# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Klíma** Jméno: **Matěj** Osobní číslo: **406320**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Softwarové inženýrství**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Framework na vyhodnocování vlastností testovacích scénářů pro testy konektivity v IoT systémech**

Název diplomové práce anglicky:

**Framework for evaluation of test sets for connectivity testing in IoT systems**

Pokyny pro vypracování:

Navrhněte a implementujte framework, který bude vyhodnocovat vlastnosti specializovaných testovacích scénářů určených k testování systémů Internet of Things (IoT) v případě omezeného nebo nedostupného síťového připojení. Testovací scénáře jsou specifikovány jako průchody grafem modelujícím procesy v testovaném IoT systému. Vyhodnocování bude probíhat na základě kritérií optimality zadaných školitelem půjdou vypočítat z vlastností vstupního modelu systému a sady vyhodnocovaných testovacích scénářů. Kromě toho implementujte jednoduchý test účinnosti scénářů založený na hypotetickém předpokladu výskytu chyb v různých částech procesu. Poslední částí bude sada testovacích algoritmů, které budou verifikovat konzistenci a správnost testovacích scénářů vytvořených měřenými algoritmy. Framework umožní přehledné zobrazení vlastností testovacích scénářů a srovnání výsledků různých algoritmů řešící danou úlohu. Při implementaci můžete využít stávajících knihoven platformy Oxygen.

Seznam doporučené literatury:

Jméno a pracoviště vedoucí(ho) diplomové práce:

**doc. Ing. Miroslav Bureš, Ph.D., laboratoř inteligentního testování softwaru FEL**

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

Datum zadání diplomové práce: **04.02.2019** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **20.09.2020**

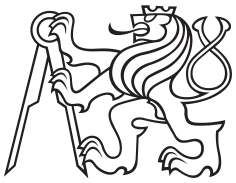
| doc. Ing. Miroslav Bureš, Ph.D.<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | prof. Ing. Pavel Ripka, CSc.<br>podpis děkana(ky) |
|---|---|---|

## 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 |

Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

# Framework na vyhodnocování vlastností testovacích scénářů pro testy konektivity v IoT systémech

Bc. Matěj Klíma

Školitel: doc. Ing. Miroslav Bureš, Ph.D.
Květen 2019

# Poděkování

Děkuji vedoucímu své diplomové práce doc. Ing. Miroslavu Burešovi Ph.D. za uvedení do problematiky testování IoT zařízení a za pomoc při analýze funkcionalit pro vyvíjený framework. Můj dík patří také Ing. Václavu Rechtbergerovi za předání užitečných informací ohledně předcházející implementace frameworku.


# 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, 22. 5. 2019

# Abstrakt

Tato diplomová práce se věnuje návrhu a vývoji frameworku pro vyhodnocování účinnosti technik pro návrh testovacích scénářů pro testy IoT zařízení v případě omezeného nebo nedostupného síťového připojení. Vyvíjený framework umožňuje specifickými kritérii vyhodnocovat kvalitu testovacích scénářů, tedy průchodů modelem procesů testovaného IoT systému. Pro kontrolu a zajištění kvality vygenerovaných testovacích scénářů framework obsahuje test jejich účinnosti ve smyslu potenciálu scénářů odhalit chyby v systému a také verifikaci jejich konzistence. Návrh a implementace obou funkcionalit jsou v této práci popsány.

Text práce obsahuje také seznámení se světem IoT zařízení a se způsoby zajištění jejich kvality. Dále je zde popsán návrh a implementace vyvíjeného frameworku. Součástí práce je naměření dat na zkušebních instancích různými algoritmy, řešícími testování systémů IoT v případě omezeného nebo nedostupného síťového připojení, a porovnání jejich výsledků.

Implementace frameworku využívá knihoven platformy Oxygen.



**Klíčová slova:** Testování systémů, internet věcí, zajištění kvality, testování na základě modelu, generování testovacích scénářů, automatizované testování, omezené síťové připojení



**Školitel:** doc. Ing. Miroslav Bureš, Ph.D.
Laboratoř inteligentního testování softwaru,
Katedra počítačů,
Karlovo náměstí 13,
121 35 Praha 2



# Abstract

This diploma thesis discusses the design and implementation of a framework that interprets the quality of specific techniques for automatic generation of test cases for testing the Internet of Things (IoT) devices in case of limited network connection. This framework allows measuring the quality of test cases using defined special criteria. Test cases discussed in this thesis are paths through a model of processes in an IoT System Under Test (SUT). To ensure the quality of generated test cases, the framework supports measuring the potential of test cases to find a bug in a SUT and to verify the consistency of generated test cases. The design and implementation of both of these functions are also part of this thesis.

In its introduction part, the thesis introduces the domain of IoT devices and discusses some methods on how to ensure their quality. This part is followed by the design and implementation of the framework. Measuring data on test instances by various algorithms solving testing of IoT systems in case of limited or unavailable network connection and comparing their results is also described in the thesis.

The implementation of the framework uses libraries of the Oxygen platform.



**Keywords:** System testing, Internet of Things, quality assurance, Model-based Testing, test cases generation, automated testing, limited network connectivity

**Title translation:** Framework for evaluation of test sets for connectivity testing in IoT systems

# Obsah

# Obrázky



# Tabulky

# Kapitola 1
# Úvod

Ústředním tématem této diplomové práce je testování systémů Internet of Things (IoT)[1]. Základní myšlenkou internetu věcí je implementace síťového připojení do zařízení, ve kterém připojení k síti nebylo původně běžné a která mohou díky tomuto připojení sdílet data a komunikovat s okolím bez zapojení člověka. Termín IoT se poprvé objevil již před dvaceti lety, ale teprve nyní se zařízení tohoto odvětví objevují všude okolo nás. Televize, termostaty, žárovky, zámky a také elektrické zásuvky, které lze přes síť ovládat například z mobilního telefonu odkudkoli na světě, lze dnes koupit stejně snadno, jako jejich nepřipojitelné alternativy.

Podle odhadů bude důležitost odvětví IoT i nadále narůstat. Zatímco v letošním roce je odhadováno „pouze“ 24 miliard zařízení připojených k internetu [1], k roku 2025 jich má být, například dle odhadů společnosti Huawei, až 100 miliard [2]. V souladu s růstem počtu zařízení připojených k internetu lze očekávat také nárůst důležitosti zajištění jejich kvality. To je však velmi široké téma, protože oblastí zajištění kvality IoT zařízení je celá řada. Jednou z nich, pro kterou zatím nebyly specifikovány přesné postupy, je testování IoT zařízení v případě omezeného nebo nedostupného síťového připojení některé z jeho vzdálených součástí. A framework, vyvinutý v rámci této diplomové práce, bude pomáhat vyvíjet postupy pro zajištění kvality právě v oblasti konektivity IoT systémů.

Pro testování konektivity IoT systémů je možné testovací scénáře, tedy sekvence kroků, které jsou určené pro otestování nějaké funkce systému, generovat automatizovaně. To spočívá v automatizovaném vytváření průchodů orientovaným grafem, který zachycuje zřetězené procesy probíhající v testovaném systému. Jednotlivé techniky, kterými by bylo možné testovací scénáře z modelu IoT zařízení generovat, jsou však zatím teprve předmětem výzkumu, například na katedře počítačů Fakulty elektrotechnické ČVUT. Toto automatizované testování přináší oproti manuálním testům mnoho výhod, například snížení času potřebného pro testování, snížení nákladů nebo zvýšenou spolehlivost zařízení. Podrobnější porovnání výhod a nevýhod automatizovaných a manuálních testů je dále uvedeno v textu této práce.

Usnadnění výzkumu v oblasti vývoje technik generování testovacích scénářů pro testování konektivity IoT zařízení umožní také tato diplomová práce.

[1] Termín je v textu občas zaměňován za český překlad „internet věcí“.

Vyvinutý framework totiž umožní specifikovanými technikami vygenerovat sady testovacích scénářů nad zadanými grafy a následně tyto sady porovná. To bude provedeno specifickými kritérii optimality, vypočítanými z vlastností vstupního modelu systému a sady vygenerovaných testovacích scénářů. Framework umožní také přehledné zobrazení vlastností testovacích scénářů a srovnání výsledků různých algoritmů řešící danou úlohu.

Dalším z cílů práce je implementovat test účinnosti scénářů, který je založený na hypotetickém předpokladu výskytu chyb v různých částech procesu. Implementovat je nutné v této práci také sadu testovacích algoritmů, které verifikují konzistenci a správnost testovacích scénářů vytvořených měřenými algoritmy.

Při implementaci frameworku je využito stávajících knihoven platformy Oxygen, která je v rámci práce doplněna o nové funkcionality.

Text práce je strukturován do několika kapitol. Kapitola 2 obsahuje vysvětlení základních pojmů použitých v této diplomové práci. V kapitole 3 je čtenář podrobněji seznámen s odvětvím internetu věcí a technologiemi, které jsou v zařízení IoT nejčastěji použity. Kapitola 4 obsahuje seznámení s problematikou testování IoT zařízení v případě omezeného nebo nedostupného síťového připojení. Uvedeny jsou zde také současné přístupy k testování, popis řešeného problému a model, použitý pro generování testovacích scénářů. Na konci kapitoly jsou popsány možnosti škálování intenzity testovacích scénářů a kritéria optimality. Kapitola 5 začíná specifikací funkčních požadavků na framework, pokračuje popisem použitých technologií a knihoven a také navržené architektury. Součástí této kapitoly je také návrh testu účinnosti scénářů a algoritmu na ověření konzistence scénářů. Kapitola 6 obsahuje popis implementace frameworku, testu účinnosti scénářů a algoritmu pro ověření konzistence scénářů. Nechybí v ní také popis zobrazení výsledků frameworku a na úplném konci této kapitoly uživatelská příručka. Kapitola 7 popisuje ověření funkcionalit frameworku díky spuštění vybraných algoritmů nad modelovými instancemi, které jsou v této kapitole taktéž popsány.

# Kapitola 2

# Vymezení základních pojmů

V této diplomové práci jsou použity některé odborné termíny, především z oblasti testování software. Čtenáři nemusí být vždy přesný význam těchto specifických termínů známý, navíc terminologie v oblasti testování systémů je poměrně nejednoznačná, proto následuje tabulka 2.1 s jejich vysvětlením a původním zněním v anglickém jazyce.

| **Název pojmu** | **Překlad** | **Význam** |
|---|---|---|
| Test case | Testovací scénář | Množina předpokladů, vstupů, akcí, očekávaných výsledků a výstupů, vytvořených pro účely otestování nějaké části software [3]. |
| Test set / suite | Sada testovacích scénářů | Sada často souběžně vytvořených testovacích scénářů, která je určená pro vykonání ve specifickém testovacím cyklu (část testování věnovaná jedné jasně identifikovatelné části systému) [3]. |
| Software development lifecycle (SDLC) | Životní cyklus vývoje informačního systému | Souhrn činností a návazností mezi nimi, které je potřeba učinit pro vznik a zprovoznění informačního systému. Mnoho činností a modelů věnujících se SDLC využívá následující fáze, identifikované W. Roycem: sběr požadavků, analýza, návrh, implementace, testování a nasazení [4]. |
| Testability | Testovatelnost | Odhad zda a jak snadno test odhalí přítomnost chyby v testovaném software či jeho části [5]. |
| Code coverage | Pokrytí kódu | Procentuální poměr řádků kódu pokrytých sadou testů [6]. |
| Test coverage | Testové pokrytí | Změření pokrytí funkčních požadavků sadou testů. [6]. |

| Název pojmu | Překlad | Význam |
|---|---|---|
| Model-based testing (MBT) | Testování na základě modelu | Návrh testů software z abstraktního modelu, který reprezentuje jeden či více aspektů daného software. Modely jsou většinou zachyceny pomocí UML diagramů [5]. |
| Coverage criteria | Kritérium pokrytí | Umožňuje strukturovat množinu vstupních hodnot a díky tomu efektivněji testovat. Kritéria pokrytí se dají rozdělit do čtyř kategorií, vycházejících z matematických struktur. Jsou jimi: kritéria pro rozdělení dle definičního oboru vstupů, grafy, logické výrazy a gramatiky [5]. |
| Optimality criterion | Kritérium optimality | Kritérium pro vyhodnocení optimality testovacího scénáře či celé sady scénářů, které umožňuje porovnávat efektivitu technik pro jejich generování [5]. |
| Unit test | Jednotkový test | Je určené pro testování „jednotek“ (například jednotlivých metod), produkovaných během implementační fáze. Jedná se o nejnižší úroveň testování a často je za ně zodpovědný sám autor implementace dané jednotky [5]. |
| Regression test | Regresní test | Spouští se většinou po nasazení nové verze systému. Slouží k ověření, že změny neovlivnily dříve fungující funkcionality ostatních částí systému [5]. |
| Functional test | Funkční test | Vyhodnocení, zda systém či jeho část odpovídá specifikovaným funkčním požadavkům [3]. |
| Security test | Test zabezpečení | K změření zabezpečení softwarového produktu [3]. Atributy, na kterých spočívá zabezpečení softwarového produktu, jsou důvěrnost (confidentiality), celistvost (integrity) a dostupnost (availability) [7]. |
| Performance test | Výkonový test | K ověření výkonu testovaného software [3]. |

| Název pojmu | Překlad | Význam |
|---|---|---|
| Stress test | Zátěžový test | Typ výkonového testu vykonávaný za podmínek, které odpovídají stanovenému limitu vytížení, či při omezené dostupnosti zdrojů (paměti, serverů) [3]. |
| Concurrency test | Test souběžného zatížení | Ověření, jak systém zvládne reagovat na souběžné požadavky od více uživatelů [6]. |
| Code coverage verification | Ověření pokrytí kódu | Analytická metoda, která vyhodnocuje které části softwaru byly pokryty danou sadou testovacích scénářů a které ne [3]. |
| Prime path | hlavní cesta | Hlavní cesta je definována jako cesta z uzlu $n_i$ do uzlu $n_j$, která není jako podcesta obsažena v žádné jiné cestě v grafu [5]. |
| Test requirement | testovací požadavek | Je specifický prvek softwarového artefaktu (tedy např. UML diagramu, kódu či seznamu požadavků), kterou musí dodržet či pokrýt testovací scénář [5]. |

**Tabulka 2.1:** Vymezení použitých termínů

# Kapitola 3

# Internet of Things

Tato kapitola seznamuje s odvětvím Internet of Things. Představují se zde základní prvky IoT infrastruktury a zařízení internetu věcí jsou zde klasifikovány do čtyř kategorií podle služeb, které nabízí. V poslední části je nastíněno, jak probíhá pro IoT zařízení zajištění kvality.

## 3.1 Svět internetu věcí

Termín Internet of Things se poprvé objevil v roce 1999 v prezentaci Kevina Ashtona pro Procter & Gamble. Ta se týkala nových nápadů pro využití RFID[1] v dodavatelských řetězcích a proběhla naprosto mimo zraky a zájem široké veřejnosti [8]. Nyní, dvacet let od první zmínky termínu Internet of Things, jsme na prahu plného využití IoT zařízení v každodenním životě. To bylo umožněno díky vývoji chytrých senzorů, protokolů Internetu a komunikačních technologií obecně [9].

Obecná definice pojmu IoT bohužel chybí [10]. Jeho podstatou je však všudypřítomnost výpočetního prostředí[2], ve kterém velké množství malých počítačů sdílí své informace, a konektivita k internetu, což umožňuje připojeným zařízením spolupracovat na dosažení požadavků z reálného světa. Oproti osobním počítačům, součástí všudypřítomného výpočetního prostředí může být téměř jakékoli zařízení v jakémkoli prostředí [9].

O důležitosti tohoto odvětví svědčí odhady expertů z firmy Cisco, kteří očekávají více než 24 miliard zařízení připojených k internetu v letošním roce [1]. Morgan Stanley pak odhaduje připojení více než 75 miliard zařízení k internetu k roku 2020 [10] a Huawei dokonce 100 miliard k roku 2025 [2]. V souladu s těmito odhady stoupá také očekávání na finanční dopad IoT zařízení na globální ekonomiku, který McKinsey Global Institute vyčíslil mezi 3,9 až 11,1 bilionů amerických dolarů k roku 2025 [11].

[1] Z naglického: Radio Frequency Identification. Této technologii je věnována část 3.2.1.
[2] V angličtině označovaného jako „ubiquitous computing environments“.

## 3.2 Stavební bloky

Stavebními bloky, ze kterých se sestávají zařízení internetu věcí, jsou prvky umožňující identifikaci, sběr dat, komunikaci, výpočty a interpretaci dat [11]. Každý z těchto prvků je podrobněji vysvětlen v následujících podkapitolách.

### 3.2.1 Identifikace

Jak již bylo zmíněno v úvodu, zařízení IoT se pohybují ve všudypřítomném výpočetním prostředí. V tom je však naprosto nezbytné, aby bylo každé zařízení jasně identifikovatelné a adresovatelné. Pro tyto potřeby existuje mnoho metod. Jedním z nich jsou ubiquitous codes (uCode), které spravuje organizace Ubiquitous ID Center, vzniklá v roce 2003 a zastřešující více než 500 organizací, které se podílí na vývoji standardů a specifikací [12]. Další možností identifikace je využití electronic product codes (EPC), univerzálního identifikátoru, který poskytuje unikátní a neměnnou identitu každému fyzickému objektu kdekoli na světě. Příkladem EPC identifikátoru je technologie Radio frequency identification (RFID).

#### Technologie RFID

Realizace RFID představuje malý štítek, skládající se z integrovaného obvodu a antény. Má schopnost komunikovat přes radiové vlny s RFID čtečkou, které může odesílat ale i od ní přijímat, zpracovávat a ukládat si informace. Informace, které čtečce odesílá, představují především unikátní identifikaci věci, se kterou je spjat. RFID čtečka pak tato data předává dále do zařízení, které data zpracuje a vyhodnotí. Čtečka může mít různé podoby. Od klasického osobního počítače po celý podnikový informační systém [13].

Největší využití je však v pasivním ultra vysokofrekvenčním systému RFID, který může fungovat bez baterie a s velmi malou anténou. Toho se využívá například v dodavatelských řetězcích. Zboží je totiž možné za vynaložení relativně nízkých nákladů označit těmito RFID štítky, a tak mít o něm aktuální informace při jeho cestě do cíle [13].

#### Adresace

Zajištění unikátní identifikace jednotlivých zařízení napomáhá, kromě identifikačního kódu zařízení, skrývajícího se například v RFID štítku, také jeho adresa v síti.

Způsoby adresace IoT zařízení zahrnují obě používané verze internetového protokolu, tedy IPv6 i IPv4.

Internetový protokol verze 6 (IPv6) umožňuje při neustálém nárůstu počtu zařízení připojených k internetu jejich přímou komunikaci a to bez nutnosti překladu adres, jako je tomu u staršího protokolu IPv4. Kapacita přibližně 4,3 miliard adres tohoto stále ještě velmi používaného protokolu by totiž k přímé komunikaci odhadovaných 100 miliard připojených IoT zařízení

v nadcházejících dekádách určitě nestačilo. Adresní prostor protokolu IPv6 má oproti tomu kapacitu celkem $2^{128}$ položek [10].

Vznikla také varianta IPv6 nazvaná IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), která poskytuje možnost adresovat i ta nejmenší zařízení s nízkým odběrem proudu připojená k internetu.

### 3.2.2 Sběr dat

Jedním z největších případů užití pro IoT zařízení je sběr dat ze senzorů a jejich přeposílání k dalšímu zpracování, uložení do databáze či vystavení na cloudu [9].

Zařízení internetu věcí mohou obsahovat inteligentní senzory (smart sensors), aktuátory nebo sensory použité pro nositelnou elektroniku (wearables).

**Inteligentní senzory** mají schopnosti přesahující pouhé generování naměřených či sledovaných hodnot. Oproti tomu nabízí funkce významně zjednodušující zapojení těchto senzorů do výsledné aplikace či celého síťového prostředí [14].

**Aktuátory** lze chápat jako zařízení svou činností protikladné k senzorům. Jejich úkolem je totiž převést digitální informace na mechanickou aktivitu. V IoT zařízení je lze nalézt například v inteligentních domech při aktivaci osvětlení, vytápění, závlahového systému či zabezpečení [14].

Jako **nositelná elektronika** jsou označována zařízení, která mohou být nošena člověkem v běžném každodenním prostředí a to jak přímo na těle, tak v kapse či dokonce jako součást oblečení. Senzory pro nositelnou elektroniku dnes umožňují měřit intenzitu sportovní aktivity, lokalizaci v prostoru ale i sběr důležitých zdravotních informací. Senzory pro měření tlaku, srdečního pulsu nebo tělesné teploty se dnes stále více objevují jako součásti zařízení určených pro každodenní nošení [15].

### 3.2.3 Komunikace

Komunikace, jako vzájemná výměna dat mezi IoT zařízeními, může fungovat ve čtyřech různých modelech [10]:

- **Zařízení se zařízením**[3]
  Je příkladem běžného komunikačního modelu, který funguje bez nutnosti jakéhokoli prostředníka. Můžeme jej najít při ovládání jednoduchých elementů chytré domácnosti, jako například žárovek.

- **Zařízení s cloudovou službou**[4]
  Dalším modelem komunikace je spojení zařízení s cloudovou službou, která ukládá a vyhodnocuje naměřená data. Příkladem tohoto modelu je chytrý termostat od firmy Nest, který do cloudu odesílá data o spotřebě energie domácnosti. Nazpátek posléze uživateli přijde například měsíční porovnání s ostatními majiteli tohoto termostatu.

[3] V originále „Device-to-Device“
[4] V originále „Device-to-Cloud“

- **Zařízení s místní bránou**[5]
  Třetím možným modelem komunikace je spojení zařízení s místní bránou, která má funkci prostředníka mezi zařízením a cloudovou službou a která obstarává například zabezpečení či překlad dat. Dobrým příkladem tohoto modelu jsou různé fitness pomůcky, které pro správnou činnost potřebují jako prostředníka chytrý telefon.

- **Zařízení sdílející data**[6]
  Posledním komunikačním modelem je model umožňující sdílení dat do zařízení z různých zdrojů a cloudových služeb. To ovšem vyžaduje schopnost vzájemné výměny a interpretace informací mezi těmito platformami.

Společným požadavkem na výše zmíněné modely a komunikaci mezi IoT zařízeními je nízký odběr elektrické energie jejich součástí. Prvky účastnící se komunikace se navíc často musí vypořádat se šumem a ztrátami v síti [9]. Těmto požadavkům vyhovuje několik běžně používaných a níže popsaných komunikačních technologií, přičemž jejich výběr se napříč různými kategoriemi IoT zařízení často liší.

### Celulární rádiová síť

Mezi nejpoužívanější technologie komunikující na dlouhou vzdálenost patří jednoznačně celulární rádiová síť. Síťové technologie 3G, 3GPP LTE a 4G se v dnešních IoT zařízeních vyskytují v masovém zastoupení a k nim přibude také nastupující technologie 5G. Toho je docíleno díky relativně nízkým nákladům za nasazení, vysoké úrovni zabezpečení a jednoduché správě. Nevýhodou však zůstává, že tyto technologie jsou navrženy primárně pro širokopásmové připojení a tedy ne pro přímou komunikaci zařízení se zařízením [16].

### IEEE802.11 Wi-Fi

Do této kategorie patří standard IEEE802.11 Wi-Fi. Navržen v roce 1997 se záměrem nahradit Ethernet, a tedy mimo zájmy prostředí IoT, ztrácí tento standard oproti svým „konkurentům“ v několika oblastech. Například Bluetooth umožňuje při komunikaci na podobnou vzdálenost menší spotřebu energie a ZigBee umožňuje komunikaci na delší vzdálenost. V určitých oblastech IoT je však, například díky vysoké míře pokrytí Wi-Fi signálem, používaným nástrojem [16].

### Bluetooth

Dalším komunikačním prostředkem na krátkou vzdálenost je Bluetooth, který zažívá v poslední době, i díky nástupu IoT zařízení, veliký rozmach. Od verze 4.0, uvedené v červnu 2010 [17], obsahuje standard Bluetooth chytrou a nízkoenergetickou verzi nazvanou Bluetooth Low Energy. Zároveň se v dalším

[5] V originále „Device-to-Gateway“
[6] V originále „Back-End Data-Sharing“

vývoji pracuje na zvyšování přenosové rychlosti, která v aktuální[7] verzi 5.1 dosahuje až 2Mbit/s a umožňuje komunikovat až do vzdálenosti 200 m [17]. Snižování potřebné energie, zvýšení odolnosti proti šumu a akcent na zlepšování zabezpečení komunikace činí z technologie Bluetooth pro IoT zařízení velmi atraktivní komunikační prostředek. Dle odhadů společnosti Allied Business Intelligence (ABI) bude ze 48 miliard zařízení, připojených k roku 2021 k internetu, dokonce 30 % z nich v sobě skrývat technologii Bluetooth [18].

#### IEEE 802.15.4 Zigbee

Levným a nízkoenergetickým bezdrátovým zařízením pro komunikaci je síťový standard IEEE 802.15.4 Zigbee. Jeho užití je možné v podobném typu zařízení jako Bluetooth, oproti kterému vyniká větším dosahem, avšak ztrácí na přenosové rychlosti a energetické účinnosti [9].

#### Z-Wave

Dalším populárním komunikačním protokolem je Z-Wave, jehož užití je možné najít především v chytrých domácnostech a malých podnicích. Dosah okolo 30 m a model komunikace zařízení se zařízením s možností výměny malého množství dat je totiž ideální pro zařízení jako jsou požární senzory, osvětlení chytré domácnosti, topení, klimatizace a podobně [9].

#### Near Field Communications (NFC)

Populárním protokolem pro komunikaci zařízení v bezprostřední vzdálenosti (jednotky až několik desítek centimetrů [19, 20]) je protokol NFC. Ten funguje na frekvenci 13,56 MHz a podporuje datový tok až 424 kbps [19]. Jeho princip vychází z technologie RFID a byl navržen mimo jiné i pro zajištění lepšího zabezpečení. Tomu napomáhá možnost šifrované komunikace mezi NFC prvky a také nutnost jejich bezprostřední vzdálenosti [20]. Díky tomu je možné dnes protokol NFC využít například při placení v obchodech mobilním telefonem prostřednictvím platebního terminálu.

### 3.2.4 Výpočetní část

Procesorové jednotky (např. mikrokontroléry, mikroprocesory, programovatelná hradlová pole a systémy na čipu) a na nich běžící software reprezentuje mozek a výpočetní potenciál IoT. V nedávné době dokonce vzniklo pro potřeby IoT zařízení několik hardwarových platforem a to například Arduino, UDOO, FriendlyARM, Intel Galileo, Raspberry PI, Gadgeteer, BeagleBone, Cubieboard, Z1, WiSense, Mulle, and T-Mote Sky [9].

Existují již také celé softwarové platformy přizpůsobené pro poskytování IoT funkcionalit. Jednou z nich je například operační systém Contiki RTOS, který prostřednictvím simulátoru Cooja usnadňuje vývoj IoT aplikací [9].

[7] K březnu 2019

Pro výpočetní účely mohou sloužit také cloudové platformy, které umožňují v reálném čase zpracovávat obrovské množství dat. Některé z nich své funkce dokonce poskytují zcela zdarma [9].

### 3.2.5 Interpretace dat

Všechny předchozí části by byly zbytečné, kdyby se data naměřená a poskytnutá IoT zařízení nedala efektivně interpretovat. Pro tyto účely byly uvedeny technologie jako Resource Description Framework (RDF) či Web Ontology Language (OWL). Konsorcium World Wide Web (W3C) pak pro možnost velmi efektivní komunikace bez nutnosti rekonstrukce textové reprezentace zpráv (jako například u formátů JSON, XML, JavaScript a HTML) zavedlo formát Efficient XML Interchange (EXI) [21].

## 3.3 Kategorizace a příklady IoT zařízení

Jedním z problémů IoT prostředí je jeho různorodost a rozlehlost. Existuje nepřeberné množství konceptů a technologií, které se zde uplatňují a ve kterých je najít nějaký systém velmi obtížné. Co však rozdělit lze, jsou služby, které IoT zařízení poskytují. A to do čtyř kategorií na služby: identifikující zařízení[8], sbírající data[9], rozhodující[10] a všudypřítomné[11] [22]. Popis jednotlivých zařízení, rozdělených dle stejného schématu, je uveden v následujících bodech.

### 3.3.1 Zařízení pro identifikaci

Nejjednodušší službou, kterou IoT zařízení mohou poskytnout, je identifikace. Toho je ve velké míře využíváno například ve výrobě, při sledování zásilek, u dodavatelských řetězců ale i v komplexnějších IoT zařízeních [22]. Pro provoz je kromě nějakého identifikátoru, nejčastěji RFID štítku, nutná také čtečka, která s identifikátorem komunikuje a dále zpracovává přijaté informace. Identifikátory svou identitu mohou aktivně vysílat do okolí, nebo naopak pouze pasivně odpovídat na požadavek ze čtečky. Pro tento druh zařízení je charakteristická absence přímého zdroje energie, jako je tomu například u pasivních RFID štítků [13].

### 3.3.2 Zařízení pro sběr dat

Proces sběru, zpracování a předání dat získaných ze senzorů do aplikace spadá do této kategorie služeb poskytovaných mnohými IoT zařízeními. Pro splnění mnohých komplexnějších požadavků na aplikaci se pak často používá více komunikačních kanálů současně. Například RFID pro získání identity zařízení, ZigBee pro sběr dat ze senzorů a webové služby (Web Services), jako XML či

[8] V originále „Identity-Related Services“
[9] V originále „Information Aggregation Services“
[10] V originále „Collaborative-Aware Services“
[11] V originále „Ubiquitous Services“

JSON, pro agregaci dat do aplikace [22]. Monitorování důležitých ukazatelů typu teploty, vlhkosti, světla či zabezpečení v inteligentní domácnosti je jedním z příkladů využití této kategorie služeb IoT zařízení.

### 3.3.3 Rozhodující zařízení

Oproti předchozí kategorii, tato jsou určené přímo pro vykonání akce, odvislé od hodnot naměřených z připojených senzorů. Zařízení v této kategorii mezi sebou přímo komunikují, což klade vyšší požadavky na odezvu, výpočetní výkon, kapacitu úložiště, spotřebu energie ale i zabezpečení těchto IoT zařízení. Technologie IPv6, diskutovaná v části 3.2.1, umožňuje rozmach těchto zařízení, neboť pro potřeby přímé komunikace je nutná adresace všech připojených zařízení v síti [22].

### 3.3.4 Všudypřítomná zařízení

Tato všudypřítomná zařízení internetu budou jednou tím, co se lidem představí pod pojmem IoT. Budou přístupná pro všechny a za každých okolností, ovladatelná z mobilního telefonu, počítače či hodinek. V současné chvíli brání plnému rozvoji těchto zařízení hlavně odlišnost používaných protokolů, kvůli čemuž spolu zařízení nemohou sjednoceně komunikovat. Využití RESTful webových služeb a IPv6 však nabízí do budoucna pro vznik univerzální architektury pro komunikaci napříč světem IoT zařízení veliký potenciál [22].

## 3.4 Zajištění kvality

S očekávaným rozvojem technologií, podílejících se na fungování zařízení internetu věcí, musí držet krok také zajištění jejich kvality[12]. Nositelné zařízení, které by měřilo tepovou frekvenci s pouze 50 % přesností, žárovka, která by reagovala na pokyny s několikaminutovým zpožděním či IoT kamera s nedostatečným zabezpečením, by používány jistě nebyly.

QA v běžném procesu tvorby softwarového produktu ošetřuje každý krok jeho tvorby, což umožňuje existenci chyb v produktu významně snížit. V prostředí vývoje IoT zařízení je však situace složitější. Existují zde oblasti, například zabezpečení a zajištění soukromí, pro které již techniky zajištění kvality existují a v praxi se používají. Stále ale zbývá i několik oblastí, pro které techniky jak změřit jejich kvalitu chybí [23].

Málo pokryté oblasti zajištění kvality IoT zařízení jsou například [23]:

- Testování vzájemné kompatibility jednotlivých prvků IoT systému.
- Testování chování IoT systému v případě omezeného síťového připojení.
- Techniky, kterými je možné zredukovat vysoký počet možných konfigurací testovaného systému (různých kombinací, která zařízení se připojí).

[12] V textu je termín „zajištění kvality“ zkrácen na „QA“, pocházející z anglického termínu „quality assurance“.

### 3.4.1 Hrozby pro IoT zařízení

Ahmed a kol. identifikují devět významných problémů, které IoT zařízení mohou ohrozit. Jsou jimi [23]:

1. Vysoká konkurence a tlak na snižování cen jednotlivých zařízení, což může vést k nevynaložení nutných výdajů na zajištění kvality.
2. Fyzická dostupnost senzorů či celých zařízení, což umožňuje jejich poškození.
3. Obtížná aktualizace některých typů zařízení. To může být problém především při objevení nového způsobu, jak obejít zabezpečení daného zařízení.
4. Požadavky na nízkou energetickou náročnost IoT zařízení, což může také vést k jejich nedostatečnému zabezpečení.
5. Vzhledem k různorodosti standardů, širokému spektru použitelných softwarových i hardwarových technologií a i jisté nevyzrálosti celého odvětví, může zajištění kvality IoT zařízení vyžadovat testování i dalších vrstev IoT infrastruktury, než jen samotné vyvíjené aplikace. To však v případě omezeného rozpočtu nemusí být vždy možné.
6. Zařízení IoT jsou připojeny k internetu a jejich počet bude neustále narůstat. Při připojení do jedné vnitřní sítě však bude zabezpečení všech těchto zařízení významně záviset na tom nejméně zabezpečeném. Pro bezpečnost systému totiž platí axiom, který širší veřejnost zná pod rčením používaným pro pevnost řetězu, tedy že je jen tak silný, jak silný je jeho nejslabší článek [24].
7. V rozlehlém světě IoT zařízení člověk nemusí mít detailní přehled o všech připojených zařízení, navíc vystavených různým aktualizacím, což může opět vést k mnoha bezpečnostním rizikům.
8. Potenciální bezpečnostní rizika mohou plynout také z amatérsky postavených zařízení, nerespektujících standardy daného odvětví.
9. Možnou hrozbou je také zvyšující se využívání IoT zařízení v oborech, na kterých závisí lidské zdraví. Tím se významně zvyšuje dopad škod, které může IoT zařízení způsobit při svém selhání.

### 3.4.2 Automatizace testování

Fáze testování je ve dnes používaných metodikách pro tvorbu softwarového produktu vedle sběru požadavků, analýzy, návrhu, implementace a nasazení jednou ze základních fází životního cyklu informačního systému. Často se však může stát tou částí nejdražší. Je dokonce zdokumentován příklad, kdy testování zabralo přes 50 % všech zdrojů vyčleněných na vytvoření tohoto systému [25].

Jedním ze způsobů, jak při tvorbě softwarového produktu ušetřit, je snížit výdaje nutné na testování. Toho lze docílit pomocí automatizace [26].

Automatizované testování software může mít několik podob. Starší definice z roku 1999 zní, že je to správa a provedení činností, jako je vývoj a exekuce testovacích skriptů pro ověření testovaných požadavků, s využití automatizovaného nástroje [27]. Oproti tomu definice o deset let novější říká, že automatizace je vytvoření a zavedení softwarové technologie napříč celým testovacím cyklem s cílem vylepšit jeho účinnost a efektivitu [6].

Pohled na automatizované testování se tedy mění. Od vytváření jednotlivých skriptů s využitím externího nástroje se přechází k automatizaci celého testovacího procesu. Dnešní volná definice by tedy mohla znít, že automatizované testování je tvorba software, který napomáhá efektivnímu a opakovatelnému vyhodnocování kvality softwaru určeného k otestování.

### Výhody automatizace testů

Rafi a kol. ve svém výzkumu ověřili v praxi platnost několika tvrzení o důsledcích začlenění automatizace do zajištění kvality systému.

Ve výsledných systémech byl díky automatizovaným testům [26]:

- Zjištěn nižší počet chyb a tedy větší kvalita produktu.
- Naměřeno větší testové pokrytí kódu.
- Snížen čas potřebný pro testování. A to především díky možnosti spustit v čase souběžně více testů.
- Zvýšena spolehlivost oproti manuálním testům. Jejich exekuce je totiž vždy stejná a neliší se tím, kdo jí provádí, jako se to může stát u manuálních testů.
- Zvýšená důvěryhodnost ve kvalitu systému např. z pohledu vývojářů.
- Zajištěna lepší opakovatelnost testů. Ovšem pouze za předpokladu, že byly správně navržené.
- Snížená potřeba lidských zdrojů, díky automatizované exekuci testů.
- Při vysokém stupni automatizace byly potřeba, oproti manuální exekuci, nižší náklady.
- Byl nalezen vyšší počet chyb v testovaném systému.

### Omezení automatizace testů

Rafi a kol. ve svém výzkumu identifikují také omezení a nevýhody automatizovaných testů, oproti testům manuálním [26]:

- Obtížná automatizace některých druhů testů. Například těch, které pro vytvoření vyžadují rozsáhlé znalosti a zkušenosti v oboru.

- Nedosažení očekávaných výsledků kvůli nedostatečnému pochopení výhod a způsobu testování pomocí automatizovaných testů.
- Složitější údržba kvůli rychlému zastarávání použitých technologií a změnám v testovaném software.
- Náročnější příprava testů a přidružené infrastruktury při snaze o docílení dlouhodobého užitku z automatizovaných testů.
- Chybná očekávání od automatizovaných testů, například redukovaná v pouhou snahu o úsporu nákladů. Tím jsou přehlíženy ostatní výhody automatizace.
- Potenciál chybně zvolené strategie automatizace, tedy výběru jaké části software a s jakými vstupy a parametry automatizovat. Kvůli tomu mohou být opominuty některé výhody automatizace a vložené náklady se nemusí vyplatit.
- Nedostatek lidí s požadovanými znalostmi. Pro tvorbu automatizovaných testů je potřeba mít zkušenosti s používání vhodných automatizačních nástrojů, programováním, znalost architektury systémů apod.

### Testy vhodné pro automatizaci

Rozhodnutí o tvorbě automatizovaných testů by vždy měla předcházet analýza návratnosti investice. Na jejím základě se pak lze rozhodnout, které testy je hospodárné automatizovat a které ne. Protože všechny testy sice lze automatizovat, ale zdaleka ne u všech je to potřeba a u některých to také nemusí být ekonomicky efektivní [25, 28, 29].

M. Bureš ve svém výzkumu o situaci automatizovaných testů v České republice píše, že návratnost investice vložené do tvorby automatizovaných testů, která byla vyhodnocována pouze ve 43 % z 28 sledovaných projektů, byla dosažena jen u 26 % projektů. U zbylých přibližně 17 % návratnost investice dosažena nebyla. Z dalších výsledků tohoto výzkumu vyplývá, že největšími faktory negativně ovlivňujícími návratnost investic do tvorby automatizovaných testů, jsou podcenění výdajů na správu testů a nedostatečně zkušený a kvalifikovaný tým [30].

Pro některé druhy testů je tedy jejich automatizace vhodnější, než pro jiné. Dustin a kol. označují ve své knize následující druhy testů za vhodné k automatizaci [6]:

- **Jednotkové testy**
  Jsou určené pro testování „jednotek“ (například jednotlivých metod), produkovaných během implementační fáze. Jedná se o nejnižší úroveň testování a často je za ně zodpovědný sám autor implementace dané jednotky [5].
- **Regresní testy**
  Spouští se většinou po nasazení nové verze systému. Slouží k ověření, že změny neovlivnily dříve fungující části systému [5].

- **Funkční testy**
  Vyhodnocují, zda systém či jeho část odpovídá specifikovaným funkčním požadavkům [3].

- **Testy zabezpečení**
  Používají se k změření kvality zabezpečení softwarového produktu [3]. Atributy, na kterých spočívá zabezpečení softwarového produktu, jsou důvěrnost, celistvost a dostupnost [7].

- **Výkonové testy**
  Slouží k ověření výkonu testovaného software [3]. Jednotlivými indikátory pro měření výkonu software jsou dostupnost, odezva, propustnost a vytíženost [31].

- **Zátěžové testy**
  Jedná se o typ výkonového testu, který je spouštěn za podmínek, odpovídajícím stanovenému limitu vytížení, či při omezené dostupnosti zdrojů (paměti, serverů) [3].

- **Testy souběžného zatížení**
  Slouží k ověření jak systém zvládne reagovat na souběžné požadavky od více uživatelů [6].

Nejčastěji automatizovanými testy jsou dle výsledků výzkumu J. Kasurinena a kol. jednotkové testy a regresní testy [25].

### Automatizace v prostředí IoT

Do prostředí internetu věcí se postupně zavádí mnoho standardních druhů testů. Jedním z takových příkladů je testování založené na modelu systému. Tento model může být například UML model tříd, UML sekvenční diagram či model vytvořen jinými metodami specifickými pro prostředí IoT. Z těchto modelů jsou následně strojově vygenerovány testovací scénáře, což zvyšuje přesnost a pokrytí těchto testů [23].

Některé druhy testů je však v současné době komplikované automaticky otestovat. Bureš a kol. ve svém výzkumu z praxe zjistili, že za nejnáročnější považují firmy testování v případě omezeného nebo nedostupného síťového připojení [23]. I proto je tento druh testování tématem této diplomové práce.

# Kapitola 4

# Testování IoT zařízení v případě omezeného nebo nedostupného síťového připojení

V této kapitole je specifikován řešený problém při testování IoT zařízení v případě omezeného nebo nedostupného síťového připojení a jsou zde popsány současné přístupy k řešení tohoto problému. Je zde vysvětlen model procesů IoT systému a definován testovací scénář, který se z modelu systému generuje. Na závěr kapitoly jsou uvedeny možnosti škálování intenzity testovacích scénářů a také kritéria optimality, které umožňují porovnat efektivitu vygenerovaných scénářů.

## 4.1 Popis problému

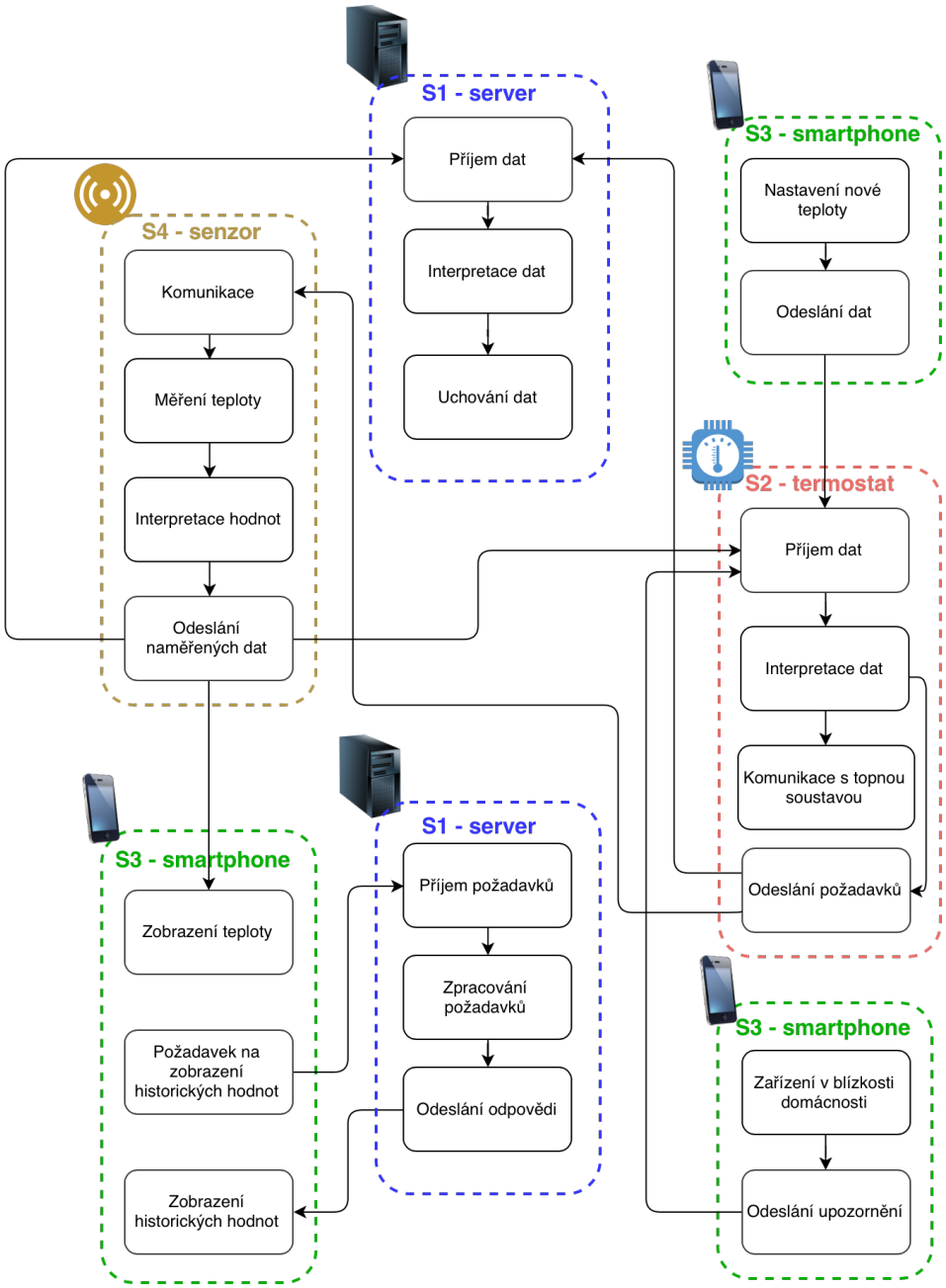
Pro pochopení popisované problematiky je dobré si představit podobu propojených a komunikujících IoT zařízení. Příkladem může být přibližný model vzájemné komunikace pro potřeby řízení vytápění v chytré domácnosti, uvedený na obrázku 4.1. Ten obsahuje zařízení: S1 - server, S2 - termostat, S3 - smartphone (chytrý telefon) a nakonec S4 - senzor teploty. Server v tomto modelu slouží pro příjem, interpretaci a uchování dat a pro odpovědi na požadavky vyslané z připojených zařízení. Termostat přijímá data ze senzoru teploty a také přijímá požadavky z chytrého telefonu. Na jejich základě řídí domovní vytápění a dále komunikuje se serverem, tepelným senzorem a chytrým telefonem. Chytrý telefon umožňuje také získání historických hodnot o spotřebě energie topnou soustavou ze vzdáleného serveru.

Pro testování IoT zařízení v případě omezeného nebo nedostupného síťového připojení je nutné pokrýt dvě situace, které mohou nastat:

1. Během určité části procesu komunikace mezi jednotlivými prvky systému bylo přerušeno připojení.

2. Dříve přerušené síťové připojení bylo opět obnoveno.

V prvním případě, tedy že připojení bylo přerušeno, je nutné zkontrolovat například:




**Obrázek 4.1:** Znázornění procesů v IoT systému vytápění chytré domácnosti

- Informovanost uživatelů a závislých prvků v systému o této skutečnosti.
- Podmínky konzistentního uchování dat, během jejichž odesílání bylo připojení přerušeno. Tedy zda to systém dokáže a případně po jak dlouhou dobu.
- Zda tím není ovlivněn výkon ostatních zařízení v systému.

Po obnovení připojení je nutné zkontrolovat například:

- Předání informací o této skutečnosti uživateli a závislým prvkům v systému.
- V případě, že systém uchovává data, během jejichž odesílání bylo připojení přerušeno, zda byla tato data úspěšně obnovena a nedošlo k jejich ztrátě či modifikaci.
- V případě, že systém funguje na základě transakcí, zajištění, že při obnovení připojení je tato přerušená transakce dokončena, nebo kompletně zrušena (nemůže tedy nastat situace, že by byla provedena jen z části).
- Zda po obnovení připojení není neakceptovatelným způsobem ovlivněn výkon zařízení.

K ověření chování testovaného systému v situacích zmíněných v předchozí části je vhodné zvolit testování na základě modelu. Hlavním důvodem je možnost efektivního a opakovaného generování testovacích scénářů z modelu systému, což se vyplatí především při změnách testovaného systému [5].

Pro vytváření testovacích scénářů se v testování na základě modelu často využívá strategie založená na průchodech[1]. To znamená, že zmíněný model představuje zřetězení všech procesů, které probíhají v testovaném systému. Z nich lz vhodnou technikou vybrat výsledné testovací scénáře [32].

Problém testování IoT zařízení v případě omezeného nebo nedostupného síťového připojení je tedy redukován na nalezení techniky, která vygeneruje z modelu procesů optimální sadu testovacích scénářů, pokrývající procesy podléhající výpadku připojení.

## 4.2 Řešení problému

Použití tradičních technik pro generování testovacích scénářů z modelu procesů, například PCT (popsané v části 7.2.3) je však v prostředí IoT nevyhovující. Ve výsledných scénářích jsou totiž průchody, probíhající uvnitř systémů a tedy neohrožené výpadkem připojení, zastoupeny ve stejném množství, jako průchody podléhající možnému výpadku připojení [33]. Ruční výběr pouze takových scénářů, které testují průchody podléhající výpadku připojením, by také nemusel vést k optimálním výsledkům a pravděpodobně by obsahoval zbytečně vysoký počet průchodů systémem.

Vývoj nové techniky, řešící tento problém, je v současné době předmětem výzkumu na katedře počítačů. Stručný popis její implementace je uveden v části 7.2.1.

## 4.3 Model procesů IoT systému

Pro vytvoření optimální techniky pro generování scénářů z modelu průchodů testovaným IoT systémem je nutné tento model přesněji specifikovat. V této

[1] Anglicky „Path-based Test Selection Strategy“

práci používáme model systému definovaný vedoucím práce, který je výsledkem vývoje a výzkumu techniky pro testování chování IoT systému v případě omezeného síťového připojení v rámci výzkumné skupiny STILL na katedře počítačů ČVUT FEL. Jedná se o orientovaný graf $G = (V, E, v_s, V_e)$. Množina $V$ značí vrcholy a $E$ hrany. Prvky množiny $E$ jsou podmnožinou $V \times V$. Uzel $v_s \in V$ je počátečním uzlem $G$ a může být jen jeden. Množina $V_e \subset V$ obsahuje všechny koncové uzly.

Pro možnost vygenerovat testovací scénáře z $G$ je nutné splnění podmínek:

- $G$ je jednoduchý a souvislý.
- Existuje v něm právě jeden uzel $v_s \in V$, do kterého nevede žádná hrana a existuje z něj orientovaný sled do všech ostatních vrcholů v grafu.
- V $G$ existuje neprázdná množina uzlů $V_e \subseteq V$, ze kterého nevede žádná hrana.
- Pro každý uzel $v_i \in V$ existuje orientovaný sled do uzlu $v_e \in V_e$.

Výše zmíněná definice podkladového grafu, jako modelu testovaného zařízení, se nijak neliší od definic pro klasické „Path-based“ testování [34, 35]. Pro IoT zařízení je však nutné zavést do grafu další prvky, popsané v následujícím textu.

### Pravděpodobnost výpadku připojení na hraně (PVP)

Označuje pravděpodobnost výpadku připojení na kterékoli hraně $e \in E$. V textu práce je tato skutečnost značena jako $pvp(e)$.

### Hraniční PVP

Znamená hraniční pravděpodobnost výpadku připojení, pro kterou generujeme testovací scénáře $T$. Nastavením tohoto údaje, značeného v textu práce jako $limit$, na číslo $n$ předpokládáme, že všechny hrany $e$, které mají $pvp(e)$ větší rovnou $n$ budou ohroženy výpadkem síťového připojení a jsou tedy určené k otestování.

### Zóna omezeného připojení (ZOP)

Pro vygenerování testovacích scénářů je nutné zavést koncept specifických zón v grafu, označených jako: „zóna omezeného připojení“ (ZOP). K definici této zóny je nejprve nutné zadefinovat pojem **ZOP hrana**, což je každá hrana $e \in E$, pro kterou $pvp(e) \geq limit$ a pojem **nZOP hrana**, což je hrana $e \in E$, pro kterou $pvp(e) < limit$.

**ZOP** $Z$ definujeme jako souvislý podgraf $G$, který obsahuje pouze ZOP hrany. Pro určitou hodnotu $limit$ může $G$ obsahovat více než jednu ZOP.

### Množina VSTUP uzlů

Uzel $v \in V$ patří do množiny VSTUP uzlů zóny $Z$, pokud splňuje jednu z následujících podmínek:

1. $v = v_s$ a $v$ obsahuje výstupní hranu, která je hranou v $Z$.
2. $v$ obsahuje výstupní hranu, která je hranou v $Z$ a $v$ obsahuje vstupní hranu, která není hranou v $Z$.

### Množina VÝSTUP uzlů

Uzel $v \in V$ patří do množiny VÝSTUP uzlů zóny $Z$, pokud splňuje jednu z následujících podmínek:

1. $v \in V_e$ a $v$ obsahuje vstupní hranu, která je hranou $Z$.
2. $v$ obsahuje vstupní hranu, která je hranou $Z$ a $v$ obsahuje výstupní hranu, která není hranou v $Z$.

### Hraniční uzel

Je buď VSTUP uzel, nebo VÝSTUP uzel zóny $Z$. Množina všech hraničních uzlů v $Z$ je označena jako $hraniceZOP(Z)$, případně všech hraničních uzlů v $G$ jako $hraniceZOP(G)$.

### Příklad modelu

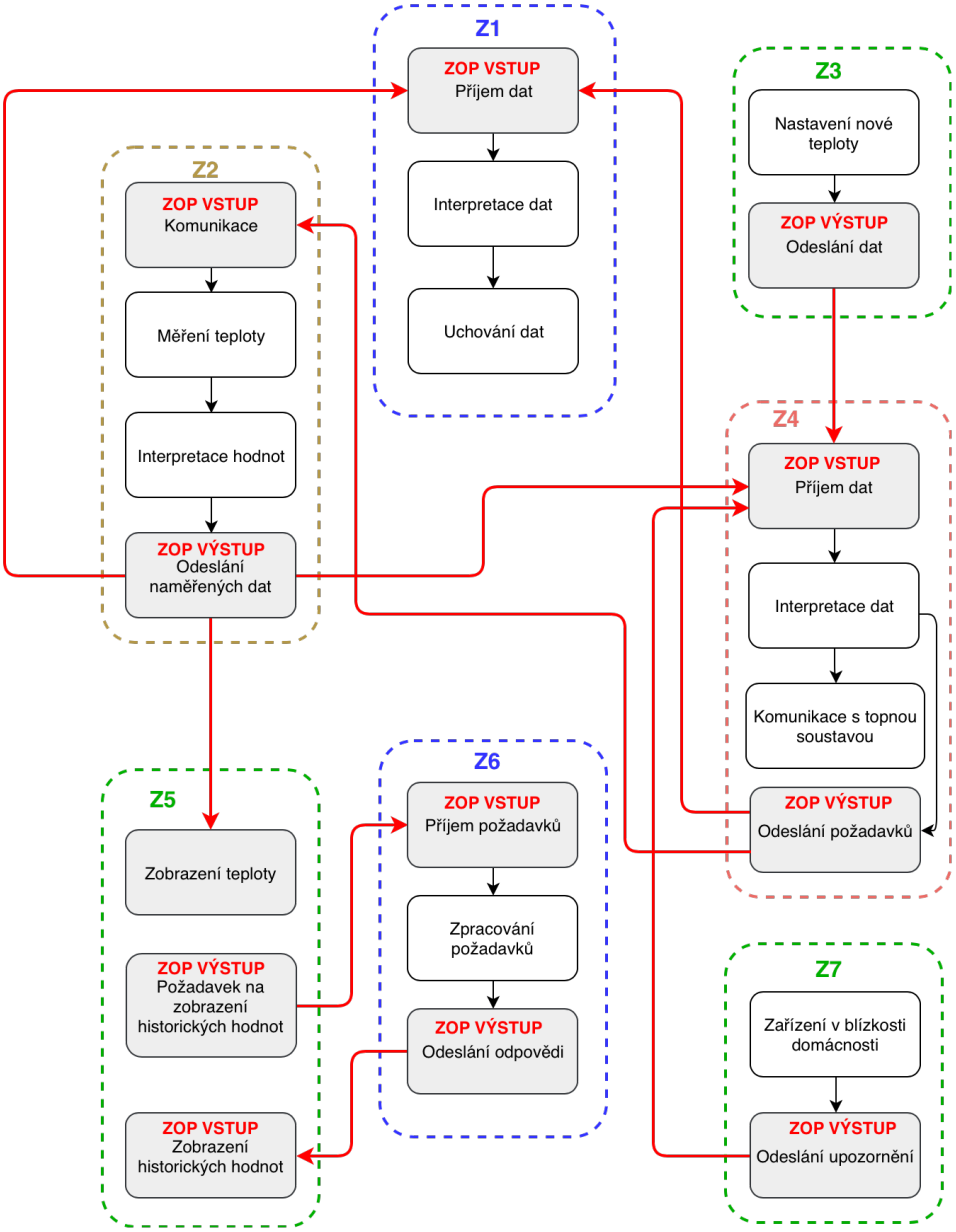
Příklad modelu procesů IoT systému, ze kterého lze technikami popsanými v části 7.2 generovat testovací scénáře, je znázorněný na obrázku 4.2. Jedná se extrakci z modelu na obrázku 4.1. Jsou zde uvedeny jak ZOP, tak hraniční uzly pro jednotlivé zóny. Červenou barvou je označená taková hrana $e$, která má hodnotu $pvp(e)$ větší nebo rovnu, než je stanovená hodnota kritéria $limit$.

## 4.4 Testovací scénáře

Vytváření testovacích scénářů pro testování zařízení v případě omezeného nebo nedostupného síťového připojení se v této práci provádí pomocí nacházení a spojování specifických průchodů orientovaným grafem $G = (V, E, v_s, V_e)$ (popsaného v části 4.3). Testovací scénář $t$ je definován jako [36]:

- Sekvence uzlů $v_1$, $v_2$,..., $v_n$ a hran $e_1$, $e_2$,..., $e_{n-1}$, kde hrana $e_i = (n_i, n_{i+1})$, $e_i \in E$, $v_i \in V$, $v_{i+1} \in V$.
- Začíná v počátečním uzlu $v_s$ ($v_1 = v_s$) a končí v jednom z koncových uzlů ($v_n \in V_e$).

Sada testovacích scénářů $T$ je množinou testovacích scénářů.




**Obrázek 4.2:** Model IoT systému pro testování konektivity

## 4.5 Škálování intenzity testovacích scénářů

Otestovat všechny možné vstupy testovaného systému většinou není možné. Proto se používají určitá kritéria, která umožní systematicky rozdělit množinu vstupů tak, aby byl počet chyb odhalitelných v každém testovacím scénáři co největší. Tato kritéria jsou nazývána kritéria pokrytí a dále se dělí do mnoha skupin. Dají se však shrnout do pouhých čtyř kategorií, vycházejících z matematických struktur, tedy: kritéria pro rozdělení dle definičního oboru vstupů, pro grafy, pro logické výrazy a pro gramatiky [5].

Model testovaného zařízení v případě omezeného nebo nedostupného síťového připojení je zachycen orientovaným grafem. Pro škálování intenzity testovacích scénářů se tedy použijí kritéria pokrytí definovaná pro grafy a těch je mnoho druhů. Příklady, seřazené dle intenzity testovacích scénářů jsou [5]:

- Uzlové pokrytí[2], vynucující v testovacích obsažení všech uzlů v grafu.
- Hranové pokrytí[3], vynucující v testovacích scénářích obsažení všech hran v grafu.
- Pokrytí hranových dvojic[4], které v testovacích scénářích vynucuje pokrytí všech cest s délkou (počtem hran) rovnou dvěma v grafu.
- Pokrytí hlavních cest[5], které v testovacích scénářích vynucuje pokrytí všech hlavních cest (viz 2.1) v grafu.

Kritérium pokrytí může být také specifikováno hloubkou pokrytí[6], jako je tomu u generování testovacích scénářů technikou PCT (viz 7.2.3). Aby sada testovacích scénářů naplňovala toto kritérium, musí v ní být alespoň jednou obsaženy všechny cesty z každého uzlu v grafu s délkou odpovídající právě hodnotě hloubky pokrytí [33].

Pro grafy, které obsahují hrany ohodnocené nějakými tranzitivními prioritami, lze použít kritérium úrovně priority[7]. Zvolením hodnoty tohoto kritéria udáváme, že všechny hrany s touto a vyšší prioritou musí být v sadě testovacích scénářů alespoň jednou zastoupeny [36].

Pro škálování intenzity testovacích scénářů, vygenerovaných pro testování zařízení v případě omezeného nebo nedostupného síťového připojení, bylo školitelem definováno kritérium „úroveň pokrytí hraničních uzlů“ (*UPH*). To v sadě testovacích scénářů vynucuje pokrytí vstupních a výstupních uzlů ze všech ZOP identifikovaných ve vstupním grafu. V jednom případě, nazvaném „VŠECHNY_KOMBINACE“, je nutné mýt zastoupené v sadě testovacích scénářů všechny kombinace uzlů VSTUP a VÝSTUP ve všech ZOP v grafu. Druhá možná hodnota kritéria, nazvaná „KAŽDÝ_HRANIČNÍ-_UZEL_JEDNOU“, je mírnější a vyžaduje v sadě testovacích scénářů zastoupení každého z uzlů VSTUP a VÝSTUP všech zón omezeného připojení v grafu.

## 4.6 Kritéria optimality testovacích scénářů

Cílem výběru vhodného kritéria pokrytí je snaha o zefektivnění testování. Nejde tedy jen o to, odhalit v testovaném systému co nejvíce chyb, ale také o to, aby se testované případy a tím i odhalené chyby zbytečně neopakovaly.

[2] Z anglického: „node coverage“.
[3] Z anglického: „edge coverage“.
[4] Z anglického: „edge-pair coverage“.
[5] Z anglického: „prime path coverage“.
[6] Z anglického: „test depth level“ (TDL).
[7] Z anglického: „priority level“ (PL).

Li a kol. proto zavádějí „problém minimální ceny testovacích průchodů[8].“ (MCTP). Řešením tohoto problému je nalezení minimálního počtu testovacích průchodů grafem, za zachování pokrytí všech testovacích požadavků.

Pro redukci počtu testovacích průchodů nachází Li a kol. možnosti [32]:

- Zkrácení jednotlivých testovacích scénářů (obsažení menšího počtu uzlů).
- Vytvoření menšího počtu testovacích scénářů.
- Obsažení menšího počtu průchodů grafem v testovacích scénářích.
- Snížení počtu testovacích požadavků, pokrytých v jednotlivých testovacích scénářích.

Pro porovnání, která z technik pro generování testovacích scénářů pro testování zařízení v případě omezeného nebo nedostupného síťového připojení vykazuje lepší výsledky, je možné z výše uvedených možností vytvořit základní metriky.

M. Bureš a kol. identifikují v práci o prioritizovaných procesních testech další kritéria [34]. Pro účely této práce používáme kritérium porovnávající počet unikátních hran ve vygenerované sadě testovacích scénářů vůči počtu všech hran v grafu.

Dwarakanath a kol. navíc při porovnání optimality testovacích scénářů pracují s časem generování dané sady [35].

Všechny výše uvedené metriky jsou shrnuty v tabulce 4.1.

| **Kritérium optimality** | **Popis** |
|---|---|
| \|T\| | Počet testovacích scénářů. |
| edges(T) | Počet hran v testovacích scénářích; hrany nemusí být unikátní. |
| uedges(T) | Počet unikátních hran v testovacích scénářích. |
| nodes(T) | Počet uzlů v testovacích scénářích; uzly nemusí být unikátní. |
| unodes(T) | Počet unikátních uzlů v testovacích scénářích. |
| bnodes(T) | Počet hraničních uzlů v testovacích scénářích; uzly nemusí být unikátní. |
| ubns(T) | Počet unikátních hraničních uzlů v testovacích scénářích. |
| $\mathrm{er(T)} = \frac{uedges(T)}{\lvert E\rvert} \times 100$ | Poměr unikátních hran v testovacích scénářích vůči celkovému počtu hran. |
| $\mathrm{bnr(T)} = \frac{hranice(T)}{\lvert N\rvert}$ | Podíl hraničních uzlů v testovacích scénářích vůči celkovému počtu uzlů. |

**Tabulka 4.1:** Kritéria optimality testovacích scénářů

[8] Z anglického: „minimum cost test paths problem“ (MCTP)

Pro vybrání nejlepší sady testovacích scénářů pro daný model testovaného systému je použit postup popsaný M. Burešem a B. Ahmedem [36]. Spočívá ve třech krocích:

1. Nejprve se vybere množina algoritmů a jejich vstupních parametrů pro vygenerování testovacích scénářů T pro každý model testovaného systému G, určí se kritérium pokrytí a určí se kritéria optimality testovacích scénářů.
2. Následně se algoritmy spustí a jsou vygenerovány sady testovacích scénářů.
3. Nakonec se výsledné sady vygenerovaných testovacích scénářů analyzují.

### Funkce optimality

Za optimální je označena ta sada testovacích scénářů, která má nejvyšší hodnotu funkce optimality $o(T_x)$. Tato funkce byla inspirována funkcí popsanou M. Burešem a B. Ahmedem [36], a byla školitelem definována jako:

$$o(T_x) = w_{|T|}(1 - |T_x|/\frac{\sum_{T=T_1}^{T_m} |T|}{m}) + w_{edges(T)}(1 - edges(T_x)/\frac{\sum_{T=T_1}^{T_m} edges(T)}{m}) + w_{uedges(T)}(1 - uedges(T_x)/\frac{\sum_{T=T_1}^{T_m} uedges(T)}{m})$$

Konstanty $w_{|T|}$, $w_{edges(T)}$ and $w_{uedges(T)}$ představují váhy odpovídajících kritérií optimality testovacích scénářů, přičemž platí:

$$w_{|T|} \in \langle 0; 1\rangle, \quad w_{edges(T)} \in \langle 0; 1\rangle, \quad w_{uedges(T)} \in \langle 0; 1\rangle$$

A zároveň musí platit:

$$w_{|T|} + w_{edges(T)} + w_{uedges(T)} = 1$$

U každé sady testovacích scénářů je spočítán procentuální rozdíl daného kritéria $(w_{|T|}, w_{edges(T)}, w_{uedges(T)})$ s průměrnou hodnotou pro všechny techniky generování testovacích scénářů. Pokud je hodnota pro $T_x$ nižší než průměr, výsledek je kladný. Pokud je naopak vyšší, výsledek je záporný.

# Kapitola 5

# Návrh frameworku na vyhodnocování účinnosti testovacích scénářů

V této kapitole jsou popsány požadavky na framework, technologie a knihovny, které framework používá a jeho architektura. Na závěr je uveden návrh testu účinnosti scénářů a popis pravidel pro ověření konzistence scénářů.

## 5.1 Požadavky na framework

Ve spolupráci se školitelem byly specifikovány na framework následující požadavky:

1. Spuštění frameworku pomocí příkazové řádky.
2. Využití datových struktur a knihoven aplikace Oxygen.
3. Začlenění frameworku jako jedné z funkcí aplikace Oxygen.
4. Načtení parametrů příkazové řádky, kterými lze specifikovat cestu ke XML souborem se vstupním modelem, cestu k textovému konfiguračnímu souboru a volbu ze dvou funkcionalit frameworku: „--meassure“, či „--optimality“.
5. Načtení modelu testovaného systému z XML souboru ve formátu, odpovídajícímu aplikaci Oxygen.
6. Načtení textového konfiguračního souboru, obsahujícího názvy algoritmů vybraných pro generování testovacích scénářů a hodnoty jejich vstupních parametrů.
7. Spuštění algoritmů pro generování testovacích scénářů zvolených v konfiguračním souboru nad načteným projektem s grafem, či více grafy, modelujícími testovaný systém.
8. Porovnání optimality výsledných scénářů, při spuštění frameworku s parametrem: --optimality.

9. V případě, že byla při spuštění frameworku vybrána možnost pro porovnání optimality výsledných scénářů, je nutné načíst z konfiguračního souboru hodnoty konstant, jež představují váhu jednotlivých kritérií optimality testovacích scénářů. Tyto konstanty jsou popsány v části 4.6.
10. Vyhodnocení a zobrazení vlastností grafu, či více grafů, obsažených v načteném projektu modelujícím testovaný systém.
11. Vyhodnocení a zobrazení vlastností vygenerovaných testovacích scénářů. V případě volby pro porovnání optimality testovacích scénářů (parametr: --optimality) navíc také zobrazení kritérií specifikovaných v části 4.6.
12. V případě, že jsou v některém z načtených grafů zaznamenány chybové dvojice ZOP VSTUP a ZOP VÝSTUP uzlu pro test účinnosti testovacích scénářů, popsaný v části 5.4. Navíc také zobrazení odhalených chybových dvojic v sadách testovacích scénářů vygenerovaných zvolenými technikami.
13. Export naměřených výsledků, popsaných v bodech 10, 11 a 12 do souboru ve formátu CSV.
14. Kontrola konzistence vygenerované sady testovacích scénářů dle pravidel specifikovaných v části 4.4.
15. Generování logů, exportovaných do zvláštního souboru, obsahujících informace o aktuálním běhu frameworku a případně chybách, pokud nastaly.
16. Vytvoření souboru s modelem testovaného systému ve formátu aplikace Oxygen s testovacími scénáři, které byly vygenerovány během aktuální exekuce frameworku.
17. Doba exekuce frameworku v řádu maximálně jednotek vteřin.
18. Vygenerování chybového souboru v případě, že kroky uvedené výše neskončí úspěšně.

## 5.2 Použité technologie a knihovny

Pro splnění funkčních požadavků bylo nutné framework přizpůsobit tak, aby dokázal pracovat v souladu s aplikací Oxygen. Na funkcionalitách (například algoritmech pro generování testovacích scénářů) a datových strukturách této aplikace je framework, i vzhledem k požadavkům na něj, zčásti závislý.

### 5.2.1 Oxygen

Oxygen, dříve nazývaný PCTgen, je zdarma dostupná desktopová[1] Java aplikace vyvíjená v rámci laboratoře inteligentního testování softwaru na

[1] Aplikace s grafickým uživatelským rozhraním spustitelná v operačním systému osobního počítače.

katedře počítačů Fakulty elektrotechnické Českého vysokého učení technického v Praze. Hlavním účelem aplikace je významně usnadnit uživatelům tvorbu testovacích scénářů a to díky jejich automatickému generování z vytvořeného modelu testovaného systému. Oxygen se používá také pro akademické účely při navrhování a ověřování nových testovacích technik [37]. Jádro aplikace Oxygen využívá knihovny JGraphX pro vizualizaci i reprezentaci grafů, Log4j pro logování, JUnit pro jednotkové testy a Commons CSV pro práci se soubory ve formátu CSV.

V současné době je v Oxygenu možné vytvářet modely testovaných systémů prostřednictvím orientovaných a neorientovaných grafů a diagramů aktivit. Z těchto grafů lze vygenerovat testovací scénáře technikami Process Cycle Test [33], Prioritized Process Test [34], Database CRUD Test [38] a nově také Limited Connectivity Process Test pro testování konektivity IoT zařízení.

Ve frameworku je z aplikace Oxygen využíváno především balíčku situations_generators, který obsahuje další balíčky a třídy pro generování testovacích scénářů výše uvedenými technikami. Důležitým pomocníkem pro framework je třída MetricsUtils v balíčku situations_generators.paper_algorithms.measuring, která obsahuje metody pro výpočet metrik pro zjištění kritérií optimality testovacích scénářů (více v části 4.6).

### 5.2.2 Java

Vzhledem k tomu, že aplikace Oxygen je napsána v programovacím jazyce Java, byl pro implementaci frameworku taktéž zvolen tento jazyk (konkrétně Java SE 8). Java je vyšší objektově orientovaný programovací jazyk, který byl k březnu 2019 nejpoužívanějším programovacím jazykem na světě [39].

Běh frameworku využívá funkcí platformy Java, která se skládá z Java Virtual Machine (JVM) a z Java Application Programming Interface (API). Výhodou výběru této platformy pro vývoj frameworku je především její portabilita. Programy napsané v programovacím jazyce Java se totiž nekompilují do strojového kódu konkrétního počítače, jako je tomu například u jazyka C, ale do byte kódu, který je interpretován v JVM. Díky tomu je možné program spustit na kterémkoli počítači, jehož operační systém umožňuje instalaci některé implementace JVM. Zdarma a v aktivním vývoji je aktuálně 21 implementací JVM, přičemž většina z nich je odvozena od Oracle implementace jménem HotSpot [40]. Ta je pro verzi Java SE 8 dostupná na operačních systémech Windows od verze Vista SP2, Mac OS X 10.8.3 a vyšší, Solaris od verze 10 a na různých distribucích operačního systému Linux [41].

### 5.2.3 Framework

Součástí aplikace Oxygen byla prvotní verze frameworku ještě před vznikem této diplomové práce. Umožňovala již porovnávat techniky generování testovacích scénářů, avšak pro jinou oblast testování, než IoT. Některé požadavky na framework však byly shodné a tak se části kódu, které je řešily, daly znovu použít. Konkrétně se jednalo o požadavky: 2, 4, 5, 6, 7, 10, 11, 13, 15 a 16, definované v části 5.1.

## 5.3 Architektura frameworku

Pro popis architektury frameworku bylo v této diplomové práci použito způsobu 4+1 pohledů, navrženého Philippe Kruchtenem [42]. Pohledy různých účastníků tvorby softwarového produktu na jeho architekturu lze podle Kruchtena shrnout na logický, procesní, implementační, fyzický a pohled případů užití, který je společný pro všechny (onen plus první pohled).

Logický pohled obsahuje náhled na strukturu systému z hlediska výsledné funkčnosti pro koncové uživatele systému a je určen analytikům. Využívá se v něm především UML diagramů tříd. Procesní pohled zohledňuje nefunkční požadavky na chování systému a je určen systémovým integrátorům. Implementační pohled popisuje softwarové komponenty, ze kterých se systém skládá a je určen pro vývojáře. Pohled fyzický pak obsahuje navázání systému na topologii hardwarových a dalších softwarových komponent a často se v něm využívá UML diagramu nasazení. Poslední pohled je pohled případů užití, který shrnuje předchozí čtyři pohledy a zobrazuje malé množství důležitých scénářů, instancí nejdůležitějších požadavků [42].

### 5.3.1 Pohled případů užití

Diagram na obrázku 5.1 zobrazuje pohled případů užití na architekturu frameworku. Jsou v něm obsaženy nejdůležitější scénáře, které ve frameworku probíhají. Scénáře jsou abstrakcí nejdůležitějších požadavků, na které se pod zkratkou PF (požadavek na framework) odkazuje.

### 5.3.2 Logický pohled

Diagram na obrázku 5.2 zobrazuje logický pohled na architekturu frameworku. Je na něm zobrazeno 9 skupin tříd a jejich navržené asociace pro zajištění požadovaných funkcionalit frameworku.

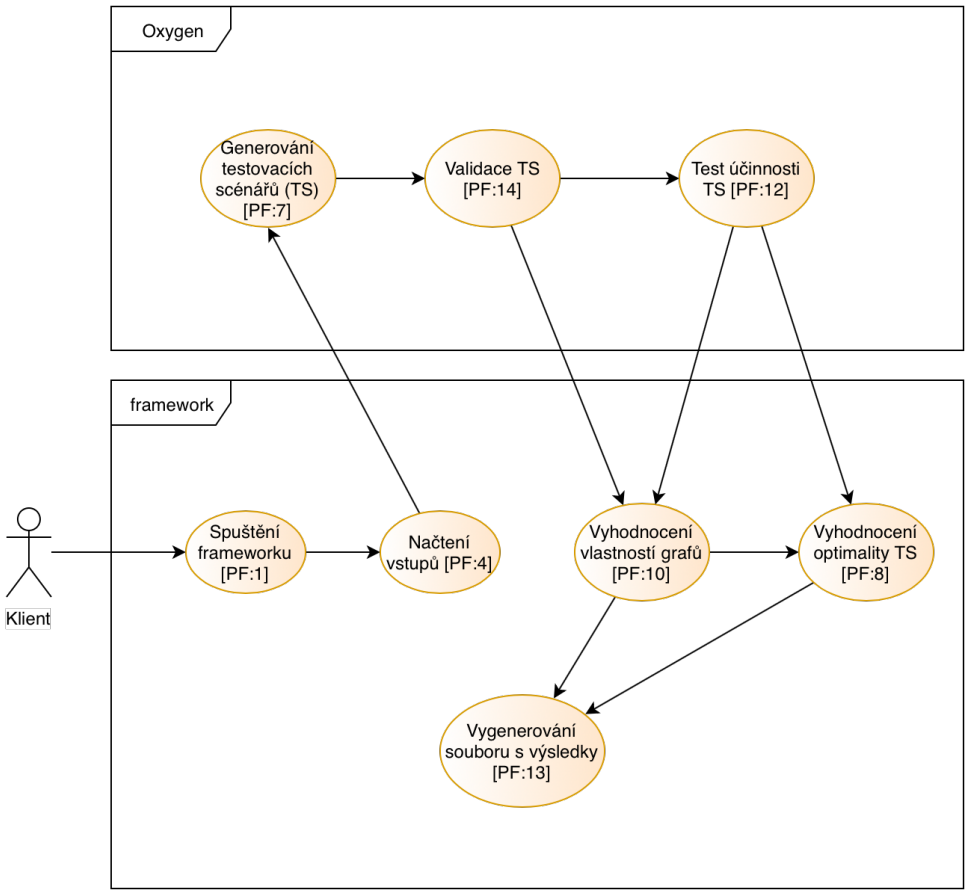
Jednotlivé skupiny tříd jsou:

- **Strategie spouštění frameworku**
  Framework lze spustit ve dvou možných funkcionalitách, jak je specifikováno v požadavku: 4. To řídí třídy v této skupině.

- **Práce se soubory**
  Framework umožňuje načítání několika souborů (požadovaných například v požadavcích: 5 a 6) a sám některé také generuje (což je požadováno například v požadavku: 13). To umožňují třídy v této skupině.

- **Generování testovacích scénářů**
  Obsahuje třídy spojené s funkcí frameworku pro generování testovacích scénářů zvolenými technikami nad načtenými grafy.

- **Vyhodnocení optimality**
  Obsahuje třídy spojené s funkcí frameworku pro vyhodnocení optimality




**Obrázek 5.1:** Pohled případů užití na architekturu frameworku

vygenerovaných testovacích scénářů zvolenými technikami nad načtenými grafy.

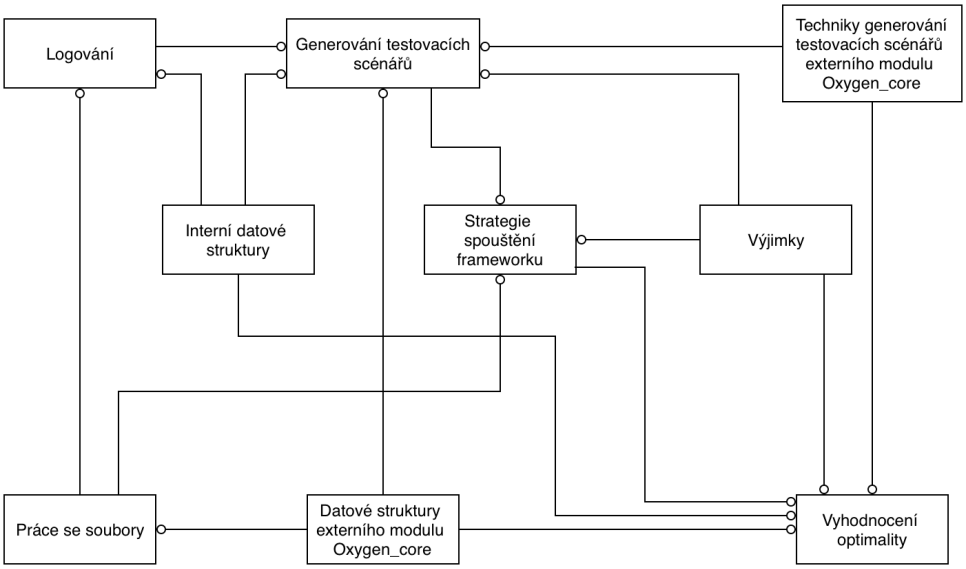
- **Techniky generování testovacích scénářů externího modulu Oxygen_core**
  Skupina tříd v externím modulu Oxygen_core, které obsahují implementace jednotlivých technik pro generování testovacích scénářů, které framework při této činnosti využívá.

- **Interní datové struktury**
  Obsahuje interní datové struktury, vytvářené frameworkem. Používají se například pro reprezentaci tabulek v generovaných souborech.

- **Datové struktury externího modulu Oxygen_core**
  Vzhledem k požadavku: 2 je nutné používat datové struktury z aplikace Oxygen, nacházející se v této skupině tříd.

- **Logování**
  Tato skupina obsahuje všechny třídy, které umožňují logování výstupů aplikace (jak je uvedeno v požadavku: 15).

- **Výjimky**
  Součástí požadavků na framework je také logování případných chyb (požadavek: 15). To usnadňují třídy v této kategorii.




**Obrázek 5.2:** Logický pohled na architekturu frameworku

### 5.3.3 Procesní pohled

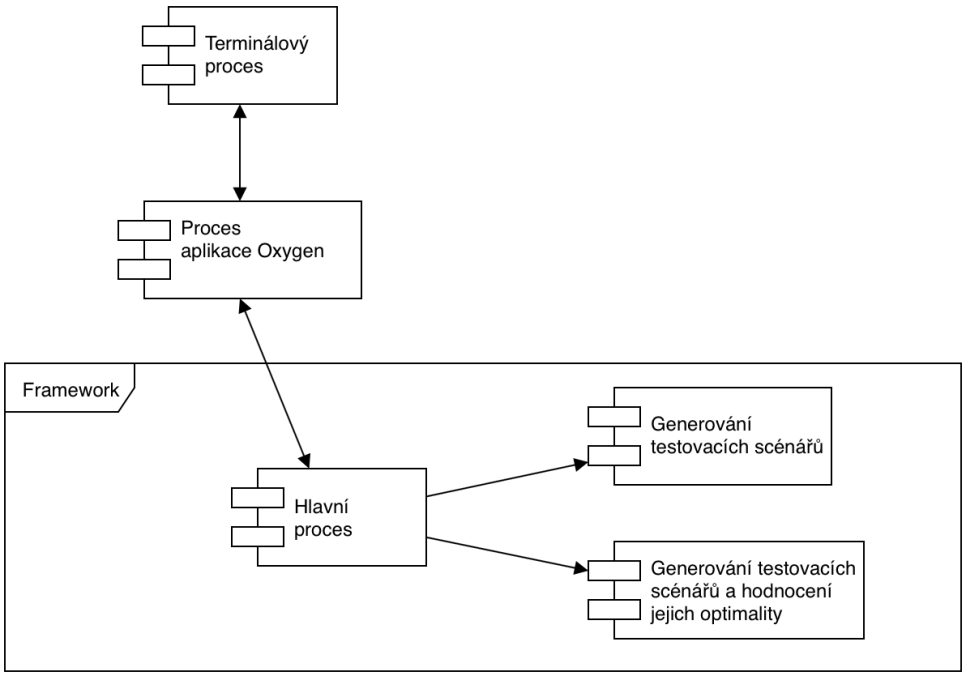
Diagram na obrázku 5.3 zobrazuje procesní pohled na architekturu frameworku. Jak je popsáno v požadavku 1, framework je nutné spustit z příkazové řádky, proto je na diagramu jako první uveden tento **terminálový proces**. Z terminálového procesu se spustí **proces aplikace Oxygen**, který následně dle načtených parametrů vyhodnotí, že má spustit **hlavní proces frameworku**. Ten, dle načtených parametrů příkazové řádky, spustí buď pouze proces **generování testovacích scénářů**, nebo navíc ještě **hodnocení optimality testovacích scénářů**.

### 5.3.4 Implementační pohled

Diagram na obrázku 5.4 zobrazuje implementační pohled na architekturu frameworku.

Popis jednotlivých balíčků v programu je následující:

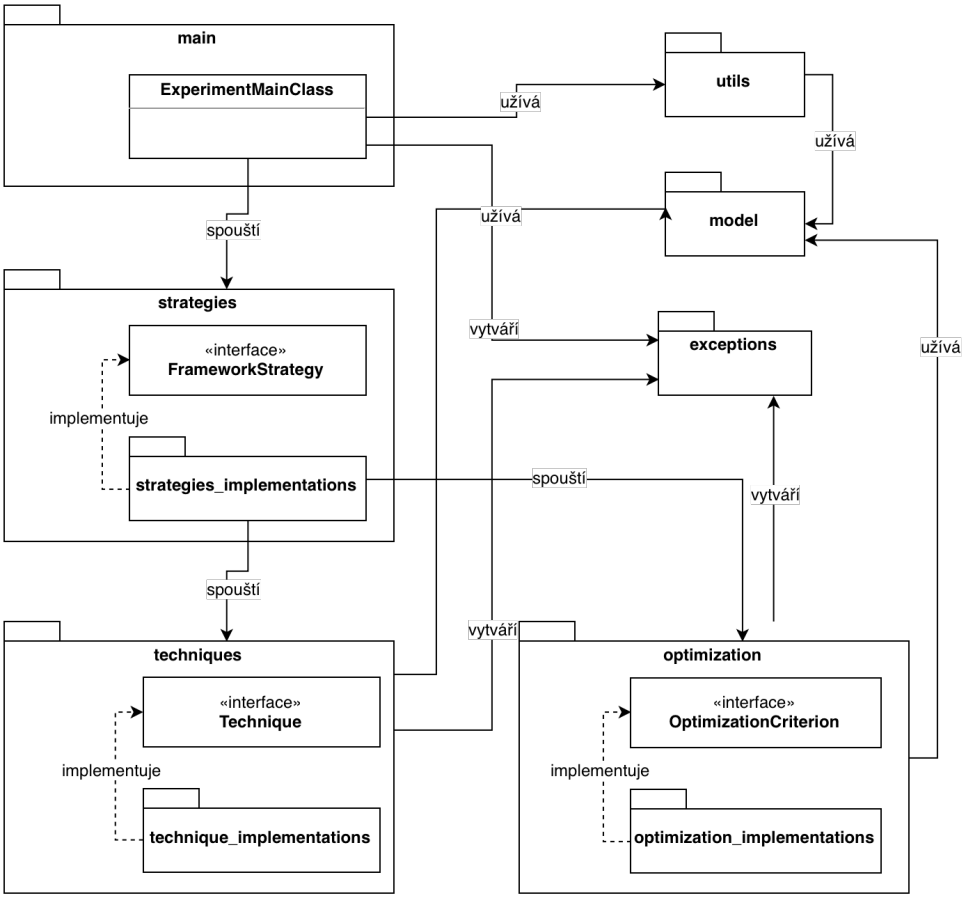
- **main**
  Obsahuje třídu ExperimentMainClass s metodou experimentMain, která umožňuje spuštění frameworku. Metoda je volána z metody main programu Oxygen, pokud je spuštěn s nějakými argumenty. To implikuje požadavek o spuštění frameworku a ne samotné aplikace.




**Obrázek 5.3:** Procesní pohled na architekturu frameworku

- **strategies**
  Obsahuje třídy implementující jednotlivé experimenty, které mohou být zvoleny parametry příkazové řádky při spuštění frameworku.
  - **HelpStrategy** vypíše nápovědu jak framework spustit a co znamenají jednotlivé přepínače.
  - **MetricsMeasuringStrategy** vybranými technikami vygeneruje testovací scénáře.
  - **TechniqueOptimalityStrategy** vyhodnotí optimalitu vygenerovaných testovacích scénářů pomocí specifických optimalizačních kritérií.
- **model**
  Vytváří struktury, které framework využívá pro reprezentaci dat.
- **techniques**
  Obsahuje třídy, které řídí generování testovacích scénářů zvolenou technikou.
- **optimization**
  Obsahuje třídy implementující měření optimalizačních kritérií nad testovacími scénáři vygenerovanými zvolenými technikami.
- **utils**
  Obsahuje funkcionality sdílené napříč frameworkem, například pro práci se soubory.

- **exceptions**
  Tento balíček obsahuje třídy s výjimkami, které umožňují uživateli identifikovat chybu při nesprávném běhu frameworku.



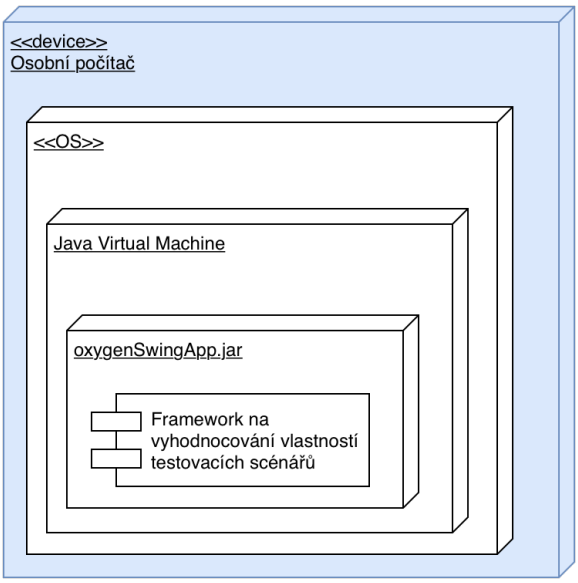

**Obrázek 5.4:** Implementační pohled na architekturu frameworku

### 5.3.5 Fyzický pohled

Diagram nasazení na obrázku 5.5 zobrazuje fyzický pohled na architekturu frameworku. Pro spuštění frameworku je nutné mít osobní počítač s instalovaným operačním systémem, který podporuje instalaci Java Virtual Machine ve verzi podporující programovací jazyk Java SE 8. Pokud byly tyto požadavky splněny, lze spustit soubor: „oxygenSwingApp.jar“, který obsahuje aplikaci Oxygen s modulem frameworku.

## 5.4 Návrh testu účinnosti scénářů

Pro test účinnosti scénářů, vygenerovaných za účelem testování konektivity IoT zařízení, byl školitelem navržen následující postup:




**Obrázek 5.5:** Fyzický pohled na architekturu frameworku

- Pro každý graf $G$ je vytvořena množina $M$ chybových dvojic ZOP VSTUP a ZOP VÝSTUP uzlů, splňující:
  - Každá chybová dvojice $(v_1, v_2) \in M$ musí být součástí jedné zóny $Z$.
  - Pro každou chybovou dvojici $(v_1, v_2)$ musí v $G$ existovat orientovaný sled.
- V sadě testovacích scénářů $T$, vygenerované zvoleným algoritmem z $G$, je naměřeno, kolik obsahuje chybových dvojic z $M$.
- Čím větší počet chybových dvojic $(v_1, v_2) \in M$ algoritmus obsáhl ve vygenerovaných scénářích $T$, tím je pro testování zařízení účinnější.

## 5.5 Pravidla pro ověření konzistence scénářů

Součástí frameworku je také funkcionalita na kontrolu konzistence vygenerované sady testovacích scénářů $T$. Pravidla, které $T$ musí splňovat, vychází zčásti z literatury [5, 32], zčásti byla dodaná školitelem.

Jsou to:

- Každý testovací scénář začíná v počátečním uzlu $v_s$ $(v_1 = v_s)$
- Každý testovací scénář končí v jednom z koncových uzlů $(v_n \in V_e)$.
- Sada testovacích scénářů je neprázdná $(|T| \neq 0)$
- Každý testovací scénář představuje souvislý orientovaný sled v grafu. Tedy mezi každými dvěma vrcholy $v_i$, $v_{i+1}$ ve scénáři existuje orientovaná hrana vedoucí z vrcholu $v_i$ do vrcholu $v_{i+1}$.

- Každý ZOP VSTUP uzel musí být alespoň jednou zastoupen v některém ze scénářů z vygenerované sady.
- Každý ZOP VÝSTUP uzel musí být alespoň jednou zastoupen v některém ze scénářů z vygenerované sady.
- Vygenerovaná sada testovacích scénářů neobsahuje duplicitní scénáře.

# Kapitola 6
# Implementace

V této kapitole je popsána implementace frameworku a funkcionalit, které byly součástí zadání této diplomové práce. Jsou zde uvedeny základní moduly frameworku, implementace testu účinnosti scénářů a algoritmu na ověření konzistence scénářů. Následuje popis, jak jsou zobrazeny výsledky testů vygenerované frameworkem. Na konci kapitoly je uvedena uživatelská příručka, tedy návod jak framework používat.

## 6.1 Základní moduly frameworku

Framework se skládá z několika základních modulů, které poskytují funkcionality specifikované v požadavcích v části 5.1. Rozdělení modulů vyplývá z architektury frameworku, popsané v části 5.3.

### 6.1.1 Načtení vstupních dat

Modul spuštěný na začátku exekuce frameworku, implementovaný ve třídě ExperimentMainClass, slouží ke kontrole, zda byly zadány všechny požadované vstupní argumenty. Dle zadaných argumentů následně spustí vykonání odpovídající části frameworku (pro pouhé vygenerování testovacích scénářů možností „--meassure“ či navíc i pro jejich porovnání argumentem „--optimality“).

### 6.1.2 Řízení běhu frameworku

Procedury vykonávané frameworkem jsou volány skrz jednotlivé třídy, které dle zvolené testovací strategie implementují rozhraní FrameworkStrategy. Metody, které třída musí implementovat řídí vykonání všech důležitých součástí frameworku, tedy načtení vstupních souborů, vykonání experimentu, vypsání výsledku, uložení výsledku, vypsání a uložení případných chyb a kontrola formátu a konzistentnosti načteného grafu.

### 6.1.3 Reprezentace projektu systému Oxygen

Pro reprezentaci grafů testovaného systému se používá třída FrameworkBasicModel. Ta data načtená z projektu aplikace Oxygen uloží do stejných

datových struktur, což umožní používat již implementované funkcionality aplikace Oxygen, jako jsou například algoritmy pro generování testovacích scénářů.

### 6.1.4 Generování testovacích scénářů

Pro generování testovacích scénářů framework využívá tříd, které dle zvolené testovací techniky implementují rozhraní Technique. Důsledkem je vytvoření specifického generátoru, implementujícího SituationsGeneratorInterface z balíčku Oxygen_core v aplikaci Oxygen. Tento generátor má vlastnosti, splňující požadavky uvedené v konfiguračním souboru uvedeném při spuštění z příkazové řádky (zvolenou testovací techniku, kritéria pro škálování intenzity testovacích scénářů a další).

### 6.1.5 Výpočet metrik pro vyhodnocení kvality testovacích scénářů

Pro počítání metrik nad grafem a testovacími scénáři framework využívá třídu MetricsUtils, začleněnou do jádra aplikace Oxygen. Příkladem metrik nad grafem je zjištění počtu hran nebo uzlů v grafu, u testovacích scénářů je to například počet unikátních hran či uzlů.

### 6.1.6 Ověření konzistence a správnosti vygenerovaných testovacích scénářů

Tento modul byl zařazen přímo do aplikace Oxygen a je tak ve frameworku používán z ní. Implementace je ve třídě TestSituationsValidator, která se nachází v balíčku validators.

### 6.1.7 Generování výstupních dat

Generování dat do výstupního CSV souboru se provádí pomocí metod ve třídě CSVTableUtils, které volá metoda *createModelCSVTable()* ve třídě AbstractMetricsMeasuringStrategy. Pro reprezentaci dat se využívá struktur CSVTable a CSVElement.

## 6.2 Implementace testu účinnosti scénářů

Test účinnosti scénářů je umožněn díky možnosti vytvoření chybových dvojic, složených z ZOP VSTUP a ZOP VÝSTUP uzlů v jednotlivých zónách v grafu. Ve vygenerovaných testovacích scénářích $T$ je následně změřeno, kolik chybových dvojic zvolený algoritmus odhalil a to pod parametrem nazvaným $lcz\_e\_p(T)$. Implementace algoritmu byla přidána do jádra aplikace Oxygen a to konkrétně do třídy TestSituations.

Činnost algoritmu je znázorněna v pseudokódu 1. Symboly v něm použité vychází z popisu v části 4.3. Fungování algoritmu začíná vytvořením kopie množiny chybových dvojic $M$. Následně se prochází testovací scénáře $T$ a pro

každý z nich se zkontroluje, zda jsou v něm obsaženy, v pořadí: ZOP VSTUP, ZOP VÝSTUP, chybové uzly, uvedené v množině $M$. Pokud ano, odmažou se tyto uzly z množiny $M'$. Výsledkem algoritmu je číslo, odpovídající rozdílu mohutnosti původní množiny chybových dvojic v grafu a její kopie, ze které byly odmazány nalezené chybové dvojice.

**Algorithm 1:** Algoritmus pro nalezení počtu chybových dvojic ve vygenerovaných testovacích scénářích

**Vstup :** Model testovaného systému $G$, sada testovacích scénářů $T$ a množina chybových dvojic $M$
**Výstup :** Počet nalezených chybových dvojic $r$

```
1  M' ← M
2  for t ∈ T do
3  |  Vytvoř prázdnou frontu N
4  |  for n ∈ t do
5  |  |  Přidej n na konec N
6  |  end
7  |  for m ∈ M' do
8  |  |  v_i := ZOP VSTUP uzel v m
9  |  |  v_o := ZOP VÝSTUP uzel v m
10 |  |  if v_i ∈ N ∧ v_o ∈ N then
11 |  |  |  idx_i := index uzlu v_i v M
12 |  |  |  idx_o := index uzlu v_o v M
13 |  |  |  if idx_i < idx_o then
14 |  |  |  |  Odstraň m z M'
15 |  |  |  end
16 |  |  end
17 |  end
18 end
19 r := |M| − |M'|
20 return r
```

## 6.3 Implementace algoritmu na ověření konzistence scénářů

Algoritmus na ověření konzistence scénářů obsahuje několik kontrolních prvků, navržených v části 5.5. Jejich implementace je popsána pomocí pseudokódu 2. Symboly, které jsou v pseudokódu použity, vychází z popisu v části 4.3. Algoritmus nejprve ověří, že vygenerovaná množina testovacích scénářů $T$ je neprázdná. Následně se prochází jednotlivé vygenerované testovací scénáře. Pro každý se zjistí, zda začíná v uzlu $v_s$ a končí v uzlu z množiny $V_e$. Poté se prochází daný testovací scénář a kontroluje se, zda je souvislý. Pro alespoň základní ověření kvality vygenerované sady testovacích scénářů $T$ je následně přidána podmínka, zda $T$ obsahuje všechny hraniční uzly zón ZOP v grafu.

Nakonec se zkontroluje seřazením množiny $T'$, kopie množiny $T$, a porovnáním vždy dvou po sobě jdoucích scénářů, zda v ní nejsou obsaženy duplicity. Výstup algoritmu je, při splnění všech kontrolních bodů, prázdný. V opačném případě je výstupem algoritmu výjimka s chybovou hláškou, vysvětlující proč sada testovacích scénářů neprošla ověřením.

**Algorithm 2:** Algoritmus pro ověření konzistence vygenerovaných testovacích scénářů

**Vstup :** Model testovaného systému $G$ a sada testovacích scénářů $T$
**Výstup :** Nic, pokud algoritmus nezjistil chybu v konzistenci scénářů. V opačném případě je výstupem výjimka s chybovou hláškou.

```
if |T| = 0 then
    Vyhození výjimky, že T je prázdný
end
B ← hraniceZOP(G)
for t ∈ T do
    v_1 := první uzel v t
    if v_1 ≠ v_s then
        Vyhození výjimky, že t nezačíná v uzlu v_s
    end
    v_n := poslední uzel v t
    if v_n ∉ V_e then
        Vyhození výjimky, že t nekončí v uzlu z množiny V_e
    end
    for n ∈ t do
        if n ∈ B then
            Odstraň uzel n z B
        end
        n' := následující uzel v T po n
        if n' není potomkem n then
            Vyhození výjimky, že t není souvislý
        end
    end
end
if B ≠ ∅ then
    Vyhození výjimky, že T neobsahuje některý z hraničních uzlů
     z množiny B
end
T' ← T
Seřazení T' podle počtu uzlů ve scénáři
for t ∈ T' do
    t' := následující testovací scénář v T' po t
    if t = t' then
        Vyhození výjimky, že T obsahuje duplicitní scénáře
    end
end
```

## 6.4 Zobrazení výsledků testů

V následující části je popsán výsledek exekuce frameworku. Pro větší názornost jsou zde přiloženy ukázky jednotlivých částí výstupních CSV souborů, přiložených do přílohy elektronické verze práce a popsaných v části C. Soubory byly otevřeny a lehce zformátovány s pomocí aplikace Microsoft Excel.

### Názvy grafů

V první části vygenerovaného výstupního CSV souboru se nachází tabulka se základními informacemi o vstupních grafech. V prvním sloupci tabulky je uvedené pořadové číslo grafu, v druhém jeho název. Příklad této tabulky je zobrazen na obrázku 6.1.

| | |
|---|---|
| 0 | Graph 1 b [MANTIS] New directed graph 1 |
| 1 | Graph 2 sumEdges [MANTIS] New directed graph 4 |
| 2 | Graph 2 b [MANTIS] New directed graph 4 |
| 3 | Graph 3 sumEdges [MANTIS] New directed graph 5 |
| 4 | Graph 3 b [MANTIS] New directed graph 5 |
| 5 | Graph 4 sumEdges [MANTIS] New directed graph 7 |
| 6 | Graph 4 b [MANTIS] New directed graph 7 |
| 7 | Graph 5 sumEdges [MANTIS] New directed graph 8 |
| 8 | Graph 5 b [MANTIS] New directed graph 8 |
| 9 | Graph 6 sumEdges [simple_grafy5] Graph_42 |
| 10 | Graph 6 b [simple_grafy5] Graph_42 |
| 11 | Graph 7 sumEdges [simple_grafy5] Graph_39 |
| 12 | Graph 7 b [simple_grafy5] Graph_39 |
| 13 | Graph 8 sumEdges [simple_grafy5] Graph_38 |
| 14 | Graph 8 b [simple_grafy5] Graph_38 |
| 15 | Graph 9 sumEdges [simple_grafy5] Graph_40 |
| 16 | Graph 9 b [simple_grafy5] Graph_40 |
| 17 | Graph 9 c [simple_grafy5] Graph_40 |
| 18 | Graph 10 sumEdges [simple_grafy5] Graph10 |
| 19 | Graph 10 b [simple_grafy5] Graph10 |

**Obrázek 6.1:** Ukázka první části výstupního souboru s výsledky testů

### Vlastnosti grafů

V další části vygenerovaného souboru jsou zobrazeny vlastnosti jednotlivých grafů. Tyto informace framework využívá pro změření kritérií optimality testovacích scénářů (jednotlivé parametry jsou popsány v části 4.6) a vyhodnocení testu účinnosti scénářů (parametr |LCZ_err_pairs| udávající počet chybových dvojic v grafu). Příklad této části výstupního souboru je uveden na obrázku 6.2.

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| multi | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| \|nodes\| | 19 | 24 | 24 | 30 | 30 | 25 |
| \|edges\| | 30 | 38 | 38 | 49 | 49 | 40 |
| \|cycles\| | 5 | 7 | 7 | 7 | 7 | 7 |
| \|LCZ_err_pairs\| | 2 | 2 | 3 | 5 | 3 | 3 |

**Obrázek 6.2:** Ukázka tabulky vlastností grafů ve výstupním souboru s výsledky testů

## Výsledky generování testovacích scénářů

Část výsledného dokumentu, která zobrazuje výsledky generování testovacích scénářů zvolenými technikami, je zobrazena na obrázku 6.3. V prvním řádku je uvedena zkratka názvu techniky a zvolené vstupní parametry. Následují metriky, jejichž význam je popsán v tabulce 4.1 a v části 6.2.

| LCPT{defaultPriority=LOW, COP_treshold=0.1,criterion=ALL_COMBINATIONS} | | | | | | |
|---|---|---|---|---|---|---|
| success | TRUE | TRUE | TRUE | TRUE | TRUE | TRUE |
| \|T\| | 2 | 2 | 3 | 2 | 3 | 2 |
| edges(T) | 17 | 18 | 23 | 17 | 25 | 15 |
| uedges(T) | 10 | 12 | 13 | 14 | 15 | 10 |
| nodes(T) | 15 | 16 | 20 | 15 | 22 | 13 |
| unodes(T) | 9 | 11 | 12 | 13 | 14 | 9 |
| er(T) | 0.333333 | 0.315789 | 0.342105 | 0.285714 | 0.306122 | 0.250000 |
| bnodes(T) | 5 | 5 | 9 | 8 | 10 | 6 |
| ubns(T) | 3 | 3 | 4 | 5 | 4 | 4 |
| bnr(T) | 0.200000 | 0.187500 | 0.200000 | 0.333333 | 0.181818 | 0.307692 |
| lcz_e_p(T) | 2 | 2 | 3 | 5 | 3 | 3 |
| time[s] | 0.000000 | 0.001118 | 0.003546 | 0.003958 | 0.109543 | 0.016375 |

| EPP{defaultPriority=LOW, COP_treshold=0.1,criterion=ALL_COMBINATIONS} | | | | | | |
|---|---|---|---|---|---|---|
| success | FALSE | TRUE | FALSE | TRUE | TRUE | TRUE |
| \|T\| | -1 | 2 | -1 | 2 | 2 | 2 |
| edges(T) | -1 | 18 | -1 | 20 | 16 | 15 |
| uedges(T) | -1 | 11 | -1 | 17 | 11 | 10 |
| nodes(T) | -1 | 16 | -1 | 18 | 14 | 13 |
| unodes(T) | -1 | 10 | -1 | 16 | 10 | 9 |
| er(T) | -1.000000 | 0.289474 | -1.000000 | 0.346939 | 0.224490 | 0.250000 |
| bnodes(T) | -1 | 5 | -1 | 8 | 7 | 6 |
| ubns(T) | -1 | 3 | -1 | 5 | 4 | 4 |
| bnr(T) | -1.000000 | 0.187500 | -1.000000 | 0.277778 | 0.285714 | 0.307692 |
| lcz_e_p(T) | 0 | 2 | 0 | 5 | 3 | 3 |
| time[s] | -0.000000 | 0.011288 | -0.000000 | 0.004837 | 0.002574 | 0.001949 |

**Obrázek 6.3:** Ukázka výsledků metrik pro dvě vybrané techniky ve výstupním souboru s výsledky testů

### Porovnání optimality testovacích scénářů

Při spuštění frameworku s funkcí pro porovnání optimality testovacích scénářů je následně zobrazena ještě jedna tabulka. Ta obsahuje na prvním řádku funkci s dosazenými hodnotami konstant, s jejíž pomocí byly vygenerované sady testovacích scénářů hodnoceny. V jednotlivých sloupcích tabulky je následně pro každý testovaný graf uveden název techniky, jejíž sada testovacích scénářů vykázala nejlepší výsledky, případně více sad, pokud měly výsledek stejný. V dalších řádcích jsou následně pro tuto nejlepší sadu uvedeny její naměřené parametry. Příklad této tabulky je na obrázku 6.4.

| Select function: 0.300000 * ( 1 - \|Tx\| ) / AVG(\|T\|) ) + 0.400000 * ( ( 1 - edges(Tx) ) / AVG(edges(T)) ) + 0.300000 * ( ( 1 - uedges(Tx) ) / AVG(uedges(T)) ) better > worse | | | | | | |
|---|---|---|---|---|---|---|
| techniques | {LCPT{defaultPriority=LOW, COP_treshold=0.1,criterion=ALL_COMBINATIONS}} | {EPP{defaultPriority=LOW, COP_treshold=0.1,criterion=ALL_COMBINATIONS}} | {LCPT{defaultPriority=LOW, COP_treshold=0.1,criterion=ALL_COMBINATIONS}} | {LCPT{defaultPriority=LOW, COP_treshold=0.1,criterion=ALL_COMBINATIONS}} | {EPP{defaultPriority=LOW, COP_treshold=0.1,criterion=ALL_COMBINATIONS}} | {EPP{defaultPriority=LOW, COP_treshold=0.1,criterion=ALL_COMBINATIONS},LCPT{defaultPriority=LOW, COP_treshold=0.1,criterion=ALL_COMBINA |
| \|T\| | {2} | {2} | {3} | {2} | {2} | {2,2} |
| edges(T) | {17} | {18} | {23} | {17} | {16} | {15,15} |
| uedges(T) | {10} | {11} | {13} | {14} | {11} | {10,10} |
| nodes(T) | {15} | {16} | {20} | {15} | {14} | {13,13} |
| unodes(T) | {9} | {10} | {12} | {13} | {10} | {9,9} |
| er(T) | {0.333333} | {0.289474} | {0.342105} | {0.285714} | {0.224490} | {0.250000,0.250000} |
| bnodes(T) | {5} | {5} | {9} | {8} | {7} | {6,6} |
| ubns(T) | {3} | {3} | {4} | {5} | {4} | {4,4} |
| bnr(T) | {0.200000} | {0.187500} | {0.200000} | {0.333333} | {0.285714} | {0.307692,0.307692} |
| lcz_e_p(T) | {2} | {2} | {3} | {5} | {3} | {3,3} |
| time[s] | {0.000000} | {0.008771} | {0.027269} | {0.021384} | {0.001893} | {0.002203,0.007213} |

**Obrázek 6.4:** Ukázka výsledku porovnání optimality testovacích scénářů ve výstupním souboru s výsledky testů

## 6.5 Uživatelská příručka

V následující části jsou uvedeny požadavky pro spuštění frameworku, postup, jak jej obsluhovat, a nakonec také rady, jak porozumět jeho výstupu.

### Požadavky na systém

Pro spuštění frameworku je třeba mít k dispozici systém s operačním systémem s nainstalovanou Java Virtual Machine ve verzi podporující Java SE 8.

### Potřebné soubory pro spuštění

Pro spuštění frameworku je třeba mít k dispozici:

- Spustitelný soubor s aplikací Oxygen ve verzi s implementovaným frameworkem.

- Projekt aplikace Oxygen s diagramem/y zobrazujícím/i model testovaného systému.
- Textový konfigurační soubor, upřesňující parametry pro běh frameworku, jehož formát je uveden níže.

### Formát konfiguračního souboru

Framework lze spustit ve dvou variantách, které požadují jiný formát a obsah konfikuračního souboru.

Popis požadovaného formátu těchto souborů je následující:

1. Při generování testovacích scénářů (zvolení parametru příkazové řádky: „--meassure“) musí konfigurační soubor obsahovat na jednotlivých řádcích hodnoty:
   - Výchozí prioritu hran (pro techniku PPT).
   - Výchozí úroveň kritéria *limit*.
   - Zadání měřených algoritmů pro generování scénářů a jejich parametry. Název algoritmu je uveden jeho zkratkou. Pro jednotlivé algoritmy je to tedy:
     - Shortest Paths Composition:
       LCPT[1] *limit UPH* [2]
     - Enforced Prime Paths:
       EPP *limit UPH*
     - Process Cycle Test:
       PCT *TDL optimalizace*[3]
     - Prioritized Process Test:
       ST[4] *PTL TDL*

   Jako příklad je uveden konfigurační soubor, použitý při testování implementace frameworku a přiložený do přílohy elektronické verze této práce. Jeho obsahem je:

```
LOW
0.5
LCPT 0.1 EACH_IN_AND_OUT_ONCE
LCPT 0.1 ALL_COMBINATIONS
EPP 0.1 ALL_COMBINATIONS
PCT 1 FALSE
ST HIGH 1
```

[1] Zkratka LCPT je z anglického: „Limited Connectivity Process Test“, což je nadřazený název pro techniku Shortest Paths Composition

[2] EACH_IN_AND_OUT_ONCE pro kritérium KAŽDÝ_HRANIČNÍ_UZEL_JEDNOU, nebo ALL_COMBINATIONS pro VŠECHNY_KOMBINACE

[3] Možnost pro odstranění duplicitních hran, vysvětlená podrobněji na stránce věnované aplikaci Oxygen [37].

[4] Zkratka ST je z anglického: „Smoke test“, což je dřívější název pro techniku Prioritized Process Test

2. Ve funkci pro nalezení optimální sady testovacích scénářů (při uvedení parametru příkazové řádky: „--optimality“) musí konfigurační soubor obsahovat v libovolném pořadí několik dvojic klíč=hodnota a následně parametry pro zadání požadovaných algoritmů. V souboru tedy musí být uvedena:

   - Výchozí priorita hran, jako hodnota ke klíči „defaultPriority“.
   - Výchozí úroveň kritéria *limit* jako hodnota ke klíči „defaultLevel“.
   - Hodnoty konstant (popsané v části 4.6):
     $w_{|T|}$ ke klíči: „statistics_w_tc“,
     $w_{edges(T)}$ ke klíči: „statistics_w_edges“,
     a $w_{uedges(T)}$ ke klíči: „statistics_w_uedges“
   - Zadání měřených algoritmů pro generování scénářů a jejich parametrů, jak je popsáno u funkce pro generování scénářů.

   Jako příklad je uveden konfigurační soubor, použitý při testování implementace frameworku a přiložený do přílohy elektronické verze této práce. Jeho obsahem je:

```
defaultPriority=LOW
defaultLevel=0.5
statistics_w_tc=0.3
statistics_w_edges=0.4
statistics_w_uedges=0.3
LCPT 0.1 ALL_COMBINATIONS
EPP 0.1 ALL_COMBINATIONS
PCT 1 false
```

### Spuštění frameworku

Pro spuštění frameworku je nutné použít příkazové řádky. Všechny potřebné soubory pro běh frameworku, popsané výše, je pak dobré mít umístěné v jedné složce, určené pouze pro tento účel. Následně je zapotřebí spustit aplikaci Oxygen a jako parametry uvést:

- Funkcionalitu frameworku, kterou chceme spustit (--meassure, nebo --optimality).
- Cestu k souboru s pojektem ve formátu aplikace Oxygen.
- Cestu ke konfiguračnímu souboru.

### Ukázka příkazu spouštějícím framework

Příklad příkazu pro spuštění frameworku z příkazové řádky se soubory přiloženými v příloze elektronické verze práce (a popsanými v přílohách A a B), je:
`java -jar ./oxygenSwingApp.jar --meassure SPC_testing_insta-nces.prj input_meassure.txt`

### Výstup frameworku

Framework vytváří několik souborů, jejichž název obsahuje vždy nejprve unikátní identifikátor (čas spuštění frameworku) a následně, za podtržítkem, zbylou textovou část názvu.

- Výsledné tabulky s naměřenými hodnotami ve formátu CSV (například: „230076674399146_log.csv“).
- Instanci projektu ve formátu Oxygen s vygenerovanými testovacími scénáři (například: „230076674399146_SPC_testing_instances.prj“)
- Log se shrnutím úspěšnosti generování scénářů jednotlivými technikami (například: „230076674399146_tc_gen.log“)

V případě, že běh frameworku neskončí úspěšně, vygeneruje se pouze soubor s výpisem chyby (například: „173848023420268_error_log.txt“.

# Kapitola 7

# Ověření frameworku

V této kapitole je uvedeno ověření, že výsledná funkčnost frameworku odpovídá specifikovaným požadavkům. Jsou zde také popsány algoritmy, které byly při testování použity pro generování testovacích scénářů, instance, na kterých byly algoritmy spuštěny a nakonec výsledky tohoto testování.

## 7.1 Testování funkčnosti frameworku

Nejdůležitější funkcionality frameworku ze seznamu v části 5.1, byly ověřeny sadou „end-to-end“ testů, tedy spuštěním celé aplikace a porovnáním výsledků jejího běhu s očekávaným výstupem. Ten byl vytvořen pro pět vybraných instancí z tabulky 7.3. Pro každou z těchto instancí byly ručně nalezeny testovací scénáře technikou, popsanou v části 7.3, a oběma kritérii škálování intenzity testovacích scénářů, popsanými v části 4.5. Podrobnější informace o kontrolních bodech a pokrytých funkčních požadavcích tímto testem, jsou uvedeny v tabulce 7.1.

| **Čísla požadavků** | **Kontrolní bod** |
|---|---|
| 1, 2, 3, 5, 13 | Exekuce frameworku nad projektem vytvořeným v aplikaci Oxygen, zakončená vygenerováním CSV souboru s validními daty. |
| 1, 3, 4, 8, 13 | Exekuce frameworku z příkazové řádky s oběma definovanými přepínači s kontrolou výsledku ve vygenerovaném CSV souboru. |
| 8, 6, 7, 13 | Kontrola generování testovacích scénářů všemi specifikovanými technikami při změnách vstupního konfiguračního souboru. |
| 6, 7, 13 | Kontrola generování testovacích scénářů s různými kritérii škálování intenzity testovacích scénářů, specifikovaných ve vstupním konfiguračním souboru. |

| Čísloa požadavků | Kontrolní bod |
|---|---|
| 9 | Kontrola odpovídající hodnoty konstant ve funkci pro výběr optimální sady scénářů z části 4.1 a výpočtu hodnot metrik z výstupu frameworku v souladu se specifikací z části 4.6. |
| 10 | Shodu parametrů testovaných grafů (počet hran, počet uzlů,...) s danými hodnotami ve výstupu frameworku. |
| 11 | Podobnost parametrů frameworkem vygenerovaných testovacích scénářů s ručně nalezenými. |
| 12 | Shoda počtu frameworkem nalezených chybových dvojic ve vygenerovaných testovacích scénářích s ručně nalezenými. |
| 13 | Kontrola formátu vygenerovaných dat v CSV souboru. |
| 15 | Porovnání výsledného stavu generování testovacích scénářích jednotlivými technikami ve výstupních logovacích souborech s realitou ve vygenerovaných souborech s výsledky. |
| 16 | Načtení a kontrola vygenerovaného souboru s nalezenými testovacími scénáři v aplikaci Oxygen. |
| 17 | Dobu exekuce frameworku v řádu maximálně jednotek vteřin. |
| 18 | Spuštění frameworku s neplatným konfiguračním souborem a následná kontrola vygenerovaného chybového souboru. |

**Tabulka 7.1:** Specifikace testu funkcionalit framework

Ve všech kontrolních bodech, uvedených v tabulce 7.1, byl výstup frameworku v souladu s požadavky a jeho správná funkčnost byla ověřena.

## 7.2 Testované algoritmy

Primárním požadavkem na framework je vygenerování sady testovacích scénářů dle zadaných algoritmů. Vstupem pro tyto algoritmy je model testovaného systému, tedy propojených IoT zařízení, ve formě jednoduchého orientovaného grafu, který splňuje několik základních pravidel. Uzly v tomto grafu reprezentují IoT zařízení či jejich části, hrany pak existenci přímé komunikace mezi nimi. Podrobnější specifikace modelu, nad kterým pracují uvedené algoritmy, je uvedena v části 4.3

Jednotlivé algoritmy se od sebe liší v tom, jakým způsobem z grafu generují testovací scénáře.

### 7.2.1 Shortest Paths Composition

Algoritmus Shortest Paths Composition je jedním ze základních algoritmů navržených pro generování testovacích scénářů pro testy konektivity v IoT systémech a jeho popis bude součástí připravovaného článku do IEEE Transactions on Reliability na toto téma.

Algoritmus pracuje s orientovaným grafem, jako modelem testovaného systému IoT zařízení, jak je popsáno v části 4.3.

Fungování algoritmu se dělí do dvou částí. V první části se v každé ZOP hledají nejkratší cesty mezi hraničními uzly. V druhé části se tyto nejkratší cesty spojují do testovacích scénářů. Pokud bylo pro generování testovacích scénářů touto technikou nastaveno kritérium *UPH* na: VŠECHNY_KOMBINACE, musí být ve vygenerovaných scénářích obsaženy všechny nalezené nejkratší cesty z předchozí části algoritmu. Pokud bylo nastaveno kritérium *UPH* na KAŽDÝ_HRANIČNÍ_UZEL_JEDNOU, jsou nejkratší cesty spojovány do scénářů pouze do té chvíle, než je v nich obsažen alespoň jednou každý hraniční uzel.

### 7.2.2 Enforced Prime Paths

Tento algoritmus je založen na hledání hlavních cest (viz 2.1). A je také součástí připravovaného článku do IEEE Transactions on Reliability.

V první části algoritmu se vytvoří množina testovacích požadavků, které reflektují požadované cesty zónou omezeného připojení. Druhá část používá již existující algoritmus jménem „Prefix-Graph based Solution“ od Li a Offutta na vygenerování výsledných testovacích scénářů [32].

### 7.2.3 PCT *TDL*=1

Algoritmus Process Cycle Test (PCT) je založen na metodice TMAP Next [33]. Pro škálování intenzity testovacích scénářů se využívá parametr hloubka pokrytí (TDL) (viz 4.5).

Implementace tohoto algoritmu je, jak popisuje M. Bureš [43], rozložena do dvou částí. Nejprve se pro každý uzel v grafu s výstupním stupněm větším než jedna vygenerují kombinace s počtem hran odpovídající zvolené hloubce pokrytí. V druhé části algoritmu se kombinace spojují a vytváří se z nich výsledné testovací scénáře. To se děje do té chvíle, než jsou ve scénářích použity všechny kombinace.

### 7.2.4 PPT s entry a exit pointy s prioritou HIGH

Algoritmus Prioritized Process Test (PPT) vychází z předešlého algoritmu PCT. V grafu modelujícím testovaný systém je však navíc možné ohodnotit hrany vysokou, střední či nízkou prioritou. Kromě kritéria hloubky pokrytí je pak nutné pro vygenerování testovacích scénářů vybrat také hodnotu Prioritized Test Level (PTL) [34].

Implementace tohoto algoritmu vychází z implementace algoritmu PCT. V první části se na základě *TDL* kritéria vygenerují odpovídající kombinace, tentokrát však zůstanou pouze ty, které obsahují pouze sekvence hran, které začínají nějakou hranu, která má prioritu větší nebo rovnou kritériu *PTL*. Další část algoritmu se již od PCT neliší; nalezené kombinace se stejným způsobem spojují do testovacích scénářů.

## 7.3 Testované instance problému

Pro testování funkcionalit frameworku, ale také vyvíjených algoritmů pro testování konektivity IoT zařízení, bylo vybráno několik instancí. Tyto instance byly vybrány ve spolupráci se školitelem a odpovídají specifikacím stanoveným v části 4.3. Je jich celkem dvacet a každá obsahuje alespoň jednu ZOP. Některé instance obsahují kružnice a to jak v ZOP tak mimo ní.

| **Graf č.** | **Počet vrcholů** | **Počet hran** | **Počet hran s cop(x)** | **Počet ZOP** | **Počet VSTUP uzlů** | **Počet VÝSTUP uzlů** | **SSK v ZOP**[1] | **SSK v grafu**[2] |
|---|---|---|---|---|---|---|---|---|
| 1 | 19 | 30 | 5 | 1 | 2 | 1 | 0 | 2 |
| 2 | 24 | 38 | 7 | 1 | 2 | 1 | 0 | 3 |
| 3 | 24 | 38 | 11 | 1 | 3 | 1 | 0 | 3 |
| 4 | 30 | 49 | 24 | 2 | 4 | 3 | 0 | 1 |
| 5 | 30 | 49 | 14 | 2 | 2 | 2 | 0 | 1 |
| 6 | 25 | 40 | 14 | 1 | 1 | 3 | 0 | 1 |
| 7 | 25 | 40 | 15 | 1 | 3 | 1 | 0 | 1 |
| 8 | 33 | 52 | 15 | 2 | 2 | 4 | 0 | 2 |
| 9 | 33 | 52 | 16 | 1 | 2 | 1 | 0 | 2 |
| 10 | 28 | 48 | 19 | 3 | 7 | 7 | 1 | 3 |
| 11 | 28 | 48 | 21 | 1 | 1 | 2 | 1 | 3 |
| 12 | 25 | 38 | 11 | 1 | 1 | 2 | 0 | 2 |
| 13 | 25 | 38 | 12 | 2 | 4 | 2 | 1 | 2 |
| 14 | 26 | 41 | 17 | 3 | 4 | 5 | 2 | 2 |
| 15 | 26 | 41 | 18 | 2 | 3 | 5 | 0 | 2 |
| 16 | 40 | 57 | 18 | 3 | 4 | 7 | 0 | 3 |
| 17 | 40 | 57 | 23 | 2 | 5 | 6 | 2 | 3 |
| 18 | 40 | 57 | 22 | 2 | 5 | 5 | 2 | 3 |
| 19 | 19 | 26 | 17 | 2 | 2 | 1 | 0 | 0 |
| 20 | 19 | 26 | 13 | 2 | 2 | 3 | 0 | 0 |

**Tabulka 7.2:** Vlastnosti grafů modelujících instance testovaných systémů

## 7.4 Výsledky testování

V této sekci jsou uvedeny výsledky testu účinnosti scénářů a také frameworkem vyhodnocená optimalita sad testovacích scénářů, které byly vygenerovány

[1] Počet silně souvislých komponent s počtem uzlů větším než 1 v ZOP
[2] Počet silně souvislých komponent s počtem uzlů větším než 1 v grafu

dvěma v současné době implementovanými algoritmy pro generování testovacích scénářů pro testy konektivity IoT systémů.

### 7.4.1 Popis vstupních dat použitých v testech

Popis jednotlivých testovaných instancí a počet vybraných chybových dvojic je uveden v tabulce 7.3.

| Č. | Počet ZOP | Počet VSTUP uzlů | Počet VÝSTUP uzlů | Počet chybových dvojic v grafu |
|---|---|---|---|---|
| 1 | 1 | 2 | 1 | 2 |
| 2 | 1 | 2 | 1 | 2 |
| 3 | 1 | 3 | 1 | 3 |
| 4 | 2 | 4 | 3 | 5 |
| 5 | 2 | 2 | 2 | 3 |
| 6 | 1 | 1 | 3 | 3 |
| 7 | 1 | 3 | 1 | 4 |
| 8 | 2 | 2 | 4 | 4 |
| 9 | 1 | 2 | 1 | 2 |
| 10 | 3 | 7 | 7 | 11 |
| 11 | 1 | 1 | 2 | 2 |
| 12 | 1 | 1 | 2 | 2 |
| 13 | 2 | 4 | 2 | 4 |
| 14 | 3 | 4 | 5 | 7 |
| 15 | 2 | 3 | 5 | 6 |
| 16 | 3 | 4 | 7 | 9 |
| 17 | 2 | 5 | 6 | 10 |
| 18 | 2 | 5 | 5 | 9 |
| 19 | 2 | 2 | 1 | 2 |
| 20 | 2 | 2 | 3 | 3 |

**Tabulka 7.3:** Popis vstupních dat pro ověření frameworku

### 7.4.2 Test účinnosti scénářů

Test účinnosti byl spuštěn se všemi algoritmy popsanými v části 7.2 a jeho výsledky jsou uvedeny v tabulce 7.4. Ta, kromě počtu chybových dvojic ve vygenerovaných scénářích, navíc zobrazuje procentuální poměr vůči jejich celkovému počtu v grafu.

Výsledky v tabulce 7.4 lze interpretovat tak, že technika SPC (popsaná v části 7.2.1), která byla vyvinuta přímo pro generování testovacích scénářů pro testování konektivity IoT zařízení konektivity vykazuje v počtu odhalených chybových dvojic nejlepší výsledky. Pokud je u ní navíc nastaveno kritérium *UPH* na: VŠECHNY_KOMBINACE, odhalí ve všech testovaných případech dokonce 100 % chybových dvojic, což odpovídá ná-

vrhu této hodnoty kritéria. Avšak i hodnota kritéria *UPH*, nastavená na: KAŽDÝ_HRANIČNÍ_UZEL_JEDNOU vykazuje dobré výsledky. Oproti tomu algoritmus EPP u některých instancí testovací scénáře ani nevygeneroval. Algoritmus PCT má dobré výsledky v počtu odhalených chybových dvojic, ovšem za cenu většího počtu scénářů, jak je vidět v souboru: „output measure/229917005793463_log.csv“, odevzdaném v příloze elektronické verze práce. Poslední měřený algoritmus, PPT, obsahuje na několika instancích velmi nízké pokrytí chybových dvojic, z čehož lze vyvodit, že pro testování konektivity IoT systémů není vhodný.

| č. | SPC A[3] | % | SPC E[4] | % | EPP[5] | % | PCT[6] | % | PPT[7] | % |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 100 % | 2 | 100 % | 0 | 0 % | 2 | 100 % | 2 | 100 % |
| 2 | 2 | 100 % | 2 | 100 % | 2 | 100 % | 2 | 100 % | 2 | 100 % |
| 3 | 3 | 100 % | 3 | 100 % | 0 | 0 % | 3 | 100 % | 3 | 100 % |
| 4 | 5 | 100 % | 5 | 100 % | 5 | 100 % | 5 | 100 % | 5 | 100 % |
| 5 | 3 | 100 % | 3 | 100 % | 3 | 100 % | 3 | 100 % | 3 | 100 % |
| 6 | 3 | 100 % | 3 | 100 % | 3 | 100 % | 3 | 100 % | 2 | 67 % |
| 7 | 4 | 100 % | 4 | 100 % | 0 | 0 % | 4 | 100 % | 4 | 100 % |
| 8 | 4 | 100 % | 4 | 100 % | 3 | 75 % | 4 | 100 % | 4 | 100 % |
| 9 | 2 | 100 % | 2 | 100 % | 2 | 100 % | 2 | 100 % | 2 | 100 % |
| 10 | 11 | 100 % | 9 | 82 % | 8 | 73 % | 9 | 82 % | 6 | 55 % |
| 11 | 2 | 100 % | 2 | 100 % | 2 | 100 % | 2 | 100 % | 2 | 100 % |
| 12 | 2 | 100 % | 2 | 100 % | 2 | 100 % | 2 | 100 % | 2 | 100 % |
| 13 | 4 | 100 % | 4 | 100 % | 3 | 75 % | 4 | 100 % | 4 | 100 % |
| 14 | 7 | 100 % | 7 | 100 % | 7 | 100 % | 6 | 86 % | 7 | 100 % |
| 15 | 6 | 100 % | 6 | 100 % | 6 | 100 % | 6 | 100 % | 6 | 100 % |
| 16 | 9 | 100 % | 9 | 100 % | 9 | 100 % | 8 | 89 % | 9 | 100 % |
| 17 | 10 | 100 % | 9 | 90 % | 10 | 100 % | 8 | 80 % | 6 | 60 % |
| 18 | 9 | 100 % | 7 | 78 % | 9 | 100 % | 7 | 78 % | 4 | 44 % |
| 19 | 2 | 100 % | 2 | 100 % | 0 | 0 % | 2 | 100 % | 2 | 100 % |
| 20 | 3 | 100 % | 3 | 100 % | 3 | 100 % | 3 | 100 % | 3 | 100 % |

**Tabulka 7.4:** Výsledky testu účinnosti testovacích scénářů zvolenými technikami

### 7.4.3 Porovnání optimality vygenerovaných testovacích scénářů

Tabulka 7.5 obsahuje extrakt výsledků porovnání vybraných technik generování testovacích scénářů pro testování konektivity IoT systémů. Význam jednotlivých metrik uvedených v tabulce je vysvětlen v části 4.6. Hodnoty konstant pro vyhodnocení funkce optimality (viz část 4.6) byly, dle instrukcí od školitele, nastaveny na: $w_{|T|} = 0.3$, $w_{edges(T)} = 0.4$ a $w_{uedges(T)} = 0.3$. Kompletní výstup frameworku s tímto porovnáním je uveden v souboru: „optimality/2300766743-99146_log.csv“ v příloze elektronické verze práce.

[3]Shortest Paths Composition s *UPH* = VŠECHNY_KOMBINACE
[4]Shortest Paths Composition s *UPH* = KAŽDÝ_HRANIČNÍ_UZEL_JE-DNOU
[5]Enforce Prime Paths
[6]Process Cycle Test s *TDL*=1
[7]Prioritized Process Test

Pro generování testovacích scénářů byly použity vybrané techniky z části 7.2. Konkrétně se jednalo o algoritmy SPC a EPP s kritériem: VŠECHNY_KOMBINACE a algoritmus PCT s hloubkou pokrytí nastavenou na: 1. Pro generování naopak nebyl použit algoritmus PPT, neboť, jak je vidět ve výsledcích testu účinnosti scénářů, pro testování konektivity IoT systémů není vhodný a jeho zařazení by mohlo nepříznivě ovlivnit výsledek srovnání.

| **č.** | **technika** | **\|T\|** | **e (T)**[8] | **ue (T)**[9] | **n (T)**[10] | **un (T)**[11] | **er (T)** | **b (T)**[12] | **ub (T)**[13] | **br (T)**[14] | **ep (T)**[15] | **čas[s]** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SPC | {2} | {17} | {10} | {15} | {9} | {0.33} | {5} | {3} | {0.2} | {2} | {0} |
| 2 | EPP | {2} | {18} | {11} | {16} | {10} | {0.29} | {5} | {3} | {0.19} | {2} | {0.01} |
| 3 | SPC | {3} | {23} | {13} | {20} | {12} | {0.34} | {9} | {4} | {0.2} | {3} | {0.04} |
| 4 | SPC | {2} | {17} | {14} | {15} | {13} | {0.29} | {8} | {5} | {0.33} | {5} | {0.03} |
| 5 | EPP, SPC | {2, 2} | {16, 16} | {11, 11} | {14, 14} | {10, 10} | {0.22}, {0.22} | {7, 7} | {4, 4} | {0.29}, {0.29} | {3, 3} | {0}, {0.01} |
| 6 | EPP | {2} | {15} | {10} | {13} | {9} | {0.25} | {6} | {4} | {0.31} | {3} | {0} |
| 7 | SPC | {3} | {20} | {13} | {17} | {12} | {0.33} | {10} | {5} | {0.29} | {4} | {0.01} |
| 8 | EPP | {3} | {28} | {14} | {25} | {13} | {0.27} | {10} | {5} | {0.2} | {3} | {0} |
| 9 | EPP, SPC | {1, 1} | {10, 10} | {8, 8} | {9, 9} | {7, 7} | {0.15}, {0.15} | {4, 4} | {3, 3} | {0.33}, {0.33} | {2, 2} | {0}, {0.01} |
| 10 | EPP | {6} | {36} | {25} | {30} | {21} | {0.52} | {20} | {10} | {0.33} | {8} | {0} |
| 11 | EPP | {1} | {4} | {4} | {3} | {3} | {0.08} | {3} | {3} | {1} | {2} | {0} |
| 12 | SPC | {1} | {14} | {12} | {13} | {11} | {0.32} | {4} | {3} | {0.23} | {2} | {0.01} |
| 13 | EPP | {1} | {7} | {7} | {6} | {6} | {0.18} | {5} | {5} | {0.83} | {3} | {0} |
| 14 | SPC | {6} | {38} | {24} | {32} | {20} | {0.59} | {18} | {8} | {0.25} | {7} | {0.01} |
| 15 | SPC | {3} | {19} | {17} | {16} | {15} | {0.41} | {13} | {8} | {0.5} | {6} | {0.01} |
| 16 | EPP | {6} | {60} | {31} | {54} | {30} | {0.54} | {30} | {11} | {0.2} | {9} | {0} |
| 17 | SPC | {8} | {104} | {39} | {96} | {38} | {0.68} | {45} | {11} | {0.11} | {10} | {0.02} |
| 18 | SPC | {8} | {104} | {39} | {96} | {38} | {0.68} | {45} | {11} | {0.11} | {9} | {0.02} |
| 19 | SPC | {2} | {10} | {10} | {8} | {8} | {0.38} | {5} | {3} | {0.38} | {2} | {0.01} |
| 20 | EPP | {1} | {9} | {9} | {8} | {8} | {0.35} | {5} | {5} | {0.63} | {3} | {0} |

**Tabulka 7.5:** Výsledky porovnání vygenerovaných testovacích scénářů

Z výsledku porovnání optimality vygenerovaných sad testovacích scénářů, uvedených v tabulce 7.5 plyne, že pro testování konektivity IoT zařízení lze efektivně použít techniky SPC a EPP. Použít techniku PCT oproti tomu efektivní není, neboť jí vygenerovaná sada nebyla vyhodnocena jako optimální ani u jedné testované instance.

[8] edges(T)
[9] uedges(T)
[10] nodes(T)
[11] unodes(T)
[12] bnodes(T)
[13] ubns(T)
[14] bnr(T)
[15] lcz_e_p(T)

# Kapitola 8

# Závěr

Odvětví internetu věcí má, dle odhadů k roku 2025, obsáhnout až 100 miliard připojených zařízení za finančního dopadu na globální ekonomiku až 11,1 bilionu amerických dolarů. V souladu se vzrůstem počtu zařízení budou vzrůstat také požadavky na kvalitu IoT zařízení, přičemž jednou z důležitých a zároveň málo popsaných částí zajištění kvality IoT zařízení, je zjištění vlivu výpadku připojení na testované zařízení. Framework na vyhodnocování vlastností testovacích scénářů pro testy konektivity v IoT systémech, vyvinutý v rámci této diplomové práce, právě tuto kategorii testů posunuje dále. Umožňuje totiž snadno ověřit a porovnat kvality sad testovacích scénářů vygenerovaných jednotlivými technikami.

## Dosažené výsledky

Při řešení tohoto projektu byly dosaženy cíle stanovené v úvodu práce a byly splněny všechny požadavky zadání. Text práce obsahuje na úvod informace k technologiím, používaným ve světě internetu věcí a popis stavebních bloků, ze kterých se zařízení tohoto odvětví skládají.

Dále je v textu přesněji specifikován řešený problém a způsob, kterým je vhodné ho řešit. Napomáhá tomu upravená verze orientovaného grafu $G$, kterým je zachycen model testovaného IoT systému a ze kterého se specifickými technikami dají vygenerovat testovací scénáře. Kritérii popsanými v textu práce lze škálovat počet testovacích scénářů, ale také je ohodnotit a tak porovnat jejich optimalitu.

Text práce obsahuje také specifikaci návrhu na framework. V ní je uveden soupis funkčních požadavků, návrh architektury a popis použitých technologií a knihoven. Součástí návrhu frameworku je také návrh testu účinnosti scénářů, díky kterému je možné vyhodnotit kvalitu algoritmů použitých pro generování testovacích scénářů. Na konci této části textu je uveden návrh algoritmu na ověření konzistence testovacích scénářů, který ve frameworku garantuje splnění specifikovaných požadavků na kvalitu vygenerované sady testovacích scénářů.

Implementace vyvinutého frameworku je přiblížena popisem jeho nejdůležitějších modulů, pseudokódem implementace testu účinnosti scénářů a pseudokódem algoritmu na ověření konzistence scénářů. V kapitole o implementaci frameworku je také na konkrétních příkladech popsán způsob použitý pro

zobrazení výsledků běhu frameworku. Uživatelská příručka, která přibližuje budoucím uživatelům frameworku jeho obsluhu, je také uvedena v této části práce.

V poslední kapitole je uvedeno ověření frameworku. To bylo provedeno ověřením funkčnosti jednotlivých požadavků, specifikovaných v textu práce. Dále je zde uveden popis a základní princip fungování algoritmů, které byly použity při testování funkčnosti frameworku a které lze teoreticky pro generování testovacích scénářů pro testy konektivity v systémech internetu věcí použít. Na konci kapitoly jsou popsány modelové instance problému, které byly použity při testování frameworku, a výsledky tohoto testování.

### Možná rozšiřitelnost frameworku

Framework je vyvinut s cílem ho do budoucna rozšiřovat o nově vyvinuté techniky generování testovacích scénářů. Jedním z možných vylepšení pak je usnadnění způsobu přidávání těchto nových technik, tedy aby bylo možné to provést s žádnými, či pouze minimálními zásahy do implementace frameworku.

Dalším možným rozšířením je sjednocení formátu načítaných konfiguračních souborů, který je, pro funkci pro generování testovacích scénářů a pro funkci pro porovnání jejich optimality, odlišný. Uživatelskému komfortu by pomohlo také vyhotovení přehledného návodu k použití.

Eventuálně lze framework doplnit také o možnost nastavení formátu a obsahu výstupního souboru, který generuje. V aktuální implementaci totiž framework generuje všechny metriky pro všechny načtené grafy a specifikované techniky a neumožňuje z nich učinit nějaký výběr. Kvůli tomu může být výsledný soubor, při zvolení většího množství grafů a technik, na první pohled trochu nepřehledný.

### Využitelnost frameworku

Framework je využitelný především pro ověřování účinnosti testovacích scénářů pro testování konektivity IoT zařízení. Lze ho ale využít také pro porovnání sad testovacích scénářů, vygenerovaných různými technikami. To je umožněno díky metrikám, v textu práce popsaným, kterými jsou sady vygenerovaných testovacích scénářů jednoznačně ohodnoceny. Framework umožňuje také kontrolu konzistence vygenerovaných testovacích scénářů a test jejich účinnosti. Díky těmto funkcím lze framework využít také při vývoji nových technik pro generování testovacích scénářů při testech konektivity IoT zařízení.

Bezprostředním přínos frameworku přinese jeho použití při testování navržené implementace algoritmu Shortest Path Composition (SPC), jehož činnost je také ve zkratce popsána v textu práce. Algoritmus SPC umožňuje z modelu IoT zařízení generovat testovací scénáře, využitelné při testech konektivity tohoto zařízení. Vývoj SPC i dalších technik pro řešení tohoto problému probíhá na katedře počítačů Fakulty elektrotechnické ČVUT a jejich přesný popis bude součástí článku do odborného časopisu IEEE Transactions on Reliability, jehož jsem spoluautorem. Framework, vyvinutý v této práci, bude při ověřování nových technik v budoucnu nepostradatelným nástrojem.

Porovnání optimality vygenerovaných sad testovacích scénářů zatím implementovanými technikami nad několika modelovými instancemi je také v této práci uvedeno. Z výsledků vyplývá, že pro generování testovacích scénářů pro testování konektivity IoT systémů je technika SPC, vyvinutá na katedře počítačů Fakulty elektrotechnické ČVUT, nejlépe využitelná.

Využít lze framework také pro odhalení chyb v implementaci jednotlivých technik pro generování testovacích scénářů, což dokázal již v průběhu vypracování této práce. Test účinnosti scénářů totiž odhalil chybu právě v implementaci algoritmu SPC a po její opravě umožnil následně ověřit soulad implementace této techniky s jejími požadavky.

# Literatura


[1] "Cisco visual networking index: Forecast and trends, 2017–2022 white paper," Feb 2019. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html>

[2] L. Huawei Technologies Co., "Giv 2025 unfolding the industry blueprint of an intelligent world," May 2018. [Online]. Available: <https://www.huawei.com/minisite/giv/en/download/whitebooken20180508.pdf>

[3] "Standard glossary of terms used in software testing." [Online]. Available: <https://www.istqb.org/downloads/send/20-istqb-glossary/186-glossary-all-terms.html>

[4] W. W. Royce, "Managing the development of large software systems," in *Proceedings IEEE WESCON*, Aug. 1970, pp. 1–9.

[5] P. Ammann and J. Offutt, *Introduction to Software Testing*, ser. Introduction to Software Testing. Cambridge University Press, 2016. [Online]. Available: <https://books.google.cz/books?id=bQtQDQAAQBAJ>

[6] E. Dustin, T. Garrett, and B. Gauf, *Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality*, 1st ed. Addison-Wesley Professional, 2009.

[7] M. Bishop, *Introduction to Computer Security*. Addison-Wesley Professional, 2004.

[8] K. Ashton, "That 'internet of things' thing," Jul 2009. [Online]. Available: <https://www.rfidjournal.com/articles/view?4986>

[9] A. I. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 2347–2376, 2015.

[10] K. Rose, S. Eldridge, and L. Chapin, "The internet of things: An overview," *Internet Society*, 2015.

[11] I. S. Enables, "The internet of things: Mapping the value beyond the hype," 2015.

[12] N. Koshizuka and K. Sakamura, "Ubiquitous id: Standards for ubiquitous computing and the internet of things," *IEEE Pervasive Computing*, vol. 9, no. 4, pp. 98–101, October 2010.

[13] N. C. Wu, M. A. Nystrom, T. R. Lin, and H. C. Yu, "Challenges to global rfid adoption," in *2006 Technology Management for the Global Future - PICMET 2006 Conference*, vol. 2, July 2006, pp. 618–623.

[14] R. Frank, *Understanding Smart Sensors, Second Edition*, 2nd ed. Norwood, MA, USA: Artech House, Inc., 2000.

[15] Y. Zheng, X. Ding, C. C. Y. Poon, B. P. L. Lo, H. Zhang, X. Zhou, G. Yang, N. Zhao, and Y. Zhang, "Unobtrusive sensing and wearable devices for health informatics," *IEEE Transactions on Biomedical Engineering*, vol. 61, no. 5, pp. 1538–1554, May 2014.

[16] M. R. Palattella, M. Dohler, A. Grieco, G. Rizzo, J. Torsner, T. Engel, and L. Ladid, "Internet of things in the 5g era: Enablers, architecture, and business models," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 510–527, March 2016.

[17] M. Woolley, "Bluetooth 5: Go faster. go further | bluetooth technology website." [Online]. Available: `<https://www.bluetooth.com/bluetooth-technology/bluetooth5/bluetooth5-paper>`

[18] Allied Business Intelligence, Inc., "Bluetooth 5 evolution will lead to widespread deployments on the iot landscape," Jul 2016. [Online]. Available: `<https://www.abiresearch.com/press/bluetooth-5-evolution-will-lead-widespread-deploym/>`

[19] J. Bravo, R. Hervas, G. Chavira, and S. Nava, "Adapting technologies to model contexts: Two approaches through rfid & nfc," in *Adapting technologies to model contexts: Two approaches through RFID & NFC*, 11 2007, pp. 683 – 688.

[20] R. Want, "An introduction to rfid technology," *Pervasive Computing, IEEE*, vol. 5, pp. 25 – 33, 02 2006.

[21] "Canonical exi," Jun 2018. [Online]. Available: `<https://www.w3.org/TR/exi-c14n/>`

[22] M. Gigli and S. G. M. Koo, "Internet of things: Services and applications categorization," *Adv. Internet of Things*, vol. 1, pp. 27–31, 2011.

[23] M. Bures, T. Cerny, and B. S. Ahmed, "Internet of things: Current challenges in the quality assurance and testing methods," in *Information Science and Applications 2018*, K. J. Kim and N. Baek, Eds. Singapore: Springer Singapore, 2019, pp. 625–634.

[24] B. Schneier, *Security Is a Weakest-Link Problem*. New York, NY: Springer New York, 2003, pp. 103–117. [Online]. Available: <https://doi.org/10.1007/0-387-21712-6_8>

[25] J. Kasurinen, O. Taipale, and K. Smolander, "Software test automation in practice: Empirical observations," *Adv. Software Engineering*, vol. 2010, pp. 620 836:1–620 836:18, 2010.

[26] D. Mohammad Rafi, K. Reddy Kiran Moses, K. Petersen, and M. Mäntylä, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *2012 7th International Workshop on Automation of Software Test, AST 2012 - Proceedings*, 06 2012, pp. 36–42.

[27] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing: Introduction, Management, and Performance*. Pearson Education, 1999. [Online]. Available: <https://books.google.cz/books?id=kl2H0G6EFf0C>

[28] M. Bures, "Metrics for automated testability of web applications," in *Proceedings of the 16th International Conference on Computer Systems and Technologies*. ACM, 2015, pp. 83–89.

[29] M. Bures, "Model for evaluation and cost estimations of the automated testing architecture," in *New Contributions in Information Systems and Technologies*. Springer, 2015, pp. 781–787.

[30] M. Bures, "Automated testing in the czech republic: The current situation and issues," *ACM International Conference Proceeding Series*, vol. 883, pp. 294–301, 06 2014.

[31] I. Molyneaux, *The art of application performance testing*, 2nd ed. O'Reilly Media, Inc., 2015.

[32] N. Li, F. Li, and J. Offutt, "Better algorithms to minimize the cost of test paths," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 280–289.

[33] T. Koomen, L. v. d. Aalst, B. Broekman, and M. Vroon, *TMap Next, for Result-driven Testing*. UTN Publishers, 2006.

[34] M. Bures, T. Cerny, and M. Klima, "Prioritized process test: More efficiency in testing of business processes and workflows," in *International Conference on Information Science and Applications*. Springer, 2017, pp. 585–593.

[35] A. Dwarakanath and A. Jankiti, "Minimum number of test paths for prime path and other structural coverage criteria," in *IFIP International Conference on Testing Software and Systems*. Springer, 2014, pp. 63–79.

[36] M. Bures, B. S. Ahmed, and K. Z. Zamli, "Prioritized process test: An alternative to current process testing strategies," *arXiv preprint arXiv:1903.08531*, 2019.

[37] M. Bures, "Experimental model-based testing environment." [Online]. Available: `<http://still.felk.cvut.cz/oxygen/>`

[38] M. Bures, T. Cerny, K. Frajtak, and B. S. Ahmed, "Testing the consistency of business data objects using extended static testing of crud matrices," *Cluster Computing*, Aug 2017. [Online]. Available: `<https://doi.org/10.1007/s10586-017-1118-7>`

[39] "Tiobe index for march 2019." [Online]. Available: `<http://www.tiobe.com/tiobe_index>`

[40] "List of java virtual machines," Mar 2019. [Online]. Available: `<https://en.wikipedia.org/wiki/List_of_Java_virtual_machines>`

[41] "Oracle jdk 8 and jre 8 certified system configurations contents." [Online]. Available: `<https://www.oracle.com/technetwork/java/javase/certconfig-2095354.html>`

[42] P. Kruchten, "Architectural blueprints–the 4+ 1 view model of software architecture, software, vol. 12, number 6," 1995.

[43] M. Bures, "Pctgen: Automated generation of test cases for application workflows," in *New Contributions in Information Systems and Technologies*, A. Rocha, A. M. Correia, S. Costanzo, and L. P. Reis, Eds. Cham: Springer International Publishing, 2015, pp. 789–794.

# Příloha A

# Testovací data

Vstupní data, na kterých proběhlo testování funkčnosti frameworku, jsou součástí příloh elektronické verze této diplomové práce.

Patří mezi ně následující soubory:

- **SPC_testing_instances.prj** s projektem ve formátu aplikace Oxygen, obsahujícím modely testovaných instancí, podrobněji popsané v části 7.3. Grafy v projektu obsahují také prioritní hrany pro generování scénářů algoritmem PPT a chybové dvojice pro test účinnosti scénářů.
- **input_meassure.txt**, což je konfigurační soubor, použitý frameworkem pro generování testovacích scénářů následujícími technikami:
  - Shortest Paths Composition s kritériem *limit* nastaveným na: 0.1 a *UPH* nastaveným na: KAŽDÝ_HRANIČNÍ_UZEL_JEDNOU
  - Shortest Paths Composition s kritériem *limit* nastaveným na: 0.1 a kritériem *UPH* nastaveným na: VŠECHNY_KOMBINACE
  - Enforced Prime Paths s kritériem *limit* nastaveným na: 0.1 a kritériem: VŠECHNY_KOMBINACE
  - Process Cycle Test s *TDL* nastaveným na: 1 a neaktivní optimalizací.
  - Prioritized Process Test s *TDL* nastaveným na: 1 a kritériem *PTL* nastaveným na: HIGH.
- **input_optimality.txt**, což je konfigurační soubor, použitý frameworkem pro nalezení optimální sady testovacích scénářů technikami:
  - Shortest Paths Composition s kritériem *limit* nastaveným na: 0.1 a kritériem *UPH* nastaveným na: VŠECHNY_KOMBINACE
  - Enforced Prime Paths s kritériem *limit* nastaveným na: 0.1 a kritériem *UPH* nastaveným na: VŠECHNY_KOMBINACE
  - Process Cycle Test s kritériem *TDL* nastaveným na: 1 a neaktivní optimalizací duplicitních hran.

  Výchozí priorita hran je nastavena na hodnotu: „LOW“, úroveň kritéria *limit* je stanovena na: 0.5 a konstanty: $w_{|T|} = 0.3$, $w_{edges(T)} = 0.4$ a $w_{uedges(T)} = 0.3$.

# Příloha B

## Struktura příloh elektronické verze práce

Součástí elektronické verze práce je archiv s testovacími daty, zdrojovými kódy frameworku a souborem „oxygenSwingApp.jar“, který umožňuje spuštění aplikace Oxygen s implementovaným frameworkem. Přiložená verze aplikace obsahuje pro demonstraci funkčnosti frameworku pouze algoritmy PCT a PPT. Dle instrukcí školitele nejsou součástí tohoto balíku algoritmy SPC a EPP, neboť aktuálně probíhá jejich vývoj, a především tyto algoritmy nejsou předmětem této diplomové práce.

Pro spuštění frameworku je proto nutné v konfiguračních souborech odstranit řádky zmiňující algoritmy SPC a EPP.

| | |
|---|---|
| ├── SPC_testing_instances.prj | Oxygen projekt s testovanými instancemi |
| ├── experiment | Složka se zdrojovými kódy frameworku |
| │ ├── exceptions | |
| │ ├── main | |
| │ ├── model | |
| │ ├── optimization | |
| │ ├── strategies | |
| │ ├── techniques | |
| │ └── utils | |
| ├── input_meassure.txt | Příklad konfig. souboru pro spuštění --meassure |
| ├── input_optimality.txt | Příklad konfig. souboru pro spuštění --optimality |
| ├── output meassure | Složka se soubory, vygenerovanými ve funkci frameworku --meassure |
| │ ├── 229917005793463_SPC_testing_instances.prj | Projekt s vygenerovanými testovacími scénáři |
| │ ├── 229917005793463_log.csv | Výsledky testování |
| │ └── 229917005793463_tc_gen.log | Log soubor informující o výsledku generování |
| ├── output optimality | Složka se soubory, vygenerovanými ve funkci frameworku --optimality |
| │ ├── 230076674399146_SPC_testing_instances.prj | Projekt s vygenerovanými testovacími scénáři |
| │ ├── 230076674399146_log.csv | Výsledky testování |
| │ └── 230076674399146_tc_gen.log | Log soubor informující o výsledku generování |
| └── oxygenSwingApp.jar | Spustitelný soubor aplikace Oxygen ve verzi s implementovaným frameworkem |

# Příloha C

## Výsledky testování

Kompletní výsledky testů, které framework naměřil, jsou uvedeny v elektronické verzi práce.

Výstup frameworku, spuštěného ve verzi pro generování testovacích scénářů, je uveden v souboru: „meassure/229917005793463_log.csv“.

Výstup frameworku, spuštěného ve verzi pro porovnání optimality vygenerovaných sad testovacích scénářů, je uveden v souboru: „optimality/2300766743-99146_log.csv“.