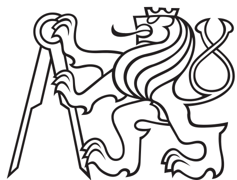
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra měření

Návrh a implementace algoritmu simulace sítí časovaných automatů

Matěj Šťastna

Vedoucí práce: Ing. Lukáš Krejčí
Srpen 2021

ČVUT ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šťastna** Jméno: **Matěj** Osobní číslo: **474619**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Návrh a implementace algoritmu simulace sítí časovaných automatů

Název bakalářské práce anglicky:

Design and implementation of simulation algorithm for timed automata networks

Pokyny pro vypracování:

1. Seznamte se s problematikou HiL integračního testování.
2. Analyzujte zdrojový kód a modelovací jazyk MBT nástroje Taster. Analyzujte proces simulace časovaných automatů využívaný nástrojem Taster.
3. Navrhněte nový simulační algoritmus pro nástroj Taster. Navržený algoritmus by měl podporovat urgentní a plošné synchronizace a urgentní a rychlé stavy. Navrhněte vhodný způsob řešení chybových situací.
4. Navrhněte vhodnou architekturu pro řízení procesu testovaní s využitím navrženého algoritmu. Navržená architektura by měla vhodně řešit zpracování chyb, opakované generování testovacích sekvencí a sběr statistických dat z testovacího procesu.
5. Navržený algoritmus a architekturu řízení testů implementujte do nástroje Taster.
6. Implementovaný algoritmus a architekturu otestujte na případové studii.

Seznam doporučené literatury:

[1] Tomáš Grus: Implementation of Integration Testing Test Cases Generation Tool. Diplomová práce, ČVUT v Praze, 2014.
[2] Mehrdad Bagheri: Analysis of Model-based Testing methods for Embedded Systems. Diplomová práce, Uppsala University, 2016.
[3] Johan Bengtsson, and Wang Yi: Timed Automata: Semantics, Algorithms and Tools. Lecture Notes in Computer Science. 2004.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Lukáš Krejčí, katedra měření FEL

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

Datum zadání bakalářské práce: **22.01.2021** Termín odevzdání bakalářské práce: **13.08.2021**

Platnost zadání bakalářské práce:
do konce zimního semestru 2022/2023

Ing. Lukáš Krejčí podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

© VUT v Praze, Design: VUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

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

.

Datum převzetí zadání

Podpis studenta

© VUT v Praze, Design: VUT v Praze, VIC

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce Ing. Lukáši Krejčímu za odborné vedení, praktické rady a nápomoc při pravidelných konzultacích. Zároveň můj dík patří rodině za trpělivost a pochopení i přes všechna vzniklá úskalí v rámci studia a velký dík též zasluhuje má přítelkyně Kristýna, která mě po celou dobu studia vytrvale podporovala a motivovala k úspěšnému dokončení.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje.

V Praze, 13. srpna 2021

....

Abstrakt

Cílem této bakalářské práce je návrh a implementace nového simulačního algoritmu průchodu sítí časovaných automatů simulačního nástroje Taster. V úvodní části je nejprve seznámení s problematikou integračního testování a s jazykem sítí časovaných automatů. Poté je analyzován modelovací jazyk nástroje UPPAAL využívaný v nástroji Taster pro modelově orientované testování. Následuje analýza zdrojového kódu procesu simulace časovaných automatů nástroje Taster a vhodný návrh nové architektury simulačního algoritmu eliminující nedostatky původní verze. Po odzkoušení návrhu na prototypu je následně nová architektura implementována do nástroje Taster. V závěru je na několika provedených experimentálních simulacích ověřena validita vytvořeného návrhu.

Klíčová slova: Taster, integrační testování, časované automaty, UPPAAL, HiL testování, SuT, Modelově orientované testování

Vedoucí práce: Ing. Lukáš Krejčí
Katedra měření, FEL

Abstract

The aim of the presented bachelor's thesis is to design and implement a new simulation algorithm for traversing through timed automata networks in the simulation tool Taster. In the Introduction, the issues regarding integration testing and timed automata network language are presented. Next, the modelling language of the tool UPPAAL, which is used by Taster for Model-based testing, and the source code of the Taser timed automata simulation process are analysed. This is followed by an appropriate design of a new simulation algorithm architecture eliminating the shortcomings of the original version. After testing the design on a prototype, the new architecture is implemented into the tool. Finally, the validity of the created design is verified through a series of conducted experimental simulations.

Keywords: Taster, integration testing, timed automata, UPPAAL, HiL testing, SuT, Model-based testing

Title translation: Design and implementation of simulation algorithm for timed automata networks

Supervisor: Ing. Lukáš Krejčí

Obsah

1 Úvod **1**

2 Analýza **3**

2.1 Časované automaty 3
  2.1.1 Stavový automat 4
  2.1.2 Časová závislost 4
  2.1.3 Sítě časovaných automatů 4
2.2 Modelační nástroj UPPAAL 5
  2.2.1 Tvorba modelu 5
  2.2.2 Reprezentace času 6
  2.2.3 Stavy 7
  2.2.4 Přechody 8
  2.2.5 Synchronizační kanály 8
2.3 Nástroj Taster 9
  2.3.1 Současný stav 10
  2.3.2 Odlišnost sémantiky 11

3 Návrh algoritmu **13**

3.1 Objektový návrh 13
  3.1.1 Systém 13
  3.1.2 Automat a proměnná 14
  3.1.3 Synchronizace 15
  3.1.4 Stav a hrana 15
  3.1.5 Simulace a statistika 16
  3.1.6 Provázanost návrhu 17
3.2 Algoritmus kroku simulace 17
  3.2.1 Inicializace 18
  3.2.2 První fáze výběru 19
  3.2.3 Kontrola uváznutí 20
  3.2.4 Druhá fáze výběru 20
  3.2.5 Provedení přechodů 21
  3.2.6 Zmrazení času 22
  3.2.7 Závěrečná synchronizace 23
3.3 Algoritmus řízení testu 23
  3.3.1 Inicializace 23
  3.3.2 Simulace 24
  3.3.3 Zachycení výjimek 25
  3.3.4 Ukončení simulace 26
3.4 Prototyp algoritmu 26
  3.4.1 Zhodnocení prototypu 28

4 Implementace návrhu **31**

4.1 Objektová struktura 31
4.2 Načtení modelu 33
4.3 Krok simulace 35
4.4 Řízení testu 37
4.5 Testování 39
  4.5.1 Test přidaných funkcionalit . . 39
  4.5.2 Test zpětné kompability 40
  4.5.3 Test načtení modelu 40
  4.5.4 Zhodnocení 41

5 Závěr **43**

A Bibliografie **45**

Obrázky

2.1 Základní časovaný automat 4
2.2 Síť časovaný automatů 5
2.3 Prostředí nástroje UPPAAL..... 6
2.4 Plynutí času 7
2.5 Prioritizace stavů 8
2.6 Synchronizace dvou automatů ... 9
2.7 Prostředí nástroje Taster 10
2.8 Příklad problémové situace způsobené invariantem 11

3.1 Třída Systém 14
3.2 Třídy druhé úrovně 14
3.3 Třídy třetí úrovně 16
3.4 Třída řízení simulace a statistiky simulace 17
3.5 Diagram kroku simulace 18
3.6 Kontrola invariantu 18
3.7 První fáze výběru hrany 19
3.8 Kontrola uváznutí 20
3.9 Druhá fáze výběru hrany 21
3.10 Provedení přechodů 22
3.11 Kontrola stavů pro zmrazení času.......................... 23
3.12 Diagram řízení testu 24
3.13 Diagram simulace............ 25
3.14 Model použitý při testování prototypu..................... 27
3.15 Výstup simulace v prototypu .. 28
3.16 Upravená druhá fáze výběru hrany kroku simulace............ 28

4.1 Proces načtení modelu 33
4.2 Zachycení kroku simulace po vykonaní broadcast synchronizace . 39
4.3 Model ovládání stahování oken - zachycení výjimky invariantu 40

Tabulky

Kapitola 1

Úvod

V dnešní době stále komplexnějších elektronických systému je již standardem, že jednotlivé komponenty systému vznikají u různých výrobců. Odvětví automobilového průmyslu je toho zářným příkladem, kdy je ve vozech posledních generací integrováno na desítky řídicích jednotek [6] (angl. Engine Control Module, zkr. ECM) od mnoha výrobců. Např. dle Jamese Priyathama, analytika výzkumu automobilové elektroniky a polovodičů ve společnosti IHS Markit ": V jednom moderním luxusním vozidle dnes může být integrováno až 150 řídicích jednotek." [17]. I přes důkladné otestování jednotlivých komponent není při takovém počtu součástí systému od různých výrobců, kdy zadání výroby komponenty může být napříč výrobci odlišně interpretováno, zaručena bezchybná komunikace mezi jednotlivými jednotkami. Z tohoto důvodu se pro odhalení chyb v distribuovaných systémech využívá proces tvz. integračního testovaní, [9] v rámci kterého je aktuálním tématem modelově orientované testování [14] (angl. Model-Based Testing, zkr. MBT). Jeho základní myšlenkou je namísto ručního vytváření testovacích případů využít automatickou generaci testů z modelu [4]. Pro možnost testování řídicích algoritmů současně přímo na připojeném reálném sytému je využívána tvz. metoda "Harware-in-the-Loop" (zkr. HiL) [3, 10]. Jejím smyslem je ověřit plnění simulace prostřednictvím interakce mezi softwarem a periferním hardwarem [11]. Jako prostředek pro využití popsaného konceptu integračního testování metodou MBT s podporou HiL testování je na Katedře měření Fakulty elektrotechnické ČVUT v Praze výzkumnou skupinou ATG (Automotive Testing Group) pod vedením Ing. Jana Sobotky Ph.D. vyvíjen simulační nástroj Taster [12]. Tento nástroj využívá modely sítí časovaných automatů vytvořené a verifikované v nástroji UPPAAL, který vyvíjí švédská Uppsala University v kolaboraci s dánskou Aalborg University [1].

Vlivem několikaletého vývoje nástroje Taster s postupným rozšiřováním jeho simulačního algoritmu, jehož architektura nebyla připravena na další vývoj, se stal simulační algoritmus nesystematický a rozsáhlou kódovou duplicitou s používáním nevhodných programovacích technik nepřehledný. Cílem této práce tak je navrhnout zcela novou architekturu simulačního algoritmu implementující doposud nepodporované modelační prvky jazyku nástroje UPPAAL.

V úvodní části je nejprve provedena analýza problematiky sítí časovaných automatů, jejich interpretací v nástroji UPPAAL a průzkumem kódu dosavadní verze algoritmu ji implantující v simulačním nástroji Taster. Následně je navržena upravené verze datové struktury a zcela nová podoba architektury simulačního algoritmu. Výsledný návrh je otestován na prototypu simulačního algoritmu s použitím modelu pokrývající rozšířenou

sadu modelačních prostředků jazyku UPPAAL. Z výsledku experimentální simulace prototypu s porovnáním simulace shodného modelu v nástroji UPPAAL jsou vyvozeny úpravy v návrhu, které jsou dále zahrnuty do následné implementace algoritmu v nástroji Taster. Na závěr je provedeno několik simulací testujících jak kompatibilitu modelů vytvořených pro původní verzi algoritmu, tak ověřujících integraci nově podporovaných modelovacích prostředků nástroje UPPAAL.

Kapitola 2

Analýza

Modelově orientované testování je metoda, kdy je chování testovaného systému (angl. System under Test, zkr. SuT) ověřováno na základě formálního popisu systému modelem, který je následně využit pro generování samotných testů [8]. Takový způsob umožňuje komplexní testování celého systému. Jeden z možných modelovacích jazyků aplikující tuto metodu je popis testovaného systému sítí časovaných automatů. Nejprve je tedy v této kapitole uvedena základní definice časovaných automatů. Na tvorbu a verifikaci časovaných automatů a jejich sítí dále navazuje uvedení nástroje UPPAAL. Rozebrány jsou jeho modelovací prostředky a sémantika. Třetí část je již zaměřena na simulační nástroj Taster. Nejprve je popsán způsob, kterým Taster provádí testování modelu. Především je prozkoumána podoba jeho dosavadního algoritmu na průchod sítěmi časovaných automatů. V rámci této práce jsou uvedeny nedostatky stávající verze a naopak použitelné části při návrhu nového simulačního algoritmu.

2.1 Časované automaty

Simulace rozsáhlých systémů s časovými závislostmi pomocí časovaných automatů (které jsou speciálním typem stavových automatů) přináší několik výhod. Hlavní výhodou využití tohoto modelačního jazyku je schopnost intuitivního a snadného popisu systémů s časovanými závislostmi a možnost automatického generování komplexních testů celého systému. Oproti běžnému způsobu testování, při kterém se vytváří individuální test pro různé scénáře, formální popis systému časovaným automatem a jeho následnou verifikací eliminujeme riziko zanechání nepovšimnuté chyby v návrhu. Současně tento přístup generuje rozmanitější varianty testů, čehož je cestou individuálních testů náročné docílit. V neposlední řadě usnadňuje znovupoužití simulace, kdy například v automobilových distribuovaných systémech nemusí být zvlášť upraveny veškeré testy zahrnující upravenou část systému, ale stačí upravit samotný model, a testy z něj následně znovu vygenerovat. Dále navíc umožňuje např. opakovatelné použití již vytvořených bloků modelu systému k modelaci dalších jeho částí, čímž zefektivňuje fázi tvorby modelu. Popřípadě lze přímo použít části modelu předešlé generace systému při tvorbě modelu systému nového. V oblasti automotiv najdeme příklad takové situace při využití stejného typu dílu systému (např. světel či ovládacího voliče světel) v rámci různých modelových řad či generací automobilů, kdy tedy model takového dílu zůstává z valné části zachován. Těmito výhodami je celkově urychlena a zlevněna fáze testování.

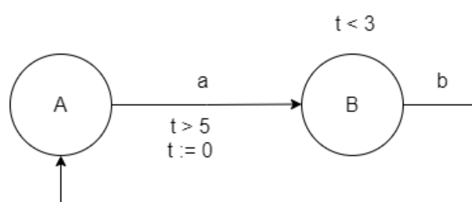
2.1.1 Stavový automat

Stavový automat je uspořádanou čtveřicí (S, s_0, U, H), kde S je množina stavů automatu, $s_0 \in S$ je počátečním stavem, U je množina událostí a $H \subseteq S \times S \times U$ je množina hran (přechodů) mezi stavy [2]. Pro potřeby této práce jsou dále uvažovány pouze konečné automaty. Stavový automat začíná ve svém počátečním stavu s_0. Svůj stav může změnit vykonáním přechodu (orientované hrany), který vychází z aktuálního stavu s. S vybraným přechodem ze stavu s do stavu s' se zároveň provede událost h odpovídací hrany. Následně automat změní svůj aktuální stav s na stav s'.

2.1.2 Časová závislost

Časované automaty jsou speciální kategorií stavových automatů, které jsou oproti automatům stavovým rozšířeny o časové závislosti. Získá se tak metoda popisu systému reálného času, u kterého je vyžadována reakce systému na podnět do konečného časového intervalu.

Pro reprezentaci toku času je zavedena proměnná typu hodiny. Čas ubíhá pouze ve stavech, přičemž iterační krok je nezáporný. Přechody mezi stavy jsou vykonány za nulový čas. Vykonání daného přechodu může být omezeno časovou podmínkou (tzv. stráž, angl. guard). Hodiny můžou být vynulovány událostí (tzv. přiřazení, angl. assignment) při přechodu. Setrvání ve stavu lze též omezit podmínkou (tzv. invariant). Časovaný automat může současně pracovat s vícero hodinami, které běží synchronně se stejným iteračním krokem. Na následujícím časovaném automatu je ukázán příklad práce s hodinami.



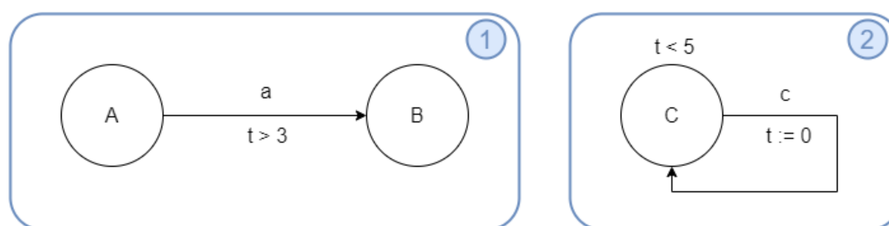
Obrázek 2.1: Základní časovaný automat

Automat začíná v počátečním stavu A s vynulovanými hodinami t. Odchod ze stavu A jedinou odchozí hranou a je omezen stráží $t > 5$, tedy automat setrvá ve stavu A minimálně po dobu 5 časových jednotek. Po uplynutí této doby může automat nadále setrvat ve stavu A (a to po neomezenou dobu) nebo použít pro odchod hranu a. V případě použití přechodu a jsou vynulovány hodiny přiřazením $t := 0$ (podporována je též verze $t = 0$ a následně automat změní svůj stav na B. Invariant stavu B dovoluje automatu v tomto stavu setrvat maximálně do $t < 3$. Před porušením invariantu tak automat musí opustit stav B přechodem b a změnit svůj stav na A.

2.1.3 Sítě časovaných automatů

Síť častovaných automatů (tzv. model systému) tvoří skupina časovaných automatů. Napříč automaty lze pracovat se společnými hodinami. Přechody mezi automaty mohou být svázány synchronizací (deklarováno jako událost hrany). Nicméně podrobněji je synchronizacím věnována kapitola 2.2.5.

Na následujícím obrázku jednoduché sítě časovaných automatů lze vidět globální užití hodin v rámci dvou automatů.



Obrázek 2.2: Síť časovaný automatů

Automat ① začíná ve stavu A, automat ② začíná ve svém jediném stavu C, hodiny t jsou při startu vynulovány. Automat ① setrvává ve svém počátečním stavu minimálně do času $t = 3$. Automat ② může kdykoliv (nejpozději do $t < 5$) využít přechodu c a vynulovat tak hodiny t. Při uplynutí $t = 3$ může automat ① nadále setrvat ve stavu A nebo použít přechod a. Přechod a je ovšem současně omezen automatem ② jen do maximálně $t < 5$, do kterého musí automat ② použít přechod c, vynulovat hodiny t a tedy znovu uzamknout přechod a automatu ①.

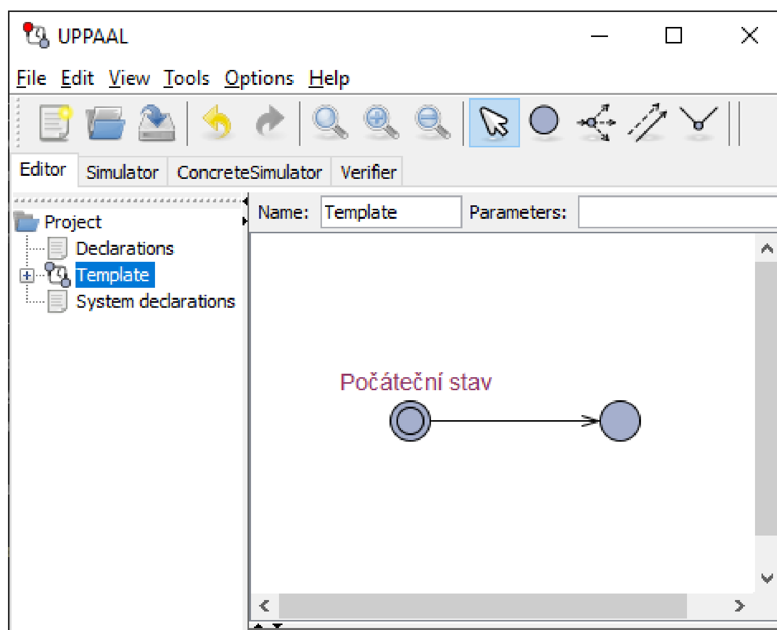
Nicméně, rozbor validace časovaných automatů a jejich sítí není cílem této práce. Proto je pro tvorbu validních časovaných automatů dále využíván nástroj UPPAAL.

2.2 Modelační nástroj UPPAAL

UPPAAL je nástroj pro validaci a verifikaci systému reálného času, který rozšiřuje časované automaty a jejich sítě o proměnné a přidává další speciální vlastnosti uzlům a hranám. V této kapitole je uveden potřebný základ jeho modelačních možností jakožto typy stavů (uzlů), přechodů (hran), synchronizačních kanálů a práci s proměnnými a časem tak [5].

2.2.1 Tvorba modelu

V nástroji UPPAAL je systém modelován jednotlivými automaty (nazývány šablony, angl. templates) pomocí grafického editoru. Celý systém je definován systémovou deklarací (využívanou pro následnou verifikaci modelu), deklarací globálních proměnných a jednotlivými automaty. Každý automat má definovaný počáteční stav (graficky znázorněn vnitřní kružnicí) a obsahuje deklaraci svých lokálních proměnných. Invariant a stráž tvoří logické výrazy využívající lokální i globální proměnné systému. Výsledný model systému je serializován do formátu Extensible Markup Language (zkr. XML) [16].



Obrázek 2.3: Prostředí nástroje UPPAAL

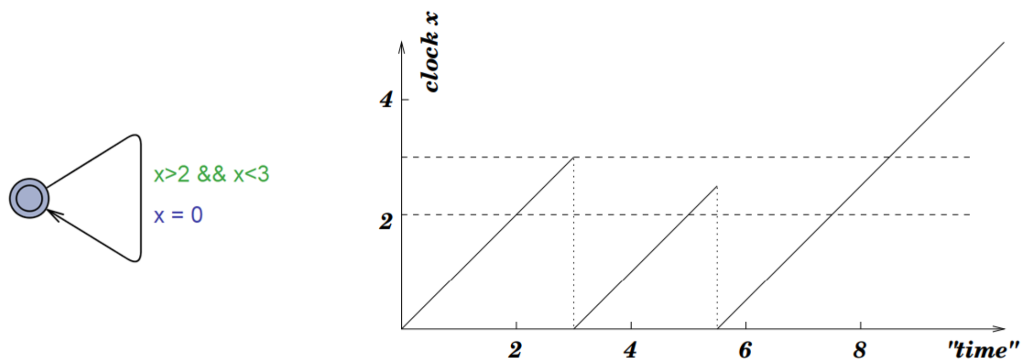
Na obrázku 2.3 je zachyceno prostředí nástroje UPPAAL. V horní části je vidět nabídka nástrojů (v této konkrétní zachycené podobě se např. v její pravé části nachází prvky pro modelaci automatů). Pod nabídkou nástrojů jsou dále záložky s editorem, simulátory a verifikátorem. Konkrétně otevřená záložka editoru je určena pro tvorbu automatů a jejich procházení. V dolní levé části se nachází strom projektu (deklarace globálních proměnných, automaty, jejich lokální deklarace proměnných a systémová deklarace) a v pravé prostor editování (v zachycené situaci otevřeného automatu s názvem "Template" tak jeho grafické vyobrazení. Simulátory jsou určeny pro grafické procházení modelovaných automatů a verifikátor pro zjištění, zda v modelu nemůže nastat např. "deadlock", kdy se model dostane do patové situace a není možná jakákoliv další změna stavu.

2.2.2 Reprezentace času

UPPAAL simuluje kontinuální plynutí času pomocí proměnné typu hodiny. Dva po sobě jdoucí přechody (nejsou-li speciálně nijak časově omezeny) proběhnou obecně v náhodně dlouhém časovém rozmezí. Tím je simulována možnost systému neprovést přechod a setrvat ve stavu bez akce.

Hodiny jsou definovány klíčovým slovem "*clock*". Při modelování lze pracovat jak s globálním časem - definovaným v globálních deklaracích systému, tak s lokálním časem automatu - definovaným v jeho lokálních deklaracích. Nicméně v obou případech plyne čas stejnou rychlostí, tedy jsou hodnoty všech hodin iterovány ve stejný moment stejným nezáporným iteračním krokem. Časová omezení lze aplikovat u stavů (invarianty) a přechodů (stráže). Stav hodin je možné resetovat pomocí přiřazení při přechodu mezi stavy.

Následující případ předvádí chování hodin u jednostavového automatu s hodinovou proměnnou x a hranou podmíněnou do časového úseku, která při použít resetuje stav hodin.



Obrázek 2.4: Plynutí času [7]

Na záznamu hodin x jsou zvýrazněny dvě významné úrovně. První úroveň $x = 2$ značí hodnotu hodin, do které je přechod ze stavu blokován. Druhá úroveň $x = 3$ značí úroveň hodin, po které je přechod blokován. Tedy stráž přechodu pouze omezujeme, že přechod je proveditelný jen ve vytyčeném časovém rozmezí 2-3 časových jednotek. Ovšem jak lze na záznamu vývoje hodin x pozorovat, k přechodu tak může dojít v náhodný okamžik ve vytyčeném časovém rozmezí, či v takto definovaném automatu k přechodu vůbec dojít nemusí.

2.2.3 Stavy

Stav (v nástroji UPPAAL pojmenováno "*location*") je definován svým jménem, logickým výrazem invariantu a typem. Možné jsou tři typy uzlů podle jejich priority.

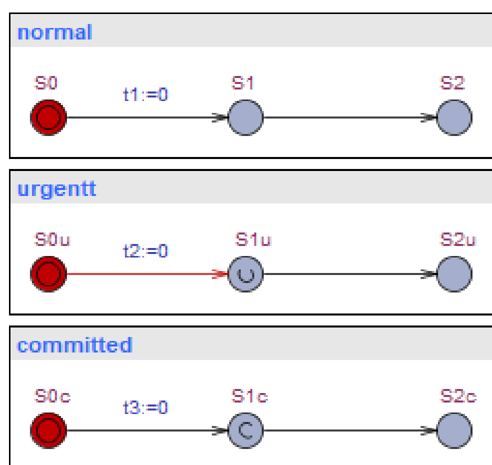
- **Normal** - Pokud se sytém nachází pouze ve stavech typu normal, běží hodinové proměnné a každý ze stavů má stejnou prioritu pro výběr následujícího přechodu.

- **Urgent** - střední priorita. Pokud se alespoň jeden z automatů systému nachází v urgent stavu, hodinové proměnné jsou po dobu setrvání v tomto stavu zastaveny - je vyžadována okamžitá reakce. Každý ze stavů má nadále stejnou prioritu pro výběr následujícího přechodu. Stav s urgent prioritou je označen písmenem "U".

- **Committed** - nejvyšší priorita. Pokud se alespoň jeden z automatů systému nachází v committed stavu, hodinové proměnné jsou stejně jako v případě urgent stavu po dobu setrvání v tomto stavu zastaveny - je vyžadována okamžitá reakce. Navíc následující hrana musí vycházet z aktivního committed stavu. Stav s committed prioritou je označena písmenem "C".

Počáteční stav je označen klíčovým slovem "*initial*". Připojit lze též komentář. Dále je stav pro potřebu simulace definován unikátním identifikátorem a pro grafické zobrazení svou polohou a polohami popisů (jména, invariant, komentář).

Jako příklad si předveďme situaci tří automatů začínajících v normálních stavech $S0$, z nich každý má své vlastní lokální hodiny t.



Obrázek 2.5: Prioritizace stavů

Při setrvávání automatů v počátečních stavech se iterují stejným krokem všechny lokální hodiny $t1 - t3$. Proveditelné jsou všechny tři aktuální výchozí hrany z počátečních stavů a to se shodnou prioritou. Jako první situaci uvažujme vybrání přechodu do normálního stavu $S1$. V takovém případě se situace nemění, hodiny jsou při setrvání ve stavech iterovány a není prioritizována žádná odchozí hrana (do stavů $S2$, $S1u$ a $S1c$)

Jako druhou možnost uvažujme vybrání přechodu do urgentního stavu $S1u$. Po dobu setrvání v tomto stavu jsou všechny hodiny $t1 - t3$ zastaveny. Nicméně stále není prioritizována žádná z odchozích hran (do stavů $S1$, $S2u$ a $S1c$)

Jako poslední možnost uvažujme vybrání přechodu do committed stavu $S1c$. Po dobu setrvání v tomto stavu jsou (stejně jako v případě urgentního stavu) všechny hodiny $t1-t3$ zastaveny. Navíc je však vyžadováno, aby následující odchozí hrana byla z committed stavu. Tedy nyní je jediný přípustný přechod do stavu $S2c$.

2.2.4 Přechody

Přechod (v nástroji UPPAAL pojmenováno "*edge*"), jakožto orientovaná hrana, je definovaný svým počátečním a koncovým uzlem. Může být podmíněn stráží (logickým výrazem) a listem přiřazení, které jsou při přechodu provedeny. Dále může mít přidělen synchronizační kanál, kterým lze provázat jednotlivé automaty systému. K přechodu lze navíc též přidat komentář skrze který je mimo jiné možné např. předat nástavbovému modelu dodatečné informace (v nástroji Taster využito pro další specifikování přechodu, a to přidání relevance, podrobněji v následující kapitole 3).

Pro grafické zobrazení jsou dále definovány polohy popisů (stráž, přiřazení, synchronizační kanál, komentář) a pozice grafického zalomení hrany tvz. hřebíků (ang. nails).

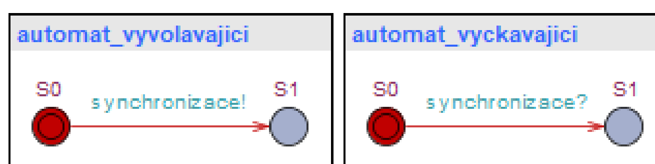
2.2.5 Synchronizační kanály

Synchronizační kanály jsou globálně definovány klíčovým slovem "*chan*", jménem a symbolem rozlišující zda se jedná o hranu vyvolávající synchronizaci (označení symbolem "!" za jménem kanálu) nebo hranou podřízenou (označení symbolem "?"), která vyčkává

na vyvolání synchronizace. Mechanismus synchronizace je realizován metodou handshaking [13], tedy vybrané synchronní hrany jsou provedeny ve stejný virtuální časový moment. Pokud je k dispozici hrana vyvolávající synchronizaci, ale není k ní umožněna žádná odpovídající hrana na synchronizaci vyčkávající (např. z důvodu nesplnění podmínky stráže), není možné provést samostatně ani hranu synchronizaci vyvolávající. Rozlišujeme následující dvě specializace kanálů.

- **Urgentní kanál** - pokud je k dispozici proveditelná jak hrana vyvolávající synchronizaci ("!") tak k ní odpovídající hrana vyčkávající ("?"), pak musí být tato synchronizace přednostně provedena bez časového zpoždění. Urgentní kanál je definován předřazením klíčového slova "*urgent*". Časové podmínky nejsou na těchto hranách povoleny.
- **Broadcast kanál** - při vyvolání broadcast synchronizace jsou provedeny všechny proveditelné hrany vyčkávající na synchronizaci ("?"). Broadcast kanál je definován předřazením klíčového slova "*broadcast*".

Kanál může být současně urgentní i broadcast. Taková definice kanálu pak má jedinou přípustnou variantu pořadí klíčových "*urgent broadcast chan*" a navazujícím názvem kanálu.

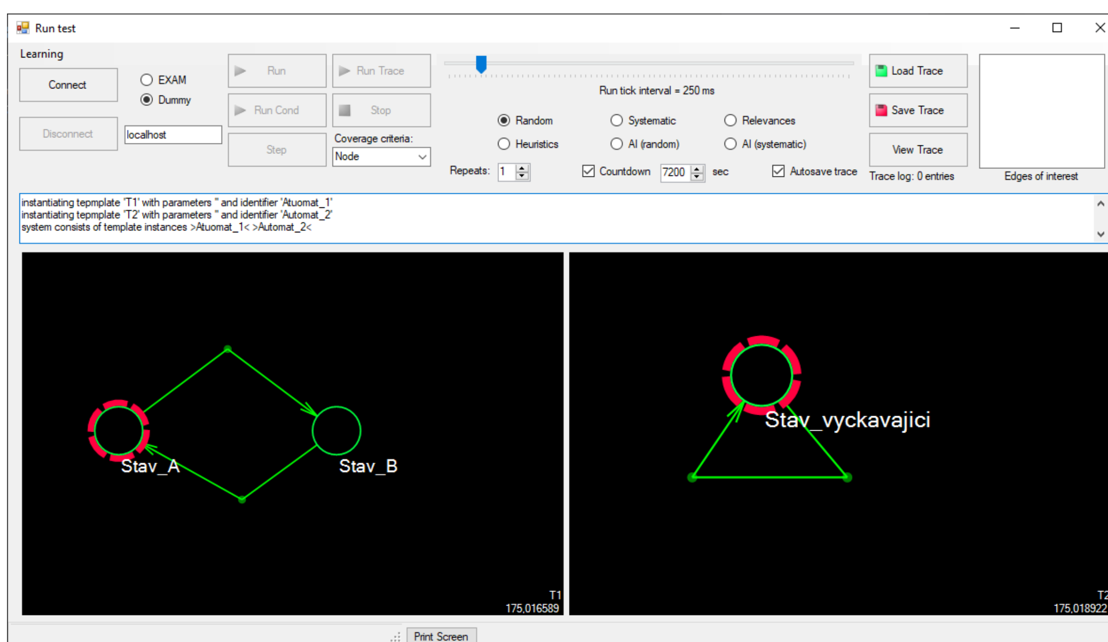


Obrázek 2.6: Synchronizace dvou automatů

V levém automatu na obrázku 2.6 vychází z počátečního stavu $S0$ hrana synchronizaci vyvolávající. V pravém automatu vychází z počátečního stavu k ní odpovídající hrana na synchronizaci vyčkávající. Ani jedna z hran není blokována žádnou podmínkou, tedy může dojít k přechodům do stavů $S1$, které v obou automatech proběhnou současně za nulový čas.

2.3 Nástroj Taster

Nástroj Taster, především vyvíjený pro automotiv k testování řídících jednotek v automobilu, implementuje myšlenku využití časovaných automatů pro simulaci chování modelovaného systému. Cílem jeho vývoje je propojit výhody testování metodou MBT s metodou HiL a vytvořit tak platformu pro výzkum a vývoj MBT metod. Jedná se o simulační nástroj umožnující využití odlišných strategií při procházení sítěmi časovaných automatů. Aktuálně podporované strategie jsou například náhodný výběr, systematické procházení, procházení dle relevance stavů, či využití strategií založených na strojovém učení. Pro simulaci využívá modely vytvořené v nástroji UPPAAL, které načítá ve formátu XML. Umožňuje jak čistě virtuální testování modelu, tak testování způsobem HiL, kdy je test prováděn přímo na připojeném reálném zařízení (např. automobilu). Průběh testovaní je současně během simulace graficky znázorňován.



Obrázek 2.7: Prostředí nástroje Taster

V horní části obrázku je vidět ovládání simulace. Zde je možné např. vybrat připojený adaptér, spustit i zastavit test, pomocí voliče nastavit velikost iteračního kroku hodin či zvolit typ strategie. Pod částí ovládání se nachází výpis hlášení posledních provedených akcí. V dolní části je jednotlivými automaty vyobrazen načtený model. Červeným přerušovaným kruhem je zvýrazněn aktuální stav, ve kterém se každý z automatů nachází.

2.3.1 Současný stav

Strukturu nástroje Taster lze rozdělit do dvou hlavních částí - TASystem věnující se vytvoření modelu z načteného XML souboru a TARuntime řízení testu a simulace. Předmětem této práce je pak především část řízení testu a především samotný simulační algoritmus. Původní architektura kroku simulace nepočítala s navazujícím vývojem, což při implementaci rozšíření vyústilo v rozsáhlou kódovou duplicitu a nesystematickou strukturu kódu. Špatné čitelnosti algoritmu též dopomáhá využívání nevhodných programovacích technik, jakou jsou např. skoky v kódu pomocí návěstí, které tak rozbíjejí kontinuálnost návrhu. To vše ve výsledku již prakticky neumožňuje jakékoliv další rozšiřování.

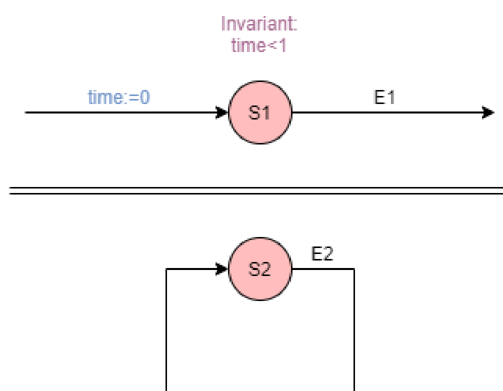
Hlavním cílem tak je vytvoření zcela nové verze systematického návrhu algoritmu s hierarchicky navázanou objektovou strukturou, která svou čitelností i do budoucna umožní další vývoj. Požadovaným vylepšením je dále přidání plné podpory synchronizačních kanálů, tedy synchronizací typu urgent a broadcast, jejíchž začlenění v aktuální verzi zcela chybí. Pro zlepšení čitelnosti kódu bude též výběr hrany (strategií) oddělen od samotného řízení simulace (kroku simulace) jakožto volaná funkce, která vrátí jednu zvolenou hranu k provedení. Současně v novém návrhu řízení testu bude též zohledněno automatické spouštění další iterace simulace a eliminováno dosavadní zahájení testu emulováním stisku uživatele.

Pro implementaci lze naopak využít dosavadní verzi syntaktického a lexikálního analyzátoru. Část načtení modelu s XML parserem bude vyžadovat určité úpravy zapříčiněné

především změnami v objektovém návrhu a kompletní podporou synchronizačních kanálů.

2.3.2 Odlišnost sémantiky

Po srovnání práce s časem v nástroji UPPAAL a v nástroji Taster byla odhalena zásadní sémantická odlišnost. Nástroj UPPAAL během simulace umožňuje neprovedení žádného kroku (tedy setrvání v aktuálním stavu) v nulovém čase. Současně pro dodržení invariantu omezující daný stav časovým omezením, přizpůsobí inkrementační krok hodinových proměnných tak, aby invariant daného stavu nebyl inkrementací porušen. Oproti tomu nástroj Taster pracuje s předem pevně určeným inkrementačním krokem a nemůže jej upravit. Tedy pokud se nenachází v urgentních stavech (typu urgent či committed), je čas vždy inkrementován shodným krokem. Problémovou situaci zachycuje následující úryvek modelu.



Obrázek 2.8: Příklad problémové situace způsobené invariantem

Mějme model dvou automatů, kdy poslední uskutečněný krok byl proveden v horním automatu vstupem do stavu S1, přičemž byly vynulovány hodiny. Druhý automat se nachází ve svém jediném stavu S2. Oba stavy S1 a S2 jsou normálními, tedy čas není zastaven. Nyní jsou možné dvě hrany - E1 a E2.

V případě simulace modelu v nástroji UPPAAL může dojít libovolně krát k vykonaní hrany E2. Pokaždé k inkrementaci času může i nemusí dojít. Inkrementační krok navíc nemusí být konstantní. Jedinou omezující podmínkou velikosti inkrementačního kroku je invariant, podle kterého se náležitě upraví. Stále je však jistotou, že k vykonání hrany E1 zaručeně dojde do hodinové hodnoty 1.

V případě simulace modelu nástrojem Tater je situace diametrálně odlišná. Předem je definována velikost inkrementačního kroku hodin. Uvažujme krok roven 1. Tedy za takovýchto podmínek by měla být jediná možná hrana, a to E1. Nicméně invariant je kontrolován pouze před výběrem hrany, kdy porušen není, tedy nic nebrání volbě jak hrany E1, tak hrany E2. V tomto případě není žádná z hran nijak prioritizována, a není tak předem jasné, která z hran se ve finále zvolí, a zdali dojde po vykonání hrany k porušení invariantu (při volbě hrany E2) či se zvolí hrana E1.

Jelikož Taster neslouží k verifikaci modelu, je nutné s touto odlišností počítat již při tvorbě modelu a model navrhnout tak, aby nemohlo dojít k nedeterministickým situacím. Nicméně v rámci Tasteru jsou nejsou invarianty z praktického hlediska využívány na

kontrolu hodin, ale pro kontrolu stavu/situace s reálným modelem. Uvažujme například model pátých dveří u osobního automobilu s motorizovaným ovládáním, které testujeme s připojeným reálným modelem (HiL testování). Při otvírání dveří, a tedy např. stavu otvírraní značící tuto akci, by nebyl v tomto stavu invariant přímo využit pro zjištění, zda se již dveře neotvírají příliš dlouho (např. z důvodu selhání motoru). Tato kontrola by byla zajištěna separátním automatem pro stav otvírání. Invariant by naopak mohl být využit pro kontrolu, zda se stav simulace shoduje s reálným stavem, tedy zda při otevřených dveřích v rámci simulace jsou dveře otevřeny i ve skutečnosti. Případně může být invariant využit jakožto nástroj pro zastavení simulace a vyvolání chyby ve stavu značící selhání. Při takovémto smyslu použití invariantu se neprojeví odlišnost v sémantice a zajistí tak planost modelu pro Taster.

Kapitola 3

Návrh algoritmu

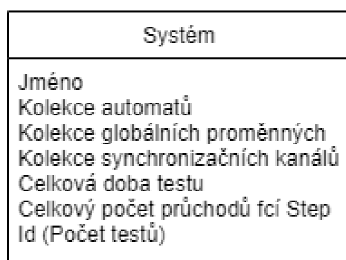
Závěr předchozí kapitoly a především pak popsaná podoba Tasteru a nedostatků jeho simulačního algoritmu v podkapitole 2.3.1 ujasňuje motivaci této práce. Následující kapitola se tak věnuje návrhu vhodně upravené objektové struktury a návrhu simulačního algoritmu, tedy algoritmus kroku simulace a algoritmus řízení testu. Následně byl dle návrhu vytvořen a vyhodnocen prototyp, testující funkčnost nové verze. Ze závěrů prototypu byly vyvozeny úpravy následně uvažované při implementaci v navazující kapitole 4.

3.1 Objektový návrh

Jedním z hlavních problémů u původní verze simulačního algoritmu jsou nedostatky v objektové struktuře. Vlivem postupného nesystematického rozšiřování původní verze, kdy kvůli neefektivnosti původní architekty bylo přistupováno ke kódové duplicitě, se stal kód nepřehledný, a tím další vývoj této verze poměrně náročný. Proto byl při návrhu verze nové kladen důraz na eliminaci zásadních nedostatků objektové struktury. Příkladem může být zlepšení podpory synchronizačních kanálů tak, aby nová objektová struktura umožňovala snadné a přehledné implementování podpory urgentních a brodcast kanálů. Zvolena tak byla provázaná struktura, kdy jednotlivé prvky systému jsou tvořeny třídami, jejíž volba přináší několik značných výhod. Využití tříd umožňuje intuitivní přistup jak k datům skrze celý systém, tak k operacím s daty vztahujícím se k dané třídě. Současně využitím tříd lze docílit minimálního datového objemu, tedy vyhnout se zbytečné duplicitě, kdy je provázanost objektové struktury řešena referencemi jednotlivých prvků. Použití tříd zároveň umožňuje i kýženou nenáročnou rozšiřitelnost. Jednotlivé třídy jsou hierarchicky řazené, dle smyslu stavby sítí časovaných automatů.

3.1.1 Systém

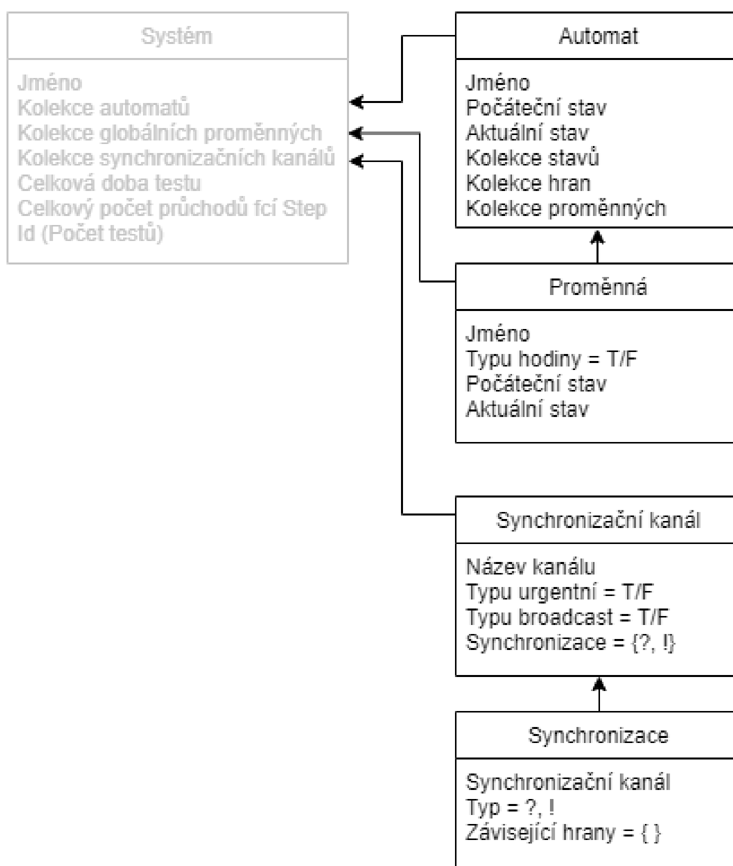
Nejvyšší třídou je třída celého Systému (obr. 3.1). Dle nástroje UPPAAL systém obsahuje kolekce kolekce tříd reprezentující automaty, globální proměnné a synchronizační kanály (schéma třídy systému na obr. 3.1). Dále obsahuje záznamové proměnné pro vedení statistiky simulace jako jsou např. celková doba běhu testu (respektive virtuální čas dle nastaveného inkrementačního kroku hodin) či celkový počet vykonaných kroků simulace.



Obrázek 3.1: Třída Systém

3.1.2 Automat a proměnná

Navazujícími třídami nižší úrovně jsou třídy automatu, proměnné, synchronizačního kanálu a synchronizace (navázání tříd automatu a proměnné na třídu systému na obr. 3.2). Každý automat je definován svým unikátním jménem a obsahuje kolekce svých stavů, hran a proměnných. Pro opětovnou inicializaci testu uchovává svůj počáteční stav. Současně nese informaci o aktuálním stavu, ve kterém se daný automat právě nachází.



Obrázek 3.2: Třídy druhé úrovně

Třídu proměnné definuje její jméno a kromě aktuálního stavu/hodnoty, uchovává též svůj počáteční stav (pro snadnou inicializaci nového testu). Speciálně pro přímočařejší práci s časem je navíc rozlišeno, zdali se jedná o proměnnou typu hodiny.

3.1.3 Synchronizace

Třetí třídou druhé úrovně je synchronizační kanál (navázání třídy synchronizace na třídu systému na obr. 3.2). V rámci nástroje UPPAAL je sice synchronizace definována mezi globálními proměnnými, nicméně pro přehlednost a značnou rozdílnost od ostatních typů proměnných je připravena separátní třída. Synchronizační kanál je definován svým názvem a obsahuje kolekce všech synchronizací (jak vyvolávajících, tak vyčkávajících), které daný kanál využívají. Dále je určeno, zda se jedná o urgentní kanál a současně zda se jedná o kanál typu broadcast.

Na synchronizační kanál je přímo navázaná třída synchronizace (vzájemné navázávaní na obr. 3.2). Ta je definována synchronizačním kanálem, kterého se synchronizace týká a typem akce. Tedy zdali se jedná o synchronizaci vyvolávající či vyčkávající na vyvolání (značeno popořadě znaky "!", "?"). Dále obsahuje kolekci závisejících hran. V případě synchronizace vyvolávající tak obsahuje veškeré hrany se synchronizacemi shodného kanálu ostatních automatů, které na danou synchronizaci vyčkávají. Přes synchronizaci tak hrana nese přímo informaci o na ni závisejících hranách. S původní objektovou strukturou bylo při vyvolání synchronizace nutné opětovně prohledávat veškeré hrany systému. S obohacením objektové struktury o synchronizace přímo odkazující na závisející hrany se značně usnadní datové prohledávání při hledání párových synchronizačních hran a urychlí se tak provedení kroku simulace.

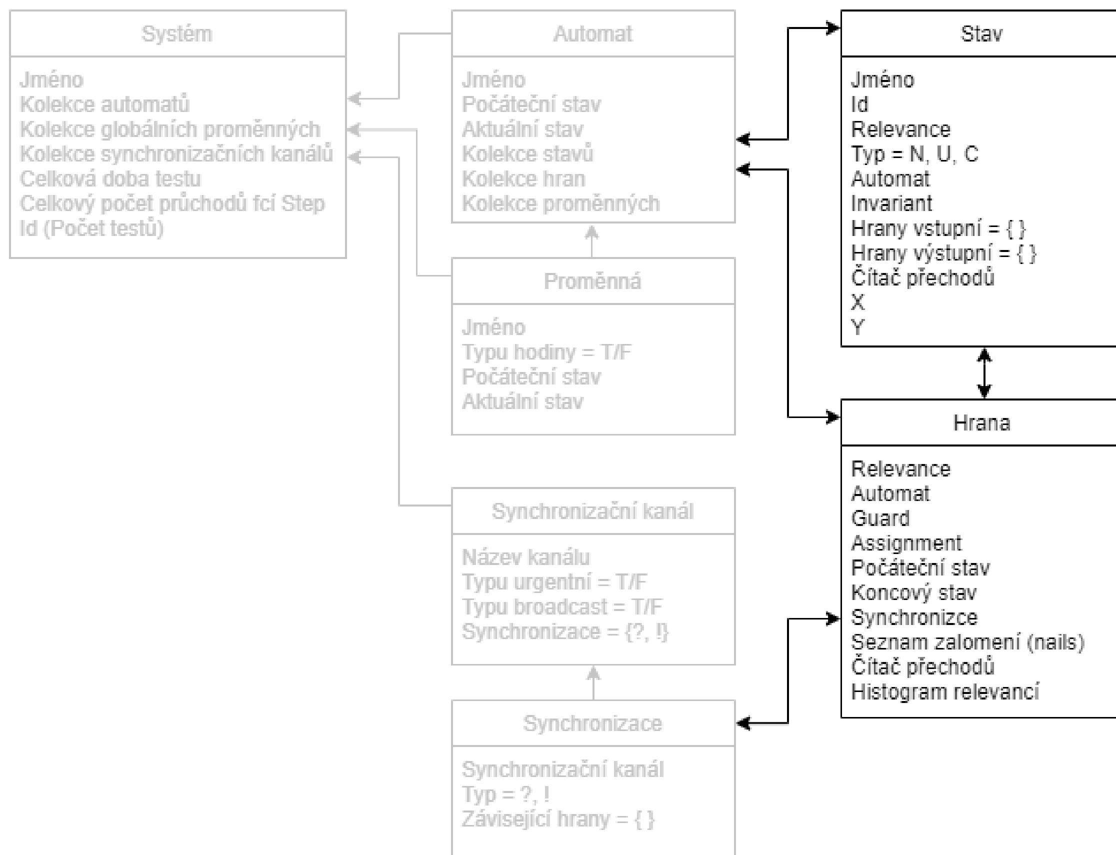
Přibližme si, jak spolu souvisí synchronizační kanál a synchronizace. Uvažujme dvě hrany nacházejících se v odlišných automatech systému, které jsou vzájemně synchronizačně vázány. Jejich společnou vazbou bude synchronizační kanál názvu např. "pošťák". Vlastností první hrany pak bude synchronizace kanálu "pošťák", která synchronizaci vyvolá. Tedy tato hrana bude typu "!" a celá definice synchronizace pak zní "pošťák !". Druhá hrana bude na první vyčkávat, tedy definice její synchronizace bude "pošťák ?". Ač jsou definice obou synchronizací odlišné, obě využívají stejný kanál "pošťák", který je propojuje a který by je mohl spojovat se všemi hranami, tento kanál taktéž využívající.

3.1.4 Stav a hrana

Na třídu Automat navazujícími třídami jsou Stav a Hrana (navázání tříd stavu a hrany na třídu automatu na obr. 3.3). Třída stavu je definována svým unikátním názvem a současně typem. Tedy zdali se jedná o stav normální, urgentní či committed (značeno znaky popořadě "N", "U", "C"). Pro provázanost a snadnou datovou průchodnost obsahuje zpětnou referenci na instanci třídy automatu, kterému náleží. Dále obsahuje kolekce výstupních a vstupních hran. Každý stav může být navíc kontrolován podmínkou invariantu (logickým výrazem). V rámci držení statistiky průběžně zaznamenává počet průchodů daným stavem. Pro potřeby uvažování významnosti daného stavu je každému stavu přidělena relevance (informace využitelná strategií výběru při volbě hrany). Jedná se o číslo v rozmezí 1-9, kdy hodnota 1 je pro nejméně důležitý přechod a 9 pro nejvíce relevantní. V případě nepřiřazení relevance je hodnota nulová.

Relevance má několik možných použití. Skrze zvýšenou důležitost hran lze například upřednostnit část modelu, o které je známo, že je často problémová. Případně je možné se tímto způsobem přednostně zaměřit na poslední přepracovanou část modelu. Při testování např. automobilu pak má relevance další podstatné použití. Vyšší relevancí je možné při testování důkladněji zaměřit na části systému, které by byly svou poruchou

potenciálně život ohrožující. Například při blikání blinkrem je podstatnější rozblikání samotných venkovních signalizačních světel, před rozblikáním informačního symbolu na přístrojové desce. Dalším možným použitím je testování částí systému, které by při selhání nepříjemně snižovaly komfort každodenního užívání. Příkladem můžou být páté dveře s motorizovaným pohonem otvírani. Uvažujme tři různé způsoby otvírání kufru, které se u vozů posledních generací běžně vyskytují. Konkrétně první způsob otevření přes volič na ovládacím panelu u řidiče, dále dálkové otevření přes spínač na klíčcích a třetí způsob otevření přes venkovní spínač nacházející se přímo na dveřích kufru. Přitom právě poslední zmíněná možnost bude v praxi nejčastěji voleným způsobem otevření a její porucha by užívání pátých dveří citelně znepříjemnila.



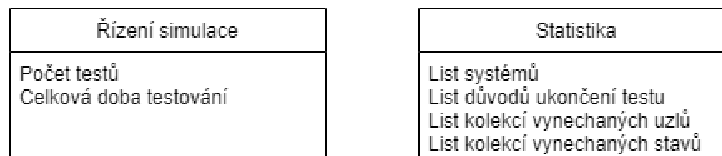
Obrázek 3.3: Třídy třetí úrovně

Druhou zmíněnou třídou je Hrana. Hrana je definována počátečním a koncovým stavem. Provázanost zajišťuje odkaz na automat, ve kterém se hrana nachází. Obsahuje logické výrazy podmínek stráže a přiřazení do proměnných. Opět pro upřednostnění podstatnějších hran při výběru je každé hraně přidělena relevance. Pro statistku a strategie je i u hrany čítač využití dané hrany. V případě synchronizační hrany obsahuje třídu Synchronizace.

3.1.5 Simulace a statistika

Separátními třídami pro uchování záznamu testování jsou třída řízení simulace a statistika (obr. 3.4). Ta pro zpětné vyhodnocování průběhu testování může obsahovat např. list tříd

systému (kopie z konce testování před resetováním systému pro opakování testu). Díky objektové struktuře a provázanosti je tak již zaručen přístup ke kompletnímu detailnímu záznamu výsledku testu a všech jeho částí. Dále může uchovávat např. seznam důvodů ukončení každého z testu (např. vypršení času, pokrytí všech stavů či chybovou hlášku) popřípadě např. list vynechaných hran a uzlů.



Obrázek 3.4: Třída řízení simulace a statistiky simulace

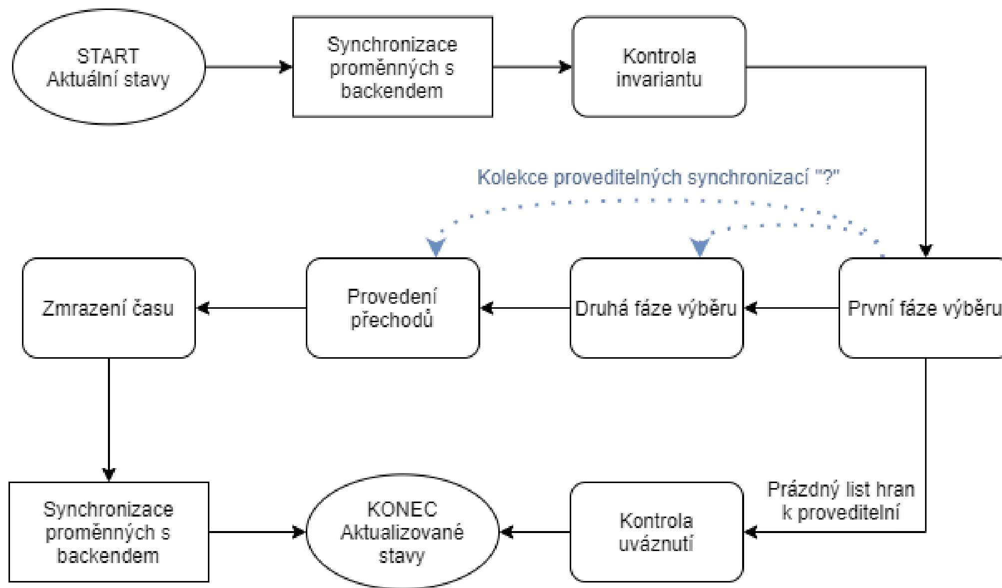
3.1.6 Provázanost návrhu

Výraznou změnou v původní objektové struktuře je kompletní provázanost tříd. Ta je na diagramu 3.3 znázorněna obousměrnými šipkami mezi druhou a třetí úrovní. Tedy každý stav zná kromě svých vstupních a výstupních hran také odkaz na automat, ve kterém se nachází. Přes automat je tak cesta ke všem stavům, hranám a proměnným daného automatu. Totožnou provázaností disponuje též každá hrana systému. Tímto způsobem lze při simulaci co nejefektivněji procházet systémem a je umožněno rychlé vyhodnocení dalšího kroku simulace. Zároveň je pro strategie připraven snadný přímý přístup k veškerým informacím konkrétní konstelace systému.

Mírnou nevýhodou takové provázanosti je složitější sestavení systému při prvotním parsování modelu z XML souboru. Jelikož např. při sestavování stavu je třeba znát vstupní i výstupní hranu a současně každá hrana zná svůj cílový a počáteční stav, je nutné sestavování částí modelu provést ve více krocích a odkazy zpětně doplnit. Nicméně jelikož je sestavení systému provedenou pouze jedenkrát při načtení modelu, klady takové objektové struktury především při několika opakovaném simulovaní modelu převyšují zmíněnou nevýhodu.

3.2 Algoritmus kroku simulace

Algoritmus vykonávající krok simulace (tedy průchodu modelem systému) byl navržen tak, aby byl snadno čitelný, přehledný a systematicky provedl celý proces výběru a uskutečnění dalších přechodů v systému. Efektivní vytřídění všech odchozích hran všech aktuálních stavů systému je založeno na využití výhod předělané objektové struktury. Na obr. 3.5 je vyobrazen digram algoritmu kroku simulace. V další podkapitole jsou postupně rozebrány jednotlivé kroky algoritmu s detailním rozkreslením daných částí.

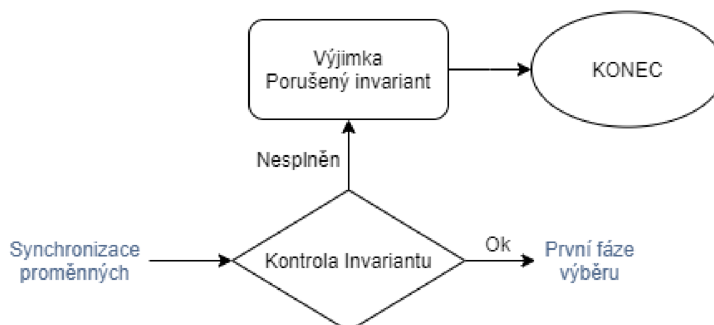


Obrázek 3.5: Diagram kroku simulace

3.2.1 Inicializace

Vstupní informací algoritmu jsou aktuální stavy modelu, ve kterých se systém zrovna nachází. Před zahájením vykonávání samotné sekvence kroku simulace, je nejprve uskutečněna synchronizace s připojeným systémem. Jsou tak vyčteny hodnoty proměnných reálného systému a jimi aktualizovány hodnoty virtuálních proměnných modelu. Následně dojde ke kontrole invariantů aktuálních stavů modelu, které reprezentují sestavu systému, ve které se zrovna simulace nachází, čímž je skrze model ověřena validita reálného systému v aktuální sestavě. Jako příklad uveďme situaci testovaní na připojeném modelu automobilu, kdy se ověřuje, zda jsou otevřené dveře v simulovaném modelu současně otevřené i u reálného automobilu.

V případě porušení invariantu je vyvolána výjimka (obr. 3.6), pomocí které je vyšší úrovni řízení simulace předána informace o chybě, kterou již následuje ukončení testu (podrobněji v podkapitole 3.3). Po této úvodní části navazuje několik kroků provedených čistě virtualizovanou podobou systému - modelem, kdy je již přistoupeno k vyhodnocování aktuálních stavů a výběru hran k provedení.



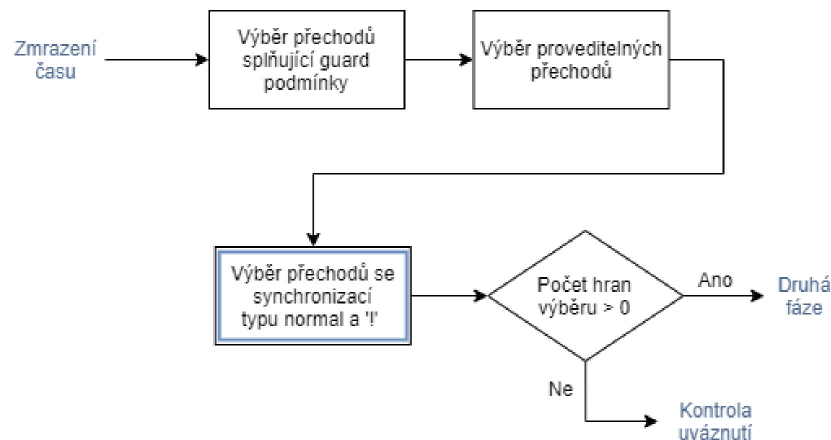
Obrázek 3.6: Kontrola invariantu

3.2.2 První fáze výběru

V prvním kroku fáze výběru se projdou veškeré odchozí hrany z aktuálních stavů a vytvoří se kolekce odchozí hran, které splňují podmínku hlídače. Následně se z této kolekce vyřadí hrany se synchronizačním kanálem, které v této kolekci nemají svou párovou hranu, a to z jiného automatu, než ve kterém se nacházejí.

Příkladem uvažujme odchozí hranu z aktivního stavu automatu A1 synchronizačního kanálu SK se synchronizací typu "!", která není omezena hlídačem. Pokud ale nebude v kolekci vybraných hran též hrana stejného synchronizačního kanálu SK typu "?" nacházející se v jiném automatu než A1, bude tato hrana vyřazena. Stejně tak i obráceně, pokud hrana s vyčkávající synchronizací nemá v jiném automatu svůj protipól, je též vyřazena. Díky objektové struktuře, kdy synchronizace hrany zná přímo své závisející hrany, je možné snadno přímo provést vyřazení osiřelých hran. Touto selekcí se z kolekce odchozích hran vytvoří tzv. výběr proveditelných přechodů.

Dalším krokem je rozdělení kolekce proveditelných hran na kolekci zvolitelných hran a kolekci hran s vyčkávající synchronizací. V kolekci zvolitelných hran tak zůstávají přechody bez synchronizace a přechody obsahující synchronizaci typu "!". Naopak druhá kolekce obsahuje pouze hrany se synchronizacemi typu "?". Druhá zmíněná kolekce bude využita v pozdějších fázích simulace (v druhé fázi výběru a případně též ve fázi provedení přechodu). Proto je krok na obr. 3.7 zvýrazněn modrým obdélníkem a jeho dopad též zvýrazněn modrými přerušovanými šipkami v přehledu celého algoritmu kroku simulace na obr. 3.5.



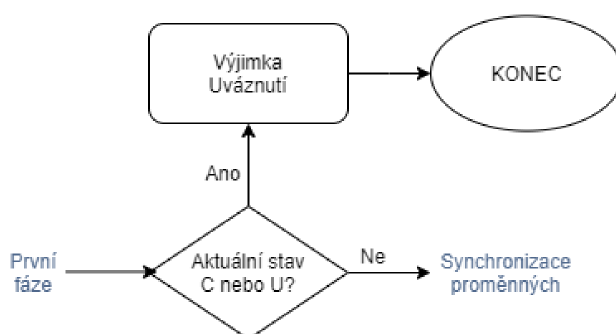
Obrázek 3.7: První fáze výběru hrany

Výsledkem první fáze výběru je tak kolekce odchozích hran obsahující proveditelné přechody bez podřízených synchronizací vyčkávajících na vyvolání. Výsledným počtem hran v této kolekci jsou pak možné následující dva scénáře.

- **Kolekce odchozích hran je prázdná** - v takovém případě se výběr hrany ukončí a přejde se ke kontrole možného uváznutí v simulaci (následující podkapitola 3.2.3).
- **Kolekce odchozích hran není prázdná** - v tomto případě, kdy kolekce obsahuje alespoň jednu hranu, nic nebrání přístupu k druhé fázi výběru hrany (podkapitola 3.2.4).

3.2.3 Kontrola uváznutí

Uváznutí by mělo být odhaleno již při prvotní tvorbě a validaci modelu v nástroji UPPAAL. Nicméně pro potřeby podchycení možného vzniku chyby a zacyklení simulace právě špatně navrženým modelem, je tato kontrola přidaná i do samotné simulace modelu. Tedy v případě prázdné kolekce odchozích hran po první fázi výběru je zkontrolováno, zda-li není některý z aktuálních stavů typu urgent či committed, čímž by byl současně zmrazen čas. Pokud ano, dochází k zacyklení, kdy výsledek příštího kroku simulace by byl stejný. Tudíž aktuální simulace nikam dál nevede a je ukončena s vyvoláním výjimky (obr. 3.8). V případě nezmraženého času je možnost, že inkrementace hodin vyhoví některému hlídači odchozích hran a v příštím kroku simulace bude možné provést právě alespoň jednu z odchozích hran a k zacyklení tak nedojde.



Obrázek 3.8: Kontrola uváznutí

Samozřejmě možných důvodů zacyklení simulace je více, nicméně jejich odhalení už není tak přímočaré a je třeba se jich vyvarovat již při vytváření modelu, a to jeho validací. Tedy tento krok simulace je přidán pouze pro urychlení celé fáze testování právě díky snadné odhalitelnosti vzniku takového typu chyby.

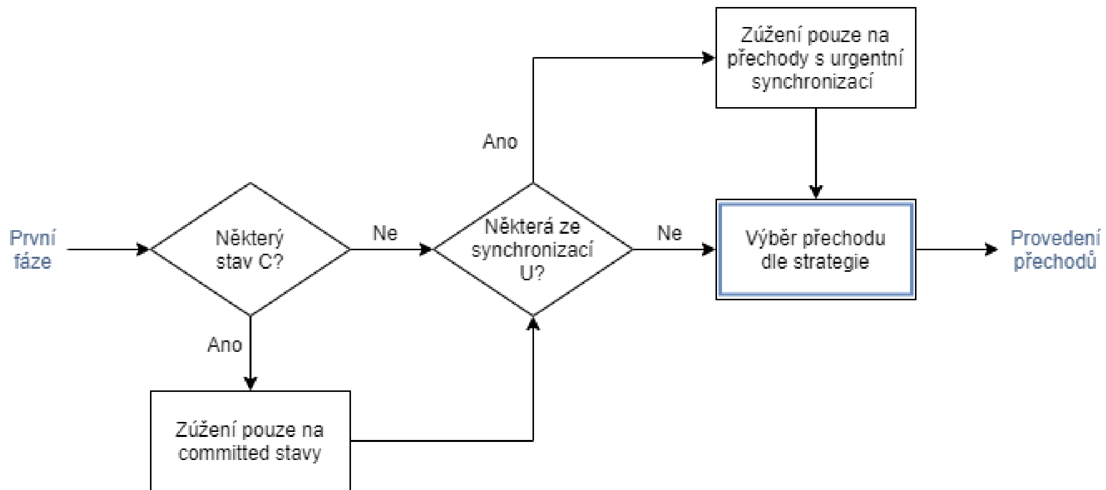
3.2.4 Druhá fáze výběru

V druhé fázi výběru hrany k provedení je ve dvou krocích zkontrolováno a provedeno další možné zúžení kolekce výstupních hran z první fáze (obr. 3.9). Nejprve jsou zkontrolovány typy všech hran kolekce výstupních hran. V případě, že je mezi hranami alespoň jedna typu committed, jsou tyto hrany vybrány a ostatní z kolekce opět vyřazeny.

Druhý krok možného zúžení je obdobný. U všech zbylých hran kolekce je zkontrolováno, zda obsahují synchronizaci a jakého typu. V případě existence alespoň jedné takové hrany jsou přechody s tímto typem synchronizace vybrány a ostatní z kolekce výběru vyřazeny.

Zde je možné zamyšlení nad pořadím těchto dvou kroků zúžení výsledné kolekce výstupních hran. Jelikož na sobě vzájemně nijak nezávisejí, stejně validní jsou obě možné varianty pořadí, tedy výše zmíněná možnost (nejprve o committed stavy, pak o urgentní synchronizace), či prohozená (nejprve o urgentní synchronizace, pak o committed stavy). Jelikož v dosavadní verzi Tasteru nebyla podpora pro urgentní synchronizace, a tudíž jsou veškeré dosavadní modely systému vytvořeny bez využití tohoto typu synchronizace, je v tomto návrhu upřednostněna první zmíněná varianta pořadí, jelikož pravděpodobněji dojde k zúžení kolekce právě z důvodu committed stavů. Nicméně ukáže-li se v budoucnu

jako efektivnější varianta druhá, bude tuto změnu pořadí možné jednoduše kdykoliv provést.



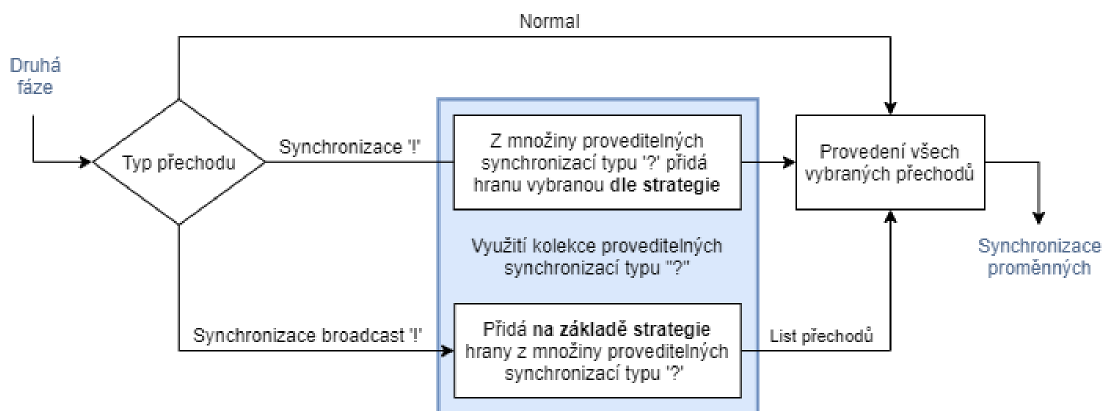
Obrázek 3.9: Druhá fáze výběru hrany

Posledním krokem druhé fáze výběru je samotný výběr finální hrany k provedení, které již provede zvolená strategie simulace. Strategii je předána výsledná vytříděná kolekce výstupních hran a současně z důvodu úplnosti pro rozhodnutí strategie je též předána kolekce proveditelných hran se synchronizací typu "?" (zmíněná kolekce z první fáze 3.7). Od strategie je očekávána právě jedna vybraná hrana, se kterou se pokračuje do navazující fáze provedení přechodů.

3.2.5 Provedení přechodů

Ve fázi provedení zvoleného přechodu se nejprve zkontroluje, zdali vybraná hrana obsahuje synchronizaci a případně je-li typu broadcast. Možné jsou tři scénáře (obr. 3.10). V prvním vybraná hrana synchronizaci neobsahuje. V takovém případě se přejde rovnou v provedení dané hrany. Tedy obsahuje-li hrana přiřazení, je toto přiřazení uskutečněno. Následně je změněn aktuální stav automatu vybraného přechodu na vybraný a dále se přistoupí ke kontrole zmrazení času.

Druhou možností je situace, kdy vybraná hrana obsahuje obyčejnou synchronizaci typu "!". V tomto případe se nejprve vytvoří kolekce z průniku závisejících hran vybrané vyvolávající synchronizace s kolekcí proveditelných synchronizací typu "?" (zmíněná kolekce z první fáze výběru 3.7). Z tohoto průniku se dále strategií provede výběr synchronizačního protipólu, kdy je strategii k výběru pro úplnost k rozhodnutí předána jako závisející hrana právě původní vybraná hrana (vyvolávající synchronizace typu "!"). Strategií zvolená hrana (synchronizační protipól "?") je přidán ke vstupní zvolené hraně (synchronizace typu "!"). Vytvořen je tak list dvou hran k provedení a postupně dojde k jejich vykonání (počínaje vyvolávající synchronizací typu "!"). Tedy u každé se provede její přiřazení, následně dané automaty změní svůj stav a přejde se ke kontrole zmrazení času.



Obrázek 3.10: Provedení přechodů

Třetím možným případem je, kdy vybraná hrana obsahuje synchronizaci typu broadcast. V této situaci je nejprve vytvořen list hran k provedení, do kterého je jako první hrana přidána hrana synchronizaci vyvolající. Dále je proveden průnik kolekce závisejících hran (na hranu vyvolávající) s kolekcí proveditelných synchronizací typu "?" (opět zmíněná kolekce z první fáze výběru 3.7). Při tomto průniku je vytvořen slovník, kdy klíčem je automat a hodnotou je kolekce hran z daného automatu (tedy hrany nacházející se ve shodném automatu). Následně je tento slovník procházen a mohou nastat tyto dvě situace.

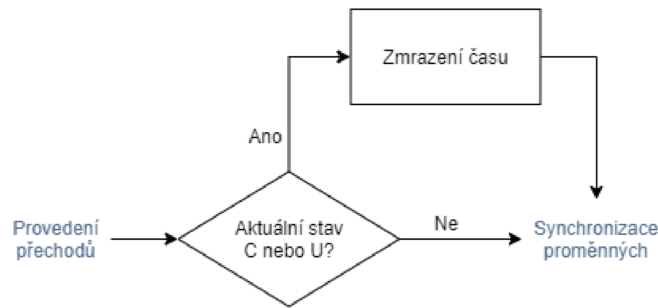
- **Průnik obsahuje hrany stejného automatu** - tedy je-li hodnotou slovníku kolekce s více jak jednou položkou. V tomto případě je na výběr jedné hrany z kolekce hran shodného automatu opět použita strategie. Krom zmíněné kolekce je strategii též pro úplnost hrana předána hrana synchronizaci vyvolávající. Strategií vybraná hrana je následně přidána do listu hran k provedení.

- **Průnik obsahuje hranu, která je jedinou možnou svého automatu** - tedy je-li hodnotou slovníku kolekce s právě jednou položkou. v tomto případě je daná hrana přímo přidána do listu hran k provedení.

Vytvořen je tak list hran k provedení, obsahující původní hranu vyvolávající synchronizaci a současně k ní závisející hrany v maximálním počtu jedna vybraná hrana z jednoho daného automatu. Poté dojde k postupnému vykonání všech hran seznamu, a to počínaje hranou synchronizaci vyvolávající (typu "!"). Tedy u každé se provede její přiřazení a následně dané automaty změní svůj aktuální stav a přejde se ke kontrole zmrazení času.

3.2.6 Zmrazení času

Při kontrole zmrazení času se projdou typy aktuálních stavů. Je-li alespoň jeden z aktuálních stavů typu urgent nebo committed, dojde k zmražení času (obr. 3.11). Informace o zmrazení času je poznamenána a později použita jako návratová hodnota funkce kroku simulace, čímž je vyšší úrovni řízení simulace předán pokyn pro práci s časem modelu (detailněji v podkapitole navazující kapitoly o řízení simulace 3.3.2). Následně se přejde k synchronizaci proměnných modelu s připojeným systémem.



Obrázek 3.11: Kontrola stavů pro zmrazení času

3.2.7 Závěrečná synchronizace

V závěrečné fázi dojde k opětovné synchronizaci virtuálních proměnných s proměnnými připojeného systému. Tímto způsobem jsou vykonaná přiřazení v rámci modelu (uskutečněných během vykonání hran) do systémových proměnných. V případě jakéhokoliv důvodu neúspěchu synchronizace dat dojde k vyvolání výjimky a následně k zastavení simulace.

Výsledkem úspěšně provedeného simulačního kroku tak jsou aktualizované stavy v odpovídajících automatech a aktualizované proměnné systému díky provedeným přiřazením na vybraných hranách. V takovém případě je v návratové hodnotě vrácena informace o zamrznutí času, kdy kladná návratová hodnota značí aktivní zmrazení. V případě neúspěšného kroku simulace je výsledkem vyvolaná výjimka oznamující typ vzniklé chyby.

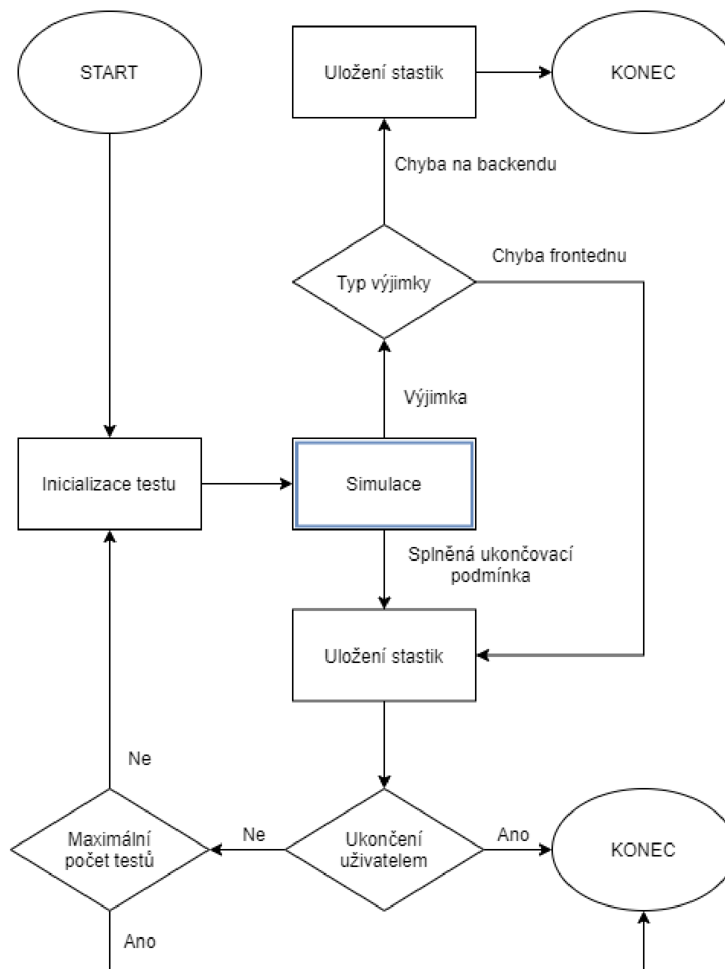
3.3 Algoritmus řízení testu

Pro zkvalitnění přehlednosti celého algoritmu simulace bylo opět snahou zlepšit strukturovanost kódu. Z tohoto důvodu bylo nově navrženo řízení simulace odděleně od samotného simulačního kroku. Řízení se tak stará o spouštění simulace, kdy volá a vyhodnocuje simulační krok. Dále archivuje statistická data a zajišťuje propojení s uživatelem skrze ovládací panel uživatelského rozhraní.

Algoritmus řízení, jehož diagram je rozkreslen a na obr. 3.12, lze rozdělit do několika fází. Jimi jsou inicializace testu, fáze samotné simulace, zachycení a vyhodnocení závažnosti výjimek a vyhodnocení právě proběhlé simulace, případně opětovné spuštění další iterace simulace. Ve fázi simulace je vyhodnocen simulovaný krok a dochází ke kontrole podmínek ukončení. Během této fáze je využíván algoritmus kroku simulace. Z toho důvodu je tento blok v digramu zvýrazněn.

3.3.1 Inicializace

Během první fáze je nejprve inicializován model systému, včetně synchronizace s připojeným adaptérem. V modelu systému jsou tak resetovány všechny proměnné, aktuální stavy automatů jsou nastaveny zpět na počáteční a restartovány jsou také veškerá statistická počítadla (jakožto čítače přechodů, využití konkrétních automatů či stopky simulace). Po úspěšné inicializaci systému již následuje spuštění samotné simulace.



Obrázek 3.12: Diagram řízení testu

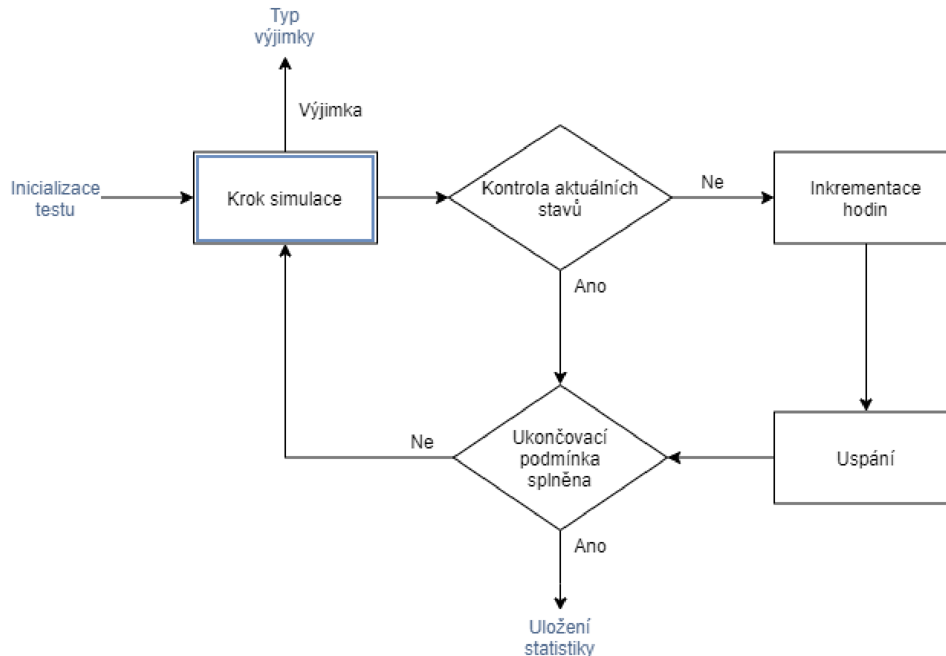
3.3.2 Simulace

Fáze simulace začíná voláním kroku simulace (tento krok v diagramu na obr. 3.13 zvýrazněn). V případě úspěšně provedeného simulačního kroku je zkontrolována návratová hodnota volané funkce. Ta odpovídá zamrznutí hodin simulovaného systému, kdy kladná návratová hodnota značí zamrznutí času. Tedy systém se v takovém případě nachází pouze v normálních stavech a dojde tak k inkrementaci hodinových proměnných. Aby čas simulace reflektoval reálné zpoždění odpovídající inkrementačnímu kroku hodin, je následně simulace "uspána"(doba uspání odpovídá zvolenému ikrementačnímu kroku hodin). Po uspání dále zkontrolována podmínka ukončení simulace.

Při záporné návratové hodnotě kroku simulace se systém právě nachází v alespoň jednom stavu typu urgent nebo committed. V takové případě je vyžadováno jeho neprodlené opuštění. Z toho důvodu je pouze zkontrolována podmínka ukončení simulace a následně rovnou opětovně volána funkce kroku simulace. Tím je co nejrychleji umožněno opuštění daného stavu.

Oddělením inkrementace hodin od funkce kroku simulace je respektována sémantika nástroje UPPAAL, kdy význam urgentních a committed stavů hraje roli až při setrvání

v daném stavu nebo jeho opouštění, nikoli však při vstupování do daného stavu. Tento smysl práce s časem je tak v nástroji Taster převzat. Současně je zmíněným oddělením inkrementace hodin zpřehleděna celá fáze kroku simulace.



Obrázek 3.13: Diagram simulace

Při kontrole ukončovacích podmínek může být simulace zastavena hned z několika následujících důvodů.

- **Ukončení uživatelem** - vynucené zastavení.
- **Překročení maximální délky testu** - reálného času běhu simulace.
- **Překročení maximálního počtu kroků** - počtu volání funkce kroku simulace.
- **Pokrytí všech stavů** - navštívení všech stavů.
- **Pokrytí všech hran** - použití všech hran.
- **Pokrytí všech párů hran** - neboli všech kombinací dvou po sobě jdoucích hran.

Sada vybraných podmínek ukončení je zvolena před zahájením simulace. Sadu podmínek lze sestavit libovolně, přičemž platí, že simulace probíhá do první splněně podmínky. Architektura je navržena tak, aby do budoucna umožňovala jednoduché přidání nových ukončovacích podmínek. V případě splnění kterékoliv vybrané podmínky se následně přechází k uložení statistky zachycené v průběhu simulace. Naopak v případě nesplnění žádné z podmínek dojde k další iteraci simulace opětovným voláním simulačního kroku.

3.3.3 Zachycení výjimek

Pokud byla během kroku simulace vyvolána výjimka, je tato vzniklá chyba zachycena a vyhodnocen její typ a závažnost. Rozlišujme dva základní typy výjimek dle místa vzniku.

- **Výjimka vyvolaná chybou v modelu** - jedná se převážně o chybu logického charakteru. Příkladem může být uváznutí/zacyklení simulace. Chyba tohoto typu je méně závažná, dojde pouze k zastavení dané iterace simulace, zaevidování chyby a uložení statistiky a přes další již popsané kroky je možné opět spustit novou iteraci simulace.

- **Výjimka vyvolaná chybou adaptéru** - chyba závažného charakteru. Příkladem se jedná o porušení invariantu v nějakém z aktuálních stavů. V případě testovaní na připojeném automobilu by taková chyba mohla znamenat například špatnou polohu některé z mechanických částí, kdy by při pokračovaní v simulaci mohlo hrozit poškození připojeného zařízení. Z toho důvodu je již jen uložena statistky a následně simulace kompletně zastavena. Nové zahájení simulace je tak vynuceno vědomím spuštěním uživatelem.

3.3.4 Ukončení simulace

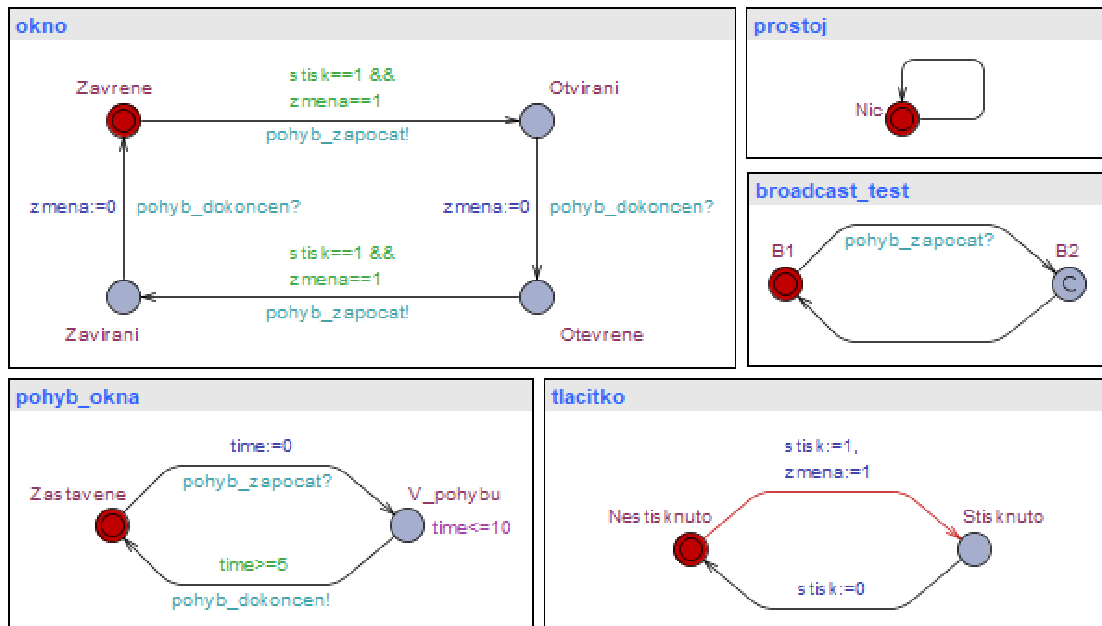
Při korektním ukončení iterace simulace, kdy nedošlo k vyvolání závažné výjimky, je po uložení statistiky zkontrolováno, zda simulace nebyla zastavena samotným uživatelem. V takovém případě vynuceného ukončení uživatelem je simulace kompletně zastavena a k dalšímu testovaní dojde až po uživatelem opětovném spuštění zcela nové simulace. Pokud ukončení není vynuceno uživatelem, navazuje kontrola počtu provedených testů. V případě naplnění předem daného počtu testování je simulace ukončena. Opačném případě se přejde k inicializaci testu nového, čímž se smyčka cyklu uzavírá pokračováním v simulacích.

3.4 Prototyp algoritmu

Pro otestování efektivnosti práce s daty v nově navržené provázané objektové struktuře a odzkoušení hlavních změn v samotném algoritmu simulace (např. nový přístup k práci se synchronizacemi) byl vytvořen zjednodušený prototyp. Jako programovací jazyk byl zvolen Python. Jedná se o objektově orientovaný, dynamicky typovaný skriptovací jazyk, s jehož volbou je možné se více soustředit především na podobu samotného algoritmu než jeho na jeho optimalizaci [18]. Díky jeho snadné syntaxi umožňuje vysokou produktivitu programování, čímž zrychluje tvorbu prototypu, a tak testování kvality návrhu [15].

Speciálně pro potřeby prototypu byl vytvořen model, který pokrývá veškeré nově přidané funkcionality a testuje kompletní podporu modelačních prostředků nástroje UPPAAL (jeho celková podoba je na obr. 3.14). V testovacím modelu tak jsou zahrnuty hlídače přechodů, přiřazení, invarianty, hodinové proměnné, urgentní stavy synchronizační kanál se synchronizací typu braodcast. Pro otestování zachycení vyvolaných výjimek vzniklých chybou v simulaci byla do modelu úmyslně přidána i možnost zacyklení, popsaná v podkapitole Odlišnost sémantiky 2.3.2). Pro zjednodušení prototypu, jsou zvoleny číselné nenulové hodnoty proměnných. Současně je práce s logickými výrazy invariantů a hlídačů přechodů zjednodušena na využití pouze jednoho logického výrazu současně, kdy je logický výraz formou porovnání hodnoty dané proměnné. Navíc navržený model v nástroji UPPAAL není prototypem načten, ale přímo předem pevně implementován do zdrojového kódu prototypu. A právě během plnění připraveného časovaného automatu do navržené

objektové struktury se projevilo mírné zkomplikování vlivem provázanosti návrhu. Kvůli obousměrným referencím na instance modelu bylo nutné při implementaci (následující kapitola 4) modelu upravit pořadí tvorby objektové struktury ve fázi sestavení modelu (podkapitola 4.2).



Obrázek 3.14: Model použitý při testování prototypu

Prototyp se zaměřuje především na část simulování kroku algoritmu, přičemž byla implementovaná pouze jedna strategie volby hrany založená na náhodném výběru z aktuálně proveditelných hran. Dále byla implementována podpora sady ukončovacích podmínek skládajících se z vypršení časového limitu simulace, dosažení všech stavů, pokrytí všech hran a ukončení uživatelem. Při simulaci je následně možné přidat k podmínce ukončení uživatelem libovolnou kombinaci ze zbývajících podmínek. Pro simulaci byl předem pevně zvolen ikrementační krok hodin na hodnotu 1. Tomuto výběru inkrementčního kroku byl současně přizpůsoben testovací model.

Pro vyhodnocení výsledků simulace a zároveň pro ukázku práci s daty statistiky, byl vytvořen jednoduchý grafický výstup. Na následujícím obr. 3.15 je pak zachycen záznam jednoho z uskutečněných testů prototypu. Během tohoto testu byla zvolena sada ukončovacích podmínek skládajících se z vypršení časového limitu a dosažení všech stavů. Současně byl zvolen extrémní časový limit na provedení simulace tak, aby byl na hranici časové proveditelnosti jedné simulace a tím mohlo dojít i k porušení této podmínky. Po ukončení simulace je tak zobrazena tabulka s přehledem provedených testů a výběrem několik statistických dat.

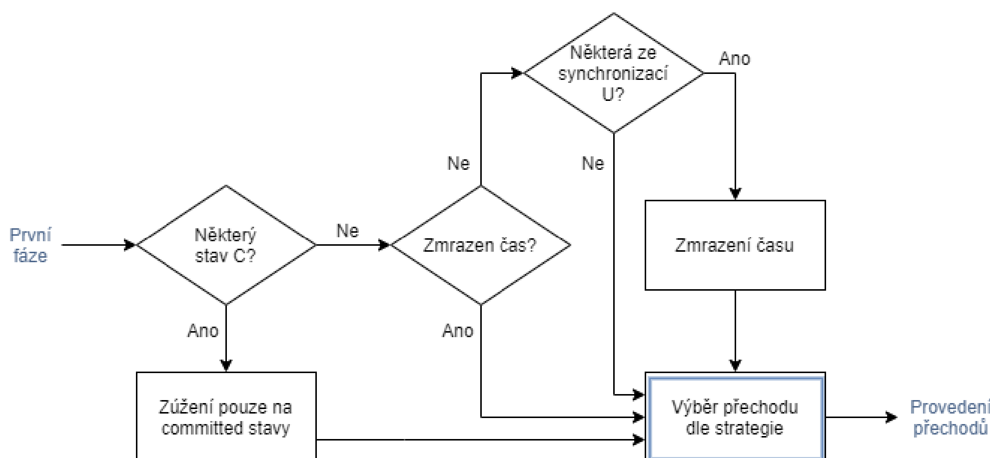
No	Doba testu [ms]	Důvod ukočení	Počet volání Step	Zbývající uzly	Zbývající hrany
1	205.276	Timeout	2	5 (45%)	9 (82%)
2	3.236	Timeout	11	1 (9%)	2 (18%)
3	2.583	All states	10	0 (0%)	1 (9%)
4	3.093	All states	13	0 (0%)	1 (9%)
5	2.338	All states	10	0 (0%)	1 (9%)
6	2.847	All states	12	0 (0%)	1 (9%)
7	3.097	Error	12	2 (18%)	4 (36%)
8	3.364	User	8	1 (9%)	2 (18%)

Obrázek 3.15: Výstup simulace v prototypu

Na obrázku 3.15 tak lze vidět několik zaznamenaných testů, lišící se průběh, kdy postupně došlo ke čtyřem různým důvodům ukončení. U prvních dvou tak došlo k vypršení časového limitu, kdy je krom doby dané simulace též vypsán počet volání funkce kroku simulace, nenavštívené uzly s nepoužitými hranami (včetně procentuálního zastoupení v celém modelu). Obdobné záznamy pozorujeme i u následujících iterací simulace, kdy došlo jak dosažení všech stavů, tak zastavení z důvodu vyvolání výjimky vzniklé chybou v modelu a v posledním testu k ukončení uživatelem. V zachyceném chybovém případě došlo k problému méně závažného charakteru vzniklého chybou v modelu - zacyklením. Tudíž se ukončil jen daný test a pokračovalo se další novou simulací.

3.4.1 Zhodnocení prototypu

V rámci tvorby prototypu došlo k několika úpravám v návrhu algoritmu kroku simulace, které již byly při psaní předchozích kapitol do popsané podoby návrhu zahrnuty. Nicméně během testování na připraveném modelu a porovnávání chování se simulací shodného modelu v nástroji UPPAAL byla odhalena misinterpretace vlivu synchronizačního kanálu typu urgent na výběr hrany (chybný návrh v podkapitole 3.2.4). Na následujícím obrázku 3.16 je tak z důvodu dodržení sémantiky jazyka UPPAAL upravená druhá fáze výběru kroku simulace.



Obrázek 3.16: Upravená druhá fáze výběru hrany kroku simulace

V novém návrhu je tak při zjištění situace, kdy se v proveditelných hranách nachází hrana ze stavu typu committed, zúžena kolekce proveditelných hran pouze na hrany z committed stavů a následně se rovnou přejde ke kroku výběru hrany. V případě, že se v kolekci proveditelných hran žádná taková hrana nenachází, je nejprve zkontrolováno, zda již není zmrazen čas. Pokud čas pozastaven není, dojde ke kontrole proveditelných hran. Jestliže se v této kolekci hran nachází hrana s urgentní synchronizací, doje ke zmrazení času. Tento krok je nejzásadnější změnou návrhu. V původním návrhu bylo při nalezení urgentní synchronizace mezi proveditelnými hranami dále vynuceno vybrání právě některé z hran s urgentní synchronizací. V upravené verzi je v tomto případě pouze zmrazen čas a kolekce proveditelných hran se již více neredukuje.

Výsledný prototyp dále již reflektuje navrženou objektovou strukturu a algoritmus kroku simulace. Současně bylo s jeho pomocí otestováno zpracování zachycených výjimek ukázaná práce s daty statistického záznamu. Další drobné změny budou již zahrnuty do samotné implantace v následující kapitole. Například v objektové struktuře tak přibudou ve třídě systému záznamy o celkovém počtu stavů a hran všech automatů (pro usnadnění a zkvalitnění tvorby statistky) a ve třídě automatu dále kolekce s referencemi na ještě nepoužité hrany a nenavštívené stavy (pro usnadnění kontroly ukončovacích podmínek, zpřehlednění tvorby záznamu a zlepšení informačního podkladu pro rozhodnutí strategie o výběru vhodné hrany). Celkově však prototyp dostatečně ověřil myšlenku nového návrhu, kvalitu provázanosti objektové struktury a bylo tak možné přistoupit k jeho implementaci do nástroje Taster.

Kapitola 4

Implementace návrhu

V rámci implementace návrhu bylo nejprve zapotřebí upravit objektovou strukturu a načítání modelu v části *TASystem* parsovací funkcí tak, aby načítání reflektovalo provázanost objektové struktury a nově podporované synchronizační kanály typu urgent a broadcast. Poté bylo již přistoupeno k implementaci samotného návrhu simulačního algoritmu v části *TARuntime* společně s úpravami v třídě *TARunform*, reprezentující okno grafického rozhraní běhu simulace. U funkce Step, zařizující vykonání jednoho kroku simulace, bylo přistoupeno ke kompletnímu nahrazení novou verzí. Stejně tak bylo přepsáno řízení testu vyhodnocující simulační krok a umožňující přímé automatické spouštění další simulace. V závěru je několik provedených experimentálních simulací s modely testující jak implementovanou podporu speciálních typů synchronizačních kanálu, tak s modely vytvořenými pro původní architekturu simulačního algoritmu, testující tak zpětnou kompatibilitu.

4.1 Objektová struktura

Objektová struktura prošla zásadním vylepšením. Hlavní objektová struktura modelu *TASystem* byla rozšířena kolekci všech synchronizačních kanálů, splněnou podmínkou ukončení simulace a kolekce automatů čekajících na dosažení všech svých stavů či použití hran. Současně z důvodu sjednocení celé fáze načtení modelu do *TASystem* (podrobněji v další podkapitole Načtení modelu 4.2), byla do třídy *TASystem* přidána též kolekce instancí automatů a id čítač.

```
public class TASystem
{
    public List<Template> Templates;
    public List<SyncChannel> SyncChannels;
    public string Declaration;
    public string Script;
    public List<Template> TemplateInstances = new List<Template>();
    public string StopReason = "";
    public int idCounter;    // for the templates id
    public List<Template> TemplesToNodeSatisfied = new List<Template>();
    public List<Template> TemplesToEdgeSatisfied = new List<Template>();
\\...
```

Dále byly přidány metody na inicializaci a kontrolu pokrytí hran/stavů a z *TARuntime* přesunuty metody zpracování systémové deklarace.

Třída hrany byla obohacena o referenci na objekt třídy automatu ve kterém se nachází, dále o třídu synchronizace, a čítač použití. U třídy stavu byla obdobně přidána reference na automat, ve kterém se nachází.

```
public class Edge
{
    public Template SourceTemplate;
    public Sync sync;
    public int takeCount = 0;
//...

public class Node
{
    public Template SourceTemplate;
//...
```

Podstatnou změnou je přidání třídy synchronizačního kanálu a synchronizace. Kanál obsahuje název, specifikovaní (typ urgent a broadcast) a kolekci všech synchronizací. Synchronizace pak obsahuje referenci na kanál, ve kterém se nachází, typ (zda se jedná o synchronizaci vyvolávající či vyčkávající) a kolekci závislých hran.

```
 public class SyncChannel
  {
    public string Name;
    public bool Urgent;
    public bool Broadcast;
    public List<Sync> AllSync;
    //...
  }

  public class Sync
  {
    public SyncChannel SyncChan;
    public char Type;
    public List<Edge> DependentEdges;
    //...
  }
```

Z důvodu využití syntaktického analyzátoru bylo naopak kompletně upuštěno od třídy proměnné (podrobněji v další podkapitole).

Ve třídě *TARuntime* byla přidána reference na systém. Dále zda je zmrazen čas, třídu ukončovacích podmínek, čítač kroků simulace, čítač simulací, maximální počet simulování a vyžádaní restartování testu.

```
public class TARuntime
  {
    public TASystem system;
```

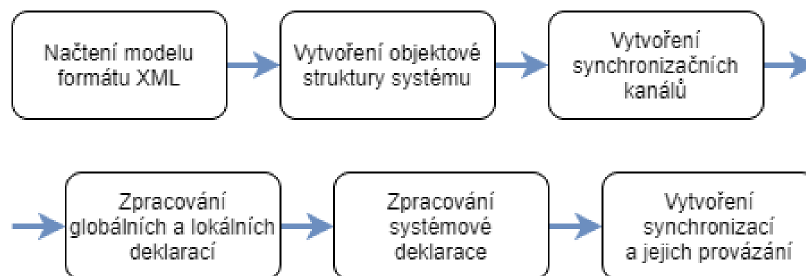
```
    public bool timeFreeze;
    public EndConditions endConditions;
    public int stepCounter;
    public int simulationCounter;
    public int maxSimulationRuns;
    public boool resetTest;
    \\...
```

Zmíněná třída ukončovacích podmínek pak obsahuje sadu vybraných podmínek z aktuálně podporovaných, které mají být pro daný test v rámci simulace kontrolovány.

```
public class EndConditions
  {
    public bool ReachAllStates;
    public bool UseAllEdges;
    public int MaxStepCalls;
  \\...
```

4.2 Načtení modelu

Ve fázi načtení modelu serializovaného ve formátu XML je postupně dle struktury načteného modelu parsováním vytvářen objektový model systému, a to počínaje deklarací globálních proměnných, jednotlivými automaty a systémovou deklarací. Posloupnost procesu této zachycuje následující schéma na obr. 4.1.



Obrázek 4.1: Proces načtení modelu

V rámci vytváření objektové struktury automatů a jejich částí je doplněno provázení objektové struktury přidáním referencí odkazujících napříč komponentami viz. obousměrné reference mezi třídou stavu, hrany a automatu na obr. 3.3. U synchronizačních hran je při vytváření automatů zaznamenána pouze definice dané synchronizace ve formátu textového řetězce (např. "kanál !"). Vytvoření synchronizačních kanálů z globální deklarace je následně provedeno až po kompletním zpracování XML souboru funkcí *CreateSyncChannels(Declaration)*.

```
private void ParseSystemFromXML(string fileName)
{
  //...
  foreach (XmlNode node in nodes)
```

```
  {
    switch (node.Name)
    {
      case "declaration":
        Declaration = node.InnerText;
        break;
      case "template":
        Templates.Add(LoadTemplate(node));
        break;
      case "system":
        Script = node.InnerText;
        break;
    }
  }
  CreateSyncChannels(Declaration);
}
```

Aby bylo možné využít stávající podobou syntaktického a lexikálního analyzátoru, je během vytváření synchronizačních kanálů globální deklarace kompletně zbavena definic synchronizačních kanálů. Důvodem této změny je zpracovávání globálních deklarací při syntaktické kontrole, kdy je v aktuální podobě syntaktického analyzátoru počítáno pouze s jednoslovnými datovými typy. Tedy víceslovné deklarace synchronizačních kanálů "urgent chan", "broadcast chan", "urgent broadcast chan" nejsou podporovány a vzhledem ke komplexnosti syntaktického analyzátoru se ukázalo jejich předzpracování jako rozumné řešení. Navíc, synchronizační kanály jsou velmi specifickým typem globální proměnné a svou odlišností od zbytku používaných proměněných není opodstatněné jejich zahrnutí do syntaktického analyzátoru.

Zpracování systémové deklarace bylo v původní architektuře provedeno až při spuštění řízení testu (*TARuntime*). Nový návrh sjednocuje vytvoření kompletního modelu rovnou do fáze načtení, tedy zpracování systémových deklarací je přesunuto do *TASystem* s vyvoláním během načtení modelu funkcí *LoadModel(fileName)*.

Po zpracování systémové deklarace funkcí *ParseScript(Script)*, v rámci které dochází k vyřazení automatů v návrhu nezahrnutých do výsledného systému (přidáno ošetření nevyužitých automatů), jsou funkcí *FillSyncChannelsWithSyncEdges(templates)* vytvořeny jednotlivé synchronizace synchronizačních kanálů. Kvůli vzájemné provázanosti synchronizací vyvolávajících s vyčkávajícími, je nejprve vytvořen slovník všech synchronizací s klíčem definic synchronizací (např. "kanál!"), pod kterými jsou do kolekce sdruženy synchronizační kanály shodné definice. Následně je tento slovník využit pro vytvoření synchronizací s odkazem na kolekci závislých hran, získaných z vytvořeného slovníku použitím klíčem protipólu.

```
public void FillSyncChannelsWithSyncEdges(List<Template> templates)
{
  // create dictionary of sync poles
  var syncPoles = new Dictionary<string, List<Edge>>();
  foreach (var t in templates)
  {
```

```
    foreach (var e in t.Edges)
    {
      if (e.Sync != "")
      {
        var sync = e.Sync.Replace(" ", "");
        if (!syncPoles.ContainsKey(sync))
          syncPoles.Add(sync, new List<Edge>());
        syncPoles[sync].Add(e);
      }
    }
  }
  // use the dictionary to create synchronizations
  foreach (var syncPole in syncPoles)
  {
    var chanName = syncPole.Key.Substring(0, syncPole.Key.Length - 1);
    var syncType = syncPole.Key[syncPole.Key.Length - 1];
    var syncAntitype = syncType == '!' ? '?' : '!';

    foreach (var edge in syncPole.Value)
    {
      var dependentEdges = new List<Edge>();
      foreach (var antipoleEdge in syncPoles[chanName + syncAntitype])
      {
        if (antipoleEdge.SourceTemplate != edge.SourceTemplate)
          dependentEdges.Add(antipoleEdge);
      }
      var chan = SyncChannels.Find(chan => chan.Name == chanName);
      edge.sync = new Sync(chan, syncType, dependentEdges);
      chan.AllSync.Add(edge.sync);
    }
  }
}
```

4.3 Krok simulace

Funkce *Step()*, která vykonává jeden simulační krok, je kompletně přepsána. Nově je přehledně a systematicky strukturována sekvencí kroků reflektující návrh algoritmu. Oproti idee návrhu je upuštěno od návratové hodnoty reprezentující zmrazení času. K tomuto účelu je využita proměnná *timeFreeze* třídy vykonávání testu *TARuntime* pro zlepšení přístupnosti k informaci o zmrazení času.

```
public void Step()
{
  var ActiveNodes = system.GetSystemAcitveNodes();
  var ChoosableEdges = new List<Edge>();
  var ChoosableSlaveSync = new List<Edge>();
```

```
  var EdgesToExecution = new List<Edge>();
  Log = string.Empty;
  stepCounter++;

  // Sync with adaptor
  _adaptor.ReadVariables(allVariables);
  CheckInvariants(ActiveNodes);
  // first stage of choose
  ChoosableEdges = CheckGuards(ActiveNodes);
  ChoosableEdges = CheckExistenceOfSyncPairs(ChoosableEdges);
  (ChoosableEdges, ChoosableSlaveSync) = SplitSyncPairs(ChoosableEdges);
  if (ChoosableEdges.Count > 0)
  {
    // second stage of choose
    ChoosableEdges = CheckCommittedStates(ChoosableEdges);
    Log += "Possible edges is " + ChoosableEdges.Count
        + Environment.NewLine;
    EdgesToExecution.Add(
        ChooseEdgeByStrategy(ChoosableEdges, ChoosableSlaveSync));
    // stage of application of taken edge
    EdgesToExecution =
       CheckTakenEdge(EdgesToExecution[0], ChoosableSlaveSync);
    DoTransitions(EdgesToExecution);
    timeFreeze = CheckUrgencyOfStates(system.Templates);
    // Sync with adaptor
    _adaptor.WriteVariables(allVariables);
  }
  else
  {
    Log += "No suitable edges to take!" + Environment.NewLine;
    // check deadlock
    if (timeFreeze)
      RaiseDeadlock();
  }
}
```

Při zjištění porušeného invariantu v rámci kroku *CheckInvariants(ActiveNodes)* je vyvolána specializovaná výjimka typu *InvariantProblem*. Obdobně při zjištění zacyklení vlivem prázdného seznamu proveditelných hran *ChoosableEdges* je vyvolána výjimka typu *Deadlock*.

Pro vyhodnocování logických výrazů při kontrole stráže, invariantu a přiřazení je využita stávající platforma třídy deklarací *Expression.Evaluate()*. Dále je pro vykonávání přiřazení typu *Execute* na připojeném adaptéru využit vyhodnocovač výrazů *ProcessExprForEXAM(expresssion)* v rámci funkce *DoTransitions(TakenEdges)*.

V rámci kroku výběru hrany funkcí *ChooseEdgeByStrategy(ChoosableEdges, ChoosableSlaveSync)*, je v aktuální verzi implementována pouze strategie náhodného výběru. Pro budoucí implementaci komplexnějších strategii je krom kolekce hran k výběru předána též

kolekce proveditelných závisejících synchronizačních hran. Strategie se tak může rozhodnout i na základě případně vyvolané závisející hraně. Zároveň při vyvolání synchronizace a volbě odpovídající závisející hrany se strategie naopak může rozhodnout na základě hrany synchronizaci vyvolávající.

Pro manuální krokování simulace byla přidána operace *SingleStepClick()*. Ta vyvolávala vykonávání kroku a stará se o zachycení a zpracovávání výjimek, kdy případně restartuje simulaci.

```
public void SingleStepClick()
{
  if (resetTest)
    Reset();
    Log += "Test was restarted." + Environment.NewLine;
  try
  {
    Step();
    if (!timeFreeze)
      IkrementTimeVariavblse()
  }
  //...
}
```

Při vykonání hrany je zaznamenán a vypsán v hlášení o průběhu simulace kromě vykonané hrany též počet zvolitelných hran. Pomocí hlášení je tak možné jednoduše přímo kontrolovat vykonávání simulace.

4.4 Řízení testu

Samotné testování s automatickým spouštěním simulace je implementováno přímo ve struktuře spuštěného simulátoru *RunForm* ve funkci *Bw_DoWork()*. Nejprve inicializován test vyvoláním procedury třídy *TARuntime.InitializeRun()*. Před zahájením každého z testů je restartován model do výchozího stavu - vynulovaním pokrytí, nastavení počátečních stavu, restartování proměnných, atd. Následně je zahájena simulace vyvoláním funkce *Step()*. Při úspěšném vykonání dojde ke kontrole zamrznutí času a případně k inkrementaci hodinových proměnných a následného uspání vlákna.

```
while (_runSimulation)
{
  try
  {
    _runtime.Step();
    if (!_runtime.timeFreeze)
    {
      _runtime.InkrementTimeVariables();
      //wait
      while (Heartbeat.MonotoneTime < time + iteration * _interval)
        Thread.Sleep();
```

```
      iteration++;
    }
    if (_runtime.CheckEndCodition())
    {
      // stop current simulation
      _runSimulation = false;
    }
  }
  //...
}
```

Dále se přistoupí ke kontrole ukončovacích podmínek operací *CheckEndCodition()*. Pokud je některá z ukončovacích podmínek splněna, vrátí tato operace kladnou odpověď a dojde k zastavení aktuální simulace. Při vyvolání výjimky v rámci funkce *Step()* je výjimka zachycena a dle vážnosti je ukončena pouze aktuální simulace, či při závaznější chybě kompletně celý test. Jsou rozlišeny tři kategorie výjimky.

- **Soft failure** - vyvolána zacyklením. Nezávažná chyba v modelu. Přistoupeno k ukončení aktuální simulace a následně lze spustit simulaci novou.
- **Hard failure** - vyvolána chybou invariantu. Závažná chyba simulace vedoucí k ukončení testu.
- **Critical failure** - vyvolána nepředpokládanou chybou simulace. Následuje k ukončení testu.

Při ukončení simulace je dále uložena statistika následně a při nepřekročení maximálního počtu testů je inicializována a spuštěna nová simulace.

```
private void Bw_DoWork(object sender, DoWorkEventArgs e)
{
//...
  while (_runTest)
  {
    _runtime.Reset();
    _runSimulation = true;
    while (_runSimulation)
    {
     //...
    }
    _runtime.saveStats();
    _runtime.simulationCounter++;
    if (_runTest)
    {
      if (_runtime.simulationCounter > _runtime.maxSimulationRuns)
        _runTest = false;
    }
  }
}
```

Touto smyčkou je tak eliminován neduh původní podoby řízení testu, kdy byl pro spuštění nové simulace emulován opětovný stisk spuštění simulace.

4.5 Testování

Implantovaný návrh byl dále podroben testování. Veškeré testy byly provedeny se simulovaným vykonáváním testu na připojeného systému. K ověření platnosti byla vybrána sada automatů, které lze rozdělit dle jejich cílů testu na tři kategorie. Výsledky těchto testů jsou rozepsány v následujících podkapitolách.

4.5.1 Test přidaných funkcionalit

První test si kladl za cíl odzkoušet přidanou podporu synchronizačních kanálů typu urgent, broadcast. Pro tento test byl zvolen již zmíněný model 3.14 použitý při otestování prototypu. Tento model současně ověřil chování urgentních stavů, práce s hodinami, podmínky guard i invariant a přiřazení do proměnných. Test proběhl v pořádku, kdy vykonání broadcast synchronizace proběhlo dle očekávání a simulace zareagovala korektně na vyvolanou výjimku z důvodu porušení invariantu při příliš dlouhém setrvání ve stavu "V_pohybu". Na obr. 4.2 je zachycen krok simulace synchronizace při použitím manuálního krokovaní.

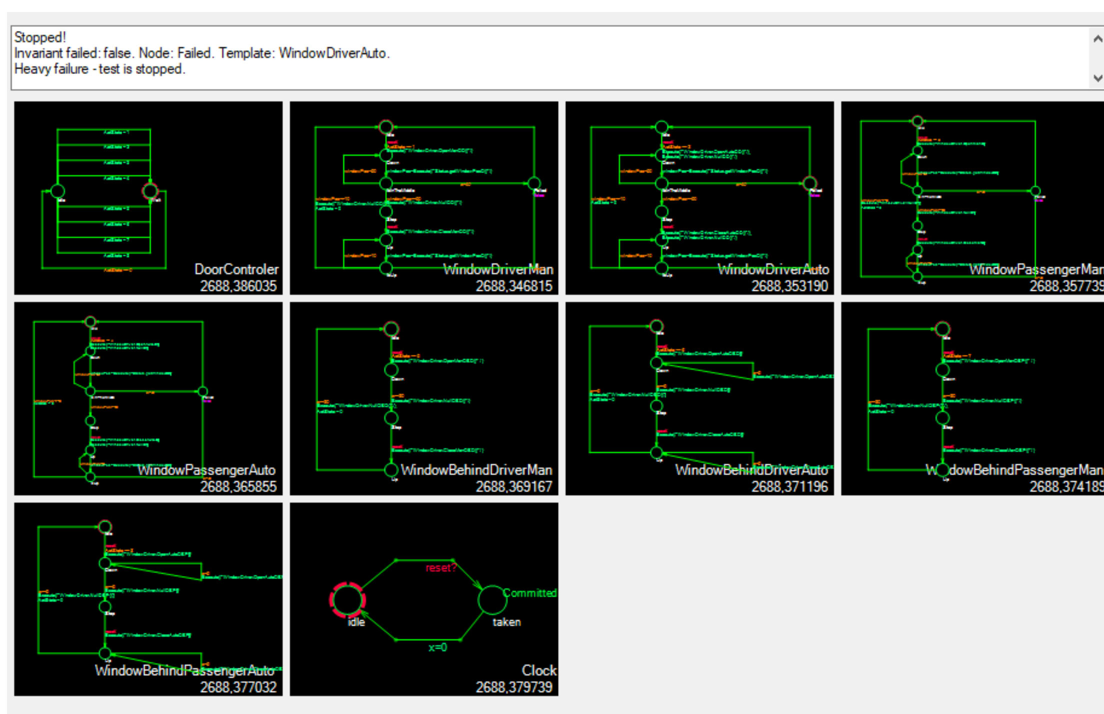
Obrázek 4.2: Zachycení kroku simulace po vykonaní broadcast synchronizace

Ve výpisu hlášení je vidět, že k výběru měla strategie tři možné přechody. První možností v tomto případě byla zvolená hrana automatu "Okno" do stavu "Otvirani" vyvolávající synchronizaci, druhou pak v automatu "Tlacitko" do stavu "Stistknuto"

a třetí v autoamtu "Prostoj" do stavu "Nic". Dále je v hlášení záznam provedených hran, tedy zvolená hran v automatu "Okno"do stavu "Otvirani"vyvolávající broadcast synchronizaci a k ní závislé hrany v automatu "Pohyb okna" do stavu "V_pohybu" a v autoamtu "Broadcast" do stavu "B2".

4.5.2 Test zpětné kompatibility

Druhým typem testem bylo ověření zpětné kompatibility modelů vytvořených pro původní simulační algoritmus na architektuře nové. Pro tento účel byly zvoleny dva modely. První byl model ovládaní stahování oken "ModelOvladaniStahovaniOken" a druhým byl modelem bezkolíčkového přístupu "ModelKessySystemu". Oba systémy se v rámci připojeného adaptéru Dummy chovaly dle očekávání. V rámci modelu ovládání stahování oken simulace korektně zareagovala na vyvolanou výjimku chybou invariantu. Naopak druhým modelem byla otestována podmínka ukončení. Konkretně opakovaným voláním funkce *Step()* nad maximální stanovený limit.



Obrázek 4.3: Model ovládání stahování oken - zachycení výjimky invariantu

4.5.3 Test načtení modelu

Posledním testem bylo korektní zpracování systémové deklarace, při implemntaci přesunuté do fáze načtení modelu. Pro tento test byl načten model obsahující automaty, které ovšem nebyly zahrnuty v systémové deklaraci. I toto ověření proběhlo úspěšně, kdy model byl v rámci načtené správně sestaven ignorováním automatů nenacházejících se v systémové deklaraci.

4.5.4 Zhodnocení

V rámci testování tak bylo pomocí zvolených modelů úspěšně ověřeno korektní chování simulace a správné načítání a sestavování modelu. Ověřena tak byla úspěšná implementace do nástroje Taster a platnost vytvořeného návrhu.

Kapitola 5

Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat novou verzi simulačního algoritmu na průchod sítí časovaných automatů simulačního nastroje Taster. Nejprve byla analyzována problematika časovaných automatů a původní podoba simulačního algoritmu nástroje Taster na průchod modelů využívajících tento modelační jazyk. V reakci na zjištěné nedostatky původní verze byl následně navržen nový simulační algoritmus společně s provázanou objektovou strukturou. V rámci návrhu byla současně rozšířena podpora modelačního jazyku UPPAAL o možnost využití urgentních a broadcast synchronizačních kanálů. Současně provázanost objektového návrhu umožnila navržení systematické architektury algoritmu vykonávající simulační krok průchodu sítí časovaných automatů. Během tvorby návrhu simulačního kroku byla též odhalena sémantická rozdílnost mezi jazyky nástroje UPPAAL a nástroje Taster. Dále byl navrhnut algoritmus řízení testu eliminující původní spouštěni simulace během automatického testování. V rámci řízení je řešeno zachytávaní chybových situací s jejich následným zpravováním.

Návrh byl dále implementován do nástroje Taster. Bylo přepracováno automatické testování využívající navržený algoritmus řízení testu a upraven nástroj pro manuální krokování simulace s průběžným výpisem vykonávání simulace. Implementace byla následně otestována na několika připravených modelech plošně ověřujících platnost navržené architektury algoritmu.

Výsledkem práce je tak rozšířená platforma pro testování systému s časovými závislostmi podporující tvorbu ještě komplexnějších modelů testovaných systémů. V dalších pracích na tomto projektu pak bude potřeba doplnit kompletní podporu všech funkcionalit původní architektury. Například v současném stavu je ze sady strategií podporována pouze strategie náhodného výběru. Kvůli světové epidemiologické situaci poslední doby, nebylo možné možné využít novou podobu Taster pro testování na připojeném reálném systému. I toto by tak mohlo být tématem navazujících prací.

Příloha A

Bibliografie

[1] Uppsala universitet a Aalborg university. *UPPAAL home*. 2021. URL: https://uppaal.org/.

[2] Rajeev Alur a David L. Dill. *A theory of timed automata*. Fundamental Study. Computer Science Department, Stanford University, 1994.

[3] M. Bacic. *On hardware-in-the-loop simulation*. Proceedings of the 44th IEEE Conference on Decision and Control. 2005.

[4] Mehrdad Bagheri. *Analysis of Model-based Testing methods for Embedded Systems*. Diplomová práce. Uppsala University, 2016.

[5] Gerd Behrmann, Alexandre David a Kim G. Larsen. *A Tutorial on Uppaal 4.0*. Department of Computer Science, Aalborg University, Denmark, 2006.

[6] Lerry Carley. *Engine Control Module Overview*. Tech Topics. 2013. URL: https://www.counterman.com/engine-control-module-the-brains-of-the-operation/.

[7] Alexandre David et al. *Uppaal 4.0: Small Tutorial*. Department of Computer Science, Aalborg University, Denmark, 2009. URL: https://www.it.uu.se/research/group/darts/uppaal/small_tutorial.pdf.

[8] Guru99. *Knuth: Computers and Typesetting*. 2021. URL: https://www.guru99.com/model-based-testing-tutorial.html.

[9] Konstantin Khokhlov. *Predikce selhání pro Model-Based integrační testování*. Bakalářská práce. ČVUT v Praze, 2020.

[10] Ing. Jaromír Krecl. *HIL simulace jako prostředek pro testování rídících jednotek v automobilu*. ŠKODA AUTO a.s. URL: http://dsp.vscht.cz/konference_matlab/matlab03/krecl.pdf.

[11] Jeong-Woo Lee, Ki-Yong Choi a Jung-Won Lee. *Collecting Big Data from Automotive ECUs beyond the CAN Bandwidth for Fault Visualization*. Mobile Information Systems. 2017. URL: https://doi.org/10.1155/2017/4395070.

[12] Jan Sobotka. *Methods for verification and validation of automotive dostributed systems*. Disertační práce. ČVUT v Praze, 2017.

[13] UPPAAL. *UPPAAL online documentation*. 2021. URL: https://docs.uppaal.org.

[14] Bc. Michal Veselka. *Integrační testování metodou Model-Based Testing - případová studie*. Diplomová práce. České vysoké učení technické v Praze, květ. 2019.

[15] Vratislav Vopička. *Programovací jazyk Python*. 2021. URL: https://adoc.pub/programovaci-jazyk-python-objektov-orientovany-citovano-z.html.

[16] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2013. URL: https://www.w3.org/TR/REC-xml/.

[17] Ally Winning. *Number of automotive ECUs continues to rise*. European Business Press SA. 2017. URL: https://www.eenewsautomotive.com/news/number-automotive-ecus-continues-rise.

[18] Andreas Zeller et al. *Prototyping with Python*. The Fuzzing Book. 2020. URL: https://www.fuzzingbook.org/beta/html/PrototypingWithPython.html.