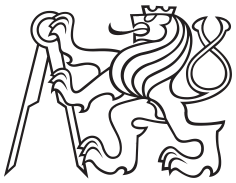
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídící techniky

Implementace rozhraní mezi nástrojem TaSysTest a softwarem EXAM

Michal Veselka

Vedoucí: Ing. Jan Sobotka
Obor: Systémy a řízení
Studijní program: Kybernetika a robotika
Květen 2016

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Michal Veselka**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Implementace rozhraní mezi nástrojem TaSysTest a softwarem EXAM**

Pokyny pro vypracování:

1. Seznamte se s testovacím nástrojem TaSysTest [1] a softwarem EXAM (http://www.exam-ta.de/) pro provádění integračních testů automobilové elektroniky používaný především koncernem Volkswagen.
2. Navrhněte vhodnou softwarovou architekturu umožňující provádění testů nástrojem TaSysTest v prostředí EXAM.
3. Implementujte navržené řešení.
4. Demonstrujte funkčnost provedené implementace formou ukázkového testu.

Seznam odborné literatury:

[1] GRUS, Tomáš. Implementace softwarového nástroje pro generování integračních testů. Praha, 2014. Diplomová práce.
[2] J. Zander, I. Schieferdecker, and P.J. Mosterman. Model-Based Testing for Embedded Systems. Taylor & Francis, 2011.
[3] SWEIGART, Al. Automate the boring stuff with python: practical programming for total beginners. 1st edition. San Francisco, CA: No Starch Press, 2014, pages cm. ISBN 1593275994.

Vedoucí: Ing. Jan Sobotka

Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 1. 3. 2016

Poděkování

Chtěl bych poděkovat vedoucímu své bakalářské práce Ing. Janu Sobotkovi za odborné vedení, za pomoc a rady při zpracování této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 27. května 2016

Podpis: .....

Abstrakt

Cílem této bakalářské práce je implementovat rozhraní testovacího nástroje TaSysTest se softwarem EXAM. TaSysTest je testovací nástroj využívající modely specifikované časovanými automaty pro provádění testů a EXAM je grafické testovací prostředí pro provádění integračních testů automobilové elektroniky používaný především koncernem Volkswagen. Propojení umožní testování systémů specifikovaných časovanými automaty metodou hardware-in-the-loop na různých testovacích platformách dostupných v testovacím softwaru EXAM. Práce nejprve zkoumá zmíněné softwarové nástroje, následně navrhuje vhodnou softwarovou architekturu k jejich propojení a nakonec je formou implementovaného ukázkového testu předvedena funkcionalita rozhraní na modelu zamykání automobilu.

Klíčová slova: TaSysTest, EXAM, UPPAAL, Časované automaty

Vedoucí: Ing. Jan Sobotka

Abstract

The main goal of this bachelor thesis is to implement the interface between the test tool TaSysTest and the EXAM software. TaSysTest is a testing tool that uses models specified timed automata for testing and EXAM is a graphical testing environment for performing integration tests of automotive electronics used primarily by Volkswagen concern. The interconnection allows testing systems specified by timed automaton using hardware-in-the-loop testing on different platforms available in testing software EXAM. The thesis examines the said software tools, then suggests appropriate software architecture to connect them and in conclusion is implemented in the form of sample test demonstrated the functionality of the interface model locking the car.

Keywords: TaSysTest, EXAM, UPPAAL, Timed automaton

Title translation: Implementing an interface between TaSysTest tool and EXAM software

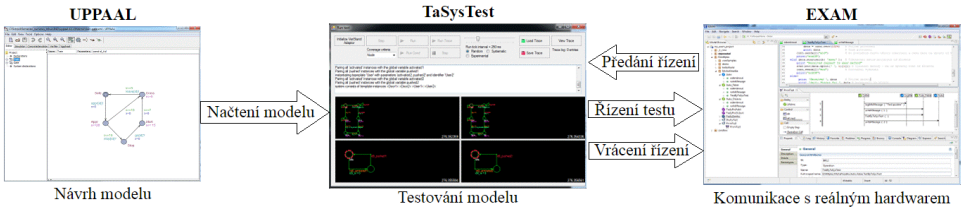
Obsah

Obrázky

Kapitola 1
Úvod

Hlavním cílem této bakalářské práce je implementace rozhraní mezi nástrojem TaSysTest a testovacím softwarem EXAM používaným především koncernem Volkswagen.

Díky tomuto propojení bude možné provádět testování systémů modelovaných časovanými automaty testovacím nástrojem TaSysTest na reálném hardwaru dostupném přes software EXAM.



Obrázek 1.1: Navrhovaná architektura

1.1 Současný stav

TaSysTest je nástroj napsaný v jazyce C# využívající pro svůj běh Microsoft .NET Framework 4.5 knihovny a na druhé straně EXAM, který pro svůj běh používá zkompilované skripty pro Python verze 2.x.

Ani jeden z programů nemá vhodné rozhraní pro vzájemnou komunikaci mezi sebou a proto je potřeba ho vytvořit.

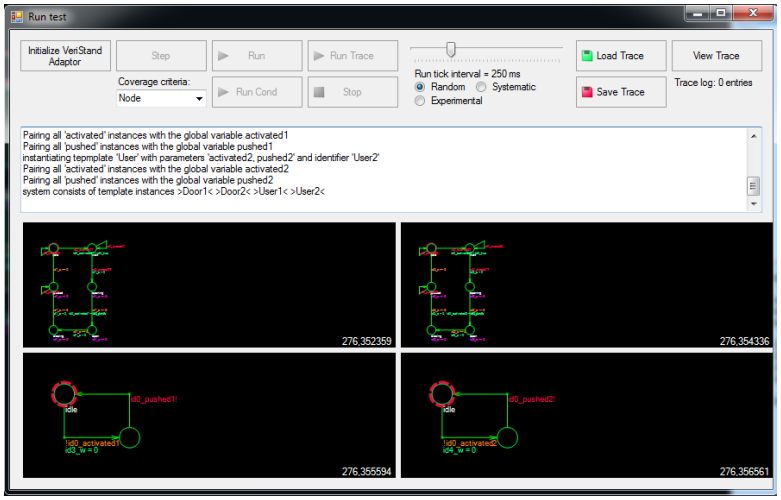
1.1.1 TaSysTest

Program TaSysTest v současném stavu umožňuje načíst navržený, odsimulovaný a verifikovaný model z návrhového prostředí UPPAALu a následně s využitím tohoto modelu provádět testy na reálném hardwaru. Ke komunikaci s hardwarem využívá knihovny a funkce z prostředí VeriStand od firmy National Instruments.

TaSysTest si načtený model nejprve pomocí vlastních algoritmů interpretuje (rozpozná vrcholy, hrany, proměnné,...), provede syntaktickou analýzu

načteného kódu a po inicializaci VeriStand adaptéru nabídne konfiguraci a spuštění testů.

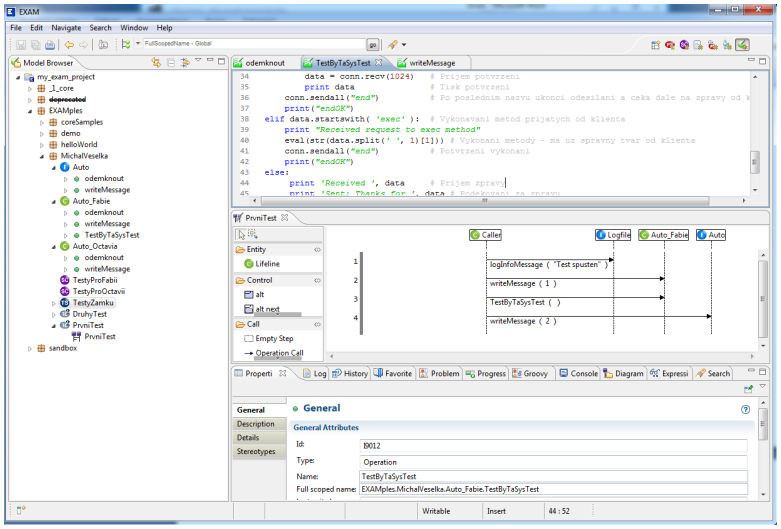
Umožňuje také ukládání a opětovné načtení a spuštění dříve provedeného testu.



Obrázek 1.2: TaSysTest

1.1.2 EXAM

EXAM je komplexní prostředí pro provádění integračních testů automobilové elektroniky používané především koncernem Volkswagen. Umožňuje grafický návrh testů pomoci sekvenčních diagramů, diagramů aktivit nebo skriptů napsaných v Python kódu.



Obrázek 1.3: EXAM

1.2 Použité názvosloví

Především v prostředí EXAMu je použito mnoho anglických názvů, pro které neexistuje odpovídající český překlad, proto se u názvů tohoto typu budu v této práci držet názvů anglických. Takto nepřeložené názvy jsou v textu vyznačeny *kurzívou*.

Kapitola 2
EXAM

2.1 Popis prostředí

EXAM (EXtended Automation Method) je grafické testovací prostředí definující komplexní metodiky založené na UML (Unified modeling language) k reprezentaci a vyhodnocování testů. Umožňuje grafické modelování testovacích procesů v sekvenčních diagramech nebo diagramech aktivit bez programovacích znalostí (zdroj [Mic16]).

EXAM tímto poskytuje jednotný a platformně nezávislý jazyk pro reprezentaci zkušebních testů a díky tomu jsou společnosti schopné dosáhnout jednotných testovacích postupů, umožnit znovupoužitelnost provedených testů a testovat společně s dodavateli, či partnery mimo vlastní testovací oddělení (zdroj [SS16]).

Od roku 2006 je EXAM vyvíjen jako společný projekt AUDI AG, Volkswagen AG a MicroNova AG.

2.2 Režimy práce a zobrazení

2.2.1 Základní typy objektů

- **Třídy a rozhraní**

  Třídy (*classes*) obsahují funkce, které mohou být použity v několika sekvencích nebo v ostatních modelech. Třída slouží jako kritérium klasifikace a zapouzdřuje spojení funkcionalit v jejich metodách.

  Každá metoda je sekvence a má buď tělo složené z Python kódu, nebo podřízeného sekvenčního diagramu. Tento diagram i Python kód jsou definované sekvencemi ve zkušebním skriptu, který bude následně generován pro danou metodu.

  Pomocí rozhraní (*interface*) můžeme "přepínat" mezi několika implementacemi dané testovací knihovny. Obecné části *TestCase* výhradně používají operace a rozhraní.

  Prostřednictvím výběru v *SystemConfiguration* je dané rozhraní během testu nahrazeno správnou třídou z knihovny testovacích funkcí.

- **Systémové nastavení**

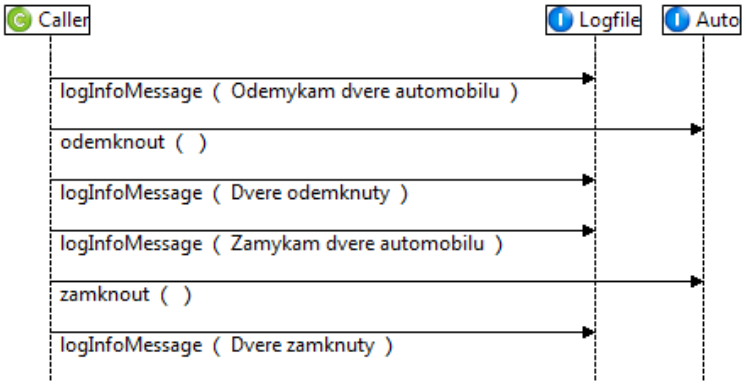
  V *System Configuration* lze nastavit k jednotlivým rozhraním jaké budou obsahovat konkrétní implementace tříd. Na toto nastavení jsme poté dotázáni při spuštění testu. Lze tedy pro jeden univerzálně vytvořený *Test case* spustit testovací sekvenci s různými nastaveními.

- **Balíčky**

  Balíčky (*package*) jsou určeny k uspořádání souborové struktury *Model browser* v EXAM modelu. Jejich obsah může být považován za samostatný modul, který může být dále distribuován.

- ***Test Case***

  Test case popisuje konkrétní akce prováděné s určitou softwarovou komponentou a jejich očekávané výsledky (dle [PAG09]). Je tedy dokument, popisující určitou činnost, kterou je potřeba otestovat. Obecně lze říci, že obsahuje kroky se skutečnými vstupními hodnotami spolu s očekávanými výsledky.(dle [Tes16]). Jedná se o definici sekvenčního diagramu, ve kterém vytvoříme postup, v jakém se mají definované metody vykonávat a s jakými parametry.



Obrázek 2.1: Test case

- ***Test Suite***

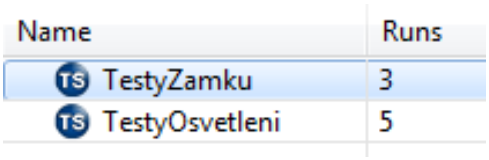
  V *Test Suite* definujeme, jaké *Test Case* mohou být vykonány v *Testrunneru*. Je možné je shlukovat do skupin (*Test groups*) a poté spouštět test pouze pro vybranou skupinu. *Test Suite* musí existovat společně s vhodnou metodou.

Name	Runs	Retries
TestOdemykani	5	0
TestZamykani	10	0

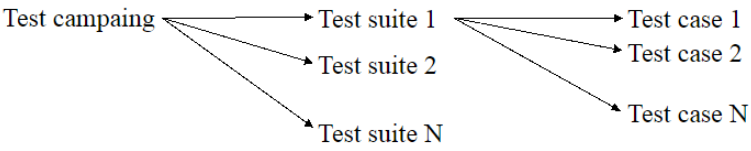
Obrázek 2.2: Test suite

- ***Test Campaign***

  Umožňuje vytvořit sekvenci několika *Test Suite* s možností nastavení počtu spuštění.



Obrázek 2.3: Test campaing

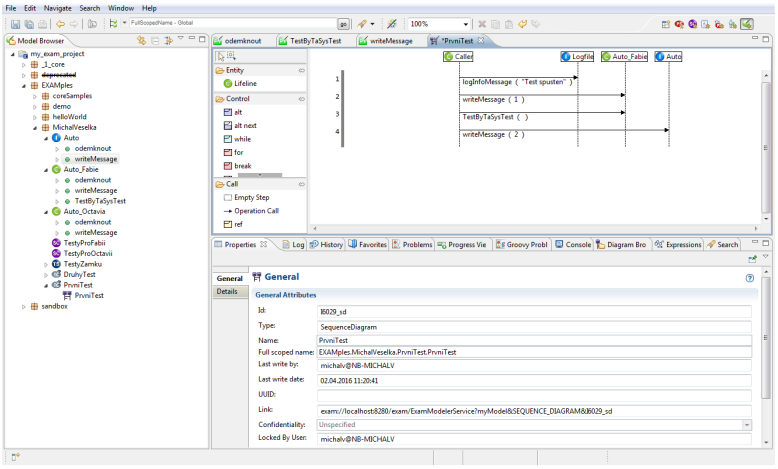


Obrázek 2.4: Struktura testovacích souborů

2.2.2 Uživatelské zobrazení

Pro přehlednost je pro jednotlivé operace připraveno jiné zobrazení uživatelského rozhraní.

- Modeler perspective



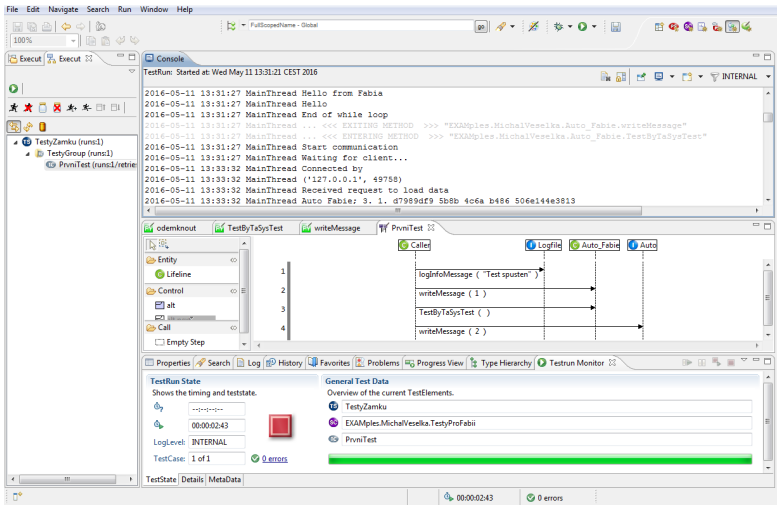
Obrázek 2.5: Modeler perspective

Obrázek 2.5 ukazuje tzv. “Modeler Perspective” neboli zobrazení v EXAMu, ve kterém je možné vytvářet a následně i provádět všechny potřebné změny modelu a nastavení běhu všech prováděných testů.

V levé části je umístěna stromová struktura programu (*Model Browser*) obsahující všechny dostupné soubory pro editaci.

V pravé části je otevřen sekvenční diagram, do kterého lze jednoduše přidat požadovanou metodu vybráním metody v *Model Browser* a přetažením na požadované místo sekvenčního diagramu. V levé části diagramu jsou zobrazeny vstupy a v pravé části výstupy pro dané metody. Samotný název metody je poté zobrazen propojkou mezi vstupem a výstupem. V závorce za názvem metody jsou uvedeny vstupní parametry dané metody. Sekvenční diagram lze vytvořit univerzálně pro více testů tím, že dosazujeme metody z definovaného rozhraní a při spuštění testu teprve vybereme, pro jakou třídu implementující toto rozhraní chceme test provést.

- Testrunner perspective



Obrázek 2.6: Testrunner perspective

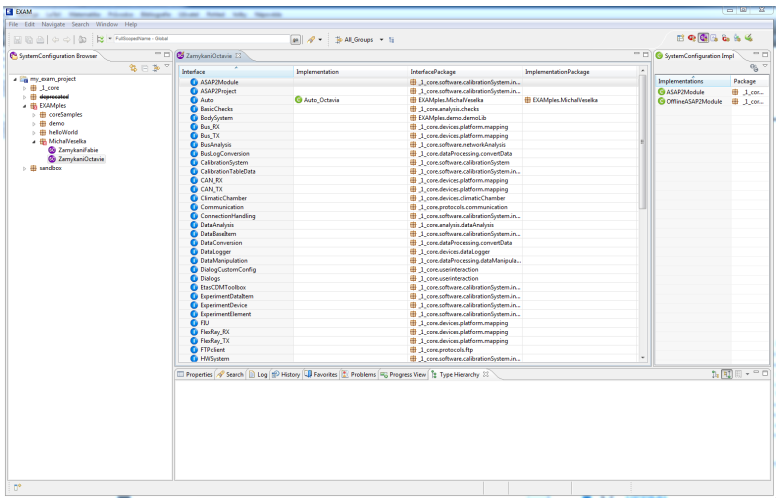
Obrázek 2.6 ukazuje "Testrunner perspective", což je zobrazení pro provádění testů námi implementovaných v *Test suite*.

V levé části se nachází struktura s testovacími daty (*Execution Data*), ve které jsou zobrazeny námi vytvořené *Test Suite* struktury s definovanými *Test case* pro testování.

V pravé části je umístěna konzole s výpisem právě probíhané části testu a pod ní je zobrazen implementovaný sekvenční diagram.

Vpravo dole je umístěn panel s informací o právě probíhajícím testu.

- SystemConfiguration perspective



Obrázek 2.7: SystemConfiguration perspective

Na obrázku 2.7 je zobrazeno tzv. “SystemConfiguration Perspective” prostředí, ve kterém je možná konfigurace rozhraní pro provádění konkrétních testů.

V levé části je umístěna stromová struktura programu (*Model Browser*) zobrazující vytvořené soubory *Systemconfiguration*.

V hlavní části okna je možné provést požadované nastavení. K rozhraní přiřadíme konkrétní implementaci třídy a při spuštění testu s tímto nastavením bude v daném testu implementované rozhraní nahrazeno přiřazenou třídou.

Kapitola 3

TaSysTest a UPPAAL

3.1 TaSysTest

TaSysTest je program, který vznikl v roce 2014 jako Diplomová práce Ing. Tomáše Gruse. Slouží jako nástroj k testování navržených modelů z prostředí UPPAALu na reálném hardwaru za pomoci NI VeriStand. Program je psaný v C# a využívá knihoven Microsoft .NET Framework 4.5.

Program umožňuje načíst kompletní navržený model z prostředí UPPAALu, který je ve strukturovaném formátu s příponou *.xml. V tomto souboru jsou uloženy všechny informace o daném časovaném stavovém automatu, což zahrnuje uzly, hrany, jejich propojení, definici systému a uživatelské funkce. TaSysTest si všechny tyto načtené části pomocí svých algoritmů interpretuje do vlastních tříd a objektů pro následné provádění integračních testů v různých módech.

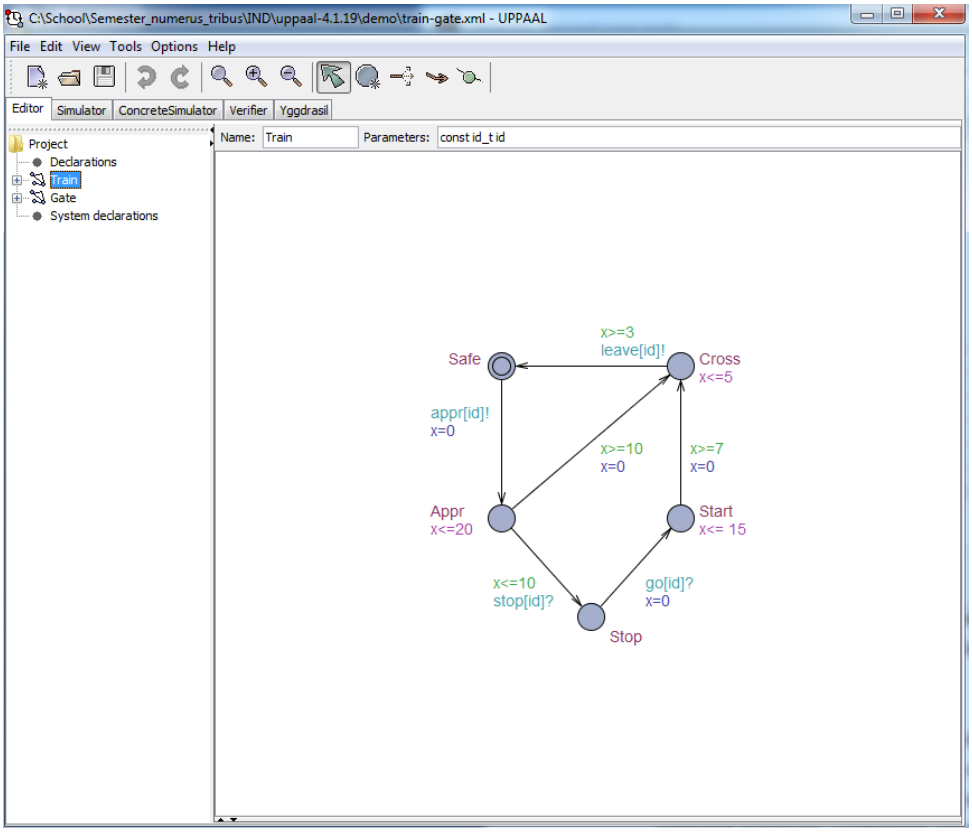
Podrobný popis viz. [Gru14a].

Interakce s hardwarem přes NI VeriStand a jeho API (viz [Gru14b]) bude nahrazena EXAMem.

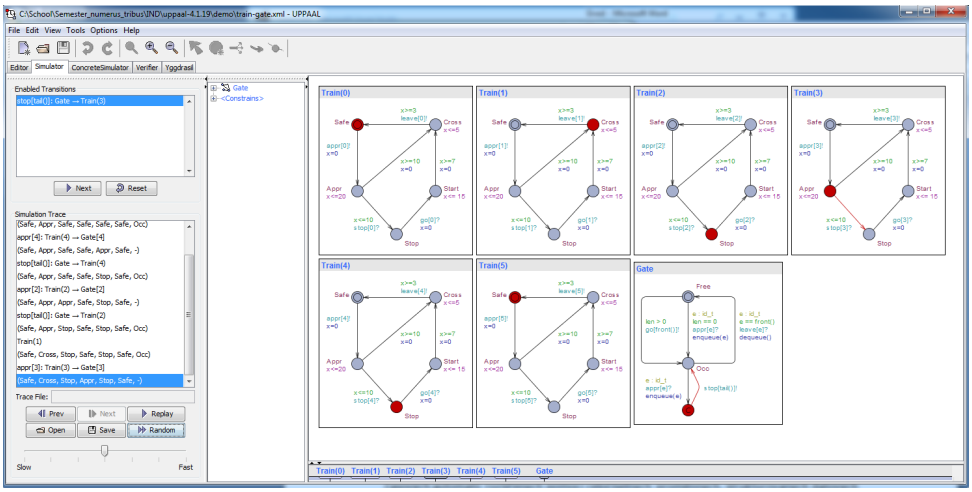
3.2 UPPAAL

Návrhové prostředí UPPAALu umožňuje tvorbu, simulaci a verifikaci systému specifikovaných časovanými automaty. Je vyvíjený jako společný projekt dvou univerzit a to švédské Uppsala University a dánské Aalborg University. Lze v něm provést ověření systémů, které jsou jako síť časových automatů rozšířených pomocí celočíselných proměnných, strukturovaných datových typů, uživatelsky definovanými funkcemi a synchronizačními kanály. První verze tohoto nástroje vznikla v roce 1995.

Program je stále vyvíjen a kromě základního nástroje existuje i velké množství jeho rozšíření (např. TIMES, Stratego, CORA, TRON, atd. více na [UPP15])



Obrázek 3.1: Návrhové prostředí UPPAALu



Obrázek 3.2: Simulační prostředí UPPAALu

Kapitola 4

Implementace rozhraní

Hlavním cílem této bakalářské práce je návrh vhodné softwarové architektury umožňující provádění testů nástrojem TaSysTest v prostředí EXAM.

Je zapotřebí si mezi těmito dvěma prostředími předávat vzájemně informace o aktuálním stavu běhu jednoho, či druhého prostředí a také posílat požadavky na navázání začátku komunikace, předání řízení prostředí TaSysTestu, odesílání požadavků na vykonání jednotlivých kroků a závěrem i řádné ukončení vzájemného spojení a předání řízení zpět prostředí EXAMu.

Nabízí se zde možnost zvolit cloudové úložiště pro oba programy, kde by se průběžně zapisovaly soubory s požadavky na vykonání od jednotlivých prostředí, a to pomocí souborů s předem definovanou strukturou (např. XML strukturou). Druhá strana by v pravidelných intervalech prohledávala společné úložiště, a pokud by hledáním nalezla soubor s novými informacemi od druhé strany, daný soubor by načetla, zpracovala, uložila výsledek pro druhou stranu a načtený soubor smazala.

Takto navržené řešení se společným úložištěm by mělo velikou výhodu v kombinaci s použitím cloudového úložiště, na které by se bylo možné připojit z jakéhokoliv zařízení podporujícího připojení k danému cloudu. Při použití internetového úložiště by pak pro komunikaci s druhou stranou bylo zapotřebí pouze internetové připojení a byla by zajištěná komunikace obou programů z kteréhokoliv počítače v jakékoliv části světa.

Vyskytuje se zde však několik problémů, a to především neustálé načítání a zapisování souborů na dané úložiště, zamčení souboru při jeho načítání/zpracování/ukládání a celkově zbytečně složité předávání dat.

Obě prostředí si budou mezi sebou předávat pouze krátké zprávy s požadavky na druhé prostředí, případně informační zprávy o úspěšně odeslaných/přijatých datech. Předpokládá se, že obě prostředí poběží v rámci jednoho počítače, případně budou prostředí rozdělena v rámci jedné sítě do dvou lokálních strojů.

Při znalosti tohoto rozdělení, se proto nabízí možnost síťové komunikace, kde by jednotlivé požadavky a zprávy byly posílány sokety skrze lokální síť na danou IP adresu v případě rozdělení prostředí do dvou lokálních strojů, případně v rámci “localhost” na stejném počítači.

Síťová komunikace se zdá pro toto spojení nejvhodnější, proto už nastává pouze otázka, zda je vhodnější využít komunikaci pomocí protokolu UDP,

nebo TCP. Dle [Bea09] je protokol UDP paketově orientovaný protokol nižší úrovně (nespojovaný), ve kterém stroje odesílají a přijímají diskrétní pakety informací, aniž by formálně vytvářely spojení. Oproti tomu protokol TCP vytváří obousměrný komunikační proud mezi stroji.

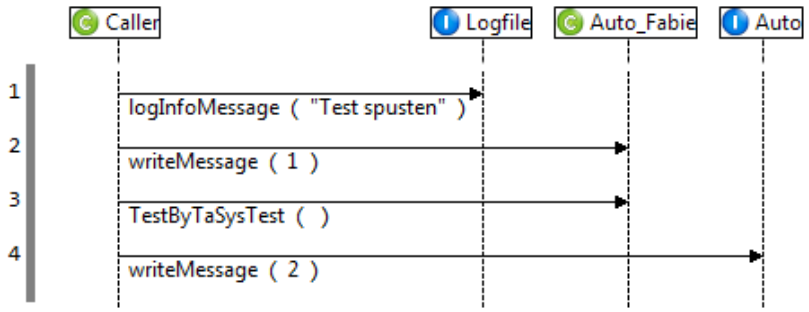
Jelikož potřebujeme zprostředkovat komunikaci, při které jedno prostředí řídí druhé, je kladen důraz na spolehlivost spojení mezi stroji a správné doručení paketu od jedné strany k druhé. Proto je zvolen protokol TCP, který vytvoří spojení mezi oběma stroji a zajistí spolehlivou komunikaci.

Komunikace pomocí TCP/IP je postavena na architektuře klient/server, což znamená, že jedna strana (server) čeká na spojení od strany druhé (klienta) a po úspěšném navázání spojení očekává server požadavky od klienta, které zpracuje a odešle odezvu zpět klientovi.

V našem případě je sice po předání řízení ze strany EXAMu straně TaSysTestu průběh testu řízen stranou TaSysTestu, ale EXAM je v pozici, kdy přijímá požadavky k vykonání určitých metod, proto je v následující implementaci zvolen jako server EXAM a stranu klienta tvoří TaSysTest.

4.1 EXAM

Na straně EXAMu je rozhraní implementování jako metoda (Python skriptu) "TestByTaSysTest()" v sekvenčním diagramu. Myšlenka je taková, že se budou postupně provádět jednotlivé kroky sekvenčního diagramu a v námi požadovaném kroku se provede navázání spojení s TaSysTestem, načež se mu předá řízení a EXAM (server) začne od klienta (TaSysTest) přijímat pokyny k vykonání metod (kroků testu).



Obrázek 4.1: Sekvenční diagram s předáním řízení TaSysTestu

Po zavolání metody "TestByTaSysTest()" se provedou následující kroky:

1. **Navázání spojení (server)**

   EXAM tvoří serverovou stranu komunikace pomocí TCP/IP protokolu, proto musí obstarat vytvoření spojení a poté čekat na připojení klientů (v našem případě pouze jednoho klienta).

   Programovací prostředí pro nás bude představovat jazyk Python, ve kterém vytvoříme skript, po jehož zavolání se vytvoří serverová strana

komunikace, která bude čekat na připojení klienta a poté na jeho požadavky.

Prvním krokem vytvoření úspěšné TCP/IP komunikace je vytvoření TCP soketu. Jazyk Python obsahuje pro tyto účely knihovnu "socket". Po jejím importu je možné využívat proměnnou, která je referencí na objekt typu socket.

Následně je zapotřebí definovat spojení pro daného hosta na určitém portu. V této práci bylo vše testováno v rámci jednoho počítače, proto nastavuji **hostname** na "localhost" a port si musíme zvolit některý z dosud volných (obsazené jsou např. 20 : FTP-Data, 23 : Telnet, 80 : HTTP(WWW), atd.). Z volných portů jsem si zvolil port číslo 13.

Dále je zapotřebí určit počet klientů, kteří se mohou připojit k serveru. V našem případě předáváme řízení pouze jediné instanci TaSysTestu. Nastavíme tedy povolený počet klientů na jednoho.

Program je nyní ve stavu, kdy čeká na připojení klienta na dané adrese a portu. Po úspěšném připojení klienta již může probíhat obousměrná komunikace mezi EXAMem a TaSysTestem.

2. **Odeslání informací potřebných ke komunikaci**

První fází, která nastane po úspěšném připojení klienta k serveru je odeslání názvů tříd, které mohou být volány ze strany TaSysTestu. Pokud není klientovi odeslán název třídy, kterou klient chce používat, je na straně klienta odmítnut (není žádané, aby měl připojený klient přístup ke všem třídám).

V EXAMu mohou být třídy, metody i parametry přidány do těla Python kódu pouhým přetažením z okna *Model browser*. Tato metoda vytvoří v kódu referenční odkaz, na který lze se současným stiskem klávesy 'Ctrl' kliknout, a to způsobí odkázání na umístění dané reference v souborové struktuře *Model browser*.

EXAM si tento referenční odkaz překládá na skutečný název, který bychom potřebovali pro úspěšné zavolání dané metody třídy, popř. parametru.

Mimo tento překlad dojde také k automatickému importu třídy do kódu, který je nutný pro volání metod z této třídy.

V současném stavu vypracování kódu je pro úspěšné načtení referenčního názvu, i skutečného názvu třídy použita struktura jménem indexovaných položek slovníku, kde index tvoří textový řetězec s názvem dané třídy a hodnotou jí je referenční odkaz, který vznikl přetažením z *Model browser* (v EXAM kódu podtržený a obarvený modře). Pro pozdější správnou interpretaci je nutno jej ještě opatřit dvojitým uvozením.

Struktura skutečného názvu třídy odpovídá souborové struktuře *CodeSynchPath*, ve které se nalézá kompilovaný Python soubor s danou třídou, jež se tímto zavolá.

```
List["Auto_Fabia"] = "Auto Fabia"
List["Auto_Octavia"] = "Auto Octavia"
```

Obrázek 4.2: Vytvoření slovníku tříd

```
import _3._1._d7989df9_5b8b_4c6a_b486_506e144e3813
import _e._9._1504fce4_182f_4426_9e5a_39cb7a009f9e
List["Auto_Fabia"] = "_3._1._d7989df9_5b8b_4c6a_b486_506e144e3813"
List["Auto_Octavia"] = "_e._9._1504fce4_182f_4426_9e5a_39cb7a009f9e"
```

Obrázek 4.3: EXAMem přeložená struktura

Skutečný generovaný Python kód vypadá následovně:

Například tedy kompilovaný soubor s názvem `_d7989df9_5b8b_4c6a_b486_506e144e3813` je uložen ve složce `C:\EXAM\EXAM\p2\myModel\_3\_1`.

V generování těchto názvů není vidět na první pohled žádná souvislost, neboť hned druhá vytvořená třída ze stejného interface má referenční odkaz překládaný jako `_3._c._c5b62fe7_8d22_4840_94f3_c38d94afb8c3`.

Pro správné fungování komunikace TaSysTest - EXAM je nutné znát tyto skutečné názvy tříd i jejich referenční názvy (pro přehlednost a smysluplné volání ze strany TaSysTestu), a proto se na straně serveru jako první odešle klientovi definovaný slovník povolených tříd a ten již poté dokáže volat správně dané třídy i jejich metody. Tento slovník je možné definovat ve třídě “TestByTaSysTest” v metodě “ListOfClasses”.

3. **Příjem metody k vykonání**

   Pokud chceme zavolat nějakou metodu dynamicky (tj. nemáme v kódu staticky přidanou referenci, jak bylo popsáno výše), potřebujeme ji zavolat jejím skutečným názvem. Jelikož jsme klientovi odeslali slovník těchto názvů, tak od klienta již dostáváme zpracovaný řetězec (viz 2) a tudíž na straně serveru již provedeme pouze zavolání metody.

   Přijatý řetězec má tvar:
   `exec SkutecneJmeno.getInstance().Metoda(vstupniParametry)`, proto na straně serveru provedeme pouze oddělení exec (umístěno jako informace pro server že má danou metodu vykonat) a příkazem `eval(SkutecneJmeno.getInstance().Metoda(vstupniParametry))` požadovanou metodu vykonáme.

   Program je navržený tak, že funkce umožňuje vracet návratovou hodnotu z právě vykonané funkce zpět do TaSysTestu.

4. **Ukončení spojení**

   Pokud budeme chtít ukončit vzájemné spojení mezi TaSysTestem a EXAMem, jsou zde dvě možnosti:

   a. Časová expirace prováděného testu (viz 3)

b. Manuální ukončení spojení (viz 3)

Obě možnosti odešlou zastavující paket, který ukončí vzájemné spojení, tím i odebere řízení TaSysTestu a EXAM pokračuje ve vykonávání dalšího kroku sekvenčního diagramu.

4.2 TaSysTest

Jelikož jsme si zvolili komunikaci postavenou na architektuře klient/server a jako serverovou stranu jsme si vybrali EXAM, vyplývá z toho, že TaSysTest bude navržený jako jediný klient tohoto serveru.

S výhodou využijeme stávajícího navrženého prostředí TaSysTestu a místo metod vykonávajících inicializaci NI VeriStand ovladače, nahradíme za metody sloužící k navázání spojení směrem k serveru.

Povolením expirační doby trvání testu a nastavením požadované hodnoty dojde při spuštění simulace k aktivaci odpočítávání, které po uplynutí nastavené doby automaticky provede zastavení probíhající simulace a odpojení od serveru. Díky tomu EXAM pokračuje v dalším kroku svého sekvenčního diagramu a opětovně předá řízení zpět TaSysTestu pouze v případě, kdy tomu tak bude požadováno v nějakém následujícím kroku sekvenčního diagramu.

1. **Příjem informací potřebných ke komunikaci**

   V TaSysTestu je strana klienta vyřešena pomocí vytvoření nové třídy "Client" využívající .NET knihovny "System.Net.Sockets", ze které vytvoříme za pomocí konstruktoru objekt této třídy, kterému se při vytváření předá informace o IP adrese serveru (při lokálním běhu obou programů na stejném stroji zvolen "localhost") a používaném portu (zvolen port 13) ([MSD16]).

   Dalším krokem, který se provede po navázání spojení, je přijmutí povolených názvů tříd, se kterými může TaSysTest operovat v rámci řízení EXAMu. Proto vyšle na stranu serveru soket se zprávou "load" a od serveru začnou přicházet data v jednotlivých soketech a jsou strukturovány jako zpráva ve tvaru "ReferenčníNázev;SkutečnýNázev". TaSysTest provede rozdělení na dva řetězce v místě se středníkem a uloží si tato data do slovníku, kde klíč je referenční název proměnné a hodnota je skutečný název proměnné.

   Takto přijímá klient data, dokud nepřijme soket se zprávou o odeslání všech povolených metod, načež klient potvrdí tuto zprávu a obě strany jsou připraveny – server k příjmu metod a klient k jejich odeslání.

2. **Odeslání metody k vykonání**

   Nyní již můžeme vybrat způsob simulace daného časovaného automatu a to buď krokováním po jednotlivých hranách, společně s výpisem logu o aktuálním dění, nebo spustit automatické procházení grafu dle různých kritérií.

V okamžiku, kdy se při přechodu z jednoho stavu do druhého dostaneme s aktualizací hrany na metodu volající některou metodu z EXAMu (poznáme jí dle tvaru "CallExam(Třída.názevMetody(vstupníParametry))"), provede se nejprve kontrola, zda je daná třída povolena nebo obsahuje-li slovník povolených metod TaSysTestu klíč s daným názvem třídy. Pokud je skutečně daný název ve slovníku, provede se v následujícím kroku tvorba řetězce ve tvaru:
"exec Třída.getInstance().metoda(vstupníParametry)", kde "exec" je informace pro server, že má metodu vykonat a následuje požadovaný tvar řetězce, ve kterém ho server očekává pro úspěšné provedení dané metody se vstupními parametry.

3. **Ukončení spojení** Ukončení navázaného spojení může nastat dvěma způsoby:

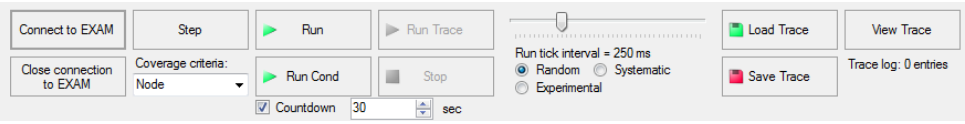
   a. Stiskneme tlačítko "Close connection to EXAM", které odešle serveru soket se zprávou o ukončení spojení a server odešle potvrzení o provedení ukončení komunikace a pokračuje v dalším kroku svého sekvenčního diagramu.

   b. Dojde k expiraci nastaveného času simulace. V první fázi se zastaví právě probíhaná simulace a poté ukončí spojení se serverem stejným způsobem jako v bodě a) při stisku tlačítka "Close connection to EXAM"

4.2.1 Úprava GUI

Metody, které se vykonají po stisku tlačítka pro připojení NI VeriStand ovladače, nahradíme metodami pro klientské připojení k serveru.

Dále doplníme tlačítko pro manuální ukončení komunikace s EXAMem a *checkbox* společně s možností povolení a nastavení hodnoty expirační doby.



Obrázek 4.4: Úprava rozhraní TaSysTestu

Kapitola 5

Test implementovaného rozhraní

5.1 Cíle testu

Po úspěšné implementaci vytvořeného rozhraní mezi prostředími TaSysTestu a EXAMu provedeme závěrečný test, ve kterém ověříme funkcionalitu celého řešení. Jako testovaný subjekt jsem si zvolil systém zámků dveří osobního automobilu.

5.2 Popis funkcionality

Systém zámků osobního automobilu jsem modeloval ze tří částí:

- systém popisující stav zámků
- systém popisující stav dveří
- systém popisující stav klíče

Pomocí klíče lze odemknout buď všechny dveře najednou, nebo pouze dveře od zavazadlového prostoru, přičemž ostatní dveře zůstanou zamknuté. Pro jejich odemknutí je zapotřebí postupovat stejně jako při odemykání všech dveří.

Zamknutí se provede buď časovou expirací při neotevření žádných dveří, nebo zamknutím pomocí klíče.

5.3 UPPAAL

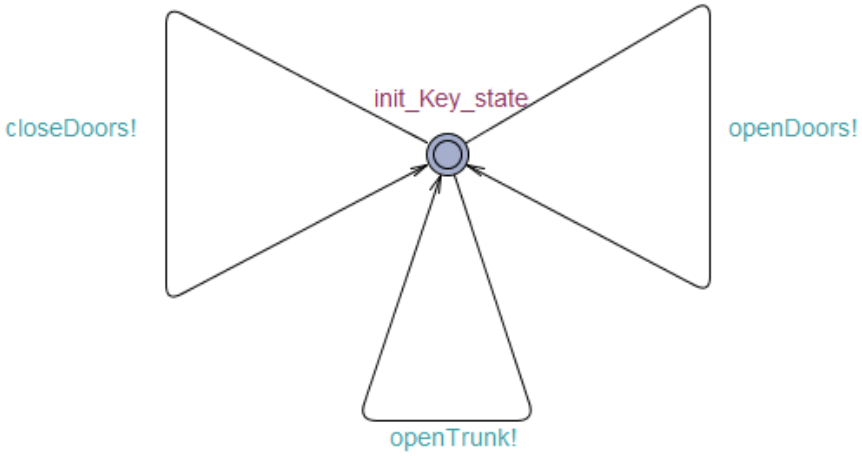
Prvním krokem návrhu je modelace systému v prostředí UPPAALu.

5.3.1 Model klíče

Klíč se skládá ze tří tlačítek, kde prvním tlačítkem lze uzamknout celý automobil, druhým lze odemknout pouze dveře zavazadlového prostoru a třetím odemknout všechny dveře.

Vycházíme z předpokladu, že výchozím stavem je klidový stav, ve kterém není zmáčknuté žádné tlačítko. Stiskem jednoho z výše uvedených tlačítek

Obrázek 5.1: Modelovaný klíč



Obrázek 5.2: Návrh klíče UPPAAL

vyvoláme akci synchronizačního kanálu, kterou zachytí naslouchající na druhém konci kanálu (v našem případě šablona popisující stav zámků). Poté se opět vracíme do klidového stavu, ve kterém není stisknuto žádné tlačítko.

5.3.2 Model dveří

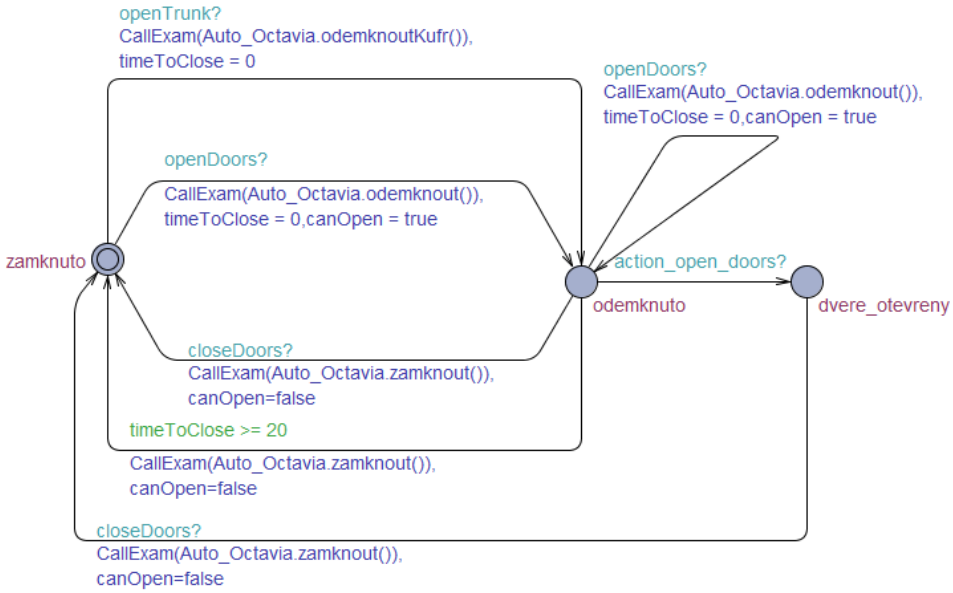
Dveře automobilu mají pouze dva stavy a to že jsou dveře otevřené, nebo uzavřené. S výhodou zde využijeme možnosti UPPAALu a to tím, že vytvoříme jedinou šablonu dveří, ze které vzniknou čtyři instance pro popis dveří automobilu.

Výchozím stavem dveří je, že jsou dveře uzamčené. Pokud jsme klíčem odemkli všechny dveře, nastavil se příznak možnosti otevřít dveře. Pokud dveře otevřeme ještě před expirací nastaveného času, vyvoláme akci synchronizačního kanálu, která je zachycena opět v šabloně popisující stav zámků.

Obrázek 5.3: Návrh dveří UPPAAL

5.3.3 Model zámků

Zámky automobilu mohou být buď ve stavu zamknuto, nebo ve stavu odemknuto. V odemknutém stavu se hlídá časová expirace při neotevření žádných dveří.



Obrázek 5.4: Návrh zámků UPPAAL

V zamknutém stavu čekáme na vyvolanou akci synchronizačního kanálu od klíče, kde buď přijde požadavek na odemknout pouze dveří zavazadlového prostoru, nebo požadavek na odemknutí všech dveří. Společně s odemknutím všech dveří nastavíme příznak pro možnost otevření dveří. V obou případech odemknutí se nastaví časovač do nuly a provede se příslušné zavolání metody v EXAMu.

V odemknutém stavu mohou nastat tři možnosti. Dojde k otevření některých dveří, přijde požadavek od klíče na uzamknutí všech dveří nebo dojde k časové expiraci neotevřením žádných dveří a automobil se zamkne.

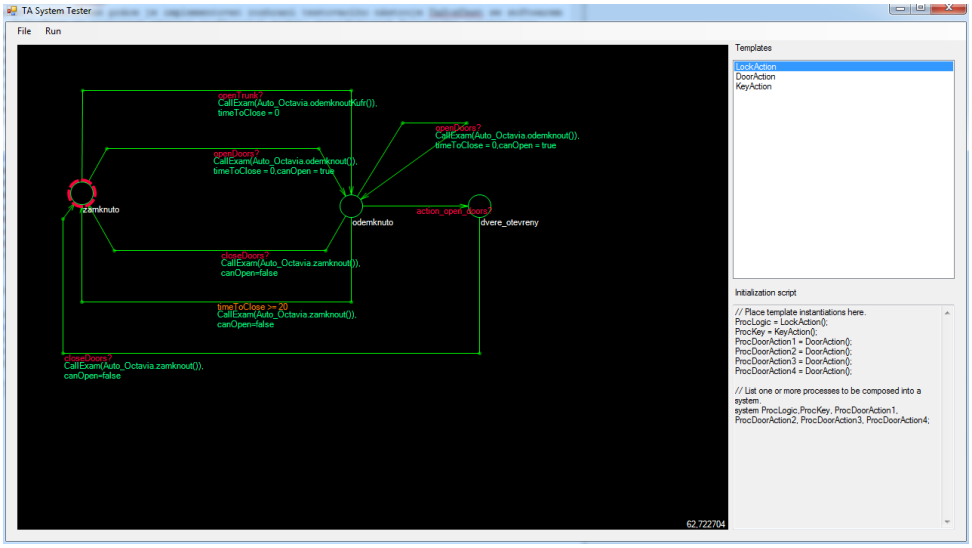
Otevřením alespoň jedněch dveří přejdeme do stavu, ve kterém je možnost zamknout automobil pouze požadavkem od klíče.

5.4 TaSysTest

Nyní již můžeme navržený model z UPPAALu nahrát do prostředí TaSysTestu, provést propojení s prostředím EXAMu a otestovat.

5.4.1 Načtení projektu

Provedení načtení modelu z prostředí UPPAALu provedeme v menu "File" stisknutím tlačítka "Open...". Před samotným zobrazením načteného projektu se provede syntaktická analýza kódu a upozorní nás na případné nesrovnalosti v právě načítaném modelu.



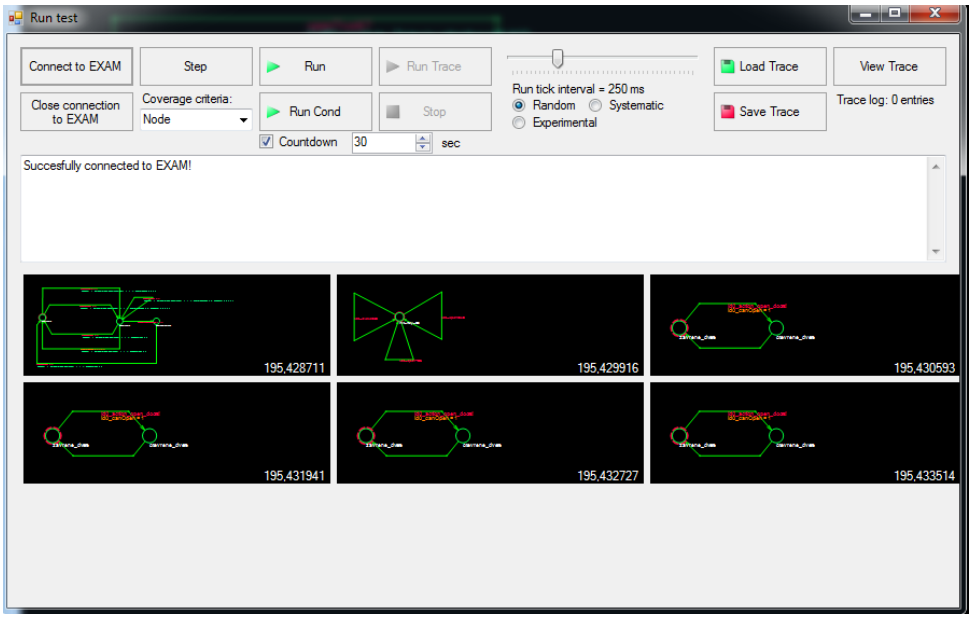
Obrázek 5.5: Načtený model v TaSysTestu

Po úspěšném načtení a provedení syntaktické analýzy již můžeme vidět interpretované šablony z UPPALLu, mezi kterými lze přepínat v seznamu na pravé straně programu. Pod tímto seznamem se nachází načtený inicializační skript, ve kterém je definováno, jaké instance se mají z těchto šablon vytvořit.

5.4.2 Spuštění testovacího prostředí

Testovací prostředí TaSysTestu lze spustit po otevření projektu v menu "Run" stisknutím tlačítka "Run".

TaSysTest dle definice inicializačního skriptu vytvoří instance z jednotlivých šablon a provede vnitřní inicializaci načteného projektu, ve které rozpozná jednotlivé hrany, stavy, proměnné, atd..



Obrázek 5.6: Testovací prostředí TaSysTestu

Ve spodní části lze vidět jednotlivé instance načtené ze šablon. V našem případě instanci odemykání zámků, systém klíče a čtyři instance dveří. Uprostřed se nachází výpis logu programu, do kterého se zapisuje aktuální dění v programu.

V horní části jsou možnosti ovládání testovací prostředí TaSysTestu.

Pokud je již spuštěna serverová část na straně EXAMu (viz. 5.5.4), lze se stisknutím tlačítka "Connect to EXAM" připojit k serveru a může probíhat komunikace a tedy i testování. V této fázi se také odešle serveru požadavek o odeslání názvů metod (jejich referenční a skutečný název, viz. 2), který po úspěšném přijetí je uložen na straně TaSysTestu, který dle požadované metody k vykonání jí interpretuje svými algoritmy tak, aby EXAM po přijetí požadavku na vykonání této metody jí již pouze vykonal. Po úspěšném připojení k serveru a načtení metod již můžeme postoupit k provádění testů. Tlačítkem Step lze provést jeden náhodně vybraný povolený krok.

Spuštění automatického testu, lze provést stiskem tlačítka "Run".

Spuštěním testu "Run Cond" a zvolením kritéria pokrytí lze provést test, který se ukončí po projití buď všech hran, nebo navštívením všech stavů.

U obou testů lze zvolit způsob výběru následující hrany k navštívení (*Random*, *Systematic*, *Experimental*). Rychlost kroku mezi dvěma stavy lze určit pomocí posuvníku. Zvolením "Countdown" a následným nastavením času společně se spuštěním jednoho z výše uvedených testů začne provádět odpočet času a po jeho expiraci se provede zastavení testu a ukončení spojení s EXAMem (viz. 3), který bude následně pokračovat dalším krokem ve svém sekvenčním testu.

Manuální ukončení spojení se provede stiskem tlačítka “Close connection to EXAM” (viz. 3).

Průběh testu je ukládán a lze si jej prohlédnout stiskem “View Trace”. Tento průchod je možné uložit a poté i opětovně načíst a spustit (více v [Gru14c]).

5.5 EXAM

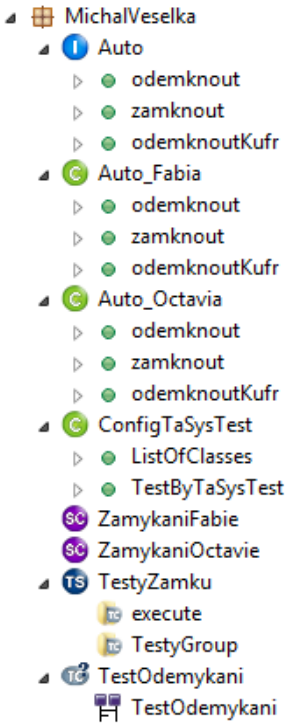
Posledním krokem je vytvoření testu, kterým ověříme komunikaci mezi TaSysTestem a EXAMem.

5.5.1 Tvorba modelu

Před tvorbou samotného testu je zapotřebí vytvořit model, na kterém budeme daný test provádět. Testujeme funkčnost zámků, proto budeme potřebovat metodu pro odemknutí všech zámků dveří, uzamknutí všech zámků dveří a odemknutí zámků zavazadlového prostoru.

Samotná realizace zavolané metody bude provedena výpisem do konzole o začátku provádění dané činnosti, prodlevou o délce 1 sekundy a následným výpisem informace do konzole o ukončení dané činnosti.

Vytvoříme proto rozhraní Auto, které bude obsahovat tři metody a to ‘odemknout’, ‘zamknout’ a ‘odemknoutKufr’. Následně vytvoříme třídy reprezentující samotný automobil vycházejícího z návrhového rozhraní ‘Auto’.



Obrázek 5.7: Struktura testu zámků

5.5.2 Tvorba komunikační třídy

Pro samotnou komunikaci s prostředím TaSysTestu je potřeba vytvoření nové třídy, kterou jsem nazval 'ConfigTaSysTest', obsahující metody 'ListOfClasses' a 'TestByTaSysTest'.

'ListOfClasses' je metoda vytvořena za účelem vytváření přehledného seznamu povolených tříd, se kterými může TaSysTest pracovat. Jedná se o slovník, který je indexovaný názvem dané třídy a hodnota daného indexu je referenční odkaz na danou třídu.

```
List["Auto_Fabia"] = "Auto Fabia"
List["Auto_Octavia"] = "Auto Octavia"
```

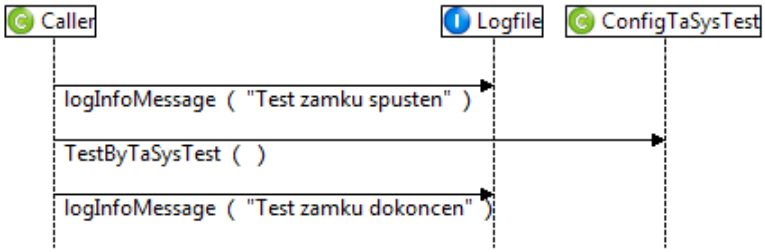
Obrázek 5.8: Struktura třídy ListOfClasses

'TestByTaSysTest' je metoda, ve které probíhá daná komunikace s prostředím TaSysTestu.

Prvním krokem je načtení seznamu povolených metod z 'ListOfClasses', načež se vytvoří serverová strana komunikace a vyčká se na připojení klienta (TaSysTestu). Po úspěšném připojení klienta se odešlou názvy povolených tříd, se kterými může TaSysTest pracovat a server čeká na příjem metod, které má vykonat. Po ukončení spojení na straně TaSysTestu je odeslán serveru požadavek na ukončení serverové strany, který následně provede a pokračuje dalším krokem sekvenčního diagramu právě prováděného testu.

5.5.3 Tvorba testu

Posledním krokem tvorby testu je vytvoření sekvenčního diagramu, ve kterém bude v daném kroku předáno řízení TaSysTestu.



Obrázek 5.9: Sekvenční diagram

Po zapsání krátké zprávy do konzole informující o spuštění testu, je předáno řízení prostředí TaSysTestu. Po ukončení řízení je opět zapsána krátká zpráva do konzole o ukončení testu.

5.5.4 Průběh testu

Abychom mohli daný sekvenční diagram spustit, je zapotřebí vytvořit TestSuite, ve kterém do nově vytvořené TestGroup přidáme výše vytvořený sekvenční

diagram.

Pro konfiguraci samotného testu vytvoříme *System Configuration*, ve kterém zvolíme parametry pro prováděný test.

Nyní již můžeme spustit vytvořenou *Test Suite* pro testování zamykání automobilu. Spustí se nám okno *Test runner perspective*, ve kterém zvolíme kterou *Test Suite* chceme spustit a následným výběrem *System Configuration* pro testovaný automobil test spustíme.

Vytvoří se server, který čeká na připojení klienta.

```
TestGroup
2016-05-21 14:14:33 Testrunner ------------------------------------------------------------------------------
2016-05-21 14:14:33 Testrunner ... current-test-name:         TestOdemykani
2016-05-21 14:14:33 Testrunner ... current-test-description:
2016-05-21 14:14:35 MainThread ... <<< INITIALIZING TEST CASE  >>> "EXAMples.MichalVeselka.TestOdemykani"_6._b._245f1368_1820_42d0_b136_f96cdede0fb6
2016-05-21 14:14:35 MainThread ... <<< ENTERING TEST SEQUENCE >>> "EXAMples.MichalVeselka.TestOdemykani"
2016-05-21 14:14:35 MainThread ... Current CallID = 00000000-0000-0000-0000-000000000000_L0
2016-05-21 14:14:35 MainThread ... PARAMETERIZATION: No parameter set loaded
2016-05-21 14:14:35 MainThread ... <<< ENTERING METHOD  >>> "_1_core.report.Logfile.logInfoMessage"
2016-05-21 14:14:35 MainThread ... <<< ENTERING METHOD  >>> "_1_core.report.details.PythonLogfile.logInfoMessage"
2016-05-21 14:14:35 MainThread ... message: Test zamku spusten
2016-05-21 14:14:35 MainThread ... <<< EXITING METHOD  >>> "_1_core.report.details.PythonLogfile.logInfoMessage"
2016-05-21 14:14:35 MainThread ... <<< EXITING METHOD  >>> "_1_core.report.Logfile.logInfoMessage"
2016-05-21 14:14:35 MainThread ... <<< ENTERING METHOD  >>> "EXAMples.MichalVeselka.ConfigTaSysTest.TestByTaSysTest"
2016-05-21 14:14:35 MainThread ... <<< ENTERING METHOD  >>> "EXAMples.MichalVeselka.ConfigTaSysTest.EnabledClasses"
2016-05-21 14:14:35 MainThread Start communication
2016-05-21 14:14:35 MainThread Waiting for client...
```

Obrázek 5.10: Spuštění serveru

Následným připojením ze strany TaSystestu se odešlou informace o povolených třídách a EXAM je připravený k vykonání přijatých metod.

```
2016-05-21 14:18:49 MainThread Odemykam kufr Octavie
2016-05-21 14:18:50 MainThread Kufr Octavie odemknut
2016-05-21 14:18:50 MainThread ... <<< EXITING METHOD  >>> "EXAMples.MichalVeselka.Auto_Octavia.odemknoutKufr"
2016-05-21 14:18:50 MainThread endOK
2016-05-21 14:18:51 MainThread Connected by
2016-05-21 14:18:51 MainThread ('127.0.0.1', 50418)
2016-05-21 14:18:51 MainThread Received request to exec method
2016-05-21 14:18:51 MainThread ... <<< ENTERING METHOD  >>> "EXAMples.MichalVeselka.Auto_Octavia.zamknout"
2016-05-21 14:18:51 MainThread Zamykam Octavii
2016-05-21 14:18:52 MainThread Octavia zamknuta
2016-05-21 14:18:52 MainThread ... <<< EXITING METHOD  >>> "EXAMples.MichalVeselka.Auto_Octavia.zamknout"
2016-05-21 14:18:52 MainThread endOK
2016-05-21 14:18:53 MainThread Connected by
2016-05-21 14:18:53 MainThread ('127.0.0.1', 50420)
2016-05-21 14:18:53 MainThread Received request to exec method
2016-05-21 14:18:53 MainThread ... <<< ENTERING METHOD  >>> "EXAMples.MichalVeselka.Auto_Octavia.odemknout"
2016-05-21 14:18:53 MainThread Odemykam Octavii
```

Obrázek 5.11: Vykonávání přijatých metod

Po přijetí požadavku na ukončení řízení se provede zastavení komunikace a vykonání následujících kroků sekvenčního diagramu testu.

```
2016-05-21 14:23:45 MainThread Received request to stop
2016-05-21 14:23:45 MainThread Code after end tcp comm
2016-05-21 14:23:45 MainThread DONE
2016-05-21 14:23:45 MainThread ... <<< EXITING METHOD  >>> "EXAMples.MichalVeselka.ConfigTaSysTest.TestByTaSysTest"
2016-05-21 14:23:45 MainThread ... <<< ENTERING METHOD  >>> "_1_core.report.Logfile.logInfoMessage"
2016-05-21 14:23:45 MainThread ... <<< ENTERING METHOD  >>> "_1_core.report.details.PythonLogfile.logInfoMessage"
2016-05-21 14:23:45 MainThread ... message: Test zamku dokoncen
2016-05-21 14:23:45 MainThread ... <<< EXITING METHOD  >>> "_1_core.report.details.PythonLogfile.logInfoMessage"
2016-05-21 14:23:45 MainThread ... <<< EXITING METHOD  >>> "_1_core.report.Logfile.logInfoMessage"
2016-05-21 14:23:45 MainThread ... <<< EXITING TEST FLOW >>> "EXAMples.MichalVeselka.TestOdemykani"
```

Obrázek 5.12: Ukončení řízení

Kapitola 6

Závěr

Cílem práce bylo vytvořit rozhraní mezi nástrojem TaSysTest a softwarem EXAM. Jako návrhová architektura komunikace mezi těmito dvěma prostředími byla zvolena síťová struktura klient/server umožňující komunikaci v rámci jednoho stroje, ale i možnost vzdálené komunikace. To zahrnovalo úpravu stávajícího kódu programu TaSysTest a přidáním metod umožňujících klientskou komunikaci a vytvořením třídy v EXAMu obsahující metody pro serverovou komunikaci a možností vykonání přijaté metody.

Pro otestování implementovaného rozhraní byl zvolen ukázkový test zamykacího systému automobilu zahrnující kontrolu systému pomocí klíčku, ovládání zámků pomocí přijatého požadavku od klíčku a dveře automobilu. Model navržený a vytvořený v prostředí UPPAALu byl následně načten a spuštěn v prostředí TaSysTestu, odkud po úspěšném připojení EXAMu byly formou automatického testu vysílány požadavky na vykonání odemknutí, či zamknutí příslušných zámků. Simulace činnosti zámků byla v prostředí EXAMu znázorněna výpisem prováděné akce do konzole. Komunikace mezi oběma prostředími probíhala dle očekávání a po ukončení řízení na straně TaSysTestu byla úspěšně provedena zbývající část sekvenčního diagramu testu na straně EXAMu.

Příloha A

Literatura

[Bea09] David Beazley, *Python essential reference*, Addison-Wesley, Upper Saddle River, NJ, 2009.

[Exa16] Exam, *About exam*, 2016.

[Gru14a] Tomáš Grus, *Implementace softwarového nástroje pro generování integračních testů*, 15–17.

[Gru14b] ———, *Implementace softwarového nástroje pro generování integračních testů*, 15.

[Gru14c] ———, *Implementace softwarového nástroje pro generování integračních testů*, 17.

[Mic16] MicroNova, *Nápověda distibuovaná s programem exam verze 4.5*, MicroNova AG, 2016.

[MSD16] Microsoft MSDN, *Tcpclient class*, 2016.

[PAG09] Ken PAGE, Alan. JOHNSTON, *Jak testuje software microsoft*, Brno: Computer Press, 2009.

[SS16] MicroNova Software and Systems, *Test automation*, 2016.

[Tes16] Testovanisoftwaru.cz, *Test case*, 2016.

[UPP15] UPPAAL, *Related tools*, 2015.

Příloha B

Obsah přiloženého CD

- TaSysTest

  Upravená verze TaSysTestu umožňující komunikaci se softwarem EXAM

- Balíček EXAM

  Balíček z prostředí EXAM obsahující rozhraní pro komunikaci s TaSysTestem. Dále balíček obsahuje i kompletní model použitý pro prováděný test

- Model zamykání dveří z UPPAALu

  Model zamykání dveří použitý v prováděném testu

- Bakalářskou práci

  Bakalářskou práci ve formátu pdf, včetně zdrojových souborů pro její tvorbu.