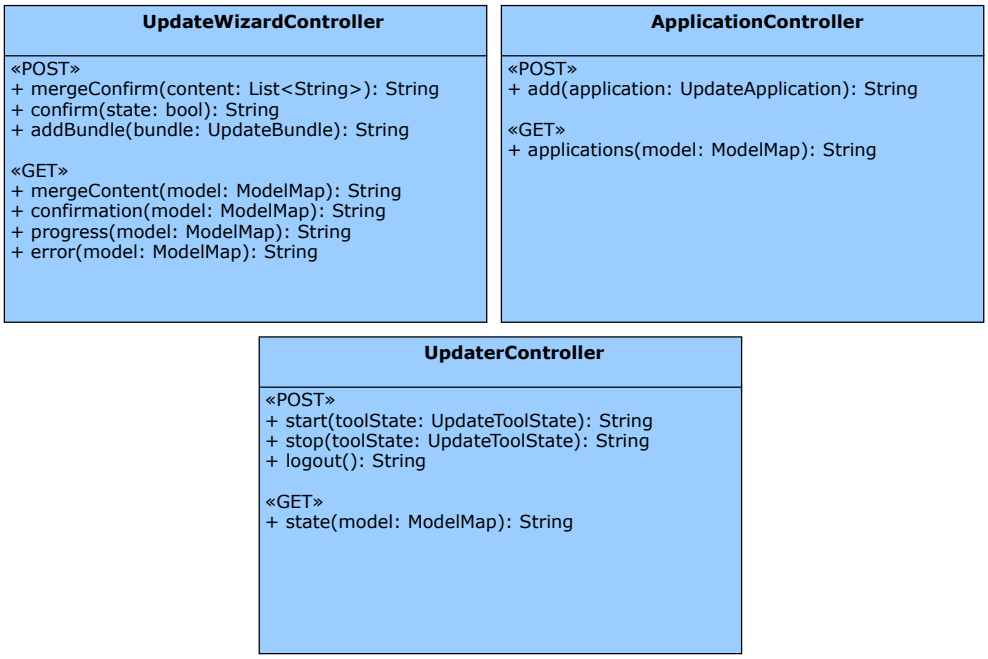
Prezentační vrstva 2.2 se stará o komunikaci aplikace s uživatelem. Bude obsahovat tři obslužní třídy typu `Controller`.

UpdateController se stará o ovládání serveru, aktualizátoru. Umožňuje ho vypnout zapnout a zjistit jeho stav. Oznamuje také serveru, že uživatel se odhlásil.

ApplicationController řeší správu známých aplikací. Umožňuje získat jejich seznam nebo případně novou aplikaci přidat do seznamu.

UpdateWizardController provádí uživatele aktualizací. Umožňuje slučování a zobrazování stavu aktualizace. Také přijímá aktualizační balíček a řeší potvrzovací obrazovky.

Potvrzovací obrazovky se synchronizují se serverem, aby se provedlo jejich zalogování. Jedná se o právně bezpečnostní ochranu. Aplikace pracuje s daty uživatele. Pokud by uživatel nesprávným zacházením o data přijde a byl před tím varován, je možnost se odvolat, že byl na tuto skutečnost upozorněn a jedná se o chybu způsobenou nesprávným zacházením.

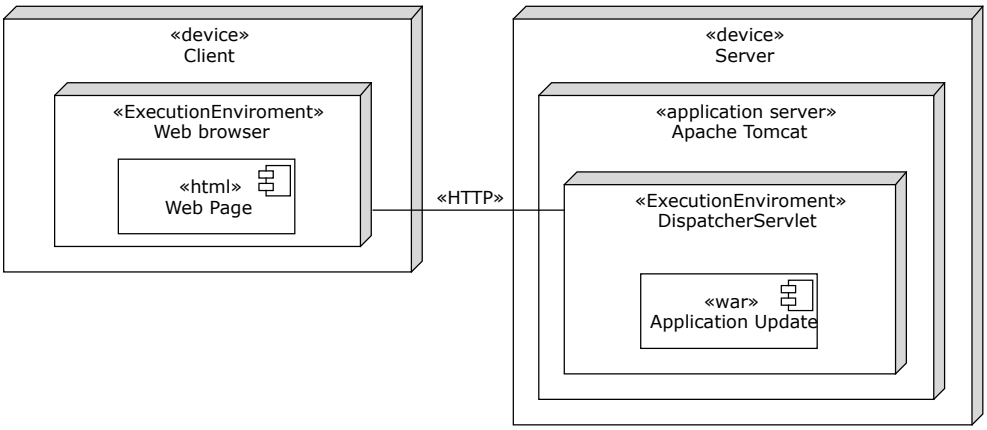


Obrázek 2.2: Diagram tříd - prezentační vrstva

2.4 Nasazení

Jako každá webová aplikace, tak i aktualizátor bude mít část běhu na zařízení uživatele. V prostředí internetového prohlížeče se bude zobrazovat webová stránka a spouštět javascript.

Na druhé straně je server s běžícím aplikačním serverem. V diagramu 2.3 je jako příklad uveden Apache Tomcat, ale nejedná se o jediné možné řešení. HTTP požadavky zpracovává `DispatcherServlet`, který na základě mapování volá odpovídající `Controller` a `View`. [21] Aplikace je distribuovaná jako balíček WAR do aplikačního serveru.



Obrázek 2.3: Diagram nasazení

Kapitola 3

Prototyp

Následující kapitoly popisují vytvořený prototyp a jeho testování.

3.1 Implementace

Součástí prototypu je implementace datové a byznys vrstvy, ke kterým lze přistupovat pomocí jednoduchého konzolového uživatelského rozhraní. Toto rozhraní umožňuje vykonat aktualizaci nebo instalaci jiné aplikace na stejném zařízení.

Pro přístup k byznys a datové vrstvě byl použit návrhový vzor abstraktní továrny.

3.1.1 Datová vrstva

Datová vrstva je zcela implementovaná pomocí standardní knihovny pro práci se soubory a Java Diff Utils pro zjišťování rozdílů mezi seznamy textových řetězců.

FileHandlerDAO Řešení operací v souborovém systému je řešeno pomocí standardní knihovny `java.nio.file`. Data objekty si pamatují své kódování. Výchozím kódováním je UTF-8. V budoucnu je počítáno s tím, že každý soubor může být jinak kódován. Tato skutečnost se bude nastavovat pomocí UI. Bude možné nastavit výchozí kódování a i kódování jednotlivých souborů.

Třída umožňuje nastavení složky, ve které se budou ukládat dočasné soubory. Pokud složka není nastavena před první použitím, tak se inicializuje na `temp` složku operačního systému. Možným alternativním řešením je ukládat dočasné soubory v adresáři aktualizátoru. Prototyp neumí mazat dočasné soubory.

Pro aktualizační balíčky je podporovaný formát archivace ZIP. Pro operaci s archivy ZIP se používá knihovna `net.lingala.zip4j`.

Záloha se nahrává do kořenového adresáře aktualizované aplikace. Vytvoří se složka `.backup` ve které se vytvoří složka s názvem tvořeným aktuálním datem a časem. V této složce se vytvoří archiv ZIP s obsahem aktualizované aplikace. Při zálohování se složka `.backup` ignoruje a v zálohách se neobjevuje.

ConfigDAO `ConfigDAO` řeší formát konfiguračního obsahu a převádí ho do datové struktury. O převod souboru na obsah se stará `FileHandlerDAO`.

MergeDAO `MergeDAO` je implementovaný podle technického návrhu. Stará se o zjišťování rozdílů mezi dvěma seznamy textových řetězců. Řádek v souboru je reprezentován jako záznam v seznamu. O převod souboru na seznam se stará `FileHandlerDAO`.

Veškerá nastavení, kromě kódování jednotlivých souborů, které musí zadat uživatel, bude ve výsledné aplikaci řešeno konfiguračním souborem aktualizátoru.

3.1.2 Byznys vrstva

Byznys vrstva se stará o logickou část aplikace. Je zde implementovaná část provádění aktualizací, ovládání serveru a slučování souborů. Část spravující verze aplikací není v prototypu implementovaná. Do těl těchto metod bylo přidáno vyhození výjimky `NotImplementedException`.

UpdateTool `UpdateTool` spouští samotný proces aktualizace, o který se stará `UpdateWorker`. `UpdateWorker` dává informace o tom, co vykonává, `UpdateTool`. Ten na základě těchto informací může poskytnou prezentační vrstvě informace o průběhu `UpdateTool`. Při dotazu, v jakém stavu se server nachází, vrací `UpdateTool` poslední známé informace.

Stav aktualizace řeší `UpdateWorker`, který nastavuje vnitřní stav balíčku aktualizace (`UpdateBundle`) pomocí referencí. Pokud prezentační vrstva potřebuje vědět stav aktualizace určitého balíčku, může si ji získat přímo z balíčku.

Metody, které pracují s atributy jednotlivých tříd, jsou opatřeny synchronizační ochranou proti vícevláknovému přístupu.

Prototyp má nastavenou hodnotu maximálního počtu konzumentů na jedna. Ve výsledné aplikaci bude umožněno tuto hodnotu měnit pomocí konfiguračního souboru.

MergeTool `MergeTool` řeší zjišťování kolizí na základě získaných rozdílů z datové vrstvy.

Z datové vrstvy se získá seřazený seznam rozdílů z originálního a změněného obsahu a druhý seznam rozdílů z originálního a aktualizovaného

obsahu. Tyto rozdíly se postupně z těchto seznamů odebírají a kontroluje se, jaký rozsah řádků v originálním souboru upravují.

Algoritmus, který zjišťuje, jestli dva rozdíly kolidují, se musí rozdělit na tři případy. Důležitá informace je, že pokud změna nese informaci o přidávání řádku, tak tento rozdíl ukazuje, že začíná na řádku předchozím.

Pouze jeden rozdíl říká, že se řádek přidává. Příklad: vkládáme mezi řádky A a B řádek C. Změna nesoucí informaci o přidání řádku C říká, že začíná na řádku A a i na řádku A končí. Aby se jednalo o kolizi, tak by musela druhá změna, se kterou se toto přidání řádku porovnává, začínat na řádku A a končit na řádku B, tedy na vloženém řádku. 3.1

Zdrojový kód 3.1: Podmínka – pouze jeden rozdíl typu vložení

```
beginInsert >= beginB && beginInsert < endB
```

Pokud obě změny vkládají řádky textu tak, aby se nejednalo o kolizi, musí se zkontrolovat, jestli rozdíly nezačínají na stejném řádku. 3.2

Zdrojový kód 3.2: Podmínka – oba rozdíly typu vložení

```
beginInsertA == beginInsertB
```

Pokud ani jedna změna nepopisuje vkládání nových řádků, tak stačí zjistit, jestli řádky, které upravují, mají neprázdný průnik.

Předchozí algoritmus přijímá jako jeden z jeho argumentů následující rozdíl ze seznamu vytvořeného z upraveného obsahu a jako druhý argument rozdíl ze seznamu aktualizovaného. Pokud algoritmus řekne, že rozdíly nekolidují, je rozdíl, který končí na dřívějším řádku, považován za vyřešený a místo něj se použije do algoritmu další čekající rozdíl. Vždycky po skončení algoritmu se považuje za vyřešený pouze jeden z argumentů algoritmu. V dalším kroku se druhý argument znovu kontroluje na kolizi.

Druhou možností je, že algoritmus označí rozdíly za kolizní. V tomto případě se musí znovu zjistit, který z těchto argumentů končí na dřívějším řádku. Druhý argument může totiž kolidovat i s dalšími rozdíly. Rozdíl končící dřív se přidá do objektu `Collision` a jeho následník se použije pro další kontrolu se zbylým argumentem. Tento proces se může opakovat. Postupně se do jedné kolize může přidat více rozdílů z obou seznamů. Jakmile se zjistí, že argumenty už nekolidují, přidá se nepřidaný kolidující rozdíl z předchozí kontroly a kolize se uzavře jako vyřešená. Celý proces zjišťování se opakuje, dokud jeden ze seznamů není prázdný. Jakmile je jeden ze seznamů prázdný, zbylé rozdíly už nemůžou kolidovat.

Složitost procesu zjišťování kolizí je $\Theta(m + n)$, kde m je počet rozdílu z jednoho seznamu a n je počet rozdílu ze seznamu druhého.

3.2 Testování

Testování aplikace proběhlo dvěma způsoby. Jedním ze způsobů bylo, že při dokončení konzolového UI 3.1 se prováděly validní a správné průchody aktualizací. Jednalo se o simulaci ovládání aplikace uživatelem. Tento test byl prováděn na obou aplikacích Manta Flow.

```
Main (3) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (3. 5. 2017 9:44:08)
Update Tool started

Path to update bundle:
c:\Users\PGondek\Documents\TEST_ENVIROMENT\!data\manta-amex-dataflow-server-1.16.1.zip
Name of application to update:
Manta dataflow server
Path to update application:
c:\Users\PGondek\Documents\TEST_ENVIROMENT\manta-dataflow-server\
Creating update bundle
Validating
Validation ok. Beginning update
Update bundle state: WAITING_FOR_UPDATE, Update tool state: RUNNING_WAITING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: UPDATING, Update tool state: RUNNING_UPDATING
Update bundle state: FINISHED_OK, Update tool state: RUNNING_WAITING
Update is done.
```

Obrázek 3.1: Výsledek testu pomocí konzolového UI

Další způsob testování byl prováděn pomocí frameworku JUnit. Nejedná se však o typické unit testy. Testy se tvořily od nejprimitivnějších funkčností, k těm složitějším, ve kterých se pro přípravu testovacích dat používaly metody již otestované. Bylo to z důvodu úspory času a složitosti přípravy některých testovacích dat. Také to umožnilo dynamicky měnit testovací data, upravovat je a opravovat. Ne ve všech případech se jedná i o testy validační. Tyto testy byly odloženy, protože nepřináší pro prototypovou implementaci zisk, který by odpovídal ceně za jejich tvorbu.

MergeToolTest kontroluje slučování na základě získaných rozdílů z datové vrstvy. Testuje plně automatickou aktualizaci, plně manuální a kombinovanou, kde je sloučeno, co sloučené může být a zbytek čeká na vyjádření. Manuální sloučení je testované na datech bez kolize a na datech s kolizí. Kontroluje se také obsah kolizí.

UpdateToolTest testuje tvorbu a přípravu dat pro vytvoření aktualizačního balíčku. Také kontroluje spouštění a vypínání aktualizace a přechody stavů aktualizačního nástroje.

UpdateWorkerTest testuje spuštění aktualizace, kontroluje přechody stavů aktualizačního balíčku. Jako testovací data se použil balíček s aplikací bez kolizí v souborech. Kontroluje výsledek aktualizace.

UpdateBundleBOTest testuje provádění validace aktualizačního balíčku a lazy inicializaci balíčků pro sloučení.

ScriptExecutorBOTest testuje spouštění skriptů pro Windows.

FileHandlerDAOTest kontroluje rozbalení archivu ZIP, nalezení konfiguračního souboru v aktualizačním balíčku, vytvoření zálohy a získávání originálního souboru pro soubor zadaný parametrem.

Z operací s dočasnými soubory se testuje tvorba a inicializace dočasné kořenové složky pro všechny dočasné soubory a složky, tvorba dočasné složky pro aktualizovanou aplikaci a tvorba dočasných souborů.

Také je testované kopírování a mazání souborů, získání rekurzivního i nerekurzivního obsahu složky, zápis do souboru a čtení ze souboru.

MergeDAOTest testuje získání rozdílů z dvou obsahů a aplikování získaných rozdílů na obsah.

ConfigDAOTest testuje překládání konfiguračního obsahu do datového modelu.

UpdateConfigTest testuje vlastní přístupové metody k datům.

FileDataObjectTest testuje kromě přetížených metod `equals` a `hashcode` připojení jiného relativního datového objektu a vytvoření datového objektu na základě určení absolutní části.

Závěr

Byl vytvořen návrh nástroje, který ulehčuje proces instalace a aktualizace jiné webové aplikace spuštěné na stejném serveru jako aktualizátor. Tento návrh byl otestován vytvořením prototypu. Na základě prototypové implementace lze prohlásit, že dokončení aplikace má smysl. Výsledný produkt opravdu může ulehčit procesy nasazení nejenom firmě Manta Tools s.r.o. Prototyp umožňuje povést instalaci nebo aktualizaci jiné aplikace včetně automatického sloučení konfiguračního souboru. Pokud by během slučování nastala kolize, prototyp na tuto skutečnost upozorní. Je tedy možné říci, že vytyčené cíle byly naplněny.

Produkt je vhodný hlavně na opakované aktualizace jiných aplikací. Pro jednorázové instalace je vhodnější využít instalátory, jako je InstallBuilder. Neznamená to, že aktualizátor je pro prvotní nasazení nepoužitelný. Pouze uživatelé nejsou s takovými průvodci, jako je aktualizátor, seznámeni.

Jak bylo řečeno, do budoucna lze určitě dokončit prototyp a udělat z něj plnohodnotnou aplikaci, která bude mít implementovanou prezentační vrstvu podle návrhu.

Pro další vývoj aplikace je jisté, že se musí dodělat validační testy a testy chybových průchodů. Aktuální testy kontrolují pouze správný a validní průchod aktualizačním průvodcem. Je pravděpodobné, že prototypová implementace obsahuje chyby, které je potřeba detekovat těmito novými testy.

Bylo by také vhodné implementovat bezpečnostní ochranu, kontrolovat digitální podpisy aktualizačních balíčků. Prototyp počítá s tím, že uživatel ví, jaký instalační balíček mu předkládá a zná jeho původ. V tento moment je toto nemalý bezpečnostní nedostatek.

Aplikace by taky mohla umět aktualizovat sama sebe. Stávající implementace s touto funkčností nepočítá, protože při instalaci (aktualizaci) aplikace provádí mnoho kontrol, zaručuje bezpečné obnovení předchozí verze při selhání a hlavně aktualizovaná aplikace musí být vypnutá. Řešením by bylo sám sebe ukončit a spustit vlastní aktualizační script. Bohužel toto přináší s sebou spoustu rizik. Otázkou k zamyšlení zůstává, jestli by na vlastní aktualizaci nešel použít nástroj třetí strany na tvorbu instalačních balíčků.

Za zvážení by také stálo, zda nevytvořit lepší vypínaní aplikací, které se mají aktualizovat. Musí se vzít v potaz, že každá aplikace může být spuštěná na jiném operačním systému a také jiným způsobem.

Nabízí se ještě mnoho dalších vylepšení jako multithreading jedné aktualizace, do konfiguračního souboru umožnit přidání operace na kořenový adresář, připravit se na možnost, že aktualizovaná aplikace bude ve formátu WAR, snadnější přejmenovávání souborů, instalace pouze scriptem a další.

Literatura

[1] BitRock Inc.: InstallBuilder Overview [online]. ©2016, [Cited 2016-12-4]. Dostupné z: https://installbuilder.bitrock.com/installbuilder.html

[2] BitRock Inc.: Purchase an InstallBuilder License [online]. ©2016, [Cited 2016-12-4]. Dostupné z: https://installbuilder.bitrock.com/purchase.html

[3] Google Inc.: *Developer Guide - Introduction [online]*. ©2010-2017, [Cited 2016-5-15]. Dostupné z: https://docs.angularjs.org/guide/introduction

[4] Dresler, R.: React - Úvod. *DžejEs [online]*, [cited 2017-5-12]. Dostupné z: https://www.dzejes.cz/react-uvod.html

[5] Facebook Inc.: GitHub - facebook/react [online]. ©2017, [Cited 2017-5-2]. Dostupné z: https://github.com/facebook/react

[6] Johnson, R. a kolektiv: *Spring Framework Reference Documentation, Part VI. The Web, Chapter 22. Web MVC framework [online]*. ©2004-2016, [Cited 2016-12-1]. Dostupné z: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

[7] JavaServer Faces.org: Get Started [online]. ©2012, [Cited 2017-5-2]. Dostupné z: http://www.javaserverfaces.org/

[8] JavaServer Faces.org: JSF in a nutshell [online]. ©2012, [Cited 2017-5-2]. Dostupné z: http://www.javaserverfaces.org/

[9] The Apache Software Foundation: What is Apache FreeMarker? [online]. ©1999-2016, [Cited 2016-12-9]. Dostupné z: http://freemarker.org/

[10] Otto, M. a Thornton, J: Bootstrap [online]. ©2016, [Cited 2016-12-9]. Dostupné z: http://getbootstrap.com/

[11] Otto, M. a Thornton, J: Bootstrap - About [online]. ©2016, [Cited 2016-12-9]. Dostupné z: http://getbootstrap.com/about/

[12] JaVers: *JaVers - Documentation [online]*. [Cited 2016-12-11]. Dostupné z: http://javers.org/documentation/

[13] Fraser, N.: Google Diff Match Patch [online]. [Cited 2016-12-11]. Dostupné z: https://code.google.com/p/google-diff-match-patch/

[14] Naumenko, D.: GitHub - dnaumenko/java-diff-utils [online]. ©2017, [Cited 2017-5-2]. Dostupné z: https://github.com/dnaumenko/java-diff-utils

[15] Coglan, J.: The Myers diff algorithm: part 1. ©2017, [Cited 2017-5-2]. Dostupné z: https://blog.jcoglan.com/2017/02/12/the-myers-diff-algorithm-part-1/

[16] Peabody, J.: Mergely [online]. [Cited 2016-12-11]. Dostupné z: http://www.mergely.com/

[17] Peabody, J.: *Mergely Reference Manual [online]*. [Cited 2016-12-11]. Dostupné z: http://www.mergely.com/doc

[18] Peabody, J.: Mergely Licensel [online]. [Cited 2016-12-11]. Dostupné z: http://www.mergely.com/license

[19] Dresler, R.: Vícevrstvé architektury aplikací. *Vývoj software není jenom o kódování ...[online]*, dubna 2011, [cited 2017-5-12]. Dostupné z: http://www.robertdresler.cz/2011/04/vicevrstve-architektury-aplikaci.html

[20] Object Oriented Design: Abstract Factory Pattern. ©2006, [Cited 2017-5-10]. Dostupné z: http://www.oodesign.com/abstract-factory-pattern.html

[21] Tutorials Point: Spring - MVC Framework. ©2017, [Cited 2017-5-4]. Dostupné z: https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm

Příloha A

Seznam použitých zkratek

AJAX Asynchronní JavaScript a XML

API Aplikační programovací rozhraní

CRUD Create, Read, Update, Delete (Vytvořit, číst, aktualizovat, mazat)

DOM Data object model

FTL FreeMarker šablonový jazyk

HTTP Hypertext přenosový protokol

IS Informační systém

Java EE Java Enterprise edice

JS JavaScript

JSF Java server faces

JSON JavaScript objektová notace

LDAP Aplikační protokol pro dotazování a modifikaci adresářových služeb

MVC Model View Controller

MVW Model View Whatever

OS Operační systém

ORM Objektově relační zobrazení

SI Softwarové inženýrství

Spring MVC Spring web model-view-component

SVN Subversion

SW Software

UI Uživatelské rozhraní

WAR Web aplikační archiv

XML Rozšiřitelný značkovací jazyk

ZIP Komprimovaný souborový formát

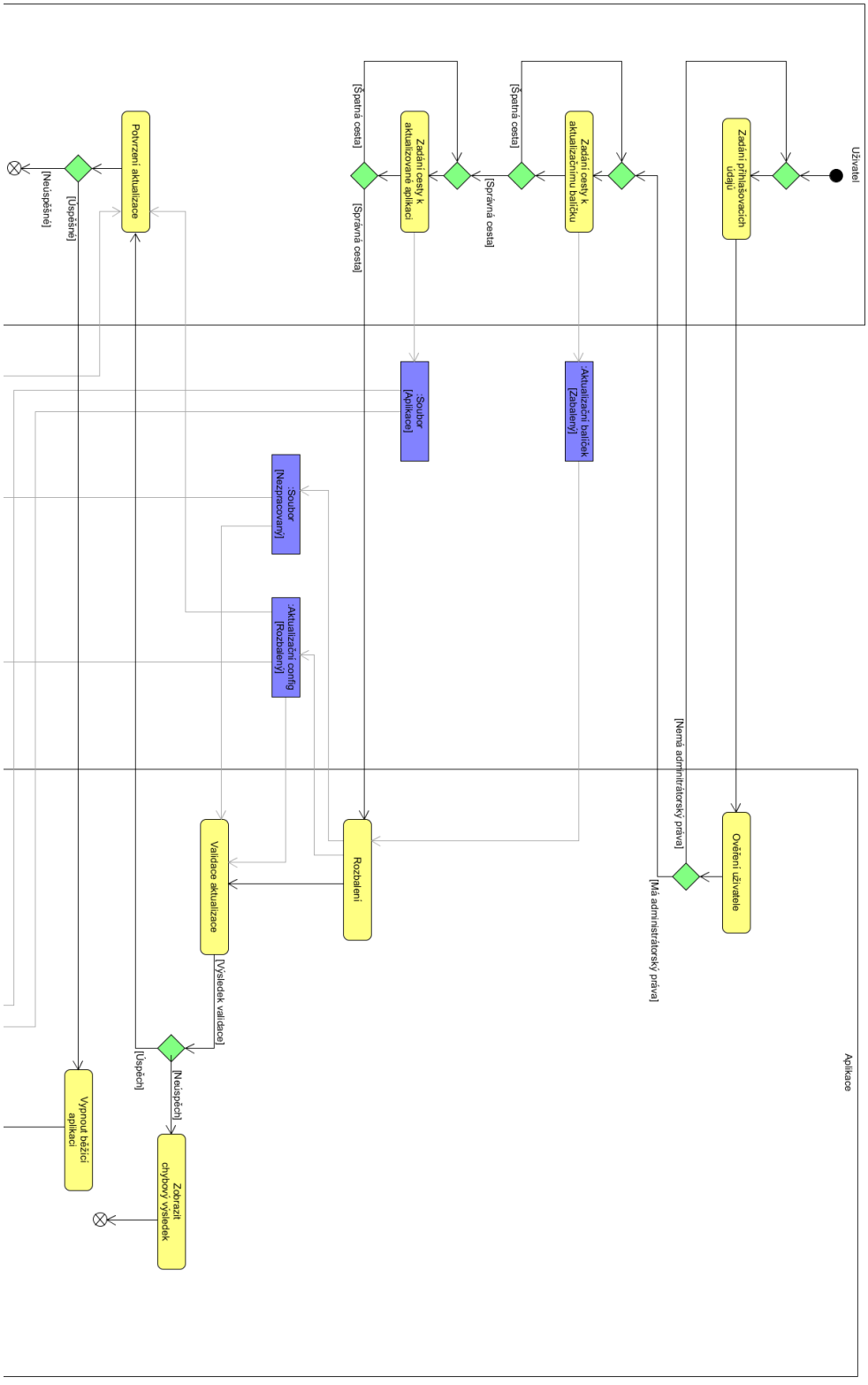
Příloha **B**

Obsah přiloženého CD

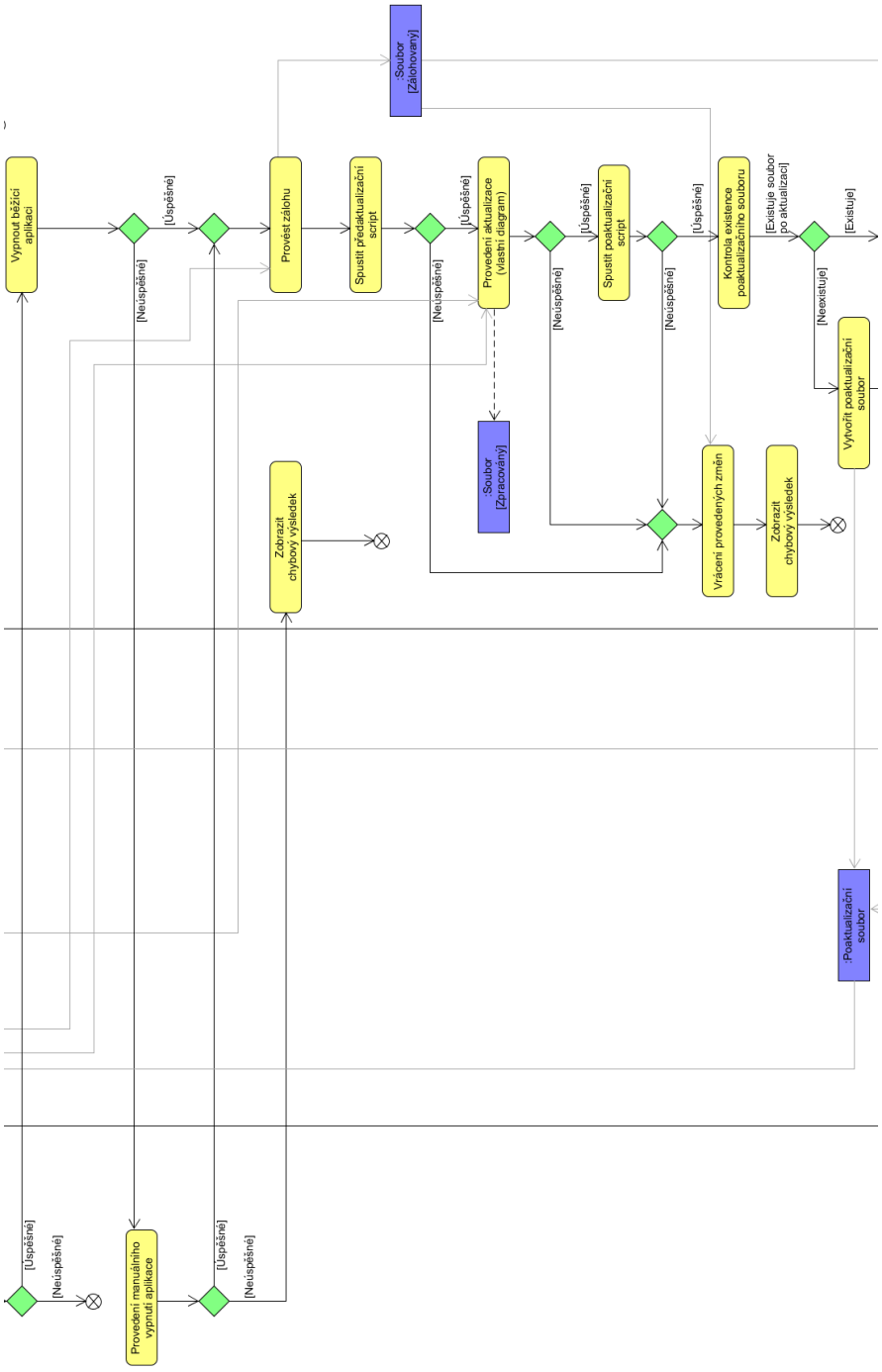
- `readme.txt` ..... stručný popis obsahu CD
- `attachments` ..... přílohy a výstupy aplikace
- `documentation` ..... dokumentace
  - `diagrams` ..... dokumentace pomocí diagramů
  - `javadoc` ..... dokumentace zdrojových kódů
- `exe` ..... adresář se spustitelnou formou implementace
- `src` ..... zdrojové kódy
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Gondek_Petr_2017.pdf` ..... text práce ve formátu PDF

Příloha C

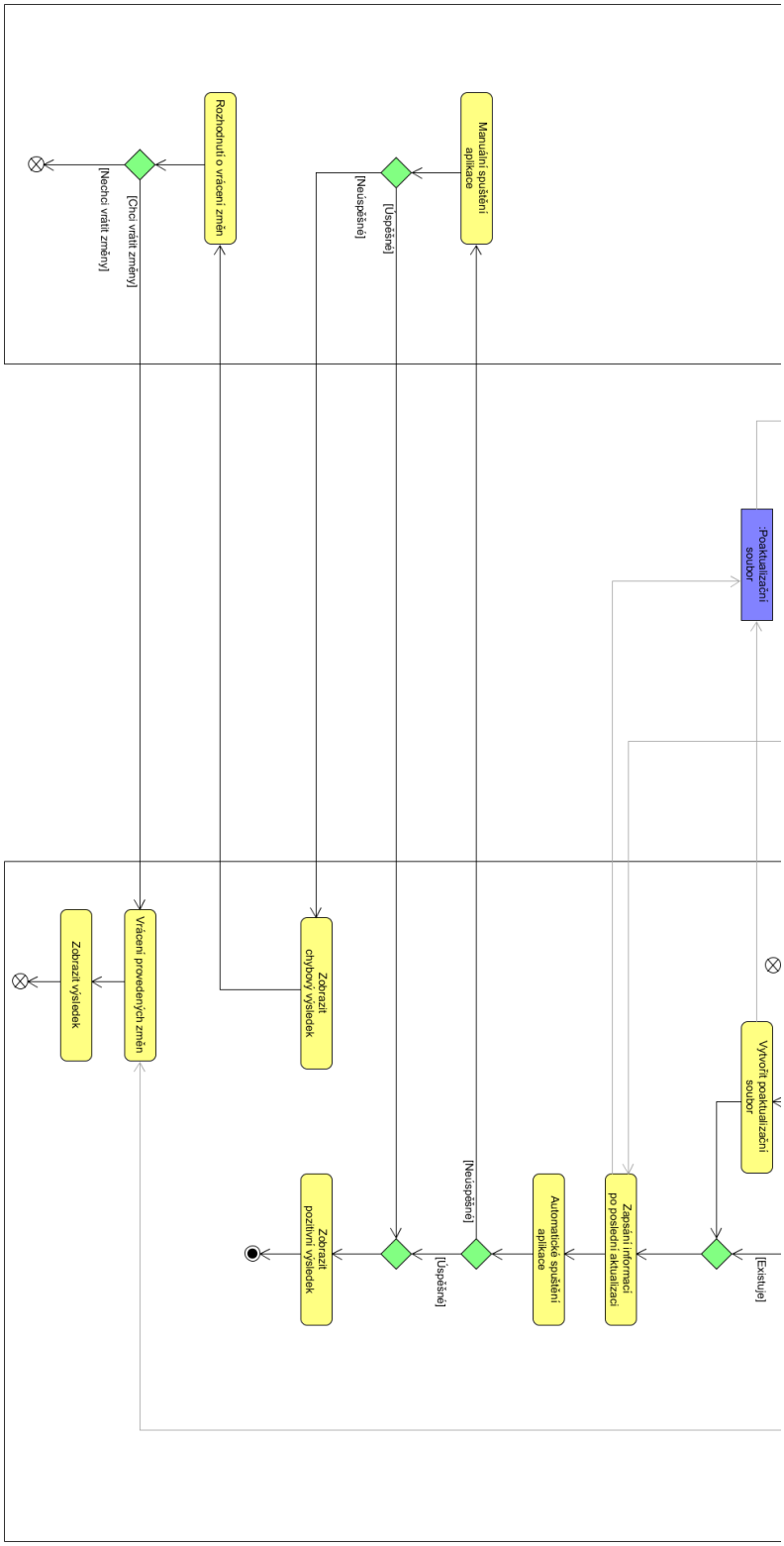
Diagram aktivit - nová aktualizace



Obrázek C.1: Diagram aktivit – nová aktualizace 1. část



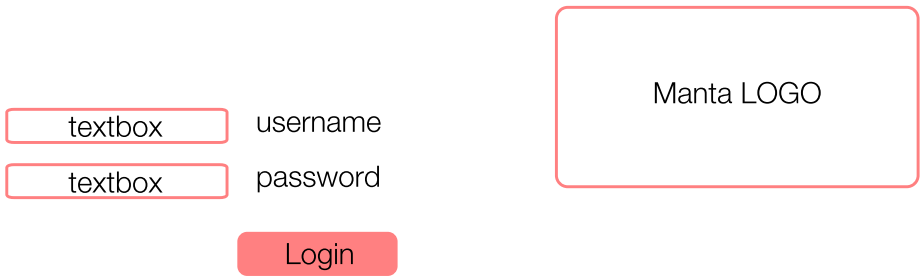
Obrázek C.2: Diagram aktivit – nová aktualizace 2. část



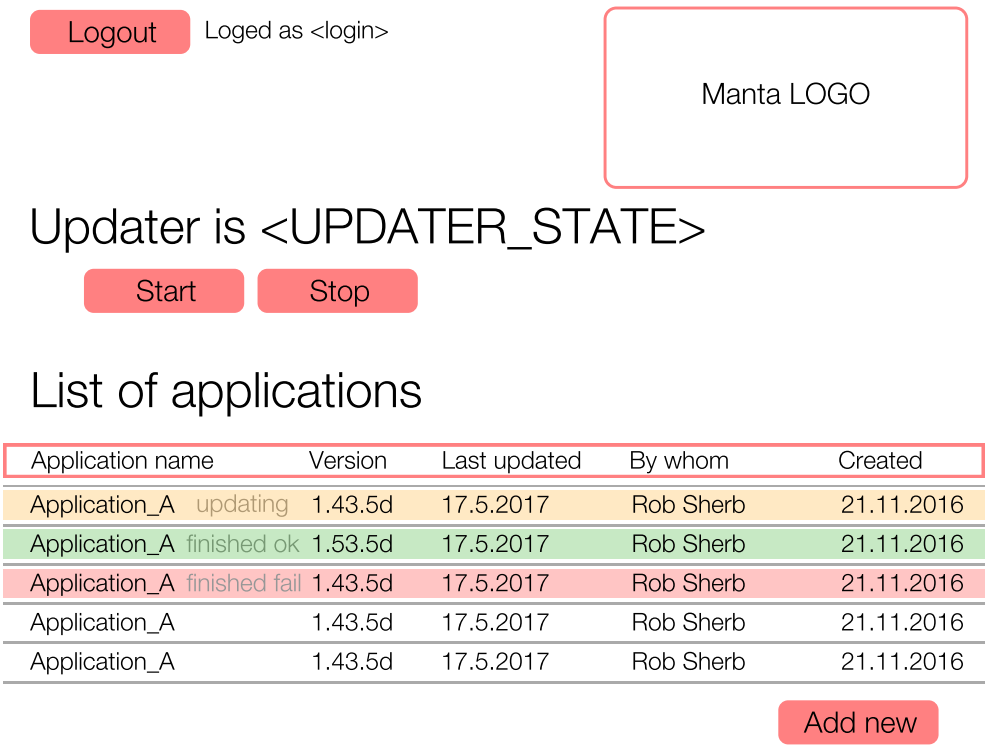
Obrázek C.3: Diagram aktivit – nová aktualizace 3. část

Příloha D

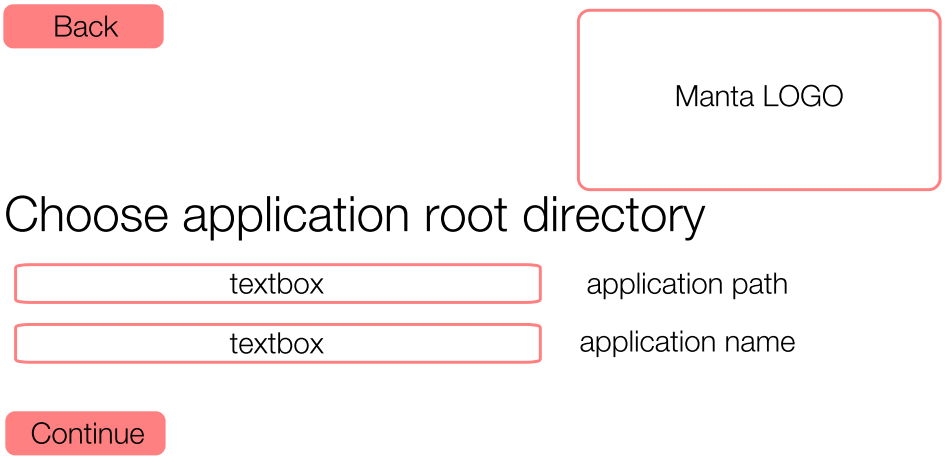
Wireframes



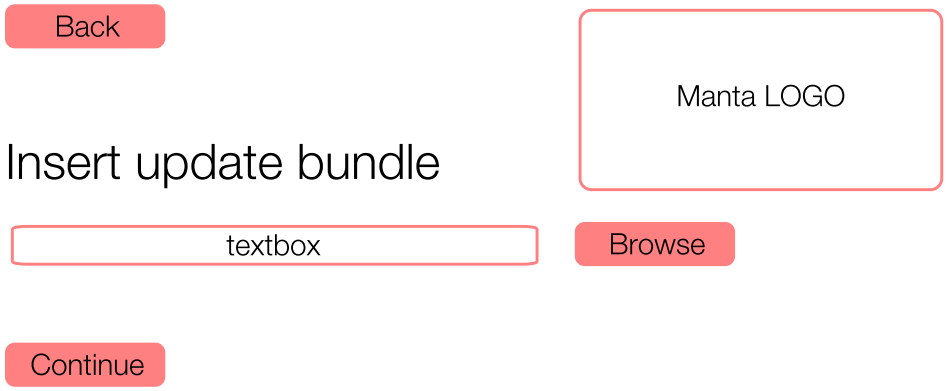
Obrázek D.1: Wireframe – Přihlášení



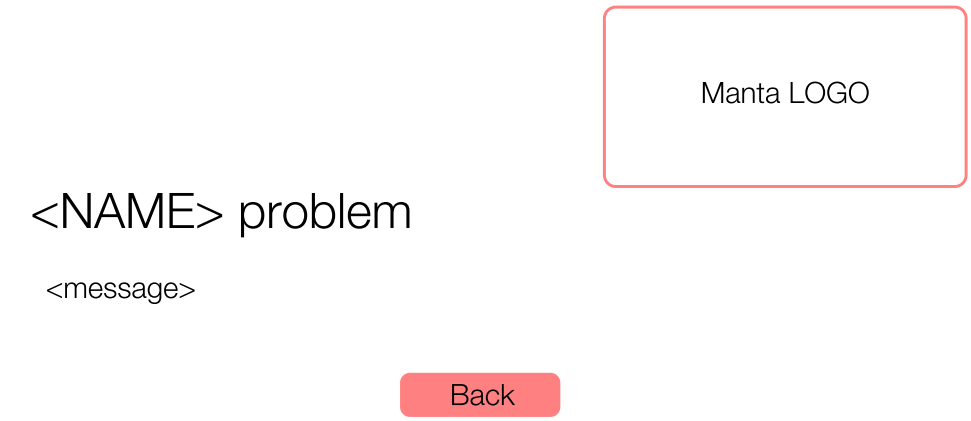
Obrázek D.2: Wireframe – Hlavní obrazovka



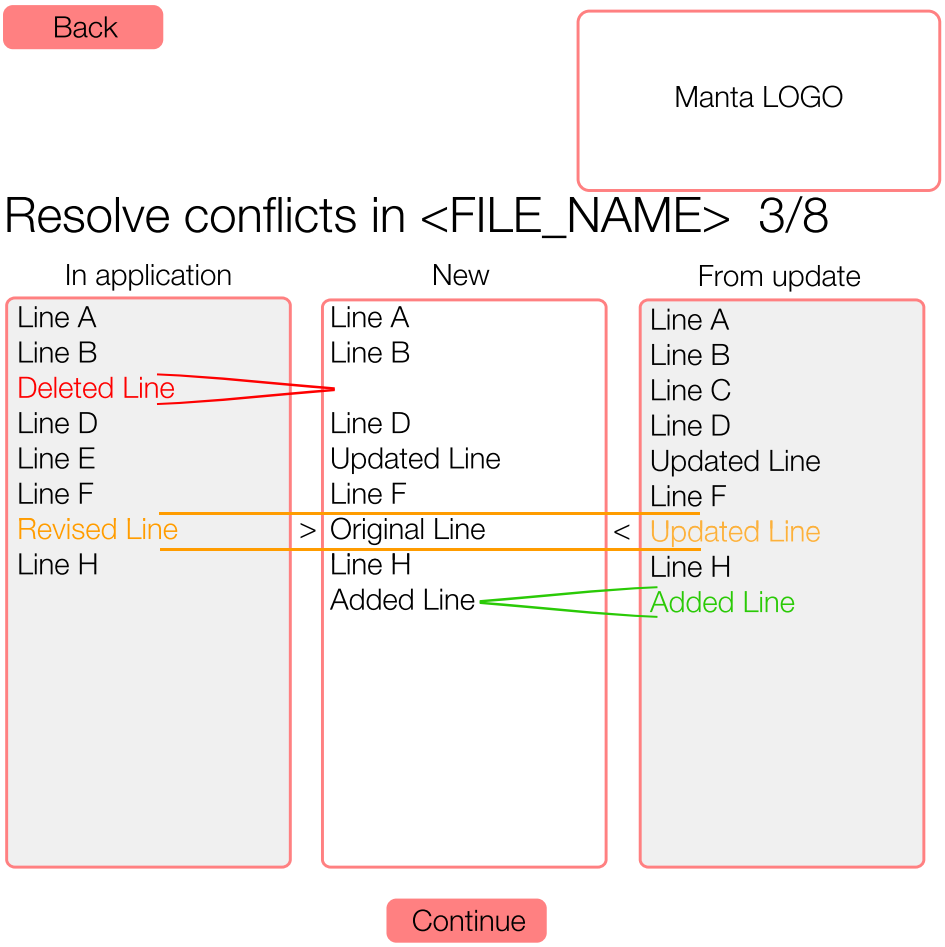
Obrázek D.3: Wireframe – Přidání aplikace



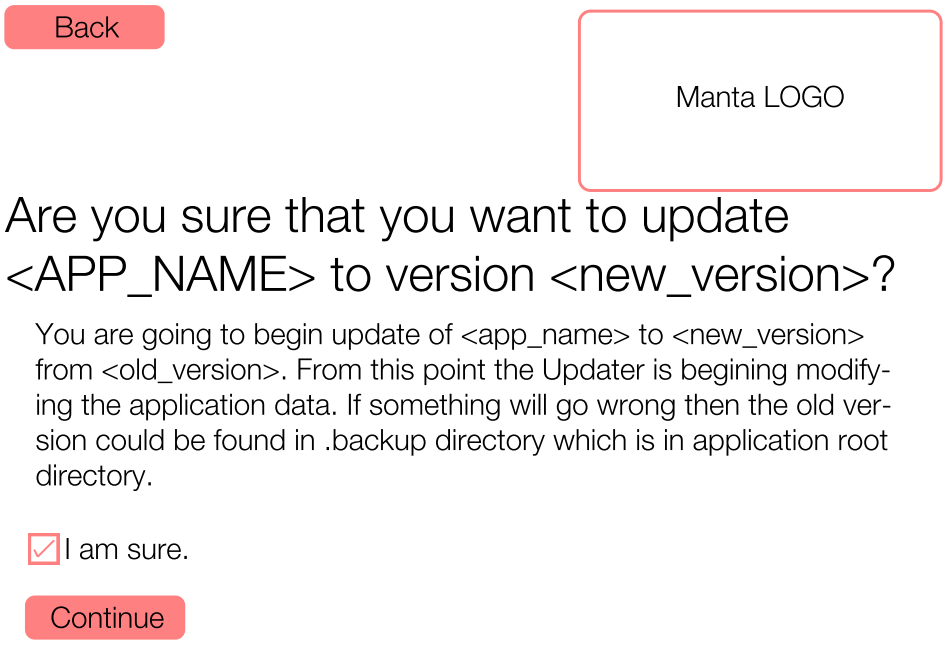
Obrázek D.4: Wireframe – Vybrání balíčku



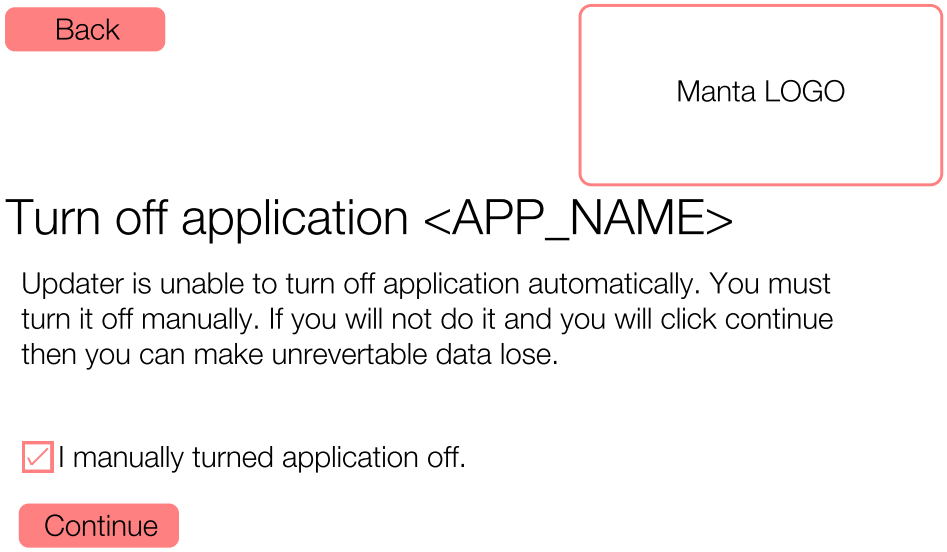
Obrázek D.5: Wireframe – Problém



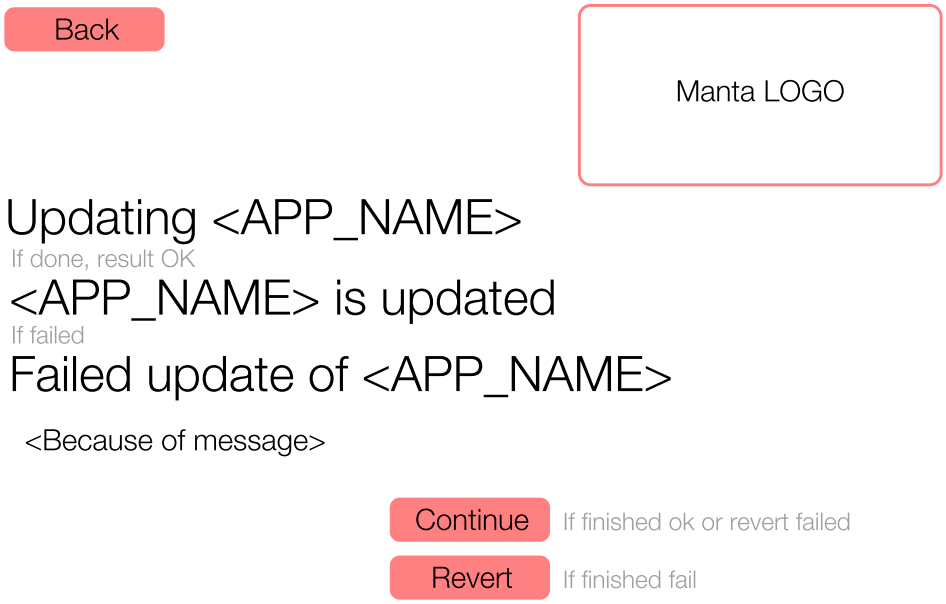
Obrázek D.6: Wireframe – Slučování



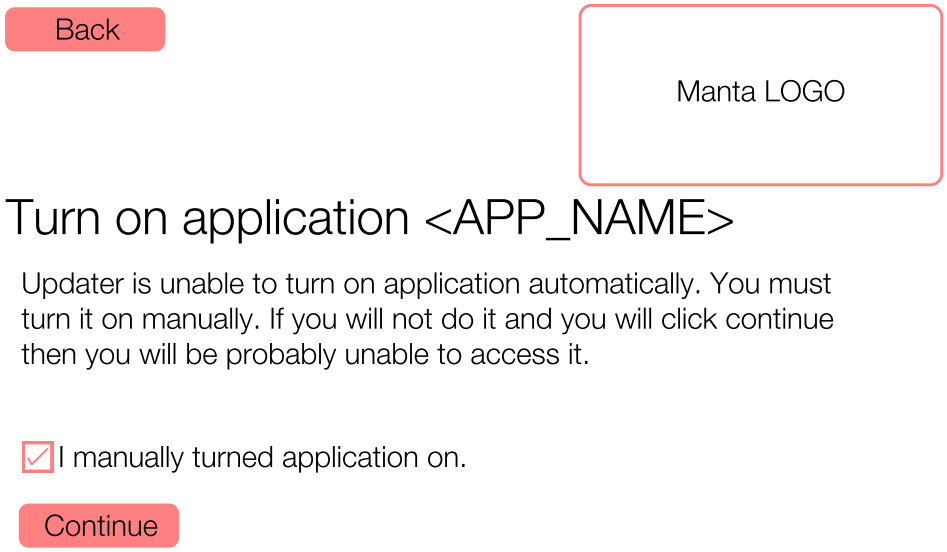
Obrázek D.7: Wireframe – Potvrzovací obrazovka: Potvrzení aktualizace



Obrázek D.8: Wireframe – Potvrzovací obrazovka: Vypnutí aplikace



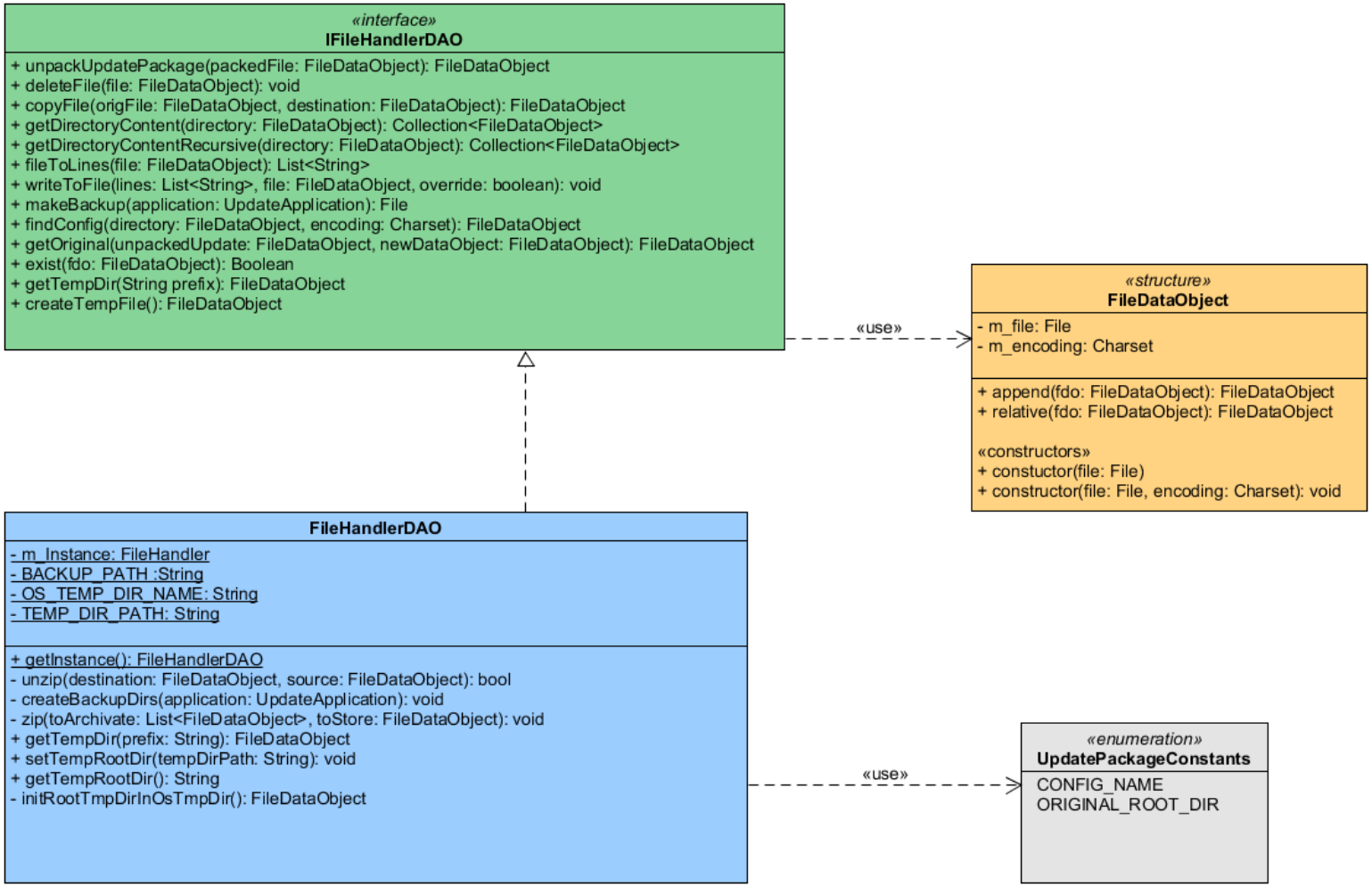
Obrázek D.9: Wireframe – Zobrazení průběhu a výsledku



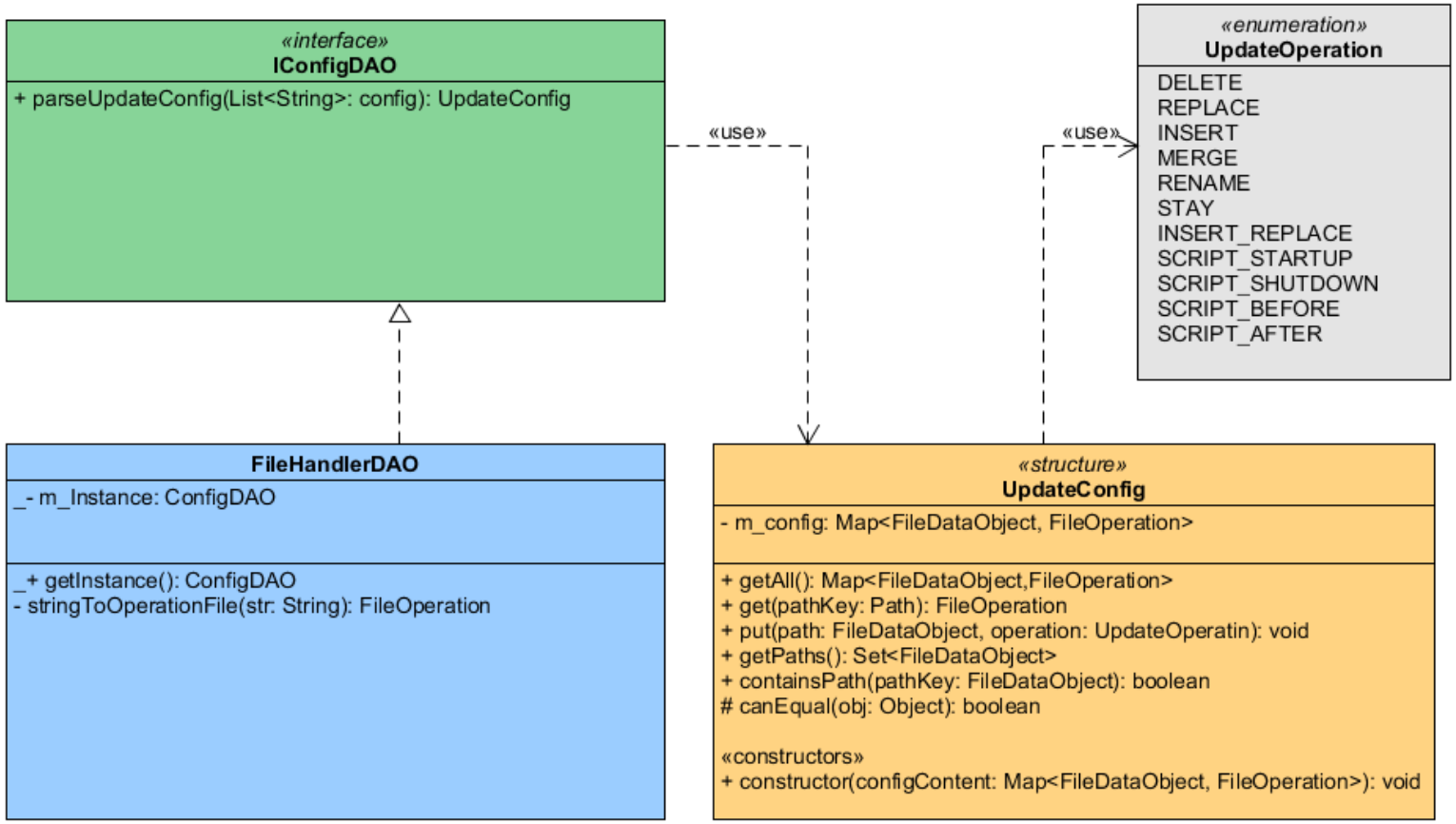
Obrázek D.10: Wireframe – Potvrzovací obrazovka: Zapnutí aplikace

Příloha E

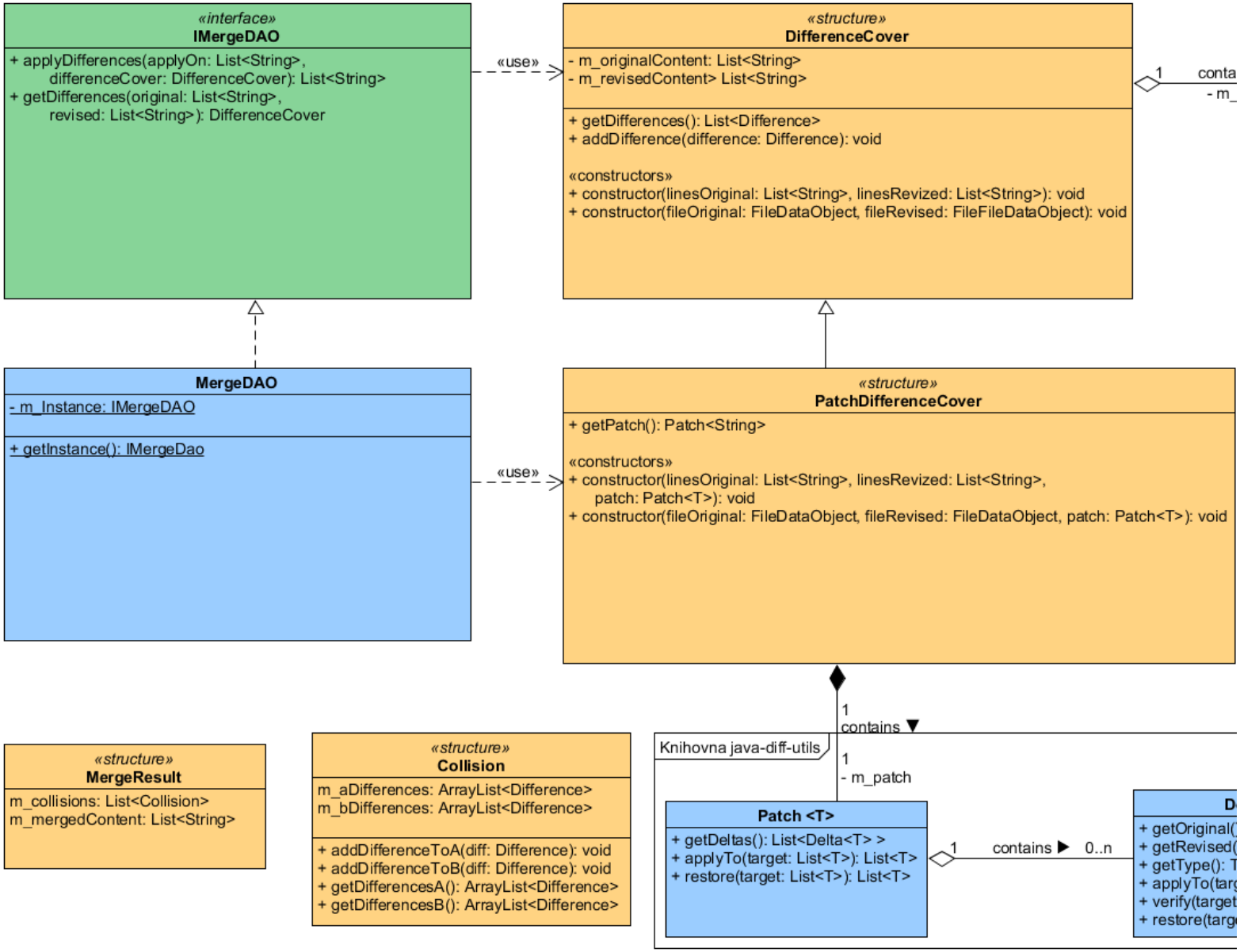
Diagramy tříd



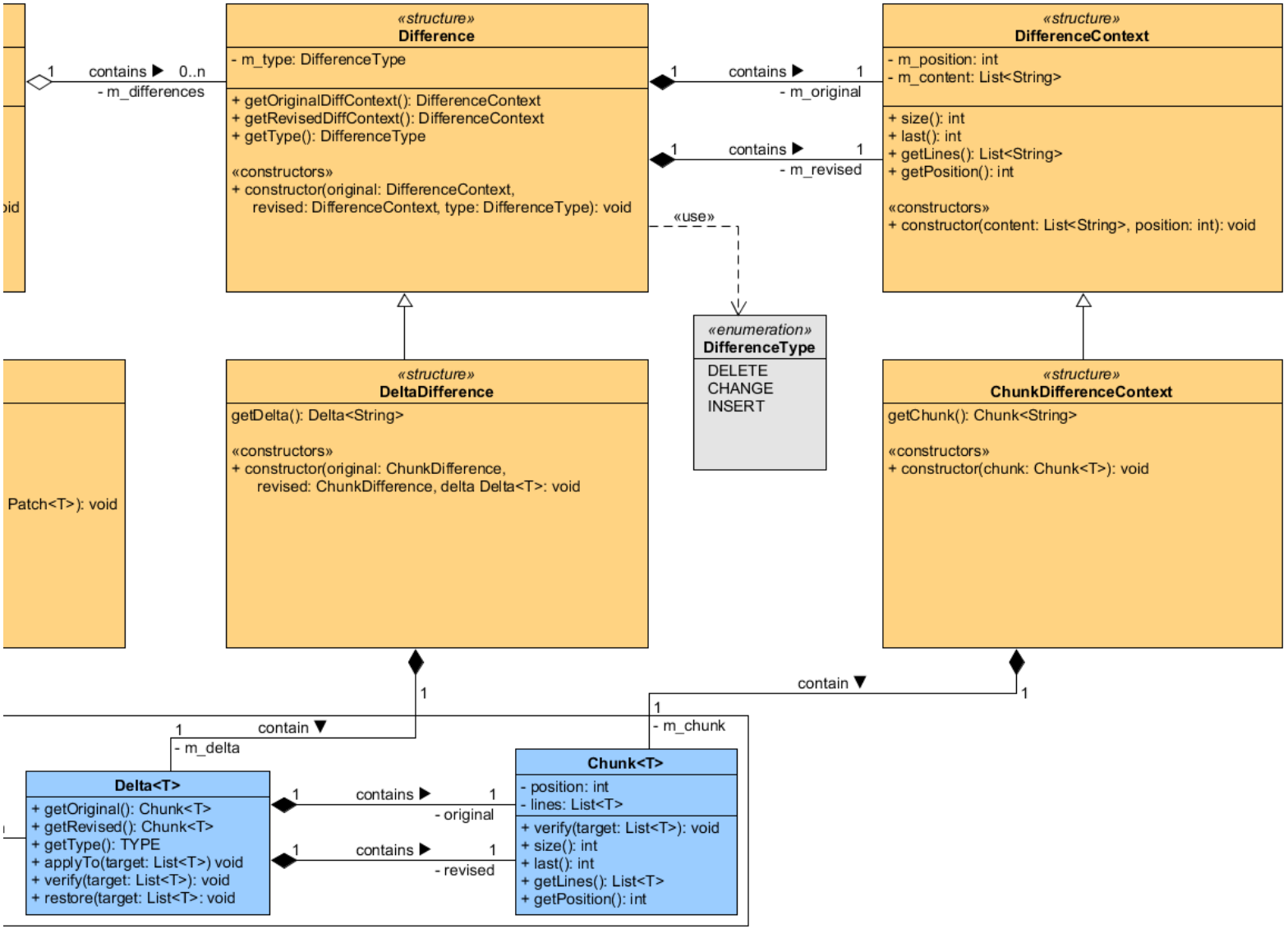
Obrázek E.1: Diagram tříd – FileHandlerDAO



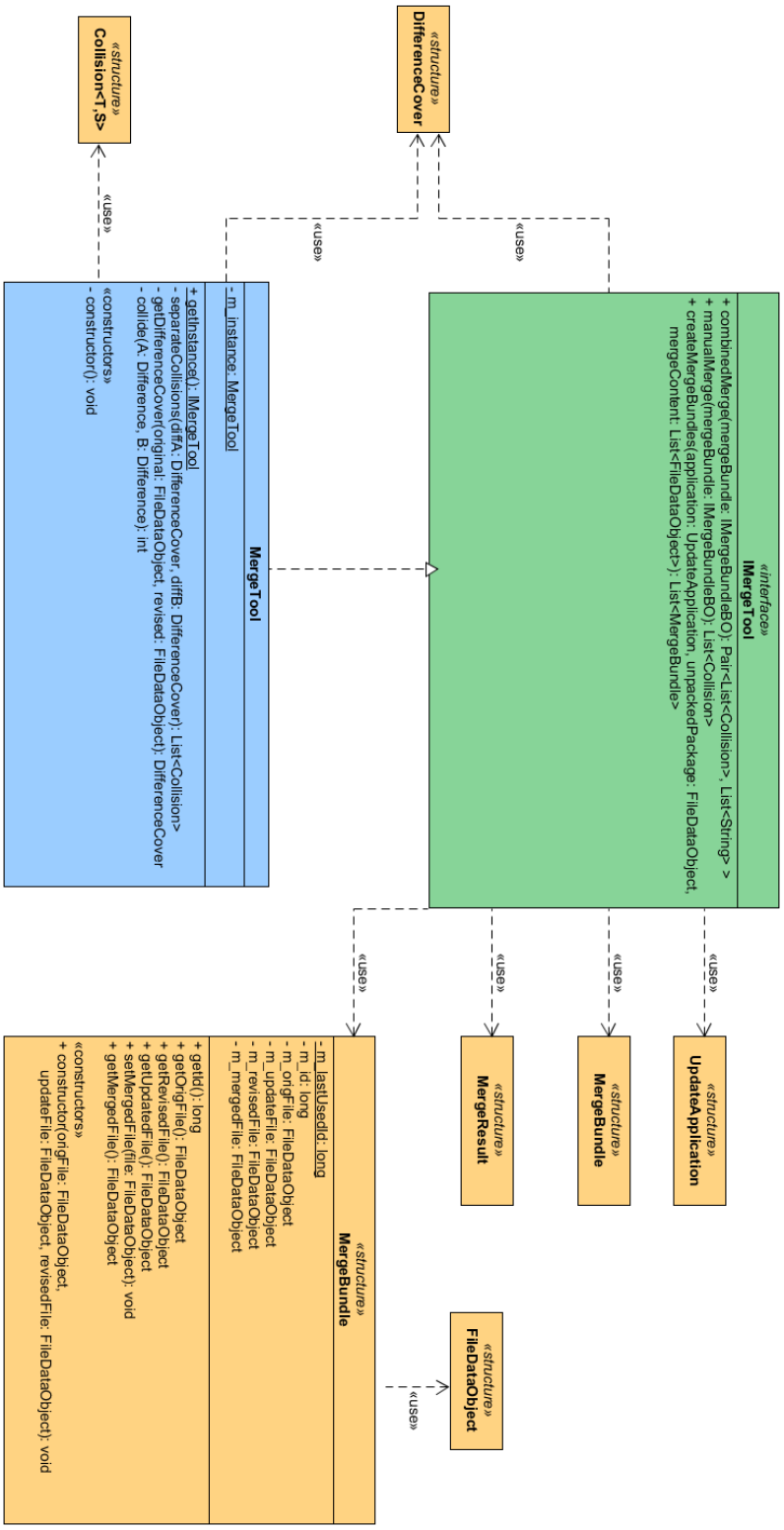
Obrázek E.2: Diagram tříd – ConfigDAO



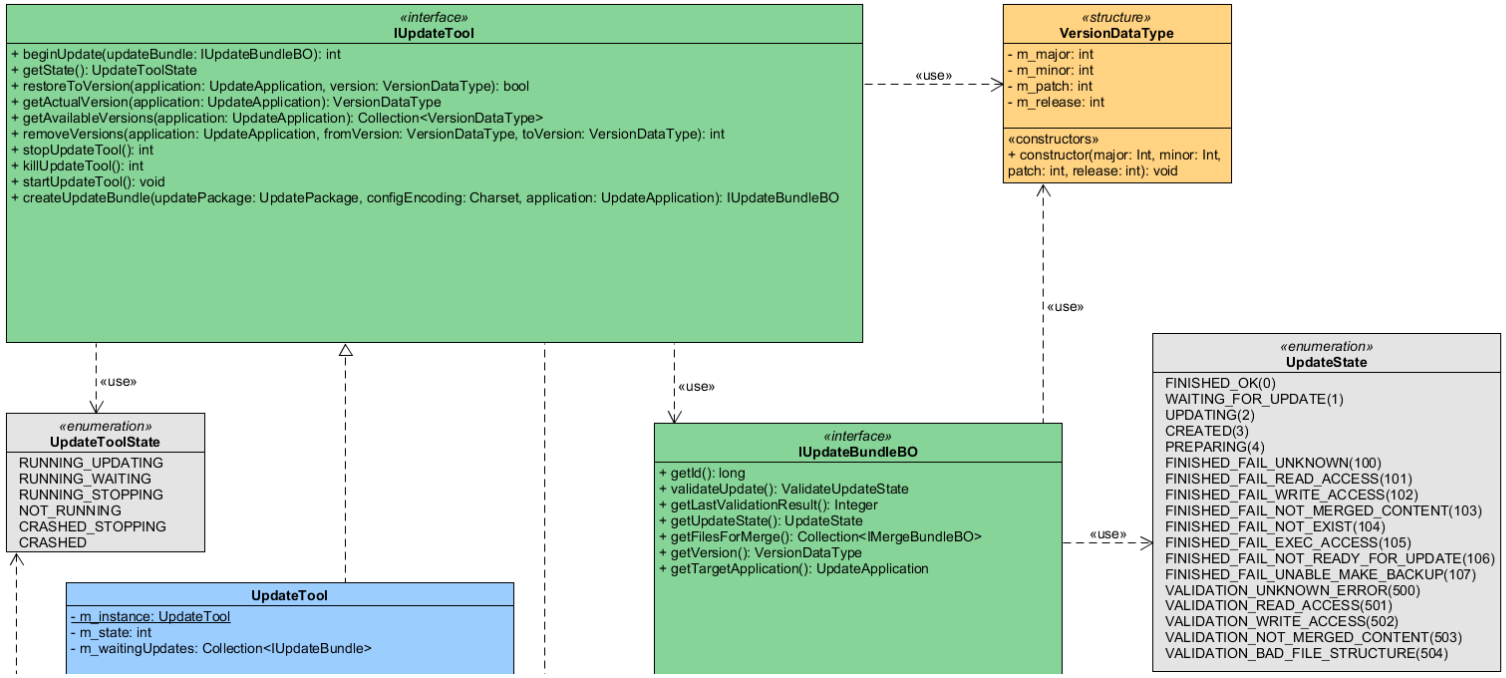
Obrázek E.3: Diagram tříd – MergeDAO 1. část



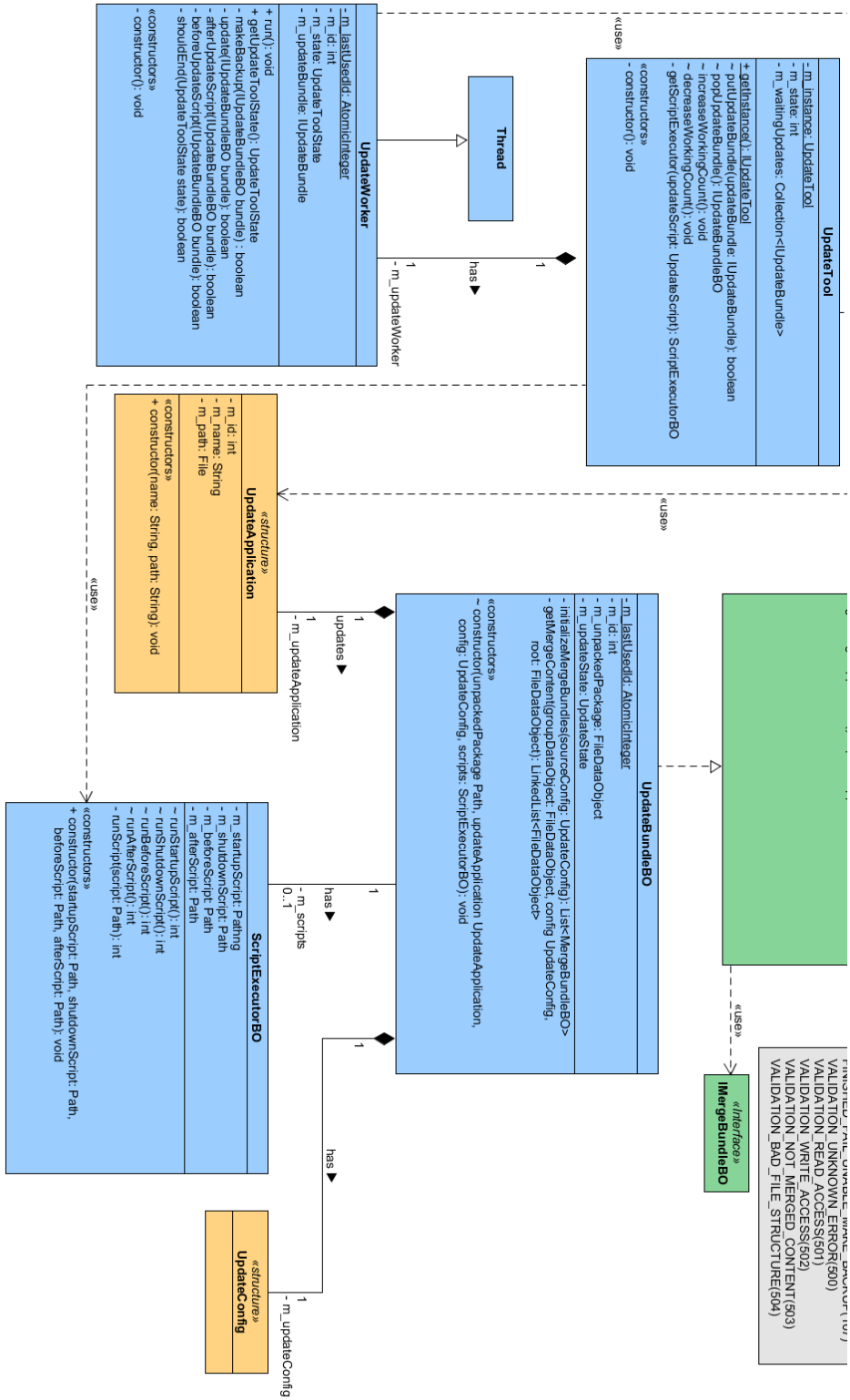
Obrázek E.4: Diagram tříd – MergeDAO 2. část



Obrázek E.5: Diagram tříd – MergeTool



Obrázek E.6: Diagram tříd – UpdateTool 1. část



Obrázek E.7: Diagram tříd – UpdateTool 2. část