České vysoké učení technické v Praze
Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

2016 Petr Košatka

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Petr Košatka**

Studijní program: Softwarové technologie a management
Obor: Web a multimedia

Název tématu: **Aplikace pro generování testovacích situací pro techniku State Transition Test**

Pokyny pro vypracování:

Navrhněte a implementujte aplikaci umožňující generování testovacích situací pro techniku State Transition Test pro zadanou hloubku testovacího pokrytí. Aplikace bude umožňovat vytvoření a aktualizaci schématu automatu pomocí tabulky a interaktivního grafického editoru a provádět kontrolu konzistence schématu. Dále bude možné tabulku automatu a vygenerované testovací situace importovat a exportovat ve formátech csv a xml. Změny schématu automatu bude aplikace propagovat do předchozích vygenerovaných testovacích situací na úrovni informace uživateli. Aplikace bude pracovat nad automaty do velikosti 50 stavů. Implementovanou aplikaci otestujte sadou funkčních testů vycházejících z uživatelských scénářů aplikace.

Seznam odborné literatury:

Kolář, J.: Teoretická informatika, Česká informatická společnost, 2004
Van Veenendaal E.: The Testing Practitioner, UTN Publishers, 2002

Vedoucí: Ing. Miroslav Bureš, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

prof. Ing. Jiří Žára, CSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 20. 2. 2014

Čestné prohlášení

Prohlašuji, že jsem zadanou bakalářskou práci „Aplikace pro generování testovacích situací pro techniku State transition test“ zpracoval sám s přispěním vedoucího práce a používal jsem pouze literaturu uvedenou na konci práce. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 12.1.2016 Podpis

Poděkování

Děkuji vedoucímu práce, Ing. Miroslavu Burešovi, Ph.D., za věcné připomínky a velkou trpělivost. Děkuji také svým rodičům za finanční podporu během studia.

Anotace

Cílem této bakalářské práce je provést návrh a vytvořit a otestovat aplikaci pro generování testovacích situací pro techniku State transition test popsanou v metodice TMap pro testování softwaru. Tato technika se používá pro testování programu, specifikace, nebo jejich částí, které lze popsat stavovým automatem. Výsledná aplikace bude uživateli umožňovat vytváření stavového automatu pomocí tabulky a interaktivního grafického editoru, kontrolu konzistence vytvořeného stavového automatu a jeho import a export do formátů CSV a XML. Aplikace bude umět generovat testovací situace a následně je exportovat do formátů CSV a XML. V práci se zabývám procesem vývoje aplikace od návrhu přes implementaci po testování. Popisuji zde požadavky na aplikaci, architekturu aplikace, použité algoritmy, formáty vstupních a výstupních dat a grafické uživatelské rozhraní.

Annotation

The goal of this bachelor's thesis is to design and create an application for generating test situations for the State transition test technique described in TMap methodics for software testing. This technique is used for testing of a program or specification or their part, which can be described by a state automaton. The resulting application will allow the user to create a state automaton using a table and an interactive graphic editor, the created state automaton's consistency checking and its import and export to CSV and XML formats. The application will be able to generate test situations and subsequently export them to CSV and XML formats. My thesis describes the process of application development from design to implementation to testing. Here I describe the requirements, architecture, used algorithms, formats of input and output data and graphical user interface.

Obsah

1 Úvod

Jako bakalářskou práci jsem se rozhodl naprogramovat softwarový nástroj pro generování testovacích situací pro techniku testování softwaru State transition test popsanou v metodice TMap. Výsledná aplikace by měla uživatelům umožňovat vytváření a editaci stavového automatu, jednak pomocí interaktivního grafického editoru a také pomocí tabulky. Aplikace bude umět zkontrolovat konzistenci vytvořeného stavového automatu a následně jeho procházením vygenerovat testovací situace. Automat bude možné importovat a exportovat z a do formátů XML a CSV. Vygenerované testovací situace bude možno vyexportovat do formátů XML a CSV.

1.1 Princip techniky State transition test

Zde se nejprve seznámíme se základními pojmy a poté si vysvětlíme princip techniky State transition test.

Konečný stavový automat
Konečný stavový automat (dále jen stavový automat nebo automat) je orientovaný graf, kde vrcholy představují stavy a hrany představují přechody mezi nimi. Musí být definovaný počáteční stav a jeden nebo více koncových stavů, graf musí být konzistentní a ze všech stavů kromě koncových musí vést alespoň jeden přechod.

State transition test
Technika State transition test patří mezi tzv. black-box testovací techniky. To znamená, že tato technika testuje systém bez znalosti jeho vnitřní struktury. Používá se, pokud lze systém, jeho část, nebo jeho specifikaci popsat konečným stavovým automatem. Cílem je projít všechny možné sekvence přechodů podle zvoleného pokrytí.

Testovací situace
Testovací situací (nebo také testovacím případem, anglicky test case) se rozumí sekvence přechodů a stavů ve tvaru stav → přechod → stav. Délka sekvence je určená testovacím pokrytím.

Testovací pokrytí
U techniky State transition test se používá testovací pokrytí N-switch coverage. Testy pokrývají všechny platné sekvence N + 1 navazujících transakcí. Pokrytí všech jednotlivých přechodů je 0-switch pokrytí, pokrytí všech dvojic navazujících přechodů je 1-switch pokrytí atd.

Technika State transition test funguje na principu testování navazujících sekvencí stavů a přechodů. Cílem je otestovat všechny možné navazující sekvence přechodů.

1.2 Přehled existujících řešení

Danou problematikou se již zabývají následující programy:

1.2.1 COVER holandské pobočky Sogeti[1]

Cover obsahuje velmi omezenou funkcionalitu pro generování testovacích situací. Graf je možno zadat pouze textově, jako množina hran a uzlů, chybí grafický editor. Aplikace je bezplatná pouze pro zaměstnance společnosti Sogeti, podmínky licence pro ostatní uživatele se mi nepodařilo dohledat.

1.2.2 GOTCHA-TCBeans od IBM Research Lab v Haifě[2]

GOTCHA-TCBeans se zaměřuje zejména na generování testovacích situací ze stavového automatu. Aplikaci se mi nepodařilo získat, IBM jí nejspíše neposkytuje veřejnosti a vyvíjí ji pouze pro interní použití.

1.2.3 Conformiq Designer[3]

Conformiq Designer je nejpokročilejší ze srovnávaných nástrojů. Nabízí široké možnosti práce se stavovými automaty a aktivity diagramy. Nástroj pro vytváření testů je ale navržen pro práci s návrhovou dokumentací v jazyce UML. Ceny licencí jsou také příliš vysoké pro malé firmy a jednotlivé uživatele.

1.3 Důvody pro vytvoření vlastního řešení

Důvodem implementace nového řešení je, že mnou vytvářené řešení, na rozdíl od výše uvedených nástrojů bude dostupné zcela zdarma. Moje řešení se také bude úzce zaměřovat pouze na generování testovacích situací pro State transition test, bude tedy vhodnější pro uživatele, kteří vyžadují pouze tuto funkcionalitu. Také bude obsahovat grafický editor pro pohodlné vytváření stavového automatu. Má aplikace bude postavena na technologii Java, tudíž by neměl být problém s její přenositelností mezi různými operačními systémy.

1 http://tstr.nl/cover/vb/pctvb1.html

2 https://www.research.ibm.com/haifa/projects/verification/gtcb/index.html

3 http://www.conformiq.com/products/conformiq-designer/

2 Návrh Aplikace

V této kapitole se budu zabývat definováním funkčních a nefunkčních požadavků na aplikaci, definicí případů užití aplikace uživatelem, dále návrhem algoritmů užitých v aplikaci a návrhem grafického uživatelského rozhraní.

2.1 Požadavky na aplikaci

2.1.1 Funkcionální požadavky

1. Informace evidované o stavech a přechodech

   1.1 Přechody budou mít atributy: id, název, popis

   1.2 Stavy budou mít atributy: id, název, popis. Stavy budou moci být počáteční a nebo konečné.

2. Konfigurace stavového automatu pomocí tabulky

   2.1 Systém bude umožňovat zadat seznam stavů automatu

   2.2 Systém bude umožňovat zadat seznam přechodů automatu

   2.3 Systém bude umožňovat zvolit počáteční stav

   2.4 Systém bude umožňovat zvolit koncové stavy

3. Konfigurace stavového automatu pomocí interaktivního grafického editoru

   3.1 Systém bude umožňovat přidání nového stavu

   3.2 Systém bude umožňovat přidání nového přechodu mezi stavy

   3.3 Systém bude umožňovat mazání stavů a přechodů

   3.4 Systém bude umožňovat přesouvání stavů ve vizualizaci

4. Import a export stavového automatu a testovacích případů

   4.1 Systém umožní import stavového automatu ve formátech XML a CSV

   4.2 Systém umožní export stavového automatu ve formátech XML a CSV

   4.3 Systém umožní export testovacích případů ve formátech XML a CSV

5. Uživatelské rozhraní

   5.1 Systém bude umět vykreslit stavový automat podle tabulky stavů a přechodů

   5.2 Při změně stavového automatu systém změny propaguje do již vygenerovaných testovacích situací, pokud nějaké existují a upozorní uživatele

   5.3 Systém umožní uživateli úpravu atributů stavů a přechodů

5.4 Aplikace bude obsahovat uživatelskou příručku

6. Kontrola konzistence stavového automatu

6.1 Systém umožní kontrolu existence počátečního stavu

6.2 Systém umožní kontrolu existence alespoň jednoho koncového stavu

6.3 Systém umožní kontrolu existence nedosažitelných stavů , jejich výpis a případně i odstranění

6.4 Systém umožní zkontrolovat existenci nepřipojených, nebo pouze na jedné straně připojených přechodů

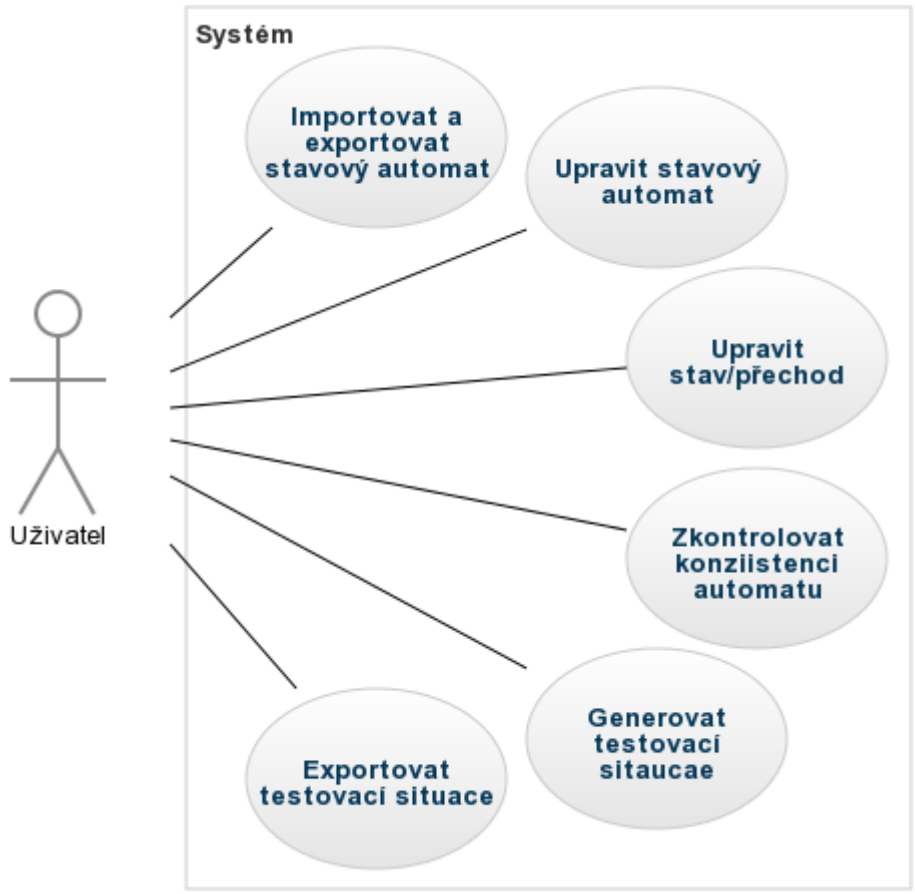
6.5 Systém umožní zkontrolovat, zda všechny nekoncové stavy mají vystupující hrany

2.1.2 Nefunkcionální požadavky

1. Systém bude modulární, část pro generování testovacích situací bude nahraditelná
2. Sytém bude generovat testovací situace bez uživatelem vnímatelného zpoždění na běžné PC sestavě pro stavový automat o velikosti 100 stavů a 100 přechodů

2.2 Případy užití

V této kapitole se budu věnovat výčtu případů užití a jejich popisu. Diagram případů užití (Obrázek 1) znázorňuje možné interakce uživatele se systémem.



Obrázek 1: Diagram případů užití

1. Importovat a exportovat stavový automat
   Tento UC umožní uživateli importovat a exportovat stavový automat z a do formátů CSV a XML.

2. Upravit stavový automat
   Tento UC umožní uživateli přidávat a odebírat stavy a přechody a měnit výchozí a cílový stav přechodu.

3. Upravit stav/přechod
   Tento UC uživateli umožní upravovat jednotlivé atributy stavů a přechodů. Pro přechody jsou to název a popis, pro stavy název, popis, počáteční stav a konečný stav.

4. Zkontrolovat konzistenci automatu
   Tento UC uživateli umožní zkontrolovat, zda je daný automat konzistentní.

5. Generovat testovací situace
   Tento UC umožní uživateli vygenerovat testovací situace pro zadané pokrytí.

6. Exportovat testovací situace
   Tento UC umožní uživateli exportovat vygenerované testovací situace do souborů ve formátech CSV a XML.

7. Zobrazit uživatelskou nápovědu
   Tento UC umožní uživateli zobrazit uživatelskou nápovědu aplikace.

2.3 Použité algoritmy

V této kapitole se budu věnovat nejdůležitějším algoritmům použitým v aplikaci. Popíšu jejich kroky pseudokódem podle vzoru z knihy Teoretická Informatika od Josefa Koláře [1].

2.3.1 Algoritmus pro vyhledání všech nedosažitelných stavů

Algorithm 1 Hledání nedosažitelných stavů

```
1: S1 = {s0}
2: S2 = {s0}
3:
4: do
5:     S1 := S1 ∪ S2
6:     for každý stav s ∈ S1 do
7:         S2 := S2 ∪ množina všech stavů, do kterých vede přechod z s
8:     end for
9: while S1 ≠ S2
10:
11: výsledek = množina všech stavů - S1
```

Na začátku mám dvě množiny, při inicializaci do obou vložím počáteční stav s_0. Opakovaně procházím všechny stavy v množině S_1 a pro každý její stav s přidávám do množiny S_2 všechny stavy, do kterých se dostanu z s. Toto dělám, dokud množiny S_1 a S_2 nejsou shodné. Po skončení cyklu obě množiny obsahují všechny stavy dosažitelné z počátku s_0. Výsledek získám jako rozdíl množiny všech stavů automatu a množiny S_1.

2.3.2 Algoritmus pro generování testovacích situací

Algorithm 2 generování testovacích situací

```
1: TC := množina všech přechodů
2: NTC := ∅
3:
4: for i = 1 to N do
5:     for testcase tc ∈ TC do
6:         TC := TC - {tc}
7:         NTC := NTC ∪ množina všech navazujících testcasů k tc
8:     end for
9:     TC := TC ∪ NTC
10:    NTC := ∅
11: end for
```

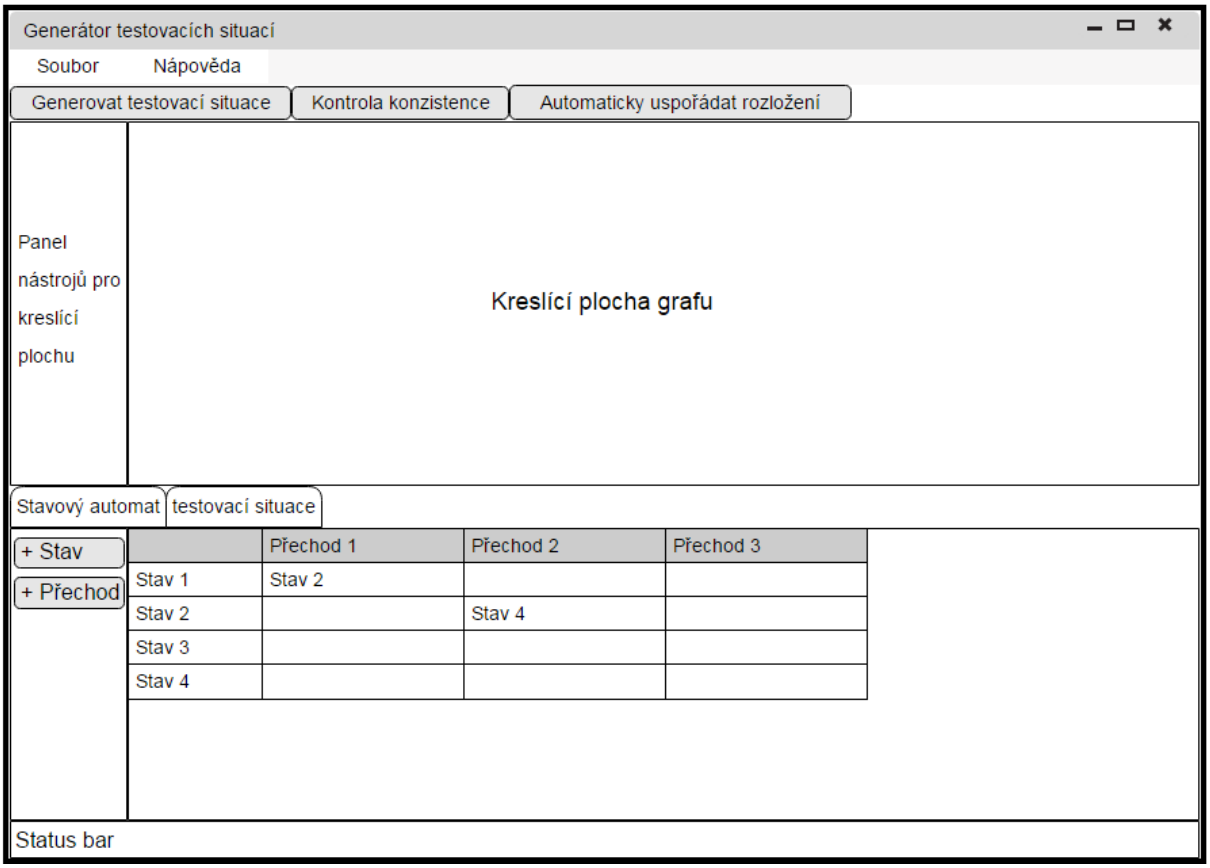
Pro generování testovacích situací v aplikaci používám vlastnoručně navržený algoritmus. Testovací situace generuji jako posloupnosti přechodů automatu, neboť z přechodů umím získat odpovídající zdrojové a cílové stavy.

Na začátku naplním množinu *TC* všemi testovacími případy o délce 1, což odpovídá množině všech přechodů automatu. Poté opakovaně procházím množinu *TC* a pro každou sekvenci *tc* v ní obsaženou přidám do množiny *NTC* všechny navazující sekvence, tedy pro každý výstupní přechod z posledního stavu sekvence *tc* přidám sekvenci *tc* prodlouženou o ten daný výstupní přechod. Z množiny *TC* odeberu sekvenci *tc*. Poté množinu *TC* naplním sekvencemi z *NTC* a *NTC* vyprázdním. Toto opakuji *N*-krát, kde *N* je zadaná hloubka testovacího pokrytí.

Po skončení algoritmu množina *TC* obsahuje testovací situace pro zadanou hloubku pokrytí.

2.4 Návrh grafického uživatelského rozhraní

Při návrhu grafického uživatelského rozhraní jsem vycházel z klasického rozložení běžných desktopových aplikací. Tedy lišta s nabídkami (Soubor, Nápověda) na horním okraji obrazovky, pod ním nástrojová lišta pro často používané funkce, dále stavová lišta u dolního okraje okna.

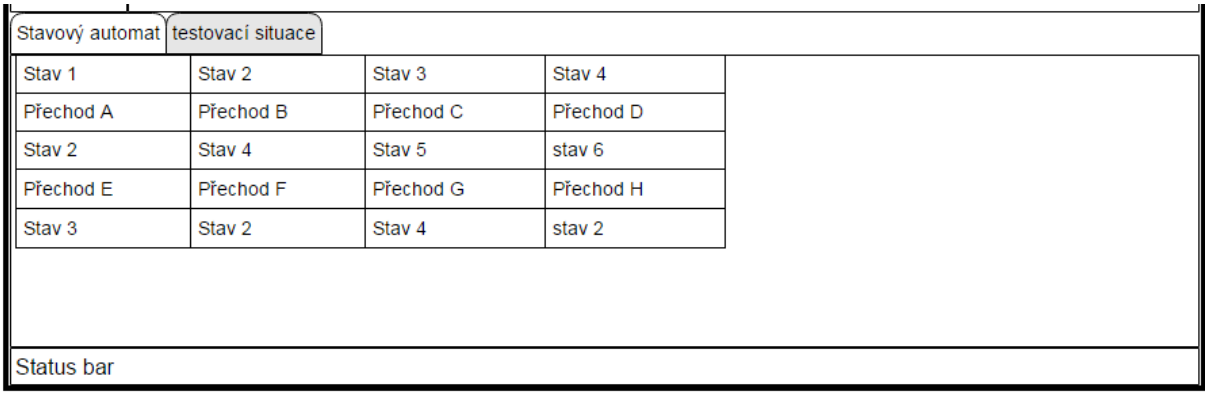


Obrázek 2: Návrh grafického uživ. rozhraní, hlavní okno aplikace

Největší část okna zabírá plátno pro vykreslení a úpravu stavového automatu, po její levé straně se bude nacházet panel nástrojů pro plátno. V dolní části okna se bude nacházet tabulka přechodů a stavů s tlačítky pro přidání přechodu a stavu. Tabulka vygenerovaných testovacích situací se bude zobrazovat jako záložka vedle tabulky stavového automatu. Odebírání stavů a přechodů pomocí tabulky bude řešeno přes kontextové menu.

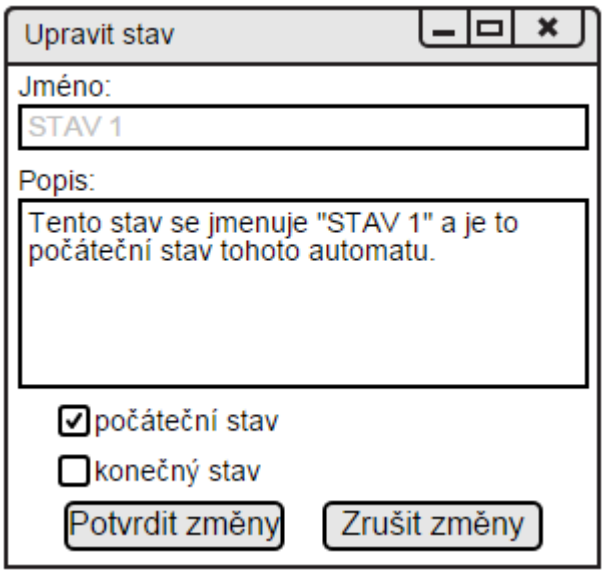
Kontextové menu bude možno vyvolat kliknutím pravým tlačítkem myši na stav či přechod na plátně i v tabulce. Toto menu bude obsahovat položky "Odebrat stav" (přechod) a "upravit stav" (přechod).

Tabulka stavového automatu (mimo hlavičky řádků a sloupců) bude editovatelná pomocí rozbalovacích seznamů s nabídkou všech aktuálně existujících stavů v rámci automatu. Tabulka testovacích situací (Obrázek 3) bude zobrazovat testovací situace jako jednotlivé sloupce tabulky a bude sloužit pouze ke čtení.



Obrázek 3: Návrh uživ. rozhraní, tabulka vygenerovaných testovacích situací

Pro úpravu stavu či přechodu bude sloužit dialog "Upravit stav" (Obrázek 4). Tento dialog bude využíván pro úpravu i pro přidávání nových stavů a přechodů, název dialogu, popisy tlačítek a upravitelné atributy se budou měnit podle konkrétní situace. V případě úpravy/přidávání přechodu v dialogu budou chybět zaškrtávací políčka "počáteční stav" a "konečný stav".



Obrázek 4: Dialog pro editaci stavů

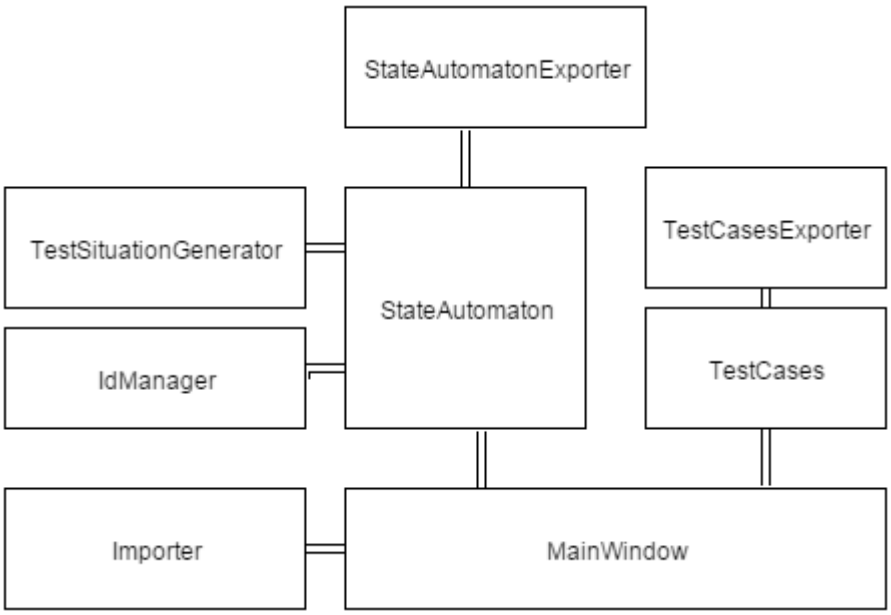
3 Implementace

V této kapitole budu hovořit o použitých technologiích v rámci aplikace, popíšu způsob, jakým je aplikace postavena a nakonec se budu věnovat formátům vstupních a výstupních souborů.

3.1 Architektura aplikace

Pro tvorbu aplikace jsem si vybral programovací jazyk Java ve verzi 1.8 od společnosti Oracle. Důvodem pro mou volbu je zejména fakt, že s touto technologií mám několikaleté zkušenosti v rámci studia na vysoké škole. Výhodou je také přenositelnost řešení mezi různými platformami. Z volby jazyka logicky vyplynula volba technologie pro tvorbu grafického uživatelského rozhraní, a to knihovny Swing.

Pro vykreslování a vnitřní reprezentaci stavového automatu jsem použil knihovnu 3. strany JgraphX (http://www.jgraph.com). Pro import a export do CSV jsem použil knihovnu OpenCSV verze 3.5. Dále jsem využil součásti knihovny Apache Commons a to commons.lang 3.3.2 a commons.io 2.4.

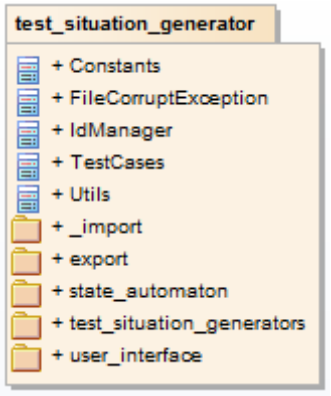


Obrázek 5: Základní blokové schéma

Nejdůležitější částí aplikace je třída MainWindow, která obsahuje metodu main() a zajišťuje veškerou funkcionalitu uživatelského rozhraní. Uchovává si odkazy na třídy StateAutomaton, reprezentující stavový automat a TestCases, reprezentující vygenerované testovací situace. Obě tyto třídy implementují rozhraní Exportable a mohou být exportovány pomocí potomků abstraktních tříd StateAutomatonExporter a TestSituationsExporter. Pro import stavového automatu třída MainWindow využívá konkrétních potomků třídy Importer. Třída StateAutomaton dále používá dvě instance třídy IdManager pro zajištění unikátního id pro každý stav a pro každý přechod v automatu.

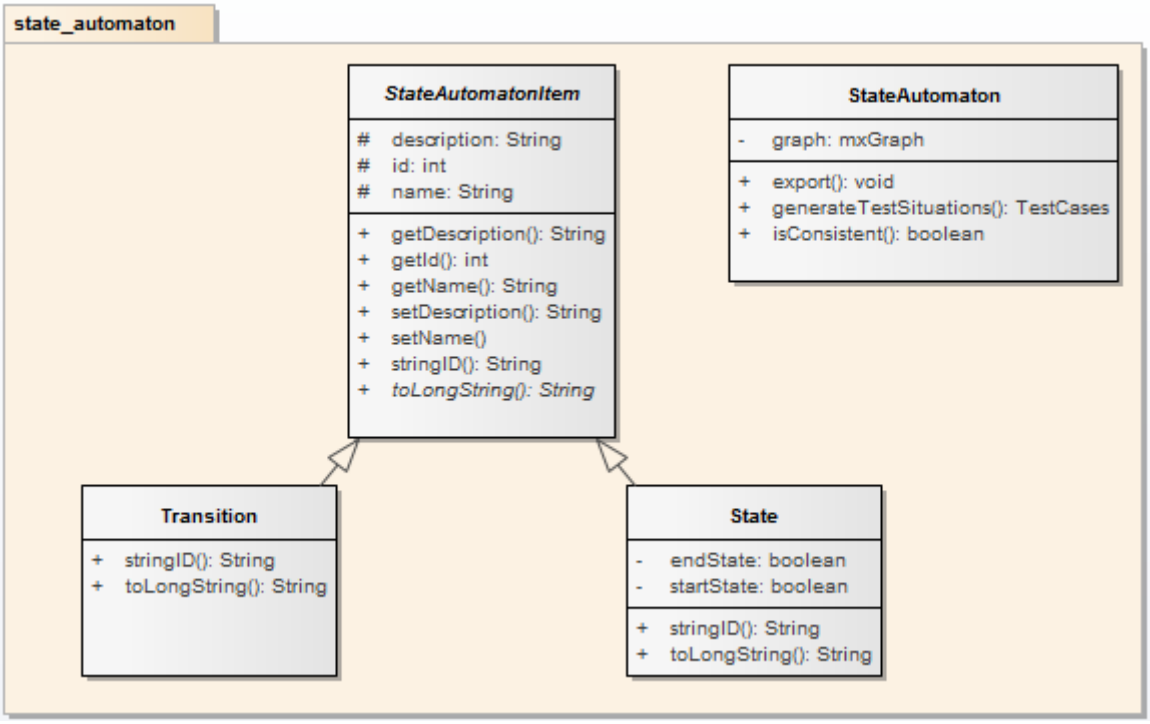
3.1.1 Přehled součástí aplikace

Všechny svoje třídy jsem umístil do balíčku s názvem „test_situation_generator“() podle názvu aplikace. Tento balíček obsahuje všechny další balíčky, které jsem vytvořil a také třídy, které se nehodily jinam.



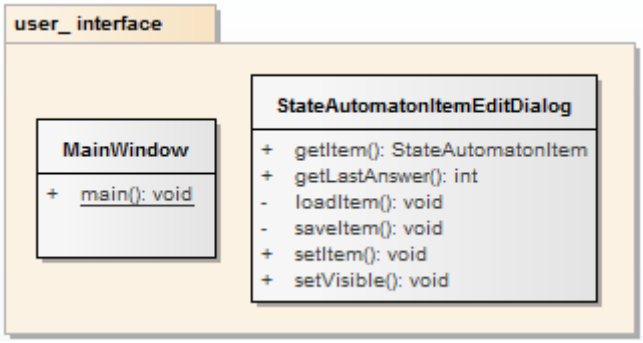
Obrázek 6: balíček test_situation_generator

Třída Constants obsahuje konstanty pro běh programu. Třída FileCorruptException představuje výjimku, která vznikne při pokusu o import souboru, který je poškozen, nebo jeho typ je nesprávný. Třída IdManager v rámci mé aplikace slouží pro spravování id pro stavy a přechody. Třída TestCases představuje vygenerované testovací případy nebo situace. Třída Utils obsahuje pomocné metody a objekty.



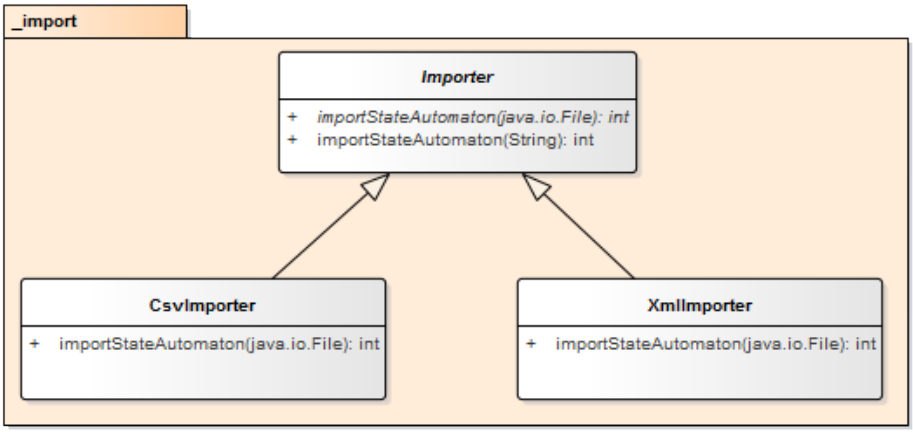
Obrázek 7: balíček state_automaton

Balíček state_automaton obsahuje nejdůležitější část aplikace - třídy pro práci se stavový automatem. Transition představující přechod, State představující stav, jejich společného abstraktního předka StateAutomatonItem a třídu StateAutomaton. Třída StateAutomaton představuje stavový automat a obsahuje graf, který uchovává všechny vazby mezi přechody a stavy. Tato třída poskytuje přístup ke grafu a zároveň implementuje rozhraní TableModel a slouží jako datový model pro tabulku stavového automatu. Dále implementuje rozhraní Exportable.



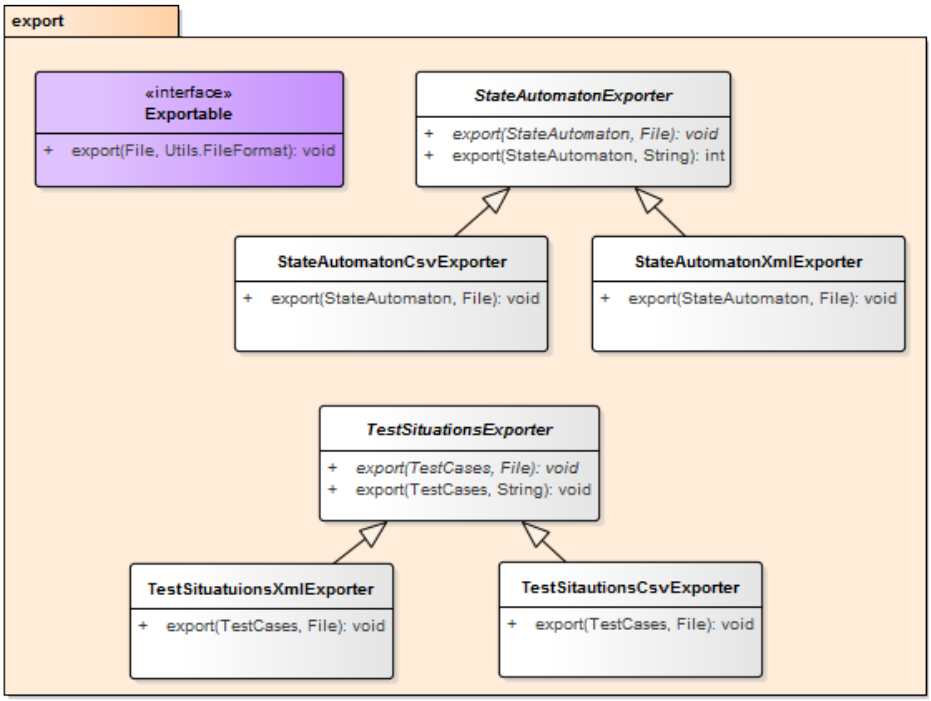
Obrázek 8: Balíček user_interface

Balíček user_interface obsahuje třídy MainWindow a StateAutomatonItemEditDialog. Třída MainWindow je hlavní třída aplikace, obsahuje metodu main(). Stará se o naprostou většinu grafického uživatelského rozhraní a obsluhu událostí. Třída StateAutomatonItemEditDialog představuje okna pro editaci stavů i přechodů, vzhled okna se přizpůsobuje podle třídy editovaného objektu (viz).



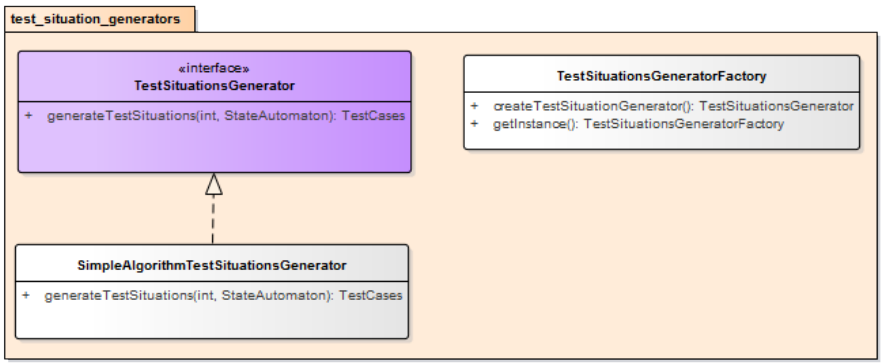
Obrázek 9: balíček import

Balíček import obsahuje abstraktní třídu Importer. a její konkrétní potomky pro import z formátů CSV a XML. Všechny třídy importující stavový automat by měly dědit ze třídy Importer.



Obrázek 10: balíček export

Balíček export obsahuje abstraktní třídy StateAutomatonExporter a TestSituationsExporter, které definují metody pro export stavového automatu a testovacích situací. Opět jsou tu potomci obou tříd pro formáty CSV a XML. Je zde také rozhraní Exportable, jež je implementováno třídami StateAutomaton a TestCases.

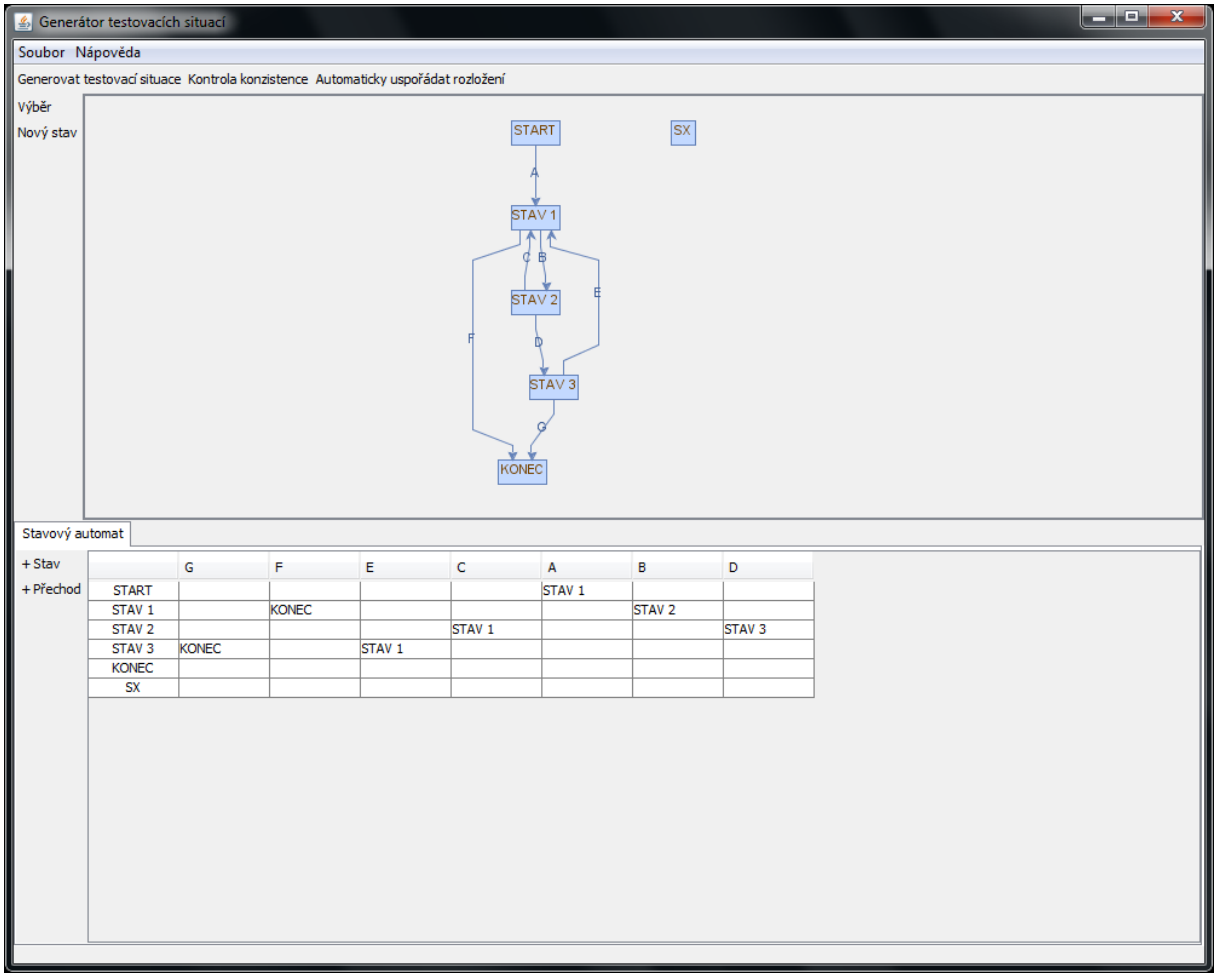


Obrázek 11: balíček test_situation_generators

Konečně balíček test_situation_generators obsahuje rozhraní TestSituationGenerator. Implementací tohoto rozhraní je třída SimpleAlgorithmTestSituatinsGenerator, která generuje testovací situace pomocí algoritmu popsaného v kapitole 2.3.1. Dále je tu třída TestSituationsGeneratorFactory, která se stará o vytváření instancí tříd implementujících rozhraní TestSituationGenerator.

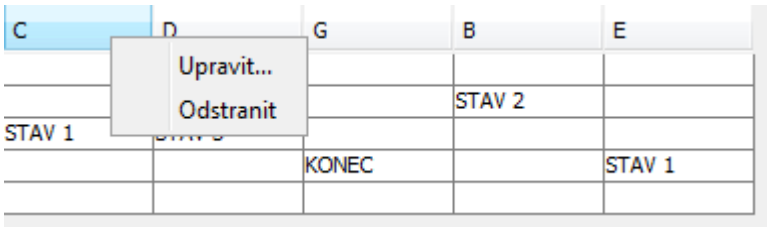
3.2 Ukázky aplikace

V této kapitole si představíme již hotovou aplikaci. Na snímku obrazovky (Obrázek 12) vidíme vykreslený stavový automat o 6 stavech, z toho je jeden stav nedosažitelný. V dolní části se nachází tabulková reprezentace daného automatu.

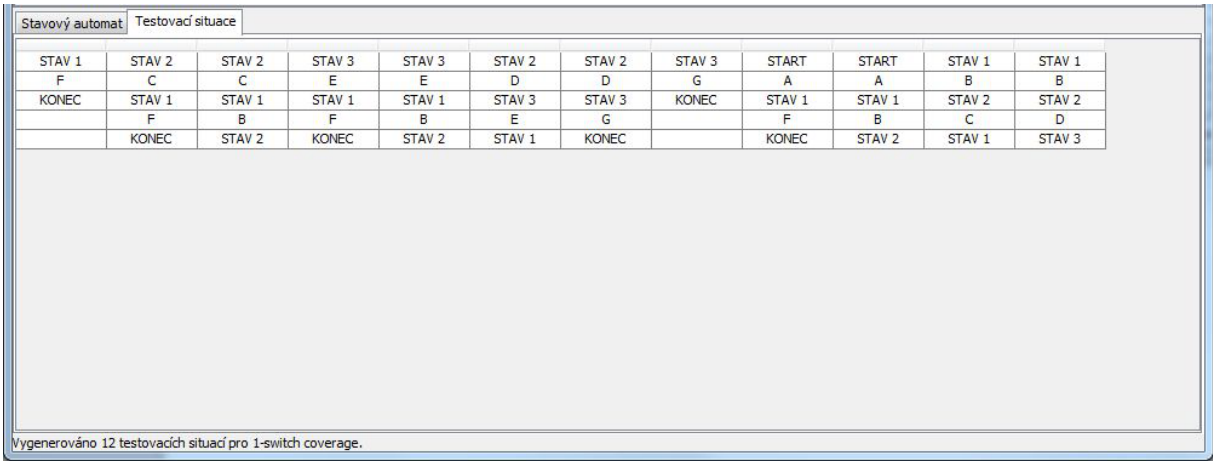


Obrázek 12: Ukázka aplikace, hlavní okno

Na druhém snímku obrazovky je vidět část tabulky stavového automatu s vyvolaným kontextovým menu.



Obrázek 13: Výřez tabulky stavového automatu



Obrázek 14: Tabulka vygenerovaných testovacích situací

3.3 Formáty souborů pro import/export

Tato kapitola se zabývá popisem souborů pro import a export testovacích situací a stavového automatu. Stavový automat je možné importovat a exportovat do formátů CSV a XML, testovací situace je možné pouze exportovat, taktéž do formátů CSV a XML. CSV souborům se věnuji níže v této kapitole, formát XML specifikuji pomocí jazyka XML Schema v příloze.

3.3.1 CSV soubor pro import/export stavového automatu

CSV formát nemá jednotnou specifikaci, vycházel jsem tedy z doporučení v RFC 4180[4]. V případě stavového automatu jsem byl však nucen formát značně upravit, z důvodu zachování atributů stavů a přechodů v exportovaných souborech.

Stavový automat tedy exportuji do CSV formátu následovně: 1. řádek obsahuje stavy, 2. řádek obsahuje přechody. Řádky jsou odděleny řetězcem CRLF (2 znaky oddělující řádky na platformě Windows), jednotlivé stavy a přechody na řádcích jsou od sebe odděleny znakem „čárka“. Každý stav a přechod je popsán svými atributy, které jsou odděleny znakem „pomlčka“. Všechny atributy jsou povinné a mají pevně dané pořadí.

Stav: *id-jméno-popis-jePočátečníStav-jeKoncovýStav*

Přechod: *id-jméno-popis-zdroj-cíl*

Atribut id je celé číslo, jméno a popis jsou řetězce, jePočátečníStav a jeKoncovýStav jsou logické hodnoty (true nebo false), zdroj a cíl jsou atributy id zdrojového a cílového stavu.

3.3.2 CSV soubor pro export testovacích situací

Testovací situace exportuji do CSV formátu stejným způsobem, jako se vykreslují v tabulce v uživatelském rozhraní, tedy posloupnost „stav-přechod-stav“ v každém „sloupci“. Stavy a přechody jsou zde reprezentovány pouze svým názvem. V souladu s RFC 4180[4] používám jako oddělovač jednotlivých polí znak „čárka“ a jako oddělovač řádků používám řetězec CRLF.

4 Testování

Vybrané nejdůležitější metody jsem testoval za použití JUnit frameworku ve verzi 4.10 na operačním systému Microsoft Windows 7 Professional SP1 64-bit.

4.1 Testovací scénáře

4.1.1 Test algoritmu pro generování testovacích situací

Název testu	Generování testovacích situací
Umístění testovací metody	test_situation_generator.test_situations_generators
Jméno testovací metody	SimpleAlgorithmTestSituationGeneratorTest. testGenerateTestSitauations();
Vstupní data	Automat s 1 stavem, N=0; automat s 2 stavy, N=0; automat o velikosti do 10 stavů a 10 přechodů s N=1,2
Kroky	1. Vygenerovat testovací situace pro stavový automat ze vstupu a dané testovací pokrytí 2. Porovnat výsledek s předpřipravenými výsledky
Očekávaný výsledek	Vygenerované testovací situace se shodují s předpřipravenými výsledky
Prošel/neprošel?	Prošel pro všechna vstupní data

4.1.2 Test importu a exportu stavového automatu

Název testu	Import a export stavového automatu
Umístění testovací metody	test_situation_generator.state_automaton
Jméno testovací metody	StateAutomatonTest.testImportExport();
Vstupní data	soubor obsahující prázdný automat, soubor obsahující pouze jeden stav, soubor obsahující pouze jeden přechod, soubor obsahující konzistentní stavový automat o velikosti do 10 stavů a 10 přechodů, soubor obsahující nekonzistentní stavový automat o velikosti do 10 stavů a 10 přechodů
Kroky	1. Importovat vstupní soubor

	2. Vytvořit z něj stavový automat sa1 3. Vyexportovat stavový automat do souboru f 4. Importovat soubor f a vytvořit z něj stavový automat sa2 5. porovnat sa1 a sa2
Očekávaný výsledek	sa1 je ekvivalentní sa2
Prošel/neprošel?	Prošel pro všechna vstupní data

4.1.3 Test hledání nedosažitelných stavů

Název testu	Hledání nedosažitelných stavů
Umístění testovací metody	test_situation_generator.state_automaton
Jméno testovací metody	StateAutomatonTest.testGetUnreachableStates();
Vstupní data	Prázdný automat, automat o 1 počátečním stavu, automat o 1 nepočátečním stavu, automat o 2 dosažitelných stavech, konzistentní automat o velikosti do 10 stavů a 10 přechodů, automat o velikosti do 10 stavů a 10 přechodů s 1 nedosažitelným stavem, automat o velikosti do 10 stavů a 10 přechodů s polovinou stavů nedosažitelných
Kroky	1. Importovat vstupní soubor 2. Vytvořit z něj stavový automat sa1 3. Vyhledat nedosažitelné stavy 4. Porovnat s předpřipravenou množinou nedosažitelných stavů
Očekávaný výsledek	Množina vyhledaných stavů je shodná s množinou předpřipravených stavů
Prošel/neprošel?	Prošel pro všechna vstupní data

5 Závěr

V rámci práce se mi podařilo vyvinout aplikaci Generátor testovacích situací pro State transition test. Domnívám se, že aplikace, tak jak je, splňuje zadání práce ve všech bodech, ale je zatím velmi jednoduchá, obsahuje pouze základní funkčnost. Pro lepší použitelnost by bylo třeba věnovat další čas tvorbě a ladění uživatelského rozhraní.

Obzvláště interaktivní grafický editor by si zasluhoval mnohá vylepšení. Bylo by vhodné přidat grafické znázornění nekonzistentních částí automatu a to jak v rámci vykresleného grafu, tak v rámci tabulky, graficky odlišit počáteční a koncové stavy od stavů běžných, přidat možnost obdélníkového výběru a značně rozšířit možnosti zarovnávání, umisťování a měnění velikosti stavů a přechodů. Dále by bylo dobré přidat funkce pro uložení momentálního stavu aplikace (při exportu se nezachovávají pozice stavů a přechodů na plátně), přidat možnosti „vrátit zpět akci“ a „opakovat akci“. Aplikaci by také určitě prospělo lepší grafické ztvárnění některých funkcionalit, například tabulky testovacích situací, panelu nástrojů pro plátno a podobně.

Pro mě osobně měla práce velký přínos v tom, že jsem si velmi prohloubil znalosti programovacího jazyka Java naučil se pracovat s dokumentací jazyka. Naučil jsem se toho spoustu o tvorbě grafického uživatelského rozhraní pomocí knihovny Swing a také jsem si vyzkoušel práci s některými knihovnami třetích stran. Vyzkoušel jsem si proces vývoje desktopové aplikace od zadání přes návrh architektury a grafického uživatelského rozhraní po implementaci a následné testování.

6 Přílohy

6.1.1 XML schéma pro testovací situace

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="test_situations">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="test_case">
          <xs:complexType>
            <xs:choice maxOccurs="unbounded" minOccurs="0">
              <xs:element name="state">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:integer" name="id"/>
                      <xs:attribute type="xs:string" name="name"/>
                      <xs:attribute type="xs:string" name="description"/>
                      <xs:attribute type="xs:boolean" name="start_state"/>
                      <xs:attribute type="xs:boolean" name="end_state"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="transition">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:integer" name="id"/>
                      <xs:attribute type="xs:string" name="name"/>
                      <xs:attribute type="xs:string" name="description"/>
                      <xs:attribute type="xs:integer" name="source"/>
                      <xs:attribute type="xs:integer" name="target"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

6.1.2 XML schéma pro stavový automat

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="state_automaton">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="states">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="state" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:integer" name="id" use="optional"/>
                      <xs:attribute type="xs:string" name="name" use="optional"/>
                      <xs:attribute type="xs:string" name="description"/>
                      <xs:attribute type="xs:boolean" name="start_state"/>
                      <xs:attribute type="xs:boolean" name="end_state"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="transitions">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="transition">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:integer" name="id"/>
                      <xs:attribute type="xs:string" name="name"/>
                      <xs:attribute type="xs:string" name="description"/>
                      <xs:attribute type="xs:integer" name="source"/>
                      <xs:attribute type="xs:integer" name="target"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Zdroje

1. KOLÁŘ, Josef. Teoretická informatika. 1. vyd. Praha: Česká informatická společnost, 1996, 168 s. ISBN 80-900853-4-2.
2. VEENENDAAL, Erik van. *The testing practitioner*. 3. Nachdr. Den Bosch: UTN Publ, 2002. ISBN 9789072194657
3. *Prezentace z předmětu TS1* [online]. [cit. 2015-05-22]. Dostupné z: http://webdev.felk.cvut.cz/~buresm3/ts1/TS1_prednaska4.pdf
4. *RFC 4180*. Dostupné také z: http://tools.ietf.org/html/rfc4180
5. KOSEK, Jiří. XML pro každého: podrobný průvodce. 1. vyd. Praha: Grada, 2000, 163 s. ISBN 80-7169-860-1.
6. *XML Schema*. Dostupné také z: http://www.w3.org/2001/XMLSchema
7. HEROUT, Pavel. Učebnice jazyka Java. 5. rozš. vyd. České Budějovice: Kopp nakladatelství, 2011, 386 s. ISBN 978-80-7232-398-2.
8. PAVEL HEROUT. Java: grafické uživatelské prostředí a čeština. 2. vyd. České Budějovice: Kopp, 2007. ISBN 8072323288.
9. PECINOVSKÝ, Rudolf. Myslíme objektově v jazyku Java: kompletní učebnice pro začátečníky. 2., aktualiz. a rozš. vyd. Praha: Grada, 2009, 570 s. Myslíme v--. ISBN 978-80-247-2653-3.

Obsah přiloženého DVD

- bp.pdf – soubor s bakalářskou prací ve formátu pdf
- bp.odt – soubor s bakalářskou prací ve formátu odt
- Složka dist se spustitelným souborem Test_situation_generator.jar
  - Složka help - obsahuje uživatelskou nápovědu ve formátu html
  - Složka lib - obsahuje knihovny potřebné pro běh aplikace
  - Složka test data - obsahuje různé stavové automaty pro testovací účely
- Složka src - obsahuje zdrojové kódy aplikace
- složka test - obsahuje testovací třídy