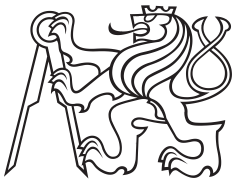
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra měření

Integrační testování metodou Model-Based Testing - případová studie

Bc. Michal Veselka

Vedoucí: Ing. Jan Sobotka, Ph.D.
Obor: Senzory a přístrojová technika
Studijní program: Kybernetika a robotika
Květen 2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Veselka** Jméno: **Michal** Osobní číslo: **393009**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Senzory a přístrojová technika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Integrační testování metodou Model-Based Testing - případová studie

Název diplomové práce anglicky:

Integration Testing by Model-Based Approach - Case Study

Pokyny pro vypracování:

1. Seznamte se s problematikou HIL integračního testování.
2. Vytvořte sadu modelů popisující základní komfortní subsystémy vozidla (zamykání, bezklíčové zapalování, vnější osvětlení, ovládání oken, …). Konkrétní podmnožinu modelovaných funkcí konzultujte s vedoucím a přizpůsobte aktuálně dostupnému hardwaru pro testování.
3. Proveďte validaci modelů pro ověření jejich věrohodnosti.
4. Navrhněte a realizujte sadu experimentů s cílem zhodnotit vhodnost MBT přístupu k integračnímu testování.
5. Experimenty proveďte pomocí softwarového nástroje Taster.
6. Výsledky vyhodnoťte a navrhněte případné úpravy.

Seznam doporučené literatury:

[1] J. Zander, I. Schieferdecker, and P.J. Mosterman: Model-Based Testing for Embedded Systems. Taylor & Francis, 2011.
[2] Johan Bengtsson, and Wang Yi: Timed Automata: Semantics, Algorithms and Tools. Lecture Notes in Computer Science. 2004.
[3] Kobza, Ondřej: Rozšíření testovacího nástroje Taster. Bakalářská práce ČVUT, Praha 2018.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Sobotka, Ph.D., katedra měření FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.10.2018** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce:
do konce letního semestru 2019/2020

Ing. Jan Sobotka, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání

Podpis studenta

Poděkování

Chtěl bych poděkovat vedoucímu své diplomové práce Ing. Janu Sobotkovi, Ph.D. 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, 24. května 2019

Podpis: .....

Abstrakt

Cílem této diplomové práce je ověřit možnosti integračního testování metodou Model-Based Testing (MBT). Práce se po úvodním seznámením s problematikou HIL (Hardware-in-the-Loop) testování zaměřuje na návrh modelů popisujících komfortní subsystémy vozidla a validaci jejich věrohodnosti. Následně je provedeno několik sad experimentů a jsou diskutovány jejich výsledky a vhodnost MBT přístupu k integračnímu testování. Závěr je zaměřen na zhodnocení přístupu MBT testování a je navrhnuto několik úprav stávajícího programu Taster pro zlepšení kvality integračního testování metodou MBT.

Klíčová slova: Taster, UPPAAL, Časované automaty, testování, integrační, Model-Based testování

Vedoucí: Ing. Jan Sobotka, Ph.D.

Abstract

The main goal of this diploma thesis is to evaluate integration testing performed by Model-Based Testing (MBT) approach. The thesis deals with the problematics of HIL (Hardware-in-the-Loop) testing. It focuses on designing models which describe comfort systems of the car and validation of their credibility. Consequently, it is done a few sets of experiments. The results and appropriateness of the MBT approach to the integration testing are discussed. In conclusion, it is evaluated overall approach MBT testing and is proposed a few changes of the Taster to the improved quality of the integration testing.

Keywords: Taster, UPPAAL, Timed automaton, testing, integration, Model-Based testing

Title translation: Integration Testing by Model-Based Approach - Case Study

Obsah

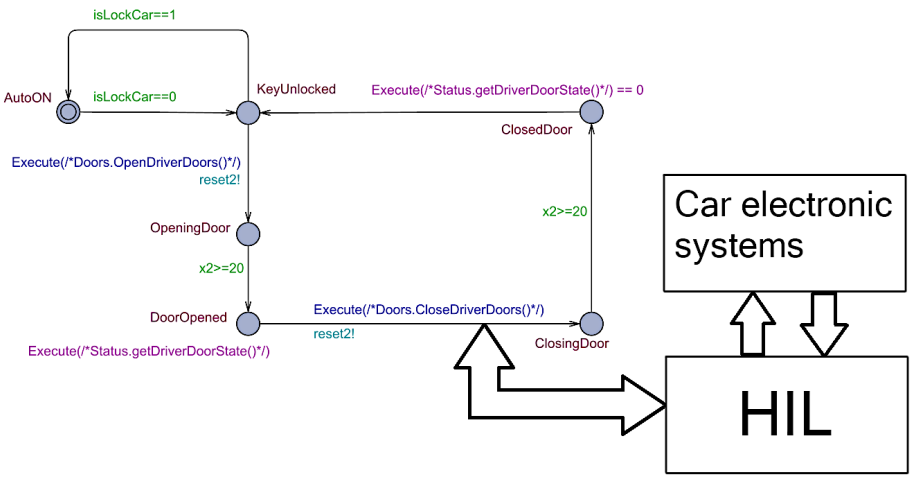
Obrázky

Kapitola 1

Úvod

V dnešní době, kdy je trh utvářen nabídkou a poptávkou, se jako nejdůležitější parametr při výrobě a následném prodeji ukazuje cena. Již není prakticky možné, aby jeden výrobce byl vývojářem, výrobcem i testerem všech dílů daného zařízení. Při výrobě automobilů to platí dvojnásob - každá součástka bývá vyrobena mnohdy jinou firmou. Problém může nastat ve chvíli, kdy se jednotlivé komponenty integrují do jednoho celku a mají spolu kooperovat. I přesto, že daná součástka může být otestována na jednotlivé vstupní a výstupní parametry, mohou se projevit nepředpokládané či skryté chyby během kooperace celého systému. Z toho důvodu je velice důležité provést kontrolu pomocí integračních testů, zahrnující testování chování složeného výrobku.

Tato diplomová práce se zaměřuje na využití stávajících softwarových nástrojů pro testování HIL (Hardware-in-the-Loop) a jejich doplnění o generování testů z modelů ve formě časovaných automatů. Rozšíření by mohlo přinést zcela nový způsob testování, případně stávající způsob do značné míry zjednodušit či zpřehlednit.



Obrázek 1.1: Testovací architektura

1.1 Použité názvosloví

Automotive je anglicko-německé prostředí a pro mnoho názvů tedy neexistuje adekvátní překlad do českého jazyka. Tyto výrazy nebudou překládány a v textu budou vyznačeny *kurzívou*.

1.2 Použitý software

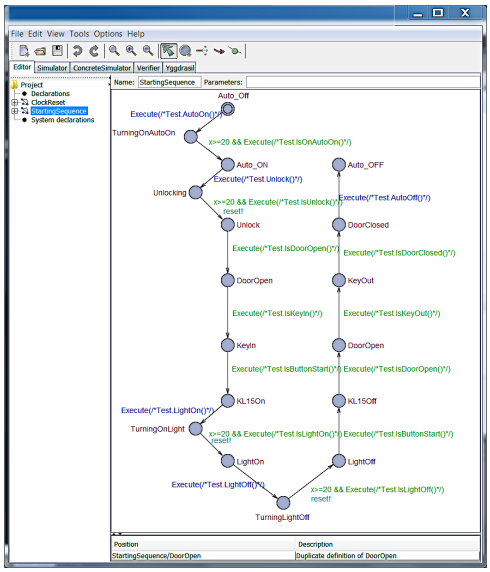
1.2.1 UPPAAL

Softwarové prostředí UPPAAL vzniklo spoluprací dvou univerzit (Upsala a Alborgh) [UPP06] a slouží k verifikaci systémů reálného času reprezentovaných časovanými automaty rozšířenými o číselné proměnné, strukturované datové typy a synchronizační kanály [Dep15].

Časovaný automat je grafem s podmínkami a resety nezáporných reálných hodin [Ale02]. Systém je modelován jako síť časovaných automatů. Přechod mezi těmito stavy modeluje chování systému. Ke kontrole, zda daný přechod může být uskutečněn, je možné doplnění o *guard*y a sychronizace. Synchronizace je mechanismus umožňující dvěma automatům přejít z jednoho stavu do druhého ve stejný čas.

Nástroj umožňuje provádět simulační kroky k ověření předpokládaného chování a pomocí *verifier*u lze zkoumat dosažitelnost jednotlivých stavů.

Výsledné modely včetně deklarací jsou uloženy v *.xml souboru, který je později načten nástrojem Taster.



Obrázek 1.2: Uppaal

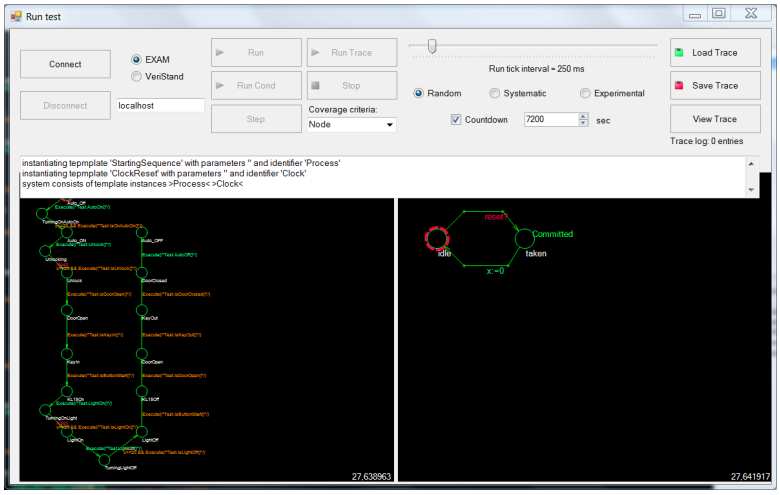
1.2.2 Taster

Taster je nástroj umožňující generování, provádění a vyhodnocování integračních testů z modelů v jazyce časovaných automatů.

Umožňuje dle vlastních algoritmů pracovat s modelem z UPPAALu, dekódovat všechny proměnné modelu (hrany, vrcholy, podmínky, ...) a spustit procházení modelu dle zadaných kritérií. [Kob18] uvádí, že průchod je založen na pseudonáhodné strategii, kde se vytvoří seznam hran z aktuálního stavu, splňující podmínku vstupu a vybere se náhodně z tohoto seznamu jedna hrana.

Průchod grafem je zaznamenáván *Trace logger*em a lze ho uložit, následně znovu načíst a projít dle uloženého průchodu.

Taster umožňuje navázat komunikaci a ovládat různé typy hardwaru pro HIL testování. Propojení s EXAMem probíhá na bázi TCP/IP spojení a odesílání zpráv. Tyti zprávy obsahují pokyny k provedení akcí a příjmu výsledků dotazů od EXAMu.



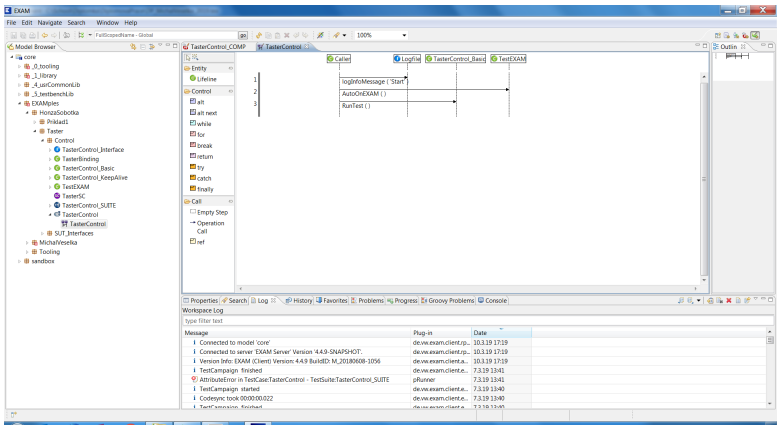
Obrázek 1.3: Taster

1.2.3 EXAM

Komunikace a testování automobilových systému HIL metodou jsou zajištěny programem EXAM, který kromě možnosti tvorby samostatných testů, obsahuje rozhraní pro ovládání testovacího stavu z prostředí Tasteru [Ves16].

EXtended Automation Method (EXAM) je dle [EXA11] jako komplexní metodika pro prezentaci, implementaci a hodnocení testovacích případů. Dále uvádí, že cílem programu EXAM je formálně popsat testovací případy v modelech jazyka UML tak, aby byly strojově interpretovatelné a částečně, nebo plně přenosné do spustitelných testovacích programů.

Primární úlohou EXAMu je pak provádění integračních testů implementovaných ve formě sekvenčních diagramů. V této diplomové práci slouží pouze jako "middleware" pro práci s hardwarem testovacího stavu. Předání řízení prováděného testu Tasteru je implementováno jako krok sekvenčního testu. Je při něm vytvořeno TCP/IP spojení a následné provádění přijmutých požadavků na vykonání z Tasteru, omezeného seznamem povolených tříd a metod, které mohou být v daném testu provedeny.



Obrázek 1.4: EXAM

Kapitola 2
Modelování systémů

V této kapitole bude diskutována tvorba modelů vybraných automobilových systémů, různé přístupy k této problematice a možnosti verifikace.

2.1 Model-Based testování

Model-based (modelově orientované) testování je softwarové testování, ve kterém je chování testovaného systému (System-Under-Test) ověřováno na základě modelu daného systému. Model popisuje chování systému a vzniká abstrakcí chování do vyšších vrstev. Chování lze popsat z hlediska vstupních sekvencí, akcí, podmínek, výstupu a toku dat ze vstupu na výstup. Měly by být prakticky srozumitelné a být znovu použitelné; musí mít přesný popis zkoušeného systému [Gur19].

Dle [Bel04] systém existuje a pracuje v čase a prostoru. Model je definován jako zjednodušující znázornění systému v určitém okamžiku v čase nebo prostoru a je určen k lepšímu pochopení skutečného systému. Dále dle [Bel04] probíhá simulace (testování) a manipulace s modelem způsobem, že pracuje s časem nebo prostorem pro jeho komprimaci. To umožňuje vnímat interakce, které by jinak nebyly patrné z důvodu jejich oddělení v čase nebo prostoru.

Výhody modelově založeného testování:

- Snadné vytvoření testů [Gur19]
- Úspora času při návrhu [Gur19]
- Včasná detekce chyb [Gur19]
- Poskytuje dobré pokrytí všech chování SUT [Mar06]
- Snížení nákladů a úsilí pro testování [Pre05]

Omezení použití MBT jsou pak dle [Sch13]:

- Nemůže zaručit nalezení všech rozdílů mezi modelem a implementací
- Vyžaduje velké zkušenosti tvůrce modelu

- Většinou (pouze) funkční testování
- Některé testy nelze snadno automatizovat

2.2 Vytvoření modelu okolí v prostředí UPPAAL

- **Model pokrývající všechny možné stavy**
  Každou akci, která je v daném modelu nabízena, lze teoreticky vykonat v kterýkoliv čas. Jen málokterá činnost vyžaduje sekvenci určitých předchozích kroků. Jako příklad poslouží spínač dveří zavazadlového prostoru u automobilu. Ten může být stisknutý v kterýkoliv okamžik: při uzamknutém automobilu, odemknutém, zavřených dveřích zavazadlového prostoru, otevřených, při nastartovaném automobilu, ... Jen občas bude mít stisknutí smysl.

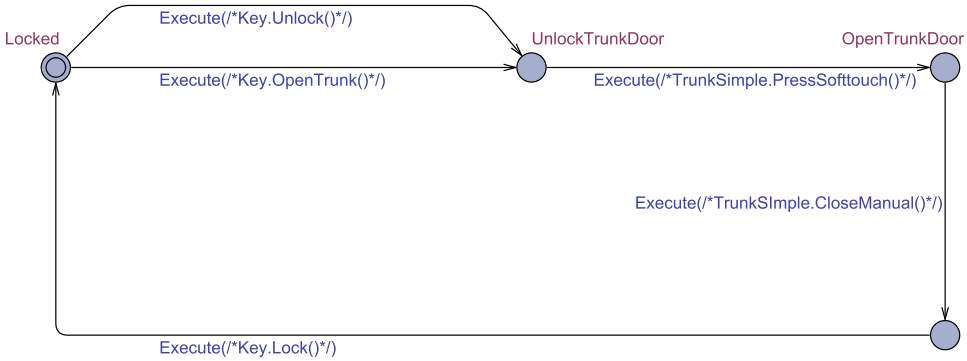
  Otázkou zůstává, jak tuto skutečnost implementovat do modelu a zda vůbec má modelování tohoto systému smysl. Tento model by znamenal, že danou akci lze vyvolat kdykoliv a model by se tedy sestával pouze z jednoho stavu. Z tohoto stavu by vedla přechodová hrana, na které by se daná akce vykonala a systém by se opět vrátil do stavu, kdy může potenciálně kdykoliv vyvolat akci znovu.

Obrázek 2.1: Model tlačítka dveří zavazadlového prostoru

  Výsledný systém by se sestavoval z velkého množství těchto "modelů", kde by cokoliv mohlo být vyvoláno kdykoliv.

  Tento přístup je podobný náhodnému generování vstupních proměnných a tedy nepřináší výhody použití modelů.

- **Sekvenční model**
  Z jiného úhlu pohledu lze předpokládat, že ne každá činnost je náhodná a tudíž, jí obvykle předchází sekvence jiných akcí. Jako příklad je uvedeno stisknutí spínače dveří zavazadlového prostoru. S největší pravděpodobností bude stisknutí spínače předcházet odemknutí automobilu, a to ať už formou odemknutí všech dveří, případně pouze dveří zavazadlového prostoru.



Obrázek 2.2: Sekvenční model spínače dveří zavazadlového prostoru

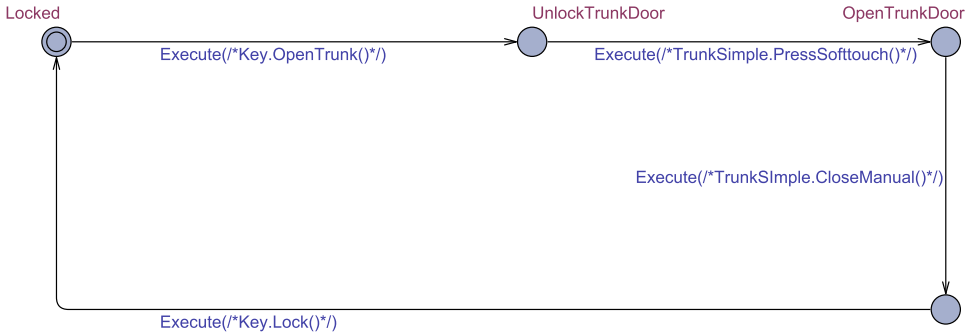
Tento model již nereflektuje plně realitu, a tudíž možnost stisknutí spínače kdykoliv. Nicméně z hlediska testování přináší možnost vytvořit model sekvencí událostí, které budou v drtivé většině případů po sobě vždy následovat. Testování nepokrytých možností (například při uzamknutém automobilu) by bylo časově náročné a je tedy efektivnější brát v úvahu, že je zámek dveří blokován mechanicky, tedy uzamknutím. Za těchto okolností nemůže nastat otevření dveří zavazadlového prostoru.

Jak je patrné z vytvořeného modelu, lze velice snadno pokrýt tímto přístupem dva možné druhy odemknutí dveří zavazadlového prostoru. Ty mohou být volány se stejnou pravděpodobností, případně využít možnost nástroje Taster. Ten umožňuje pokrytí průchodu všech hran, a tedy ověření obou možností odemykání.

Pokud bychom chtěli stejnou situaci pokrýt pomocí sekvenčního testu v EXAMu, museli bychom vytvořit smyčku a použít v určité sekvenci rozhodovací funkci a zároveň hlídat pokrytí již otestovaných částí. Složitost testu by poté mnohonásobně narůstala pro více takových možných částí testu, ve kterých by nebyla pevně daná sekvence, ale více možností průchodu.

- **Model scénáře chování uživatele**
  Posledním uvažovaným přístupem modelování může být vytvoření modelu který reflektuje jednotlivé akce uživatele a je tedy plně deterministický. Jedná se o dané sekvence kroků testů a modelují se na základě předpokládaného chování uživatele.

  Při zvolení tohoto přístupu nezískáme oproti implementaci v EXAMu mnoho výhod. Test probíhá v pevně dané sekvenci kroků za sebou. Tato skutečnost se neliší od sekvenčního testu, případně kódu, který je také vykonáván v předem dané souslednosti. Nejsou tedy aplikovány výhody MBT.



Obrázek 2.3: Model chování uživatele tlačítka dveří zavazadlového prostoru

2.3 Verifikace modelů

Verifikace je definována jako ověřování modelu pro stanovení chyb modelování [Car02]. Dále [Car02] definuje verifikace jako procesy a techniky, které vývojář modelu používá k zajištění, že jeho model je správný a odpovídá všem dohodnutým specifikacím a předpokladům.

Prokázání chyby v modelu nemusí být vždy úplně snadné a existují různé techniky, jak je lze najít.

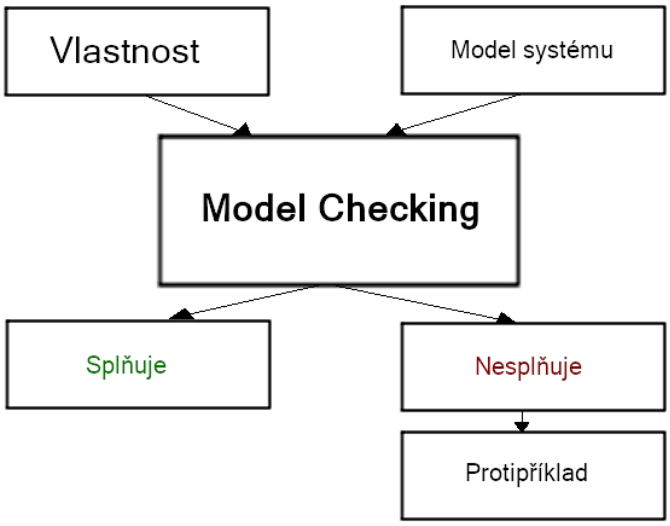
Mezi ně patří:

- Testování [Man05] představuje tradiční přístup, který detekuje chyby pomocí reálného systému s testovacími případy. Nalezená chyba se musí posoudit, zda je v chování systému, nebo modelu.
- Formální metody, např. kontrola modelu (*Model Checking*) [Cla01], který je více matematický přístup a může zajistit absenci chyb v modelovém návrhu. Kontrola spočívá v systematickém procházení matematického modelu a rozhodnutí, zda systém splňuje danou vlastnost, či nikoliv. Při nesplnění je ukázáno chování, které danou vlastnost porušuje (protipříklad).

Formální metodou lze verifikovat model přímo v UPPAALu a tradiční přístup lze aplikovat ve fázi testování.

2.3.1 Formální přístup

Verifikací modelu v UPPAALu lze zajistit pomocí *Verifier*u, používajícího dotazovací jazyk. Pro následující popis verifikací je nutné uvést co je to stavová formule a formule cesty.



Obrázek 2.4: Kontrola modelu - *Model Checking*

Stavová formule

Stavová formule je výraz, který může být vyhodnocen na stav bez ohledu na chování modelu [UPP06]. Jedná se o výrokovou logiku 1. řádu. Pokud výrok platí, je vyhodnocen jako pravda, pokud neplatí, pak je označen jako nepravda. Například to může být výraz $i > 3$, který je vyhodnocen jako pravda, kdykoliv je i větší než 3. Pokud je i menší nebo rovno 3, je výrok nepravda. Je možné otestovat, zda konkrétní proces je v určitém stavu.

Formule cesty

Jsou kvantifikonány jako cesty modelu. Lze je klasifikovat do dosažitelnosti, bezpečnosti a životnosti.

Vlastnosti stavu

- **Dosažitelnost stavu**
  Dosažitelnost stavu nám umožňuje ověřit, zda je možné v průběhu testování tento stav navštívit. Jinými slovy: existuje cesta z počátečního stavu taková, aby byl po cestě navštíven zkoumaný stav?

  V UPPAALu je syntaxe dotazu následující:

  $$E <> \ P.S$$

  P značí proces modelu a S je stav procesu P.

- **Bezpečnost stavu**
  Ověřujeme, zda nějaký stav nemůže nikdy nastat (například teplota

nikdy nepřekročí stanovenou hranici).

Syntaxe dotazu je následující:

$$A[]\ not\ (Px.Sx\ and\ Py.Sy)$$

Px značí proces modelu x, Py proces modelu y, Sx je stav procesu Px a Sy stav procesu Py.

- **Životnost stavu**
  Životnost stavu zkoumá, zda po nějakém stavu může nastat určitý zkoumaný stav. Jako příklad lze uvést, že po odeslání zprávy je očekáváno její přijetí.

  Syntaxe dotazu pro životnost:

  $$Px.Sx\ -->\ Py.Sy$$

  Px značí proces modelu x, Py proces modelu y, Sx je stav procesu Px a Sy stav procesu Py.

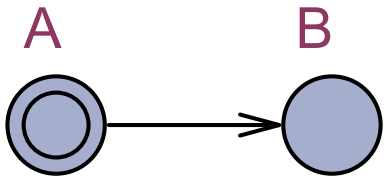
- ***Deadlock***
  Nastává v situaci, kdy k úspěšnému dokončení jedné akce je očekáváno předchozí dokončení druhé akce. V případě MBT také při čekání na synchronizační kanál, nebo splnění podmínek pro vstup do hrany.

  Nastane také v případě znemožněného pokračování v testování modelu. Dojde k němu, pokud nelze navštívit po určitém počtu kroků žádný další stav.

  UPPAAL umožňuje ověřit zda může nastat *deadlock* pomocí dotazu:

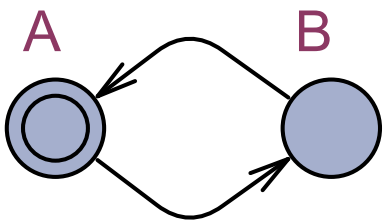
  $$A[]\ not\ deadlock$$

  .



Obrázek 2.5: Model s *Deadlock*em

Během navrhování modelů je dle požadavků na daný model vhodné zkoumat příslušné vlastnosti a jejich splnění. Můžeme tím odhalit případné skryté chyby návrhu.



Obrázek 2.6: Model bez *Deadlock*u

2.3.2 Tradiční přístup

Tradiční přístup vychází z techniky testování a hledání chyb. Je tak učiněno během testování vytvořeného modelu a zkoumání, zda se vše chová dle předpokladů. Metody tradičního přístupu umožňují najít chyby, ale již nezaručí, zda se daná chyba v modelu nevyskytuje. Může nastat situace, že je chyba v modelu, nebo naopak v SUT.

Kapitola 3
Validace modelů

Validace je popsána jako přezkoumání modelu vývojáři a lidmi znalými reálného systému [Car02]. Cílem je ověřit, jak navrhnutý model popisuje SUT.

Dle [Sar10] je definována konceptční validace modelu jako:

- Teorie a předpoklady, z nichž vychází koncept modelu, jsou správné a jsou testovány použitím diskrétní matematiky a statistických metod (například linearita, nebo nezávislost dat).
- Modelová reprezentace entity a struktury modelu, logiky, matematických a příčinných vztahů je odpovídající pro zamýšlený účel modelu.

Žádný systém nelze nikdy na 100 % verifikovat, nebo validovat. Každý model reprezentuje pouze aproximaci systému [Car02].

3.1 Navrhované metody validace

Následuje seznam možných metod pro softwarovou validaci modelu. Diskuse o vhodnosti daných metod bude provedena na konci seznamu.

- ***Antibugging*** [uoE00]
  Cílem antibuggingu je předejít chybám modelu, které by mohly nastat. *Antibugging* stojí na těchto zásadách:
  - Návrhnutí přehledného systému, ve kterém lze snadno vyčíst chování v každém stavu a jednoduše nalézt případnou chybu již ve fázi návrhu.
  - Zabránění šíření chyby do dalších částí modelu - tedy její co nejdřívější zachycení (například pomocí invariantů).

  Oproti tomu *Debugging* má za cíl odstranit chyby ve chvíli, kdy nastanou.

  Lze také čítat například kolik metod Taster požadoval a kolik jich EXAM skutečně vykonal. Tato čísla se musí rovnat, jinak by to znamenalo ztrátu požadované operace a následné nesrovnalosti testovaného modelu s reálným zařízením.

- **Jednokroková analýza** [uoE00]
  Předtím než je model spuštěn, je vhodné jednotlivé dílčí stavy modelu "odkrokovat", podobně jako program. To spočívá ve vyvolání dané akce, čekání na její výsledek a ověření, že skutečně proběhla, případně zda jsme přijmuli adekvátní návratovou hodnotu.

  Za nespornou výhodu lze považovat deterministické chování v každé časti testovaného modelu, a tedy i okamžité zachycení případné chyby. Jako nevýhoda se naopak ukazuje velká časová náročnost a potřebná znalost očekávaného chování celého modelu.

  Jako další možnost se nabízí model předat další osobě, která se na danou problematiku zaměří jiným pohledem. Tím může odhalit chyby, či potenciálně slabá místa, která při navrhování mohla být přehlédnuta.

- **Zjednodušení modelu** [uoE00]
  Před přistoupením k testování komplexního modelu, je dobré zjednodušit model na menší dílčí části, které se otestují jednotlivě a nezávisle na sobě. Na základě toho lze odhalit mnoho chyb, které by se v komplexnějším modelu hledaly mnohem hůře, či by se nepodařily identifikovat vůbec. Jednotlivé zjednodušené modely mohou proběhnout zcela korektně, ale není zde záruka, že zpětně sestavený komplexní model bude také korektní. Na druhou stranu chyba v jednoduchém modelu znamená, že tato chyba nastane i ve výsledném modelu.

- **Deterministické modely** [uoE00]
  Pokud je v nějaké části modelu použito generování náhodné hodnoty, je pro účely validace lepší nahradit všechny náhodné veličiny konkrétními. Díky tomu se model stane deterministický a až po ověření korektnosti lze postupně nahrazovat dosazené veličiny opět náhodnými.

- ***Tracing*** [uoE00]
  Sledováním (*Tracing*) běhu modelu lze dobře zpětně odhalit nesprávná místa, místa s neočekávaným chováním či chybou. Tato místa lze zpětně dohledat. Ať už se jednalo o to, jaká sekvence událostí tomu přecházela, snáze se učiní opatření (změny) modelu vedoucí k opravě. Je to vhodné pro případ, že chyba, která nastala může být rozpoznatelná a časově identifikovatelná. Poté lze provést její zpětné dohledání.

- ***Seed independence*** [uoE00]
  Jedná se o metodu zanášející náhodné malé změny do modelu a sledující změny na výstupu. Pokud model při malé změně na vstupu produkuje velké změny na výstupu, může to značit, že se někde skrývá návrhová chyba v modelu.

- **Test kontinuity** [uoE00]
  Při tomto druhu testování vycházíme z předpokladu, že abstraktní vrstva modelu může být popsána jako funkce vstupních proměnných na výstup.

U většiny funkcí by měl být výstupní průběh spojitý. Na základě toho nepředpokládáme, že malá změna na vstupu způsobí velkou změnu na výstupu.

- **Degenerační test** [uoE00]
  Tento typ testu spočívá v nastavení krajních hodnot vstupních proměnných modelu v povoleném rozsahu. Nastavením probíhá kontrola, že systém dokáže pracovat i s těmito hodnotami. Krajní hodnoty nepředstavují typické hodnoty, které model bude mít po většinu času. Díky jejich nastavení se můžou objevit chyby, které by při standardních hodnotách mohly zůstat skryty.

- **Konzistenční test** [uoE00]
  Vycházíme z předpokladu, že konzistentní model je takový, kdy při podobném zatížení dostaneme podobné průběhy. Tohoto poznatku využíváme ke kontrole toho, že model poskytuje podobné výsledky pro podobné vstupní hodnoty, které mají podobný dopad.

- **Mutantní analýza** [MV13]
  Princip analýzy spočívá ve vytvoření mutantního testovacího modelu, který je odvozen z originálního modelu a zahrnuje modelovou chybu. Předpokladem je, že takto zanesená chyba bude mít za následek nesprávné chování/vyhodnocení modelu a ten by měl při správném návrhu skončit chybou. V případě modelů časovaných automatů lze zanesenou chybu odhalit na invariantech stavů, kdy zanesená chyba způsobí nesplnění invariantu, a tedy selhání testu.

3.2 Diskuse nad navrhovanými metodami validace

Validační metody uvedené v 3.1 lze rozdělit na návrhové a testovací. Návrhové vystihuje pravidlo, že jednodušší a dobře čitelný model má mnohem menší pravděpodobnost chyby. U testovacích metod je u menších a determinističtějších modelu chyba snáze detekovatelná.

Navrhovaná validace testovaných modelů tedy bude probíhat v následujících krocích a při zjištění jakékoliv chyby se provede nápravné opatření. Postupuje se opět od bodu, ve kterém se zjistila chyba (v nižším stupni proběhla validace v pořádku, proto není důvod se vracet), dokud není seznam u konce.

1. **Otestování jednotlivých metod na jednoduchém modelu**
   Tento model může mít jenom dva stavy, kde přechodem z prvního do druhého stavu je vyvolána požadovaná akce.

   Pokud není očekávaná návratová hodnota, tak pouze proběhne kontrola provedení požadované akce. Při očekávané návratové hodnotě je provedena kontrola přijetí a její hodnoty.

Tímto lze ověřit primární funkčnost jednotlivých dílčích metod, ze kterých bude následně vystavěn celý model.

2. **Jednokroková analýza sestaveného modelu**
   Po vytvoření modelu je proveden test "odkrokováním", při kterém je podobně jako v předchozím kroku kontrolována provedenost akce, případně návratové hodnoty, s tím rozdílem, že test probíhá na navrhnutém modelu. Krokováním lze sledovat návaznost jednotlivých akcí na sebe a jejich případné vzájemné ovlivňování/nezávislost a chyby z toho vznikající.

3. **Spuštění automatického testu a *Tracing***
   Následuje spuštění automatického testu. Taster provede v každém svém kroku výběr akce z dostupných, aktualizuje tento seznam a následné opětovné vybrání.

   Mohou se zde projevit chyby způsobené brzskou/pozdní návazností na předchozí akci, případně pomalé odezvy, které nebylo možné rozpoznat pomocí jednokrokové analýzy.

   V případě výskytu chyby je test zastaven a díky *Tracing*u lze dohledat souslednosti, které k ní vedly.

4. **Opakování automatického testu**
   Automatický test je spuštěn několikrát po sobě. Zkoumá se tím konzistence systému a zda při některém spuštěném testu neprobíhají velké odlišnosti, které by mohly vést k nečekaným výsledkům testu.

5. **Zhodnocení testu a jeho shoda s reálným systémem**
   Na závěr je zhodnoceno, zda proběhnuté testy odpovídají možnému průběhu, který by mohl na daném zařízení nastat. Po vyhodnocení výsledků chování daného modelu je model prohlášen za validní, případně se provedou potřebné změny, vedoucí k jeho konečné validaci.

Kapitola 4

Model bezklíčkového přístupu - Kessy systém

Tato část se zaměří na tvorbu a popis modelu *Keyless Entry, Start and exit SYstem* (Kessy). Bude diskutována proveditelnost a shoda s reálným systémem.

4.1 Kessy systém

Kessy systém je bezklíčkové zapalování automobilu. Umožňuje uživateli pouze za přítomností klíčku odemknout, uzamknout anebo nastartovat automobil. Výhody a nevýhody tohoto systému nejsou předmětem této diplomové práce.

Dnes již je tento systém stále běžnější a najdeme ho ve stále více nově vyrobených automobilech (také již v levějších vozech). Kessy systém obsahuje mnoho funkcionalit a tím se stává poměrně komplexním. To přináší mnoho výzev na správnou funkčnost a možné chyby, které je potřeba již během testování podchytit.

Při modelování vycházíme z předpokladu vypnutého zapalování, uzamknutého vozidla a nepřítomností obou klíčků. Tato skutečnost je reflektována v inicializační sekvenci v EXAMu. Ta probíhá před spuštěním samotného testovacího modelu.

4.2 Fyzické provedení

Systém je složen z komponent různého druhu:

- **Kapacitní senzor na klice dveří**
  V přítomnosti klíčku umožňuje uzamknutí automobilu.
- **Klika dveří**
  Chycením za kliku v přítomnosti klíčku se odemknou zámky automobilu.

- **Antény**
  Systém má několik antén sloužících k detekci klíčku na různých místech. Pracují na dvou RFID frekvencích a UHF. Vnější umožňují příjem signálu dálkového odemykání/zamykání zámků dveří. Vnitřní pak mají v zavazadlovém prostoru funkci blokace uzamknutí pro případ přítomnosti klíčku v tomto prostoru. U řidiče pak umožňuje zapnutí zapalování a start motoru.

- **Klíček**
  Uživatel má dva klíče, kde každý z nich má v sobě unikátní identifikátor. Klíčkem lze buď dálkově odemknout, uzamknout, případně delším stiskem tlačítka kufru otevřít dveře zavazadlového prostoru.

  Za přítomnosti klíčku blízko automobilu je umožněno odemknutí/uzamknutí automobilu. Při přítomnosti klíčku u dveří řidiče je zatažením za kliku možné celý automobil odemknout, případně stisknutím kapacitního snímače automobil uzamknout. Obdobně u dveří spolujezdce.

  Poslední funkcí je bezklíčkové zapalování, kdy pouze přítomnost klíčku v autě umožní pomocí tlačítka nastartovat automobil.

- **Zámky automobilu**
  Uzamykají příslušné dveře

- **Dveře zavazadlového prostoru**
  Mohou být ve dvou provedeních

  - **Manuální otevírání**
    Pouze s tlačítkem pro uvolnění zámku
  - **S elektrickým pohonem**
    Obsahující kufrovou řídící jednotku - nebude modelováno

4.3 Výsledný model systému Kessy

4.3.1 Příprava na straně EXAMu

EXAM je strukturovaný na třídy a metody se vstupními parametry, které po namapování lze volat přímo nástrojem Taster. Jedná se o slovník obsahující klíč-hodnota, kde klíčem je název funkce v Tasteru a hodnotou třída a metoda v EXAMu. Jsou tedy dva seznamy: povolené třídy a povolené metody. Prvním krokem je tedy mapování již existujících tříd a metod do slovníků.

Některé funkce bylo nutné upravit, nebo vytvořit vlastní, vhodnější pro dané použití. Vyzkoušením funkce na vytvořeném malém modelu v UPPAALu se poté ověřila funkčnost volání a provedení akce.

```
# Class Key.
methodAliases['Lock'] = 'LockEXAM'
methodAliases['Unlock'] = 'UnlockEXAM'
methodAliases['OpenTrunk'] = 'OpenTrunkEXAM'
```

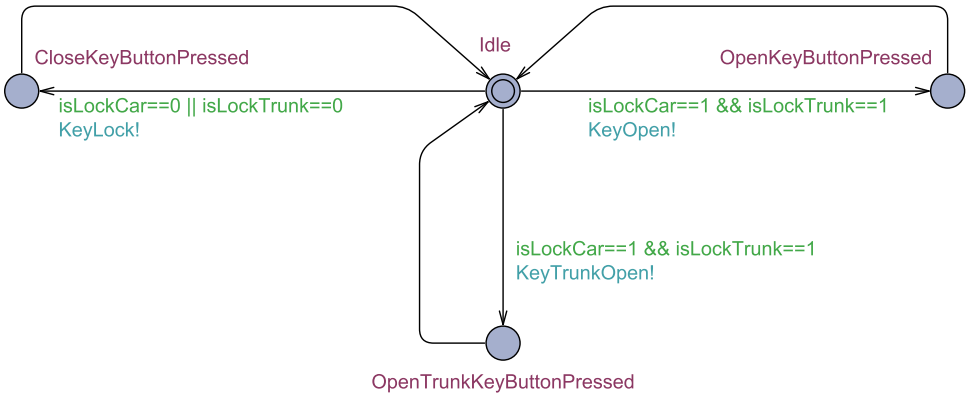
Obrázek 4.1: Slovník klíč-hodnota pro Taster

4.3.2 Navržený model

Celkový model Kessy systému se skládá z několika menších modelů, které jsou uspořádány do logických celků, dle své funkce.

Skládá se z:

- **Model klíčku**
  Klíček (dálkové ovládání odemykání automobilu) obsahuje tři tlačítka.



Obrázek 4.2: Model klíčku

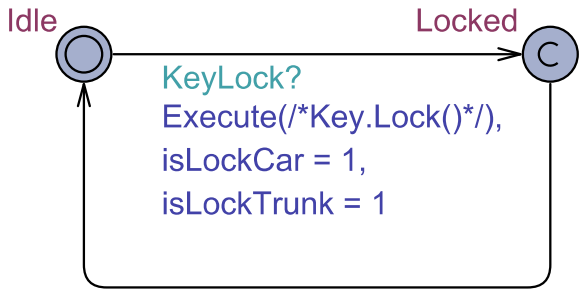
1. **Dálkové odemykání**
   Přechodová hrana navázána na tuto událost provede vyvolání synchronizace s jinou částí modelovaného systému. V té je vyvolána metoda v EXAMu, která provede krátké stisknutí tlačítka a odemknou se zámky všech dveří automobilu. Při neotevření žádných dveří do cca 20 sekund se auto opět uzamkne (zahrnuto v modelu zámku dveří). Delší stisknutí a držení tlačítka není implementováno, tudíž otevření všech oken automobilu při tomto stisku není uvažováno.
2. **Dálkové zamykání**
   Aktivace synchronizačního kanálu, ve kterém je vyvolán krátký stisk tlačítka klíče pro zamykání automobilu. Obdobně jako u dálkového odemykání není jeho delší stisk implementován, takže dálkové zavírání oken není uvažováno.
3. **Dálkové odemykání dveří zavazadlového prostoru**
   Na této přechodové hraně, po dostatečně dlouhém stisknutí tlačítka

otevření dveří zavazadlového prostoru, je vyvolána synchronizace. Ta způsobí jeho odemknutí, které proběhne v jiném modelu. Po této akci je zámek kufru uvolněn a dveře se pootevřou a lze je poté stisknutím spínače dveří otevřít.

K jednotlivým akcím jsou navázány synchronizační kanály, které vyvolají změnu stavu v dalším modelech (viz model zamykání dveří, model odemykání dveří a model zámku dveří zavazadlového prostoru).

Proměnné "isLockCar"a "isLockTrunk"jsou vnitřní pomocné proměnné modelu, uchovávající informaci o aktuálním stavu zámků. V tomto modelu slouží jako prevence k několikanásobnému vyvolání stejné akce, např. odemknutí již odemknutých zámků.

- **Model zamykání dveří**



Obrázek 4.3: Model zamykání

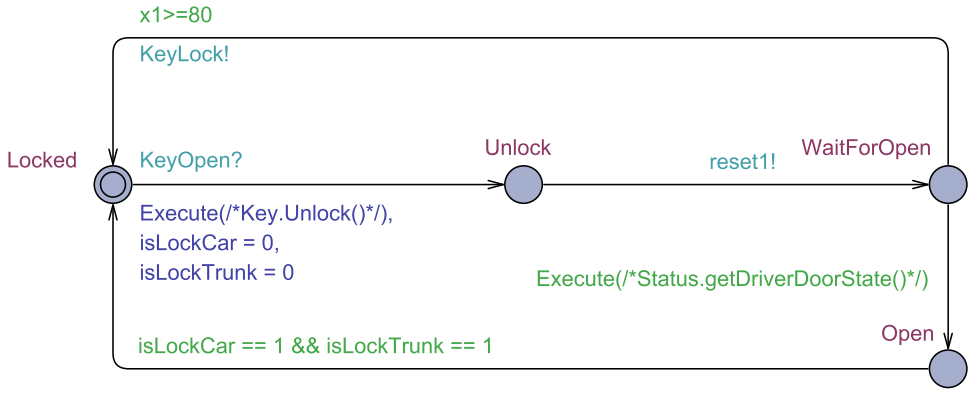
Synchronizační kanál je vyvolán stisknutím tlačítka uzamknout na dálkovém ovládání zámků automobilu. Proběhne poslání příkazu na uzamknutí dveří a nastavení proměnných.

Stav "Locked"je podmíněn co nejdřívějším opuštěním. Tím dojde k přechodu do stavu čekání na příjem ze synchronizačního kanálu.

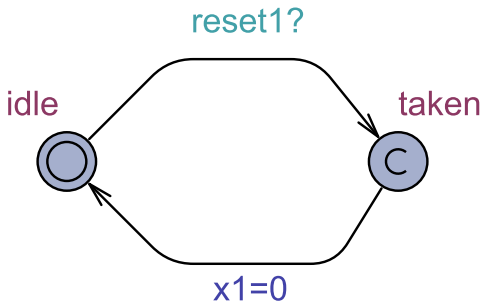
- **Model odemykání dveří**

Podobně jako v modelu zamykání zde najdeme příjem synchronizačního kanálu pro odemknutí a nastavení proměnných. Následně je vynulován časovač (viz model časovače) a čeká se 80 hodinových taktů (při 250 ms/takt = 20 sekund) na otevření dveří řidiče. Pokud dojde během této doby k jejich otevření, model přejde do stavu "Open"a čeká na uzamknutí vozu v jiné části modelu. Neproběhne-li otevření, zámky se uzamknou. V modelu jsou uvažovány pouze dveře řidiče, i když otevření jakýchkoliv dveří by mělo mít stejný důsledek.

- **Model časovače [Ger09]**
  Po vyvolání synchronizačního kanálu vyvolá na následující hraně nulování čítače. Čítaní poté probíhá v Tasteru pomocí hodinových taktů.



Obrázek 4.4: Model odemykání



Obrázek 4.5: Model časovače

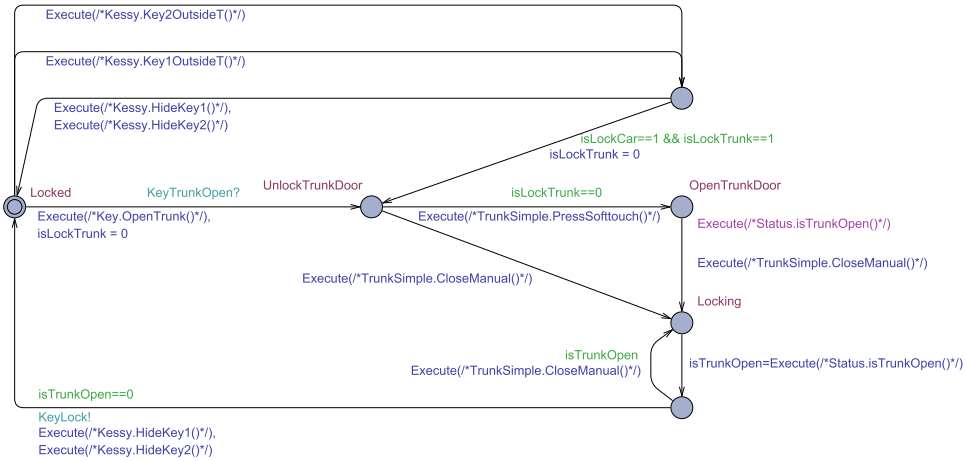
- **Model zamykání dveří zavazadlového prostoru**

  V modelu je uvažováno otevírání pomocí dálkového ovladače, případně nastavení přítomnosti klíčku poblíž dveří zavazadlového prostoru. Po rozhodnutí, která akce proběhla, dojde k otevření dveří zavazadlového prostoru. Následně k uzavření a uzamknutí, případně okamžité uzavření pootevřených dveří.

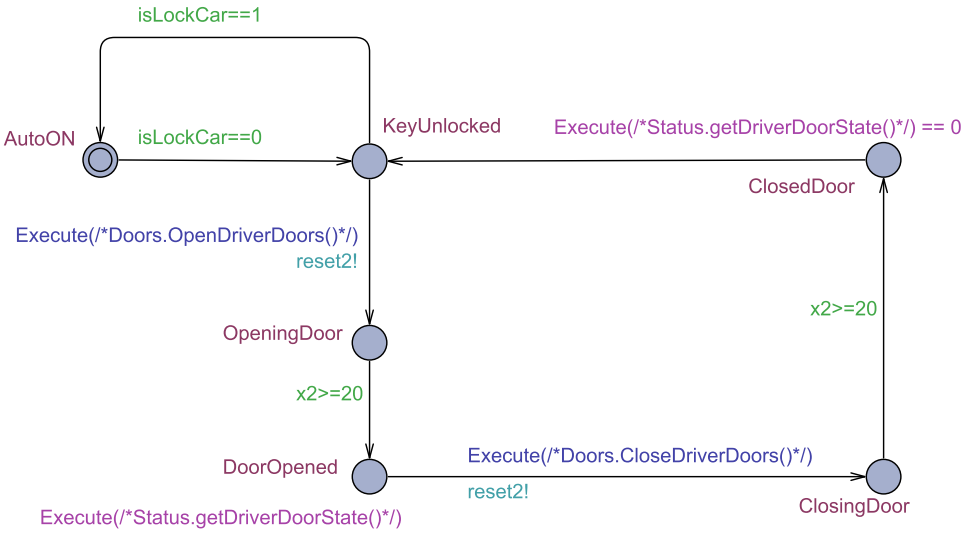
  Smyčka ověřující uzavření těchto dveří je implementována z důvodu nedeterministické chyby na straně HILu, která neprovede vždy uzavření dveří. Opakováním smyčky již k uzavření dojde.

  Následuje uzamknutí zámků a ztráta přítomnosti klíčku. Kontroluje se také zda klíček nezůstal uvnitř zavazadlového prostoru.

- **Model otevírání dveří**
  Model umožňuje při odemknutém vozidle otevřít dveře u řidiče. Po vyvolání požadavku na otevření těchto dveří je vyčkáno 20 hodinových taktů (při 250 ms/takt = 5 sekund) a invariantem se kontroluje, zda k otevření skutečně došlo.



Obrázek 4.6: Model zamykání dveří zavazadlového prostoru



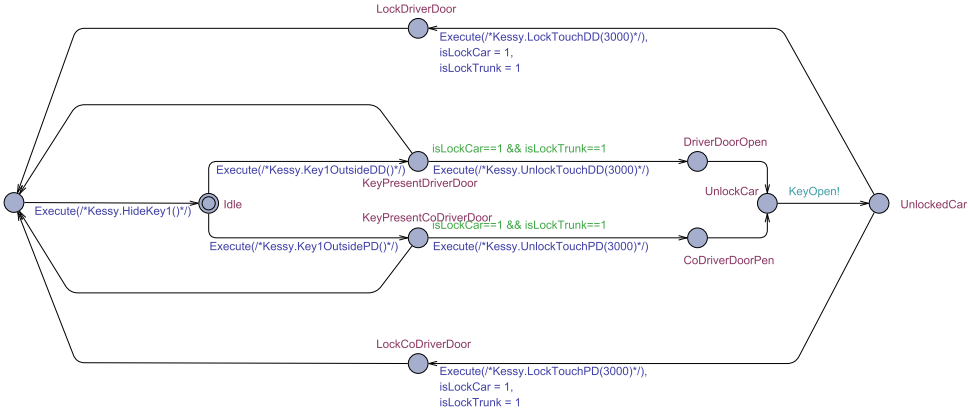
Obrázek 4.7: Model otevírání dveří

Obdobně při uzavírání dveří.

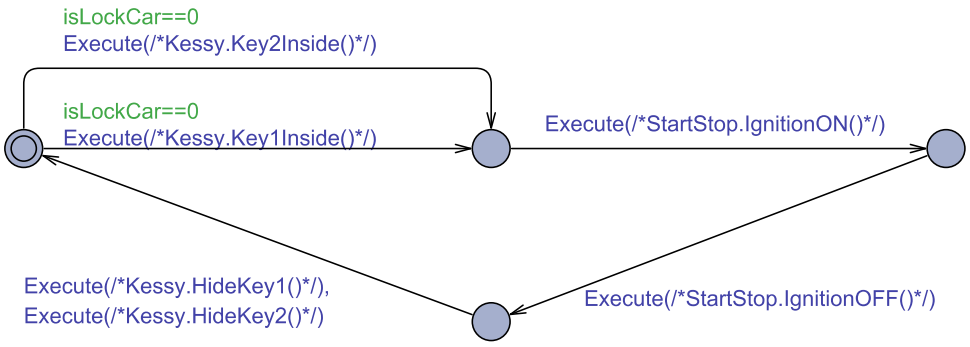
- **Model bezklíčkového odemykání**
  V modelu je prováděno nastavení přítomnosti klíčku, odebrání přítomnosti klíčku, odemykání přes kliku a zamykání kapacitním snímačem na klice. Pokud je klíč v přítomnosti dveří řidiče, je možné otevřít tyto dveře a automobil se odemkne. Obdobně pro dveře u spolujezdce. Stisknutím kapacitního senzoru lze uzamknout vozidlo a nastavit nepřítomnost klíčku u dveří.

  Standardně existují dva klíčky k vozidlu, tudíž stejný model se opakuje i pro klíček č. 2.

- **Model zapalování vozidla**



Obrázek 4.8: Model bezklíčkového odemykání



Obrázek 4.9: Model zapalování vozidla

Model nastaví přítomnost klíčku č. 1, nebo č.2 uvnitř vozidla a zapne zapalování. V následném kroku ho vypne a nastaví nepřítomnost klíčku uvnitř vozidla.

4.4 Výsledky

Model bezklíčkového přístupu (KESSY systém) se rozvinul do poměrně velkých rozměrů. Každá další drobná funkcionalita, která byla potřeba v testu pokrýt, způsobila vždy velké úpravy a zvětšení modelu.

Podařilo se pokrýt 22 z uvažovaných 29 stavů, a tedy model pokrývá ze 76 % reálný systém a ve stejném rozsahu byl i model otestován.

Model je koncipován jako model tlačítek/vstupů s předpokládaným chováním okolí, a tedy nejsou pokryty zcela náhodné stavy, jak bylo uvedeno v textu o návrhu modelů. Například možnost jakékoliv akce v kterýkoliv čas, tedy například stisknutí spínače otevírání kufru během libovolné fáze testu.

Výsledný model je už poměrně složitý model systému. Nicméně jeho namodelování může přinést zajímavé souběhy testovaných akcí, které by se sekvenčním testem pouze obtížně pokrývaly.

Model a pokrytí	Možné chování	Modelováno	Testováno
Model okolí (100 %)	Uzamknutí klíčem	Ano	Ano
	Odemknutí klíčem	Ano	Ano
	Odemknutí dveří zav. prostoru	Ano	Ano
	Bezklíčkové odemknutí (klíč č. 1 i č.2)	Ano	Ano
	Bezklíčkové uzamknutí (klíč č. 1 i č.2)	Ano	Ano
	Bezklíčkové odemknutí zav. prosotru (klíč č. 1 i č.2)	Ano	Ano
Model zámků (100 %)	Odemknutí	Ano	Ano
	Uzamknutí	Ano	Ano
	Odemknutí dveří zav. prostoru	Ano	Ano
Model dveří (40 %)	Otevření dveří řidiče	Ano	Ano
	Uzavření dveří řidiče	Ano	Ano
	Uzamknutí při neotevření dveří	Ano	Ano
	Stisknutí tlačítka dveří zav. prostoru	Ano	Ano
	Otevření ostatních dveří	Ne	Ne
	Uzavření ostatních dveří	Ne	Ne
	Otevření kapoty motoru	Ne	Ne
	Uzavření kapoty motoru	Ne	Ne
Model klíčku (100 %)	Přítomnost u dveří řidiče (klíč č. 1 i č.2)	Ano	Ano
	Přítomnost u dveří spolujezdce (klíč č. 1 i č.2)	Ano	Ano
	Přítomnost u dveří zav. prosotru (klíč č. 1 i č.2)	Ano	Ano
	Přítomnost uvnitř vozu (klíč č. 1 i č.2)	Ano	Ano
	Nastavení nepřítomnosti (klíč č. 1 i č.2)	Ano	Ano
Model zapalování (100 %)	Zapnutí zapalování	Ano	Ano
	Vypnutí zapalování	Ano	Ano
Model stahování oken (0 %)	Stažení pomocí klíčku	Ne	Ne
	Uzavření pomocí klíčku	Ne	Ne
Kontroly stavu (30 %)	Otevření dveří řidiče	Ano	Ano
	Otevření ostatních dveří	Ne	Ne
	Otevření dveří zav. prosotru	Ano	Ano

Tabulka 4.1: Pokrytí systému Kessy modelem

Kapitola 5

Model ovládání elektrických oken

Jako další model byl vytvořen systém ovládání elektrických oken, konkrétně ovládácí panel zabudovaný v řidičových dveřích. Panel umožňuje možnost obsluhy všech oken v automobilu.

5.1 Ovládání elektrických oken u řidiče

Uvažována varianta předpokládá elektrické ovládání všech oken. V testování vycházíme ze stavu, kdy je již provedená kalibrace koncových poloh oken. Ta nám umožňuje použití automatického stáhnutí a vytáhnutí oken do koncové polohy, a to bez nutnosti setrvalého držení tlačítka ovládání.

Dále model vychází z předpokladu odemknutého vozidla a sepnutého zapalování. Splnění umožňuje aktivaci ovládání stahování oken. Tato skutečnost je reflekrována v inicializační sekvenci v EXAMu. Ta probíhá před spuštěním samotného testování modelu.

5.2 Fyzické provedení

Ve dveřích řidiče je zabudované ovládání pro všechna okna v automobilu. Toto ovládání obsahuje čtyři pětipolohová tlačítka a tlačítko pro blokaci obsluhy zadních oken.

5.2.1 Ovládací panel ve dveřích řidiče

- **Tlačítko ovládání pohybu oken**
  Každé tlačítko umožňuje:
  - **Středová poloha**
    Jedná se o pozici, do které se po uvolnění tlačítko vrátí. Systém pro dané ovládání pohonu je v této situaci v nečinnosti.
  - **Manuální posun nahoru**
    Nadzvednutí tlačítka o jednu polohu uvádí systém do pohybu a okno je posouváno nahoru (uzavíráno). Uvolněním tlačítka dojde k

vrácení na středovou polohu a pozastavení okna v aktuální pozici.

Při plném uzavření okna nastane v systému deaktivace posunu nahoru, aby se zabránilo poškození systému přetížením.

- **Manuální posun dolů**
  Obdobně jako pro posun nahoru, tak při stisknutí tlačítka o jednu polohu dolů je systém uveden do pohybu a okno se posouvá dolů (otevíráno). Uvolněním tlačítka pohyb okna zastavíme. Při plném otevření se systém deaktivovuje pro další pohyb dolů.
- **Automatický dojezd do horní polohy**
  V případě nadzvednutí tlačítka o dvě polohy a je-li systém správně zkalibrován na koncové polohy, dojde k aktivaci automatického uzavření okna. Po krátkém uvedení tlačítka do druhé polohy jej uvolnit a okno zůstane v pohybu nahoru (uzavíráno), dokud nedojde k jeho plnému uzavření a systém se nezastaví.
- **Automatický dojezd do spodní polohy**
  Po stisknutí tlačítka o dvě polohy dolů je okno uvedeno do pohybu dolů (otevírání), až do plného otevření. Při plném otevření se další pohyb zastaví.

- **Blokační tlačítko**
  Na panelu ovládání blokační tlačítko umožňuje deaktivaci ovládání oken ve dveřích zadní části automobilu. Sepnutí tohoto tlačítka znemožní otevření oken ovladači v zadních dveřích (dětská pojistka) a lze s nimi manipulovat pouze pomocí ovládacího panelu od řidiče.

5.2.2 Kontrola polohy oken

Podmínkou určení správné polohy jednotlivých oken je kalibrace krajních poloh (plné otevření a plné uzavření). Kalibraci lze provést dle návodu výrobce, případně pomocí diagnostického softwaru.

Poloha okna je vyjádřena v procentech otevření, tedy plně uzavřené okno má 0 % a plně otevřené 100 %.

5.3 Modelování panelu ovládání elektrických oken u řidiče

5.3.1 Příprava na straně EXAMu

Pro testování oken byly v EXAMu namapovány dvě třídy, z nichž třída *WindowDriver* obsahuje metody pro ovládání oken a třída *Status* metody pro aktuální pozice oken.

Metody použité pro ovládání oken zahrnují možnost nastavení všech pěti poloh tlačítka. Jako příklad je uvedeno pouze ovládání okna u řidiče. Ostatní

```
allowedObjects['WindowDriver'] = 'TestWindowDriver'
allowedObjects['Status'] = 'TestStatus'
```

Obrázek 5.1: Třídy použité v modelu testování oken

tři tlačítka ovládání (okno spolujezdce a dvě zadní okna) mají podobnou kontrukci. Liší se použitou zkratou odkud probíhá ovládání okna (v tomto modelu vždy D - driver) a jaké okno ovládají (D - driver, P - passenger, BD - behind driver a BP - behind passenger). Například pro ovládání okna spolujezdce z pozice řidiče je použito zkratky DP.

```
#Class WindowDriver
methodAliases['OpenManDD'] = 'OpenManDD'
methodAliases['CloseManDD'] = 'CloseManDD'
methodAliases['OpenAutoDD'] = 'OpenAutoDD'
methodAliases['CloseAutoDD'] = 'CloseAutoDD'
methodAliases['NullDD'] = 'NullDD'
```

Obrázek 5.2: Použité metdy třídy WindowDriver pro ovládání oken

Třída WindowDriver obsahuje také blokaci ovládání zadních oken.

```
methodAliases['BlockRearWindowOn'] = 'BlockRearOnD'
methodAliases['BlockRearWindowOff'] = 'BlockRearOffD'
```

Obrázek 5.3: Použité metdy třídy WindowDriver pro blokaci ovládání oken

Třída *Status* obsahuje (kromě jiného) metody pro získávání aktuální pozice oken. Hodnota je v rozmení 0-100 % a značí procento otevření daného okna.

```
#Class Status
methodAliases['getWindowPosP'] = 'getWindowPosP'
methodAliases['getWindowPosD'] = 'getWindowPosD'
methodAliases['getWindowPosBP'] = 'getWindowPosBP'
methodAliases['getWindowPosBD'] = 'getWindowPosBD'
```

Obrázek 5.4: Použité metdy třídy Status pro zíkání aktuálních pozic oken

5.3.2 Navržený model

Model panelu ovládání oken v řidičových dveřích se skládá ze čtyřech ovládacích tlačítek. Každé pro jedno okno ve vozidle a tlačítkem blokace pro zadní okna.

Pro potřeby testování byl vytvořen model okolí, který vybere jeden testovací scénář:

- **Okno řidiče**
  - Manuální posouvání (první polohy tlačítka)

- Automatické posouvání (druhá polohy tlačítka)

- **Okno spolujezdce**
  - Manuální posouvání (první polohy tlačítka)
  - Automatické posouvání (druhá polohy tlačítka)

- **Zadní okno vlevo**
  - Manuální posouvání (první polohy tlačítka)
  - Automatické posouvání (druhá polohy tlačítka)

- **Zadní okno vpravo**
  - Manuální posouvání (první polohy tlačítka)
  - Automatické posouvání (druhá polohy tlačítka)

Při každém scénáři dojde k částečnému, nebo úplnému stažení okna (pohyb dolů) a následné uzavření okna (pohyb nahoru).

Počáteční stav testu předpokládá zapnuté zapalování a všechna okna uzavřená.

Pro přehlednost jsou uvedeny pouze modely pro ovládání okna řidiče (manuální i automatické ovládání) a ostatní tři okna jsou modelově podobná, pouze s použitím správných metod.

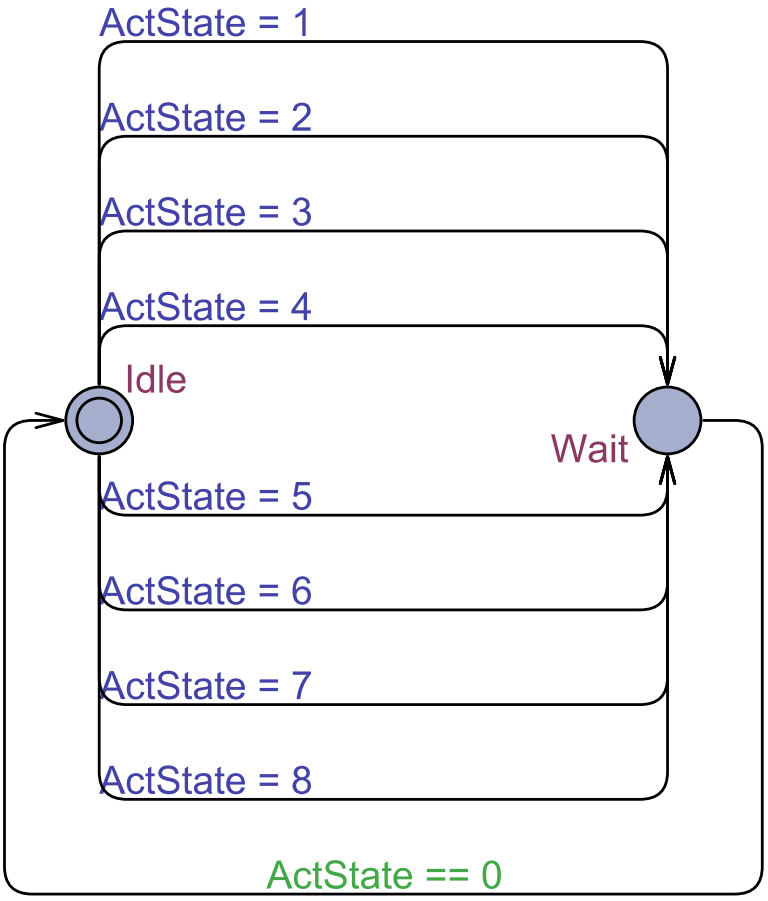
- **Model okolí**
  Model vybere dle druhu testu (náhodné procházení, nebo systematické) hranu, která nastaví proměnnou *ActState* určující druh prováděného testu. Následně čeká na dokončení testu a vybere další.

- **Model blokace ovládání zadních oken**
  Testovaný model umožňuje aktivovat a deaktivovat během testu blokaci zadních oken.

  Blokace neovlivňuje zbytek testovaného modelu ovládacího panelu ve dveřích řidiče. Blokuje pouze ovládání zadních oken, které není zahrnuto v tomto modelu. Průběh testu tedy není u tohoto modelu nijak stavově řízen modelem okolí.

- **Model manuálního ovládání okna řidiče**
  Navrhnutý model (při povolení průchodu modelem okolí) v první kroku vyvolá stisknutí tlačítka stahování okna do první polohy a setrvá v jeho držení, dokud není okno minimálně na 50 % otevřené.

  Následně uvolní tlačítko a stiskne ho do první polohy pro manuální uzavírání a zůstává v této poloze, dokud není okno uzavřeno na více jak



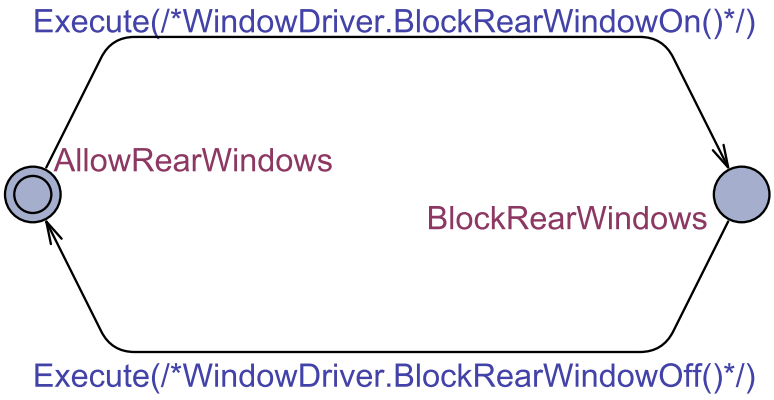
Obrázek 5.5: Model okolí pro ovládání stahování oken

90 % (otevření je menší než 10 %).

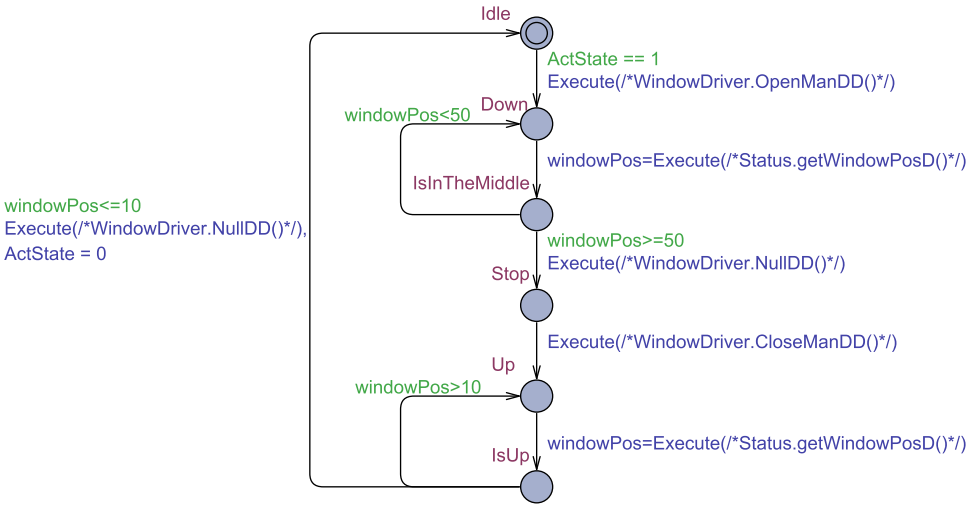
Během fáze stahování/uzavírání je průbežně zjišťována a vyhodnocována aktuální poloha okna.

- **Model automatického ovládání okna řidiče**
  Při nastavení stavové proměnné *ActState* pro volbu automatického stahování/uzavírání okna u řidiče je proveden stisk tlačítka do druhé polohy (automatické otevření okna řidiče). Následně v témže kroku proběhne uvolnění tlačítka. Po dosažení alespoň poloviční otevření je vyvoláno automatické uzavření okna stiskem tlačítka na druhý stupeň pro uzavírání a jeho následné uvolnění. Jakmile dosáhne okno otevření na méně než 10 %, je testování automatického otevírání pro okno řidiče u konce.

  Během fáze stahování/uzavírání probíhá průbežně zjišťování a vyhodnocování aktuální polohy okna.



Obrázek 5.6: Model blokace ovládání zadních oken



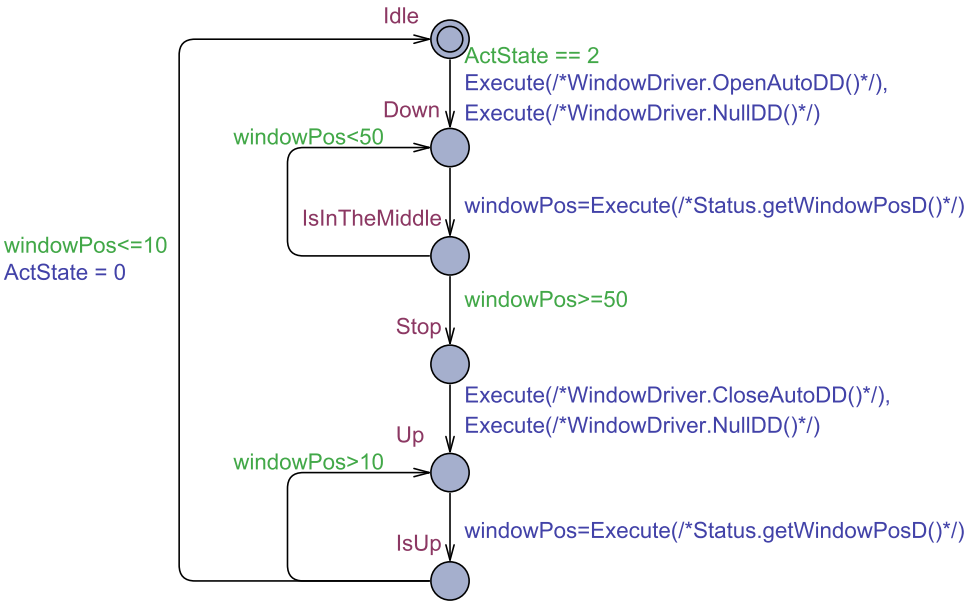
Obrázek 5.7: Model manuálního ovládání okna řidiče

5.4 Výsledky

Při tvorbě modelů pro ovládání stahování oken bylo uvažováno pouze o ovládacím panelu u řidiče, neboť obsahuje ovládání stahování všech oken.

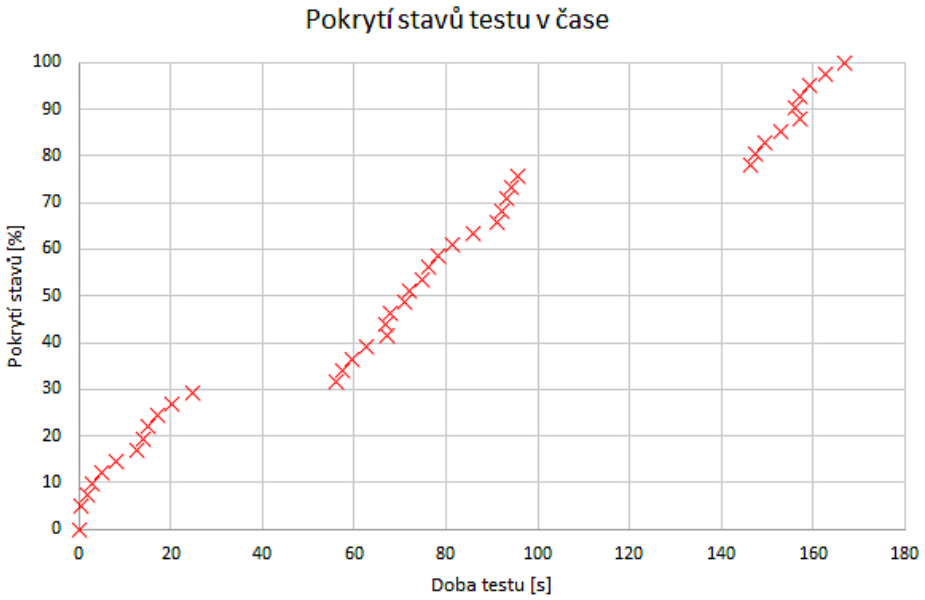
Pokrytí všech stavů panelu se podařilo namodelovat ze 100 %. Z důvodu stále vyvíjeného testovacího stavu nebylo možné úplně všechny namodelované stavy otestovat. Blokace oken ještě nebyla namapována, tedy bylo možné otestovat 96 % stavů. Tabulka 5.1 popisuje jaké stavy byly testovány. Je v ní zahrnuto pouze manuální a automatické stahování okna u řidiče. Stahování zbylých tří oken bylo vynecháno pro přehlednost.

Byl proveden test s ukončením po průchodu všemi stavy (*Node coverage*). Procházení grafem bylo náhodné. K pokrytí všech stavů došlo za 166,91 sekund.



Obrázek 5.8: Model automatického ovládání okna řidiče

Graf na obrázku 5.9 zobrazuje pokrývání stavů (první navštívení) v průběhu času. Procházení grafu probíhalo náhodnou testovací strategii - další hrana se volí náhodně z aktuálně možných a tedy mezery v grafu značí okamžiky testu, kdy byl model testován v již navštívené části. V čase 166,91 sekund byl navštívený poslední dosud nenavštívený stav a model byl testem kompletně pokryt. Kompletní trace log lze najít v přiložených souborech.



Obrázek 5.9: Navštívení stavů během testů v čase

	Stav	**Modelováno**	**Testováno**	**Čas prvního navštívení [s]**	**Pokrytí [%]**
Model okolí	Idle	Ano	Ano	0	0
	Wait	Ano	Ano	0,27	4
Blokování zadních oken	AllowRearWindows	Ano	Ne		
	BlockRearWindows	Ano	Ne		
Řidičovo okno manuálně	Idle	Ano	Ano	0	0
	Down	Ano	Ano	91,09	56
	IsInTheMiddle	Ano	Ano	92,17	58
	Stop	Ano	Ano	93,26	60
	Up	Ano	Ano	94,37	63
	IsUp	Ano	Ano	95,48	65
Řidičovo okno manuálně	Idle	Ano	Ano	0	0
	Down	Ano	Ano	66,68	38
	IsInTheMiddle	Ano	Ano	67,76	40
	Stop	Ano	Ano	67,76	42
	Up	Ano	Ano	70,82	42
	IsUp	Ano	Ano	71,9	44

Tabulka 5.1: Pokrytí stavů stahování oken

Kapitola 6
Model stěračů

Posledním vytvořeným modelem pro testování metodou Model-Based Testing je model ovládání stěračů čelního a zadního okna.

6.1 Stírací systém vozidla

Vozidla jsou standardně vybavena stěrači čelního a zadního okna. Navzdory tomu, že jsou stěrače čelního okna dva, zvažujeme je díky mechanickému propojení táhly a jejich společnému pohonu v modelu jako celek. Stěrače čelního okna umožňují setření 1x, ostřik a setření, zapnutí cyklovače s různou časovou prodlevou stěru, pomalé a rychlé stírání.

Stěrač u zadního okna má možnost kontinuálního stírání, ostřik a setření. Mnohé verze vozidel jsou vybaveny komfortní funkcí automatického setření při zapnutých předních stěračích a zařazení zpátečního rychlostního stupně. Tato komfortní nadstavba není v modelu uvažována.

Model vychází z předpokladu odemknutého vozidla a sepnutého zapalování. Jen tak je aktivována možnost manipulace se stěrači. Tato skutečnost je reflekrována v inicializační sekvenci v EXAMu. Ta probíhá před spuštěním samotného testování modelu.

6.2 Fyzické provedení

Ovládání stěračů je realizováno páčkou na pravé straně za volantem, s pomocí níž lze přepínat mezi jednotlivými režimy stírání.

6.2.1 Ovládání stěračů

- **Páčka ovládání stírání**

  - **Středová poloha**
    Všechny funkce jsou v této poloze v nečinnosti.

- **Stírání čelního skla**
  - **Setření 1x**
    Vykoná se jedno setření čelního skla (jeden cyklus).
  - **Ostřik a setření**
    Po dobu držení páčky se provádí ostřik čelního skla a zároveň stírání.
  - **Intervalové stírání**
    Často řízené pomocí dešťového senzoru.
  - **Pomalé kontinuální stírání**
    V této poloze páčky probíhá kontinuální a pomálé stírání.
  - **Rychlé kontinuální stírání**
    V této poloze páčky probíhá kontinuální a rychlé stírání.
- **Stírání zadního skla**
  - **Kontinuální stírání**
    Při zapnutí zadního stírání probíhá kontinuální stírání, které má nastavený pevný interval mezi jednotlivými stěry.
  - **Ostřik a setření**
    V této poloze páčky je ke kontinuálnímu stírání prováděn ostřik zadního skla.

6.2.2 Kontrola provedení stěru

Aby bylo možné vyhodnotit provedení stěru, je počítán počet stíracích cyklů, tedy projetí jednoho cyklu stěračů. Tato hodnota je inkrementována každým stěrem a lze jí vyčíst z HILu pro ověření funkčnosti stěračů.

6.3 Modelování ovládání stěračů

6.3.1 Příprava na straně EXAMu

Ovládání stěračů je implementováno ve třídou Wipers a kontrola stírání ve třídě Status.

```
allowedObjects['Wipers'] = 'TestWipers'
allowedObjects['Status'] = 'TestStatus'
```

Obrázek 6.1: Třída pro ovládání stěračů a kontrolu setření

Ve třídě Wipers jsou umístěné metody pro ovládání předních a zadního stěrače. Přední stěrač umožňuje vypnutí, 1x setření, ostřik a setření, intervalové stírání (bez možnosti nastavení délky intervalu), kontinuální stírání pomalé a kontinuální stírání rychlé. Pro zadní stírání je umožněno vypnutí, kontinuální stírání a ostřik a setření.

Třída Status obsahuje metodu pro vyčtení čítače stěrů pro přední a zadní stěrač.

```
#Class Wipers
methodAliases['LeverOff'] = 'LeverOff'
methodAliases['Lever1X'] = 'Lever1X'
methodAliases['LeverSpray'] = 'LeverSpray'
methodAliases['LeverInt'] = 'LeverInt'
methodAliases['LeverSpeed1'] = 'LeverSpeed1'
methodAliases['LeverSpeed2'] = 'LeverSpeed2'

methodAliases['LeverRearOff'] = 'LeverRearOff'
methodAliases['LeverRearOn'] = 'LeverRearOn'
methodAliases['LeverRearSpray'] = 'LeverRearSpray'
```

Obrázek 6.2: Metody pro ovládání stěračů

```
#Class Status
methodAliases['getFrontWiperCnt'] = 'getFrontWiperCnt'
methodAliases['getRearWiperCnt'] = 'getRearWiperCnt'
```

Obrázek 6.3: Metoda pro kontrolu setření stěračů

6.3.2 Navržený model

Systém stěračů je složen ze dvou modelů: model stěrače čelního okna a model stěrače zadního okna.

Počáteční stav testu se předpokládá při zapnutém zapalování a páčky ovládání stěračů ve středové poloze.

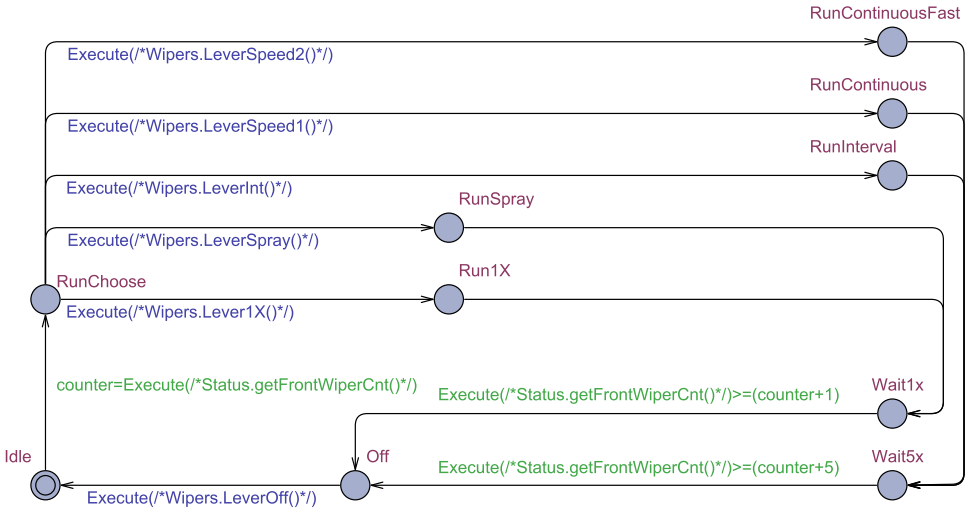
- **Model stěrače předního okna**
  Model zahrnuje všechny režimy stírání čelního okna.
  V prvním kroku je vždy vyčten aktuální stav čítače stěrů. Poté proběhne sepnutí jednoho vybraného režimu stírání a během stírání se čeká na předpokládaný počet stěrů, který musí proběhnout, aby se daný řežim stírání vypnul. Pro stírání v režimu 1x a ostřik je podmínkou jedno setření. Pro ostatní režimy pět setření.

  Funkce nastavení délky cyklu při intervalovém stírání není v modelu zahrnuta.

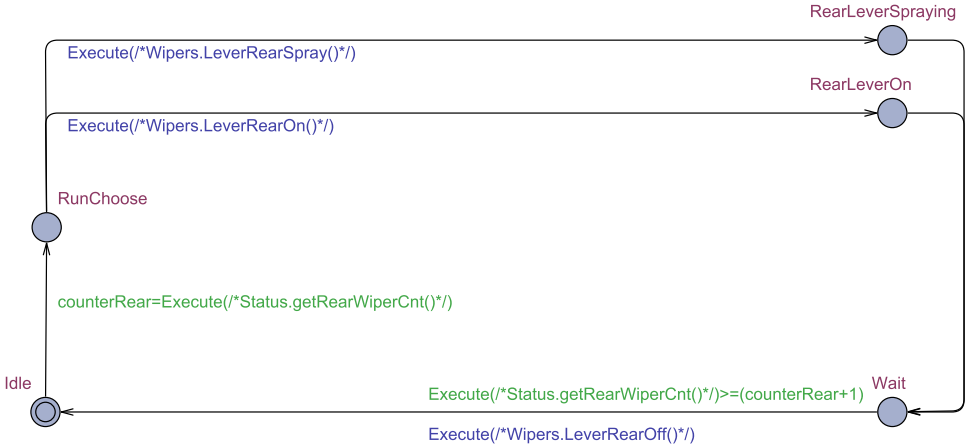
V modelu zadního stěrače proběhne v prvním kroku vyčtení stavu čítače zadního stěrače a poté spuštění režimu kontinuálního stírání, nebo stírání a ostřiku. Po provedení jednoho stěru je režim vypnut.

6.4 Výsledky

Pokrytí všech stavů stěračů se podařilo namodelovat z 91 % pro přední stěrače a z 83,3 % pro zadní stěrače. Z důvodu stále vyvíjeného HIL zařízení nebylo možné úplně všechny namodelované situace otestovat. Na předních stěračích bylo možné pokrýt testy 72,7 % a pro zadní stěrače 0 %. Tabulky 6.1 a 6.2



Obrázek 6.4: Model stírání předního okna

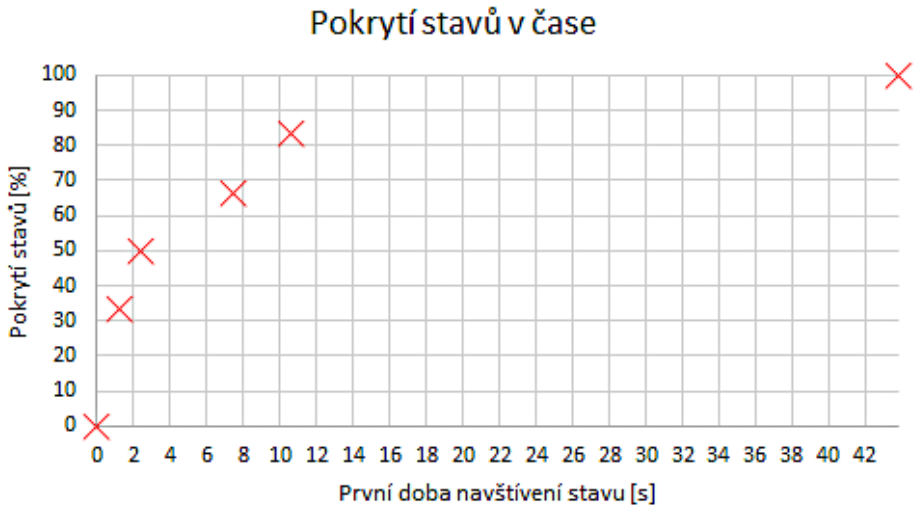


Obrázek 6.5: Model stírání zadního okna

popisují jaké stavy byly testovány.

Byl proveden test s ukončením při pokrytí všech stavů (*Node coverage*) a náhodným procházením. Celková doba testu činila 43,77 sekund.

Mezery v grafu značí procházení již navštívených stavů, které nezvýší procentuální hodnotu pokrytí modelu. Kompletní trace log lze najít v přiložených souborech.



Obrázek 6.6: Navštívení stavů během testů v čase

Stav	Modelováno	Testováno	Čas prvního navštívení [s]	Pokrytí modelu [%]
Idle	Ano	Ano	0	0
RunContinuousFast	Ano	Ano	2,38	2,38
RunChoose	Ano	Ano	1,26	33,3
Wait5x	Ano	Ano	2,38	50
Off	Ano	Ano	7,41	66,6
RunContinuous	Ano	Ano	10,65	83,3
Run1X	Ano	Ano	43,77	100
Wait1x	Ano	Ano	43,77	100
RunSpray	Ano	Ne		
RunInterval	Ano	Ne		
IntervalChange	Ne	Ne		

Tabulka 6.1: Pokrytí stavů předních stěračů

Stav	Modelováno	Testováno	Čas prvního navštívení [s]	Pokrytí modelu [%]
Idle	Ano	Ne		
RunChoose	Ano	Ne		
RearLevelOn	Ano	Ne		
RearLevelSpraying	Ano	Ne		
Wait	Ano	Ne		
RunAtReverseGear	Ne	Ne		

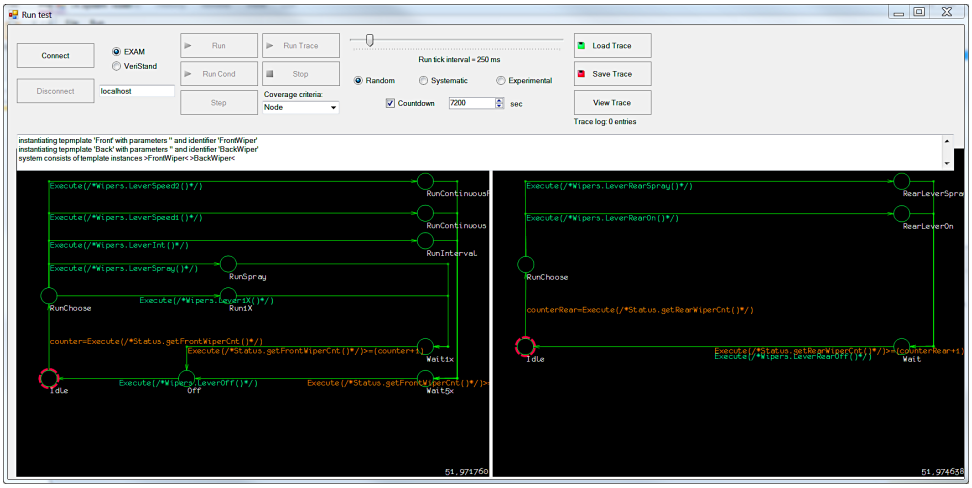
Tabulka 6.2: Pokrytí stavů předních stěračů

Kapitola 7

Ověření metody MBT pro integrační testování

Ověřování metody probíhalo na validovaných modelech ovládacího panelu stahování oken ve dveřích řidiče a modelu stěračů. Na těchto modelech byly otestovány jednotlivé volané metody malými modely, na kterých se prokázala funkčnost zapojení testovacího stavu a schopnost ovládání testovaného hardwaru z prostředí Tasteru. Poté byla provedena jednokroková analýza, spuštěn automatický test a proběhla kontrola *trace* logu. Testy byly několikrát zopakovány s různými ukončovacími podmínkami (pokrytí stavu, pokrytí hran) a diskutována validita modelů.

Před spuštěním testovací sekvence v EXAMu proběhla inicializace HILu, všech jeho komponent a nastavení do defaultních stavů. Poté bylo možné Taster připojit k EXAMu, načtený model v něm spustit a podrobit ho testování.



Obrázek 7.1: Načtený model v Tasteru

7.1 Experiment k zhodnocení přístupu MBT

7.1.1 Ověření modelu

K ověření přístupu byl vybrán model panelu ovládání oken ve dveřích řidiče.

Experiment se skládal z několika částí:

- **Ověření modelu pomocí *Verifier*u v UPPAALu**

  Jako první bylo provedeno ověření určitých vlastností návrhu modelu pomocí ověřovacího nástroje v UPPAALu. Lze jím odhalit případnou chybu návrhu ještě před samotným experimentem.

  Nástroj *Verifier* byl použit pro zjištění *deadlocku* pomocí dotazu se syntaxí "A[] not deadlock". Nástroj prověří průchodem grafu, zda existuje možnost, že by tímto způsobem test uváznul.

  Výsledek dotazu byl pozitivní a žádný *deadlock* tedy v modelu neexistuje.

  ```
  A[] not deadlock
  Verification/kernel/elapsed time used: 0s / 0s / 0s.
  Resident/virtual memory usage peaks: 7 460KB / 27 136KB.
  Property is satisfied.
  ```

  Obrázek 7.2: Ověření modelu na deadlock

- **Dosažitelnost všech stavů systému testem**
  Tímto testem ověříme, zda již navrhnutý a ověřený model dokáže navštívit každý stav testu.

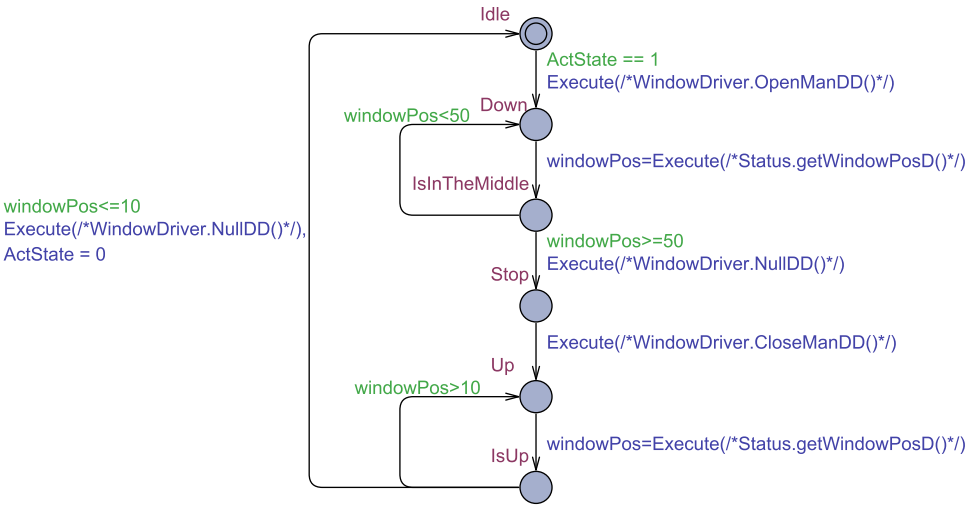
  Možností existuje několik. Je možné využít nástroj *Verifier* v UPPAALu a každý stav ověřit dotazem "E<> Proces.Stav". Jinými slovy existuje možnost se do daného stavu dostat?

  Zvolenou metodou ale byla metoda experimentální. Ta zahrnovala nahrání modelu nástrojem UPPAAL a jeho spuštění s podmínkou navštívení všech stavů. Metoda byla časově náročnější, ale výhodou je testování na reálném hardwaru, a nikoliv pouze matematicky.

  Experiment s náhodným procházením stavů byl dokončen za 166,91 sekund a navštívil všechny stavy modelu.

- **Zanesení chyby do modelu**
  Kontrola správného návrhu modelu může být provedena zanesením chyby a ověřením, že test systému zhavaroval.

  Pro ověření byl vybrán systém ovládání stahování oken ve dveřích řidiče. Z něj poté model, ve kterém je testováno stahování okna v řidičových dveřích a to pomocí první polohy tlačítka ovládání stahování (manuální ovládání).



Obrázek 7.3: Model stahování okna u řidiče - manuální ovládání

  Postupně byly v modelu vynechávány jednotlivé *guard*y a akce. Model po jakémkoliv zásahu vykazoval nesprávné chování, případně skončil v *deadlocku*

- **Pokus o namapování na neexistující třídu nebo metodu v EXAMu**
  Průběh testu zhavaruje, pokud nastane chyba v návrhu a do modelu je namapováno volání třídy, nebo metody v EXAMu, která neexistuje. K této situaci dojde také, pokud nastane chyba v pojmenování (překlep).

  Z opačného úhlu pohledu: pouze správné namapování zajistí bezproblémové testování.

  Experiment probíhal tak, že se smazala metoda manuálního otevření okna řidiče v EXAMu a test se spustil. V okamžiku průběhu testu, kdy byla Tasterem daná metoda požadována, test zhavaroval.

7.1.2 Testování proti simulované implementaci

Další částí experimentu bylo ověření modelu proti simulované implementaci v EXAMu. Na straně EXAMu byly volané metody a třídy nahrazeny metodami

```
Caught Exception in loop.py
 Error type      AttributeError
 Error message   TheClass instance has no attribute 'OpenManDD'
 Traceback       exception-traceback:   File "C:\pCode\pRunner\loop.py", line 204, in commandLoop
                 command: exec(execFileName + '.main()')
                 exception-traceback:   File "<string>", line 1, in <module>
                 command:
                 exception-traceback: Uml   "EXAMples.HonzaSobotka.Taster.Control.TasterControl", line 39, in main
                 command: testFlow('', '00000000-0000-0000-0000-000000000000_L0')
                 exception-traceback: Uml   "EXAMples.HonzaSobotka.Taster.Control.TasterControl", line 50, in testFlow
                 command: _d._9._5f77b8ba_81b1_4239_b6db_2f40c7d34e9d.getInstance().RunTest()
                 exception-traceback: Uml   "EXAMples.HonzaSobotka.Taster.Control.TasterControl_Basic", line 143, in RunTest
                 command: ret = getInstance().Execute(execute)
                 exception-traceback: Uml   "EXAMples.HonzaSobotka.Taster.Control.TasterBinding", line 97, in Execute
                 command: res = eval(command)
                 exception-traceback:   File "<string>", line 1, in <module>
                 command:
```

Obrázek 7.4: Smazání metody v EXAMu

vykazujícími podobné chování jako reálný systém (návratové hodnoty, očekávané vstupy, atd.). Tyto metody byly implementovány skripty napsanými v jazyce Python. Díky tomu bylo možné vyzkoušet model ještě před samotným testováním s reálným hardwarem HIL metodou. Lze si ověřit předpokládané chování, či přibližnou dobu za kterou může být daný test proveden. Snadno se také úpravou skriptu může vytvořit simulovaná chyba a ověřit chování modelu v takto vzniklém případě.

Pro testování byl vybrán model stěračů (předních i zadních) - viz kapitola 6, Obrázek 6.4 a 6.5. Simulováním šlo otestovat i stavy, které nejsou aktuálně na testovacím stavu zahrnuty.

V simulované implementaci byly akce vykonávány výpisem do konzole EXAMu. Návratové hodnoty poté uzpůsobeny tak, aby co nejvěrohodněji napodobovaly reálný systém. V případě stěračů se jednalo o čítač inkrementující počet stěrů. Nahrazen byl proměnnou, která je každým dotazem na stav zvýšena o hodnotu 1 (+1 stěr).

```
cntFront = cntFront + 1
print('Front wiper counter is ' + str(cntFront))
return cntFront
```

Obrázek 7.5: Simulovaná implementace čítače stěrů

```
Executing LeverSpeed1
TasterControl_Basic.RunTest(): Return value '1' sent to client.
TasterControl_Basic.RunTest(): Client '('127.0.0.1', 58852)' has connected.
TasterControl_Basic.RunTest(): Data 'execute _b._f._8e1721ff_e19b_4154_aad9_e7c1bd33aafb.getInstance().getRearWiperCnt()' received.
TasterControl_Basic.RunTest(): Execute command received: '_b._f._8e1721ff_e19b_4154_aad9_e7c1bd33aafb.getInstance().getRearWiperCnt()'.
TasterBinding.Execute(): _b._f._8e1721ff_e19b_4154_aad9_e7c1bd33aafb.getInstance().getRearWiperCnt()
Wiper counter rear is  17
TasterControl_Basic.RunTest(): Return value '17' sent to client.
TasterControl_Basic.RunTest(): Client '('127.0.0.1', 58854)' has connected.
TasterControl_Basic.RunTest(): Data 'execute _b._f._8e1721ff_e19b_4154_aad9_e7c1bd33aafb.getInstance().getFrontWiperCnt()' received.
TasterControl_Basic.RunTest(): Execute command received: '_b._f._8e1721ff_e19b_4154_aad9_e7c1bd33aafb.getInstance().getFrontWiperCnt()'.
TasterBinding.Execute(): _b._f._8e1721ff_e19b_4154_aad9_e7c1bd33aafb.getInstance().getFrontWiperCnt()
```

Obrázek 7.6: Výpis simulované implementace v EXAMu

Bylo zvoleno více testovacích scénářů. Pokrytí stavů a hran, s náhodným či systematickým procházením. Testy byly jednou zopakovány. Na závěr byl proveden dlouhodobější test bez kritérií na pokrytí. Trace logy všech testů jsou přiloženy v přílohách.

ID testu	Pokrytí	Průchod modelu	Počet kroků testu	Doba testu [s]
1	Stavy	Náhodný	97	155
2	Stavy	Náhodný	97	150
3	Stavy	Systematiký	39	73
4	Stavy	Systematiký	39	73
5	Hrany	Náhodný	137	219
6	Hrany	Náhodný	230	364
7	Hrany	Systematiký	40	74
8	Hrany	Systematiký	40	74
9	Žádné	Náhodný	1173	2234

Tabulka 7.1: Testování proti simulované implementaci

7.2 Vyhodnocení

Při prvních několika spuštěních modelu na HILu proběhlo objevení a odstranění chyb projevujících se až v kompletním zapojení.

Neočekáváným jevem během testování byla nedeterministicky definovatelná odezva HILu (různá doba trvání jednotlivých metod). Ta nastala při požadavku o provedení akce, nebo čekání na návratovou hodnotu a její následné vyhodnocení Tasterem. Řádově byla přijímána odezva HILu v čase 1-2 sekundy a během této doby Taster vyčkával, tedy hodinový takt testu (250 ms) nemohl být respektován. Zde proto vyvstává doporučení na možnou úpravu nástroje Taster. Ten může procházet model pomocí dalších možných hran a stavů. Případně bude, jako nyní, čekat na dokončení požadované akce, s tím rozdílem, že vnitřní proměnné testovaného modelu budou reflektovat změnu doby, která mezi čekáním na výsledek uplyne.

Velké zdržení také nastalo, pokud se v jednom hodinovém taktu ověřovalo více podmínek na hranách zároveň. Volání více hodnot z testovacího stavu trvá obvykle poměrně dlouho. V tomto případě se tedy výsledná doba čekání sčítá. Do velké míry tomu lze předejít ukládáním již zjištěných stavů do vnitřních proměnných Tasteru a aktualizovat jejich hodnoty, pouze je-li to nutné a dotazy provádět na co nejmenším počtu hran.

Při testování bylo také zjištěno, že nesprávně nazvaná metoda, případně neexistující metoda, která má být na straně EXAMu vyvolána, skončí neošetřenou vyjímkou a test zhavaruje. Řešením by tedy mohlo být ošetření potenciálně rizikového místa v EXAMu pomocí Try-Catch bloku a zápisem o nastalé události a pokračování v testu.

Obdobně pokud dojde v modelu k porušení invariantu stavu, na straně EXAMu při jeho porušení proběhne zastavení testu. Jelikož v průběhu testu

slouží EXAM pouze jako "middleware", nabíhání nového testu trvá díky velkému množství načítaných modulů řádově minuty, je jako další doporučení na úpravu stávající metody v EXAMu provést pouze zapis o události. Zastavení testu pak pouze na straně Tasteru, kde je tak očekáváno v případě porušení invariantu, ale nikoliv EXAMu.

Při testování proti simulované implementaci a různých podmínkách ukončení testu vyplynuly časové rozdíly průchodů modelem. Systematické pokrytí (stavů i hran) uskutečnilo v testovaném modelu výrazně menší počet testovacích kroků než náhodné. U náhodného se navíc počty průchodů mohly výrazně lišit vůči jednotlivým testům. Dlouhodobější test ověřil stabilitu prováděného testu. Během HIL testování lze simulovanou implementací nahradit chybějící hardware testovacího stavu a tím otestovat pouze aktuálně implementovaná zařízení.

Během testování byla ověřena jednak proveditelnost, ale především vhodnost MBT pro integrační HIL testování. MBT může přinést zcela nové variace integračních testů, které dokáží testovaný systém prověřit způsobem, který se svým návrhem blíží mnohem více modelu okolí a tedy věrohodněji simulovat reálné podmínky. Výhodou MBT přístupu jsou velké možnosti verifikace i validace modelů a u menších systémů i snazší tvorba testu (modelu), oproti sekvenčnímu testům, které lze implementovat v EXAMu. Do modelu se snadno zahrnou podmínky průchodů jednotlivými hranami (*guardy*), které umožní větší variabilitu prováděných testů, dle aktuálního stavu okolí.

Omezením MBT přístupu může být modelování větších systémů s velkým množstvím stavů, kde výsledný model požaduje vysoké nároky na zkušenosti tvůrce modelu. Náročné je především podchycení veškerého chování, které může v daném systému nastat. Řešením pro takto komplexnější systémy by mohl představovat hiearchický návrh modelů s vyšší abstrakcí modelovaných vrstev, kde vyšší modelované vrstvy by řídily vrstvy nižší. Zachovala by se především přehlednost modelu, zaručující deterministické chování v průběhu testování.

Kapitola 8

Závěr

Cílem této práce bylo ověřit možnosti použití metody Model-Based Testing na integrační testování. Po úvodním seznámení s návrhem modelů v prostředí UPPAAL a diskusí nad strategií vhodného návrhu, se jako nejvhodnějším přístupem jeví tvorba sekvenčního modelu okolí. Tento model předpokládá určitou souslednost činností, které po sobě následují. Díky MBT přístupu lze snadno pokrýt velké množství chování systému, při zachování dobré přehlednosti návrhovaného modelu.

Verifikace umožňuje snadné ověření základních vlastností modelu - jako *deadlock* a dosažitelnost všech stavů - a zajištění všech očekávaných předpokladů a specifikací. Po diskusi nad validováním navržených modelů a stanovení pěti kroků ověřujících validitu návrhu, je přistoupeno k vytvoření tři sad modelů, popisujících vybrané základní komfortní systémy vozidla (bezklíčkový přístup, ovládání stahování oken a stěrače).

Na Modelu bezklíčkového přístupu (Kessy systém) bylo nejdříve diskutováno fyzické provedení a z něho vyplývající tvorba modelu tak, aby proběhlo co nejlépe zachycení chování daného systému. Systém nabízí mnoho způsobů ovládání, a proto výsledný model vyrostl do větších rozměrů. Následně byl systém podroben testování a diskutovány výsledky. Navržený model pokrýval 76 % reálného systému a ve stejném rozsahu byl i otestován. MBT přístup poskytl zajímavé souběhy testovaných akcí, které by se sekvenčním testem pouze obtížně pokrývaly. Na modelu se také ověřilo chování testovacího stavu, kde byla velmi překvapující nedeterministická odezva. Na základě tohoto zjištění bylo navrhnuto několik úprav nástroje Taster, které by tuto skutečnost lépe zohledňovaly.

Na dalších dvou modelech (ovládání stahování oken a stěračů) byly provedeny testy s ukončovací podmínkou různého pokrytí modelu (stavů, či hran) a dvěma strategiemi průchodu modelem (náhodné a systematické). Především ve vybrané stategií se ukázala velká odlišnost doby trvání testu zohledňující pokrytí modelu.

V závěru práce byl proveden experiment, na kterém proběhlo ověření mo-

delu různými přístupy, a také testování proti simulované implementaci. Po něm proběhla úvaha nad společnými rysy těchto testů a vyhodnocení vhodnosti MBT pro integrační testování. Vhodnost MBT přístupu pro integrační testování na HILu se ukázala snadno implementovatelná na systémech s menším počtem možných stavů. Se vzrůstajícím počtem možných stavů systému roste náročnost na správné namodelování.

Příloha A

Literatura

[Ale02] Alexandre David, Gerd Behrmann, Kim G. Larsen, Wang Yi, *A tool architecture for next generation of uppaal*, Springer-Verlag Berlin Heidelberg, 2002.

[Bel04] Gene Bellinger, *Modeling and Simulation*, `http://www.systems-thinking.org/modsim/modsim.htm`, 2004, Accessed: 2019-05-21.

[Car02] John S. Carson, *Model verification and validation*, Model verification and validation, IEEE, 2002, Published in: Proceedings of the Winter Simulation Conference.

[Cla01] Clarke M. , Edmund and Grumberg, Orna and Peled, Doron, *Model checking*, 2001.

[Dep15] Department of Computer Science, Aalborg University, *A tutorial on uppaal 4.0*, `http://people.cs.aau.dk/~kgl/SSFT2015/SMC%20tutorial.pdf`, 2015, Accessed: 2019-05-21.

[EXA11] EXAM, *EXAM concept paper*, `http://www.exam-ta.de/en/downloads/download/4-dokumente/98-exam_conceptpaper.html`, 2011, Accessed: 2019-05-21.

[Ger09] Gerd Behrmann, Alexandre David, and Kim G. Larsen, *Uppaal 4.0 : Small Tutorial*, `https://www.it.uu.se/research/group/darts/uppaal/small_tutorial.pdf`, 2009, Accessed: 2019-05-21.

[Gur19] Guru99, *Model based testing tutorial*, `https://www.guru99.com/model-based-testing-tutorial.html`, 2019, Accessed: 2019-05-21.

[Kob18] Ondřej Kobza, *Rozšíření testovacího nástroje taster*, Master's thesis, ČVUT, the Czech Republic, 2018.

[Man05] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, Alexander Pretschner, *Model-based testing of reactive systems*, Springer, 2005.

[Mar06] Mark Utting, Bruno Legeard, *Practical model-based testing: A tools approach*, Morgan Kaufmann; 1 edition (December 11, 2006), 2006, ISBN-10: 0123725011.

[MV13] Luca Viganò (Eds.) Margus Veanes, *Tests and proofs*, ch. 2, pp. 20–38, Springer, Heidelberg Dordrecht London NewYork, 2013.

[Pre05] Pretschner A., SlotoschO., LotzbeyerH., Aiglstorfer E., Kriebel S., *Model based testing for real: The inhouse card case study*, Institut fur Informatik, Technische Universit at Munchen (2005).

[Sar10] Robert G. Sargent, *VERIFICATION AND VALIDATION OF SIMULATION MODELS*, Syracuse University (2010).

[Sch13] Gerardo Schnei, *Model-based testing*, `http://www.cse.chalmers.se/edu/year/2013/course/DAT260/files/05-Intro-MBT.pdf`, 2013, Lecture 1 - Overview of Verification and Validation.

[uoE00] The university of Edinburgh, *Computer science 4 and msc: Modelling and simulation*, `http://www.inf.ed.ac.uk/teaching/courses/ms/`, 2000, Accessed: 2019-05-18.

[UPP06] UPPAAL, *A tutorial on uppaal 4.0*, `https://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf`, 2006, Accessed: 2019-05-21.

[Ves16] Michal Veselka, *Implementace rozhraní mezi nástrojem TaSysTest a softwarem EXAM*, Master's thesis, ČVUT, the Czech Republic, 2016.

Příloha B

Obsah přiloženého CD

- **Diplomovou práci**

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

- **Model Kessy systému**

  Testovaný model bezklíčkového přístupu.

- **Model ovládání stahování oken**

  Testovaný model ovládacího panelu ve dveřích řidiče.

- **Model stěračů**

  Testovaný model spínání stěračů.

- **Trace log testu Kessy modelu**

  Průběh testování modelu Kessy zaznamenaný Tasterem.

- **Trace log testu modelu ovládání oken**

  Průběh testování modelu ovládání oken zaznamenaný Tasterem.

- **Trace log testu modelu stěračů**

  Průběh testování modelu stěračů zaznamenaný Tasterem.

- **Soubor testů modelu stěračů proti simulované implementace**

  Průběh testování modelu stěračů proti simulované implementaci v EXAMu zaznamenaný Tasterem.