České vysoké učení technické v Praze

Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

2015 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 22.5.2015 Podpis

Poděkování

Děkuji vedoucímu práce, Ing. Miroslavu Burešovi, Ph.D., za věcné připomínky a velkou trpělivost. Děkuji také své přítelky ni, která mi byla velkou oporou při vypracovávání této práce a také děkuji svým rodičům za finanční podporu během studia.

Anotace

Cílem této bakalářské práce je provést návrh a vytvořit aplikaci pro generování testovacích situací pro techniku State Transition Test.

Annotation

The goal of this bachelor's thesis is to design and create an application for generating test situations for State Transition Test technique .

Obsah

1 Úvod

Jako bakalářskou práci jsem se rozhodl naprogramovat softwarový nástroj pro generování testovacích situací pro techniku testování softwaru State Transition Test.

1.1 Základní pojmy

State Transition Test

Technika State Transition Test patří mezi tzv. Black-box testovací techniky. Používá se, pokud lze systém, jeho část, nebo jeho specifikaci popsat konečným stavovým automatem.

N-Switch pokrytí

Testy pokrývají všechny platné sekvence N + 1 navazujících transakcí.

Testovací situace

Testovací situací se rozumí přechodů a stavů ve tvaru stav → přechod → stav

1.2 Přehled existujících řešení

Danou problematikou se již zabývají následující programy:

Conformiq Designer (http://www.conformiq.com/products/conformiq-designer/)

GOTCHA-TCBeans (https://www.research.ibm.com/haifa/projects/verification/gtcb/index.html)

1.3 Zdůvodnění implementace

Důvodem implementace nového řešení je, že mnou vytvářené řešení, na rozdíl od výše uvedených nástrojů bude dostupné zdarma. Moje řešení se také bude zaměřovat pouze na generování testovacích situací pro state transition test, bude tedy vhodnější pro uživatele, kteří vyžadují pouze tuto funkcionalitu.

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 Funkční požadavky

1. Konfigurace stavového automatu pomocí tabulky

    1.1 Systém bude umožňovat zadat seznam stavů automatu

    1.2 Systém bude umožňovat zadat seznam přechodů automatu

    1.3 Systém bude umožňovat zvolit počáteční stav a 0 až n koncových stavů

2. Import a export

    2.1 Systém umožní import stavového automatu ve formátech XML a CSV

    2.2 Systém umožní export stavového automatu ve formátech XML a CSV

    2.3 Systém umožní export testovacích případů

3. Uživatelské rozhraní

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

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

4. Kontrola konzistence stavového automatu

    4.1 Systém umožní kontrolu existence počátečního stavu

    4.2 Systém umožní kontrolu existence alespoň jednoho koncového stavu.

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

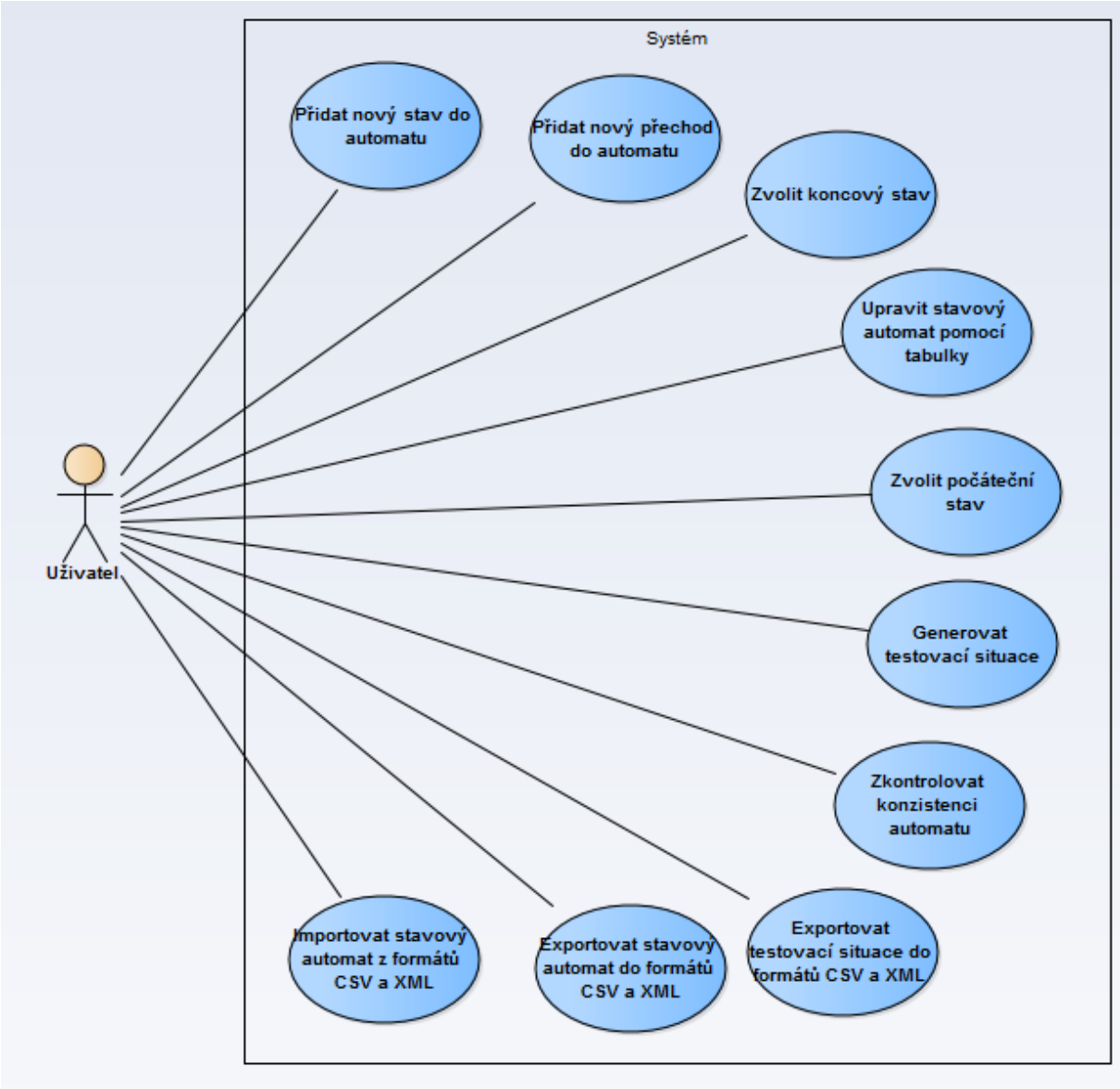
4.4 Systém umožní zkontrolovat existenci nepřipojených, nebo pouze na jedné straně připojených přechodů.

4.5 Systém umožní zkontrolovat, zda všechny nekoncové stavy mají vystupující hrany.

2.1.2 Nefunkční 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



Obrázek 1: Diagram případů užití

Diagram případů užití (Obrázek 1) znázorňuje možné interakce uživatele se systémem.

2.3 Algoritmus pro generování testovacích scénářů

Pro generování testovacích situací v aplikaci používám mnou navržený algoritmus. Algoritmus popisuji „java-like“ pseudokódem. Testovací situace generuji jako posloupnosti přechodů automatu, neboť z přechodů umím získat odpovídající zdrojové a cílové stavy.

```
List resultTestCases = testCasesFromAllTransitions();
List nextTestCases;

for (int i = 1; i < N; i++) {
   for (TestCase tc : resultTestCases) {
         resultTestCases.remove(tc);
         nextTestCases.add(subsequentTestCases(tc));
   }
   resultTestCases.add(nextTestCases);
   nextTestCases.clear();
}
```

Listy *resultTestCases* a *nextTestCases* jsou posloupnosti přechodů. List *resultTestCases* inicializuji tak, že ho naplním testovacími situacemi pro 0-switch pokrytí. Bude tedy obsahovat jednu testovací situaci o délce 1 pro každý přechod automatu.

N je hloubka pokrytí. Funkce subsequentTestCases pro každý testovací případ *t* délky *n* vrátí všechny testovací případy délky *n + 1* takové, že obsahují *t* a rozšiřují ho o navazující stav.

Po skončení algoritmu List *resultTestCases* obsahuje testovací situace pro zadanou hloubku pokrytí.

2.4 Algoritmus pro vyhledání všech nedosažitelných stavů

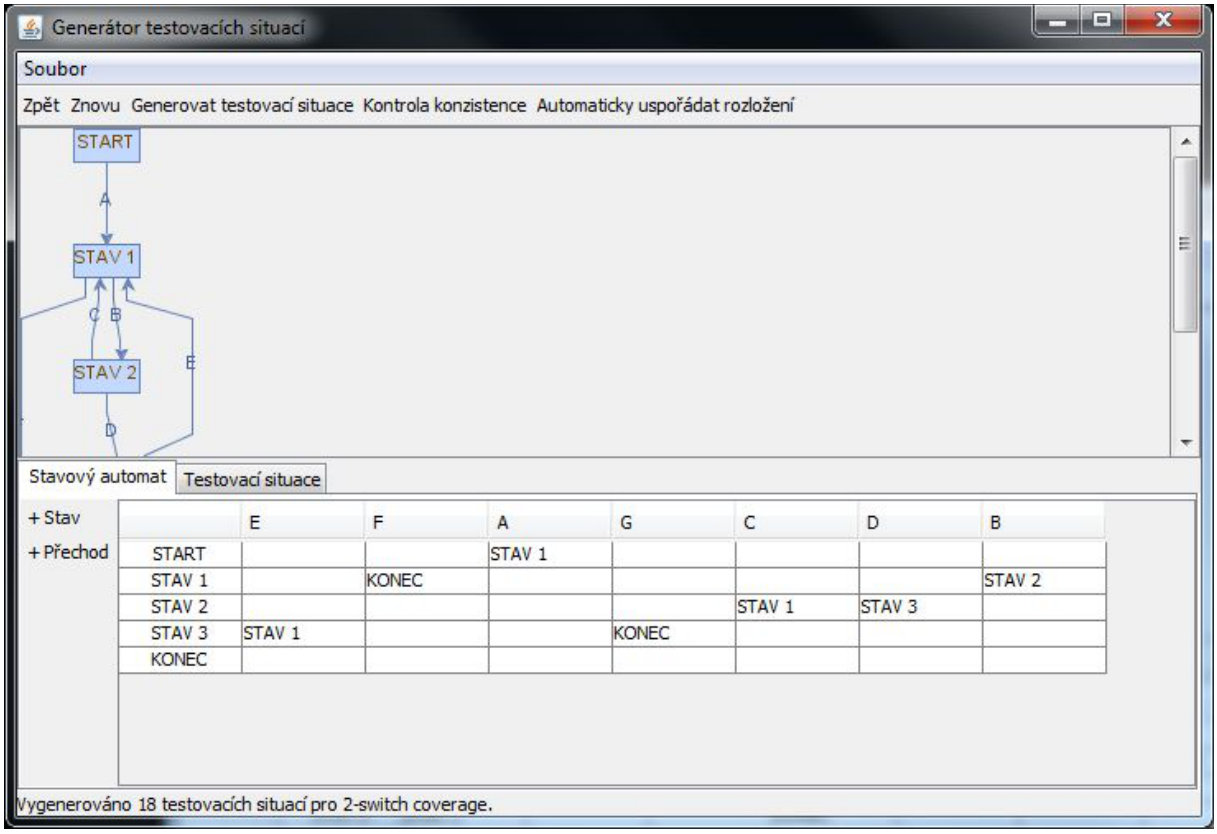
```
set1 = [ S0 ];
set2 = [ S0 ];

do {
   set1.add (set2);
   for (State s : set1)  {
         set2.add(outgoingTransitions(s));
   }
}
while (set1 != set2);

result = allVertices – set1;
```

set1, *set2* jsou množiny stavů. Metoda *outgoingTransitions()* vrací všechny výstupní přechody ze zadaného stavu. Po skončení vnějšího cyklu *set1* obsahuje všechny dosažitelné stavy. Výsledek *result* je tedy získán jako množinový rozdíl množin *allVertices* (množina všech stavů automatu) a *set1*.

2.5 Návrh grafického uživatelského rozhraní

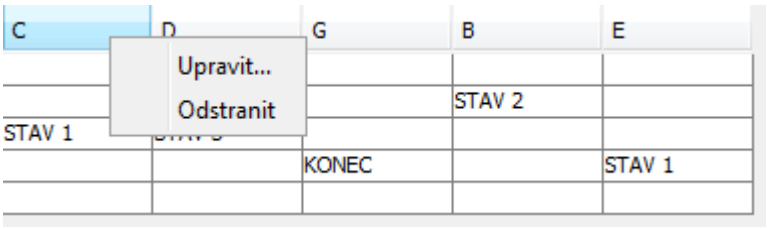
Při návrhu grafického uživatelského jsem vycházel z klasického rozložení běžných desktopových aplikací. Tedy lišta s nabídkami (Soubor) na horním okraji obrazovky, pod ním nástrojová lišta pro

často používané funkce, dále stavová lišta u dolního okraje okna a mezi tím se nachází pracovní plocha pro uživatele.



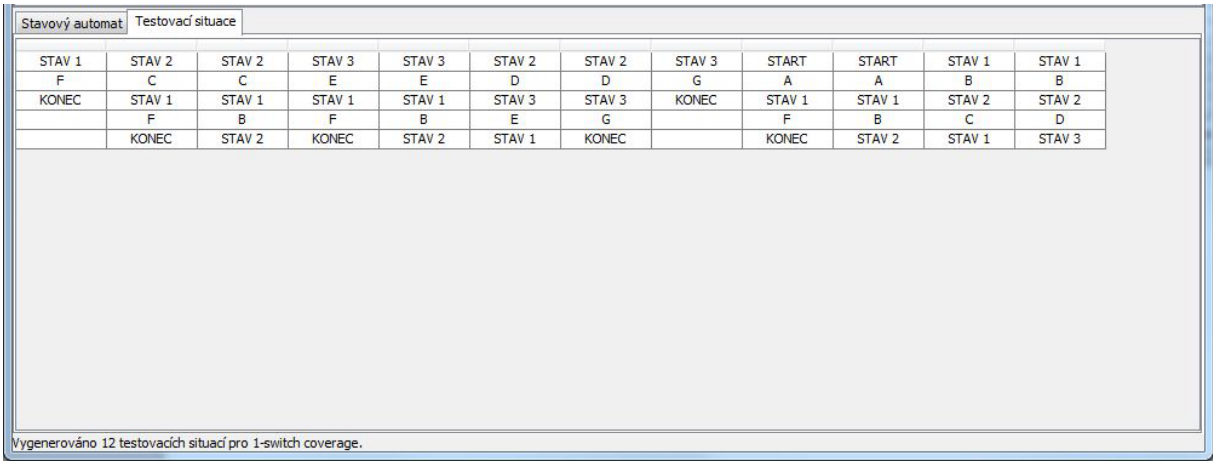
Obrázek 2: Hlavní okno aplikace

Největší její část zabírá plocha pro kreslení stavového automatu, V dolní části se bude nacházet tabulka přechodů 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 (Obrázek 3). Přes kontextové menu bude také možné vyvolat dialog pro editaci stavu (Obrázek 6) nebo přechodu (Obrázek 5).



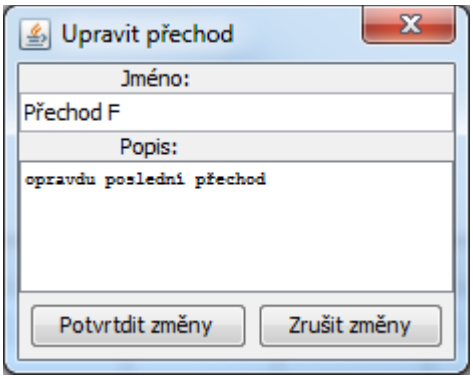
Obrázek 3: Výřez tabulky s kontextovým menu

Tabulka stavového automatu (mimo hlavičky řádků a sloupců) bude editovatelná pomocí rozbalovacích seznamů s nabídkou všech aktuálně existujících stavů v rámci automatu.

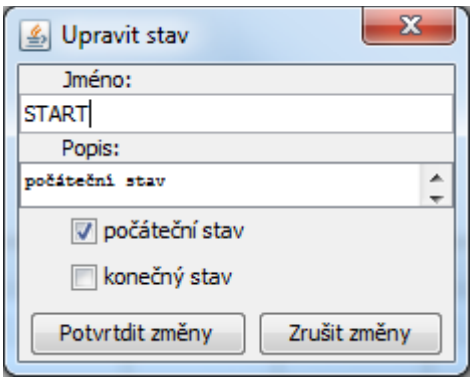


Obrázek 4: Tabulka vygenerovaných testovacích situací

Tabulka testovacích situací (Obrázek 5) bude zobrazovat testovací situace jako jednotlivé sloupce tabulky a bude sloužit pouze ke čtení.



Obrázek 5: Dialog pro editaci přechodu



Obrázek 6: Dialog pro editaci stavu

3 Implementace

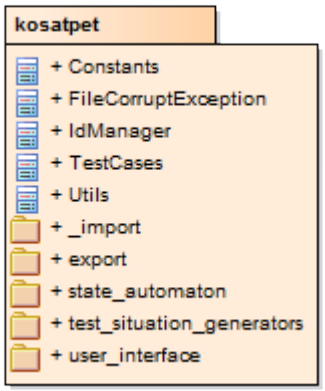
V této kapitole budu hovořit o použitých technologiích v rámci aplikace, popíšu způsob, jakým je aplikace postavena a nakonec se budu věnovat formátům vstupních a výstupních souborů.

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.

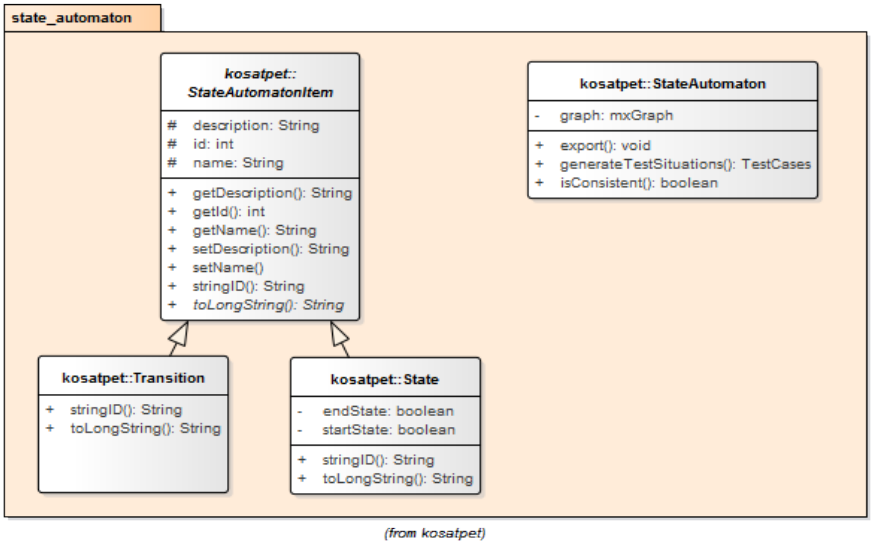
Dále používám knihovnu 3. strany JgraphX (http://www.jgraph.com) pro vykreslování a vnitřní reprezentaci stavového automatu.

Všechny svoje třídy jsem umístil do balíčku s názvem „kosatpet“(Obrázek 7) podle svého školního loginu. Tento balíček obsahuje Všechny další balíčky, které jsem vytvořil a také třídy, které se nehodily jinam.



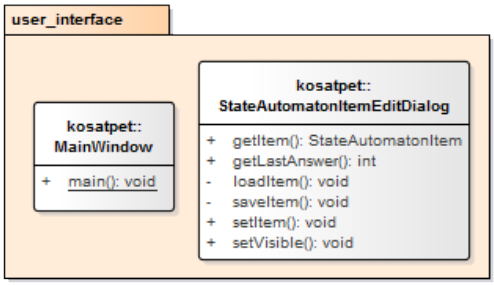
Obrázek 7: balíček kosatpet

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.



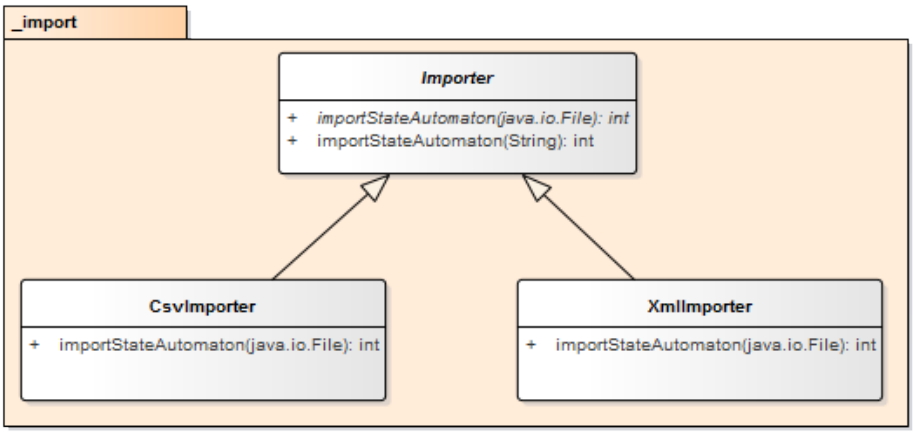
Obrázek 8: balíček state_automaton

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.



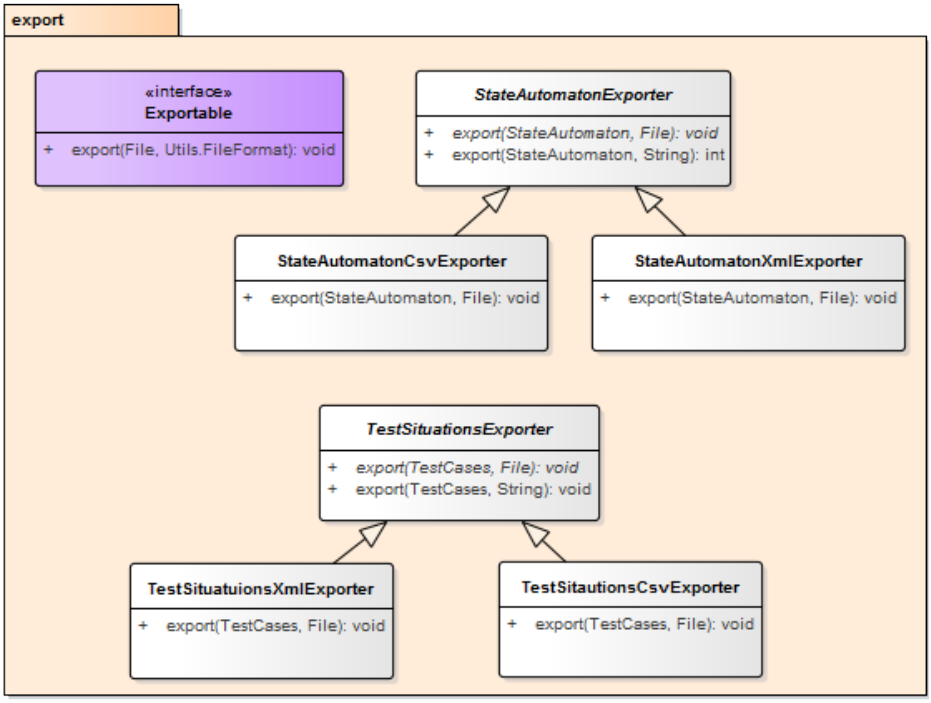
Obrázek 9: balíček user_interface

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 5, Obrázek 6).



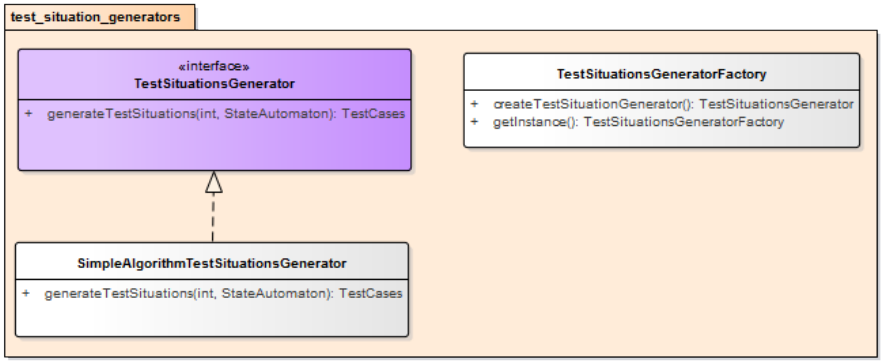
Obrázek 10: balíček import

Balíček import obsahuje abstraktní třídu Importer. a její potomci pro formáty CSV a XML. Všechny třídy importující stavový automat by měli dědit ze třídy Importer.



Obrázek 11: 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 implmementováno třídami StateAutomaton a TestCases.



Obrázek 12: 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. Dále je tu třída TestSituationsGeneratorFactory, která se stará o vytváření instancí tříd implementujících rozhraníTestSituationGenerator.

3.2 Formáty souborů pro import/export

Tato kapitola se zabývá popisem souborů pro import a export testovacích situací a stavového automatu. Stavový automat bude možno importovat a exportovat do formátů CSV a XML, testovací situace bude možno pouze exportovat, taktéž do formátů CSV a XML.

3.2.1 XML schéma pro import/export stavového automatu

Pro definici xml souborů používám jazyk XML Schema Definition Language, zkráceně XSD.

```
<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>
```

3.2.2 XML schéma pro export testovacích situací

```
<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>
```

3.2.3 CSV soubor pro import/export stavového automatu

CSV formát nemá jednotnou specifikaci, vycházel jsem tedy z doporučení v RFC 4180. 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, 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.

3.2.4 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 používám jako oddělovač jednotlivých polí znak „čárka“ a jako oddělovač řádků používám řetězec CRLF.

4 Testování

Algoritmus pro generování testovacích situací jsem testoval za použití JUnit frameworku. Nejdříve jsem vytvořil 10 stavových automatů a k nim jsem si ručně vytvořil odpovídající testovací situace. Ty jsem pak porovnával s výsledky algoritmu abych ověřil jeho správnou funkčnost.

5 Závěr

V rámci práce se mi podařilo vyvinout aplikaci, kterou najdete na přiloženém DVD. Aplikace zatím obsahuje pouze základní funkčnost a není dokonale odladěná. Pro lepší použitelnost by bylo třeba věnovat další čas tvorbě a ladění uživatelského rozhraní a také testování, na které nezbylo mnoho času v rámci vývoje.

Další rozvoj aplikace by mohl směřovat k přidání interaktivity k vykreslovanému grafu, bylo by vhodné přidat grafické znázornění nekonzistentních částí automatu a to jak v rámci vykresleného grafu, tak v rámci tabulky. Dále by bylo možné přidat funkce pro uložení momentálního stavu aplikace, přidat možnosti „vrátit zpět akci“ a „opakovat akci“.

Práce pro mě osobně měla velký přínos v tom, že jsem si prohloubil znalosti programovacího jazyka Java, zejména knihovny Swing.

Zdroje

- *Prezentace z předmětu TS1* [online]. [cit. 2015-05-22]. Dostupné z: http://webdev.felk.cvut.cz/~buresm3/ts1/TS1_prednaska4.pdf
- *RFC 4180*. Dostupné také z: http://tools.ietf.org/html/rfc4180
- *XML Schema*. Dostupné také z: http://www.w3.org/2001/XMLSchema
- KOLÁŘ, Josef. Teoretická informatika. 1. vyd. Praha: Česká informatická společnost, 1996, 168 s. ISBN 80-900853-4-2.
- VEENENDAAL, Erik van. *The testing practitioner*. 3. Nachdr. Den Bosch: UTN Publ, 2002. ISBN 9789072194657

Obsah přiloženého CD

- bp.pdf – soubor s bakalářskou prací ve formátu pdf
- bp.odt – soubor s bakalářskou prací ve formátu odt
- Složka program se spustitelným souborem Test_situation_generator.jar a knihovnami potřebnými pro běh programu
- Složka src se zdrojovými kódy aplikace