VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

AUTOMATICKÉ GENEROVÁNÍ TESTOVACÍCH DAT INFORMAČNÍCH SYSTÉMŮ

AUTOMATIC TEST INPUT GENERATION FOR INFORMATION SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ANDREJ NAŇO**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Naňo Andrej, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Informační systémy a databáze
Název: **Automatické generování testovacích dat informačních systémů**
Automatic Test Input Generation for Information Systems
Kategorie: Analýza a testování softwaru
Zadání:

1. Nastudujte principy testování založeného na datech. Nastudujte metody automatického generování strukturovaných testovacích dat podle zadaného kritéria.
2. Navrhněte nástroj pro automatické generování testovacích běhů. Nástroj bude vytvářet náhodná testovací data podobná reálnému provozu komplexního informačního systému. Zaměřte se na generování zpráv v rámci komunikace více informačních uzlů.
3. Implementujte vámi navržený nástroj. Nástroj by měl podporovat generování zpráv v rámci komunikace využívající standardní vysokoúrovňové protokoly RestAPI, OPC-UA apod.
4. Implementujte automatické testy základních funkcionality nástroje.

Literatura:

- R. Zhao, Z. Li, Q. Wang. Test Generation for Programs with Binary Tree Structure as Input. 2015. In International Journal of Software Engineering and Knowledge Engineering. Vol. 25, No. 07, pp. 1129-1151. doi: 10.1142/S0218194015500205
- M. Emmi, R. Majumdar, K. Sen. Dynamic Test Input Generation for Database Applications. In Proceedings of Intl. symposium on Software testing and analysis, 151-162, 2007.
- Standard XML 1.0, páté vydání. https://www.w3.org/TR/xml/

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Smrčka Aleš, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Nástroj ISAGEN umožňuje automatické generovanie komplexných šturktúrovaných testovacích vstupov imitujúcich reálnu komunikáciu z prostredia moderných informačných systémov. Komplexné dáta, typicky so štruktúrou stromu v súčasnosti predstavujú základný transportačný prostriedok pre prenos informácií medzi uzlami distribuovaných informačných systémov. Automatický generátor ISAGEN vychádza z metodológie *dátami riadeného testovania* a využíva konkrétne dáta z produkčného prostredia ako hlavnú charakteristiku a špecifikáciu pre riadenie generovania nových podobných dát pre testovacie prípady spĺňajúce zadané kombinačné kritéria. Hlavným prínosom tejto práce je obsiahle predloženie techník pre automatizované generovanie dát spoločne s praktickou implementáciou, ktorá demonštruje ich použitie. Vytvorené riešenie umožňuje testerom vytvárať viac relevantné testovacie dáta, ktoré vhodne reprezentujú reálnu komunikáciu z produkčných informačných systémov.

Abstract

ISAGEN is a tool for the automatic generation of structurally complex test inputs that imitate real communication in the context of modern *information systems*. Complex, typically *tree-structured data* currently represents the standard means of transmitting information between nodes in distributed information systems. Automatic generator ISAGEN is founded on the methodology of *data-driven testing* and uses concrete data from the production environment as the primary characteristic and specification that guides the generation of new similar data for test cases satisfying given combinatorial adequacy criteria. The main contribution of this thesis is a comprehensive proposal of automated data generation techniques together with an implementation, which demonstrates their usage. The created solution enables testers to create more relevant testing data, representing production-like communication in information systems.

Kľúčové slová

automatický generátor dát, testovanie založené na dátach, dátami riadené testovanie, informačný systém, stromové dáta, testovanie, automatické testovanie, testovacie dáta, testovacie vstupy, kombinatorické testovanie, t-wise, sémantická analýza, analýza dát, JSON, generovanie dát, syntéza dát, štruktúrované dáta, modelovanie komunikácie, reprodukcia dát, Testos, combine, ts-reporter

Keywords

automatic data generator, data-based testing, data-driven testing, information system, tree data, testing, automatic testing, test data, test input, combinatorial testing, t-wise, semantical analysis, data analysis, JSON, data generation, data synthesis, structured data, communication modeling, data reproduction, Testos, combine, ts-reporter

Citácia

NAŇO, Andrej. *Automatické generování testovacích dat informačních systémů*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Automatické generování testovacích dat informačních systémů

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Aleša Smrčku, Ph.D. Uviedol som všetky literárne pramene, zdroje a publikácie, z ktorých som pri písaní práce čerpal.

.....
Andrej Naňo
18. mája 2021

Poďakovanie

Rád by som sa poďakoval vedúcemu práce, Ing. Alešovi Smrčkovi, Ph.D. za cenné rady, odbornú pomoc, ktorú mi poskytol a ochotu pri konzultáciach.

Obsah

Kapitola 1

Úvod

Testovanie je v súčasnosti kriticky dôležitým procesom pri vývoji kvalitných softvérových projektov, komponentov či celých systémov. Medzi najdôležitejšie požiadavky pre úspešné testovanie akéhokoľvek programu, rozhrania alebo systému patrí použitie kvalitných vstupných *testovacích dát*. Získanie *testovacích dát* a manuálna tvorba testovacích vstupov je však často problematická, nepraktická a časovo náročná. V konečnom dôsledku pokrýva napokon iba úzku podmnožinu možných vstupov do systému alebo iba zlomok funkcionality programu pri testovacom behu. V kontraste k manuálnej tvorbe testovacích dát a testovacích prípadov je možným riešením *automatizácia* celého procesu *syntézy* pomocou *automatického generovania testovacích dát*, ktoré prácu testera výrazne uľahčuje.

Informačné systémy vyžadujú v kontexte testovania špeciálnu pozornosť. Jedná sa o komplexné systémy zložené z viacerých komponentov, umožňujúcich beh procesov, ktoré navzájom v rámci systému komunikujú prostredníctvom komunikačných kanálov na sieťovom rozhraní. Uzly informačného systému typicky využívajú na komunikáciu a výmenu dát na aplikačnej vrstve protokol *HTTP*, často vo formáte určitého komunikačného protokolu. Či už je to *REST* alebo *SOAP*, dáta majú typicky komplexnú a variabilnú štruktúru so sémanticky rozmanitými hodnotami. Pri *testovaní informačných systémov* je často požiadavkou určitým spôsobom slepé *black-box* otestovanie rozhraní bez znalosti vnútorného fungovania procesov v uzloch. Základným predpokladom tejto práce je scenár, v ktorom nie je testerovi umožnené vidieť štruktúru kódu alebo riadenia v procesoch, ale iba záznamy (*logy*) komunikácie medzi komunikačnými uzlami na sieťovom rozhraní. Jediným zdrojom toho, ako by dáta mali vyzerať, sú tak reálne dáta z komunikácie produkčného prostredia informačného systému. Proces *automatického generovania* musí dokázať vhodne zostrojiť také testovacie dáta, ktoré splnia komplexné obmedzenie a *kritéria adekvátnosti* pre *tvar štruktúry*, ale aj zachovanie *sémantických kategórií* pre *atomické hodnoty* uložené v dátových štruktúrach. Výstupné *syntetické dáta* by mali byť na jednej strane čo najviac charakteristicky podobné originálnym réalnym dátam, no na druhej strane, by nemali byť úplne totožné. Jednoducho povedané dáta by mali pôsobiť *realisticky* ale nie úplne reálne.

Cieľom tejto diplomovej práce je navrhnúť a implementovať nástroj pre automatické generovanie testovacích dát v kontexte informačných systémov. Práca systematicky prechádza a navrhuje metódy automatizovaného generovania testovacích dát, ktoré majú podobu komplexných stromových štruktúr. Ďalej je uvažovaný aj rámec komunikácie v informačnom systéme a reprodukcia komunikácie so zameraním na generovanie správ. Pri tvorbe kom-

plexných dát, typicky so stromovou štruktúrou je vo veľkej miere kladený dôraz na splnenie *kritérií adekvátnosti*, *kombinačných kritérií* ale aj *náhodnosti*.

Výsledným výstupom práce bude softvérový projekt ISGAEN, ktorý zvolené metódy generovania, ale aj *analýzy* a *abstrakcie stromových štruktúr* aplikuje do praktického a použiteľného kontextu. Nástroj bude poskytovať možnosti *automatického generovania dát*, kde vstupom pre nástroj sú *reálne dáta z komunikácie* uzlov informačného systému, ktoré nástroj transformuje na sadu nových umelých dát, ktoré sú do čo najväčšej miery *sémanticky* a *štrukturálne* podobné pôvodným reálnym dátam.

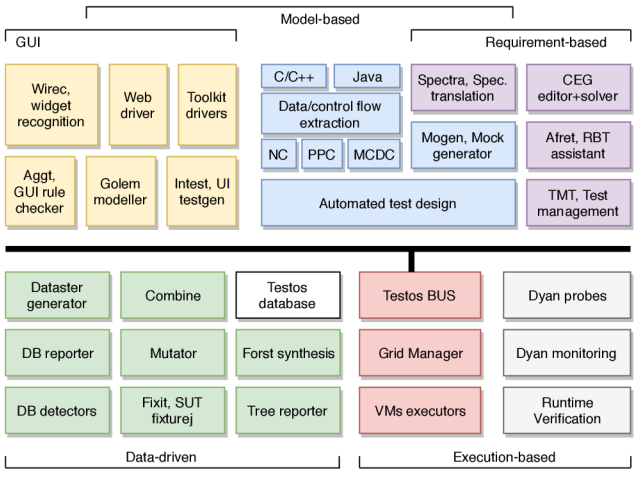
Zvyšok úvodnej kapitoly je členený nasledovne. Sekcia 1.1 neformálne definuje hlavné dôvody pre začatie práce a stanovuje základné ciele pre jej dokončenie spolu s očakávanými výzvami pri jej riešení a na záver prínosy plynúce z výsledku práce. Na záver kapitoly v sekcii 1.3 je poskytnutý organizačný prehľad o témach rozoberaných v zvyšných kapitolách tejto diplomovej práce.

1.1 Motivácie, ciele a výzvy práce

Táto sekcia slúži ako základný motivačný prehľad o hlavných dôvodoch, preberaných témach a očakávaných problémoch, ktorými sa práca bude zaoberať.

Motivácia Hlavným dôvodom prečo generovať testovacie dáta je prosté uľahčenie procesu testovania a manuálnej tvorby dát. Na druhej strane však vďaka automatizácii celého procesu získava tester istým spôsobom väčšiu moc a schopnosti nad tým, ako tieto testovacie dáta vytvárať, kombinovať alebo modifikovať. Ďalším dôvodom, pre využitie automatických generátorov dát je snaha o zrkadlenie určitého reálneho produkčného systému, ktorý je v záujme testovať, simulovať alebo len sledovať a analyzovať jeho činnosť.

Platforma Testos [42] Hlavným kontextom pre vývoj nástroja ISAGEN je platforma *Testos* (Test Tool Set), ktorá sa prevažne zaoberá *automatizovaním testovania softvéru*. Nástroje a projekty v tejto platforme sa zameriavajú na široké spektrum oblastí z prostredia testovania, medzi ktoré patrí aj stratégia *testovania riadeného dátami* (data-driven testing). Problematikou *automatickej analýzy a syntézy štruktúrovaných (stromových) dát* sa už v predchádzajúcich rokoch zaoberali aj viaceré iné projekty platformy *Testos*. Medzi najdôležitejšie súvisiace projekty patria: TS-REPORTER [34], TREAPER [45], GESTR [36], COMBINE [44], DBGENX [23], DATASTER [33] a S-DETECTORS [35]. Implementačný výstup tejto práce na vyššie spomenuté projekty vo veľkej miere nadväzuje, niektoré časti existujúcich projektov priamo využíva a iné slúžia ako silný poznatkový základ pri návrhu nového, sofistikovanejšieho nástroja. Motiváciou tejto práce je tak aj zjednotiť kolektívnu snahu o vytvorenie uceleného riešenia. Autor tejto diplomovej práce je aj súčasne autorom projektu DATASTER[1] (vytvoreného v rámci bakalárskej práce v roku 2019 [33]), ktorého náplňou bolo vytvorenie infraštruktúry a užívateľského rozhrania pre generátor DBGENX. Projekt tak zdieľa s touto prácou podobnú kategóriu použitia, taktiež ide o generátor dát, avšak charakteristika dát, použité techniky a metódy pre generovanie a takisto technológie sa medzi projektami výrazne odlišujú.



Obr. 1.1: Nástroje platformy *Testos (Test Tool Set)*. Prevzaté z [http://testos.org/].

[1] Webová aplikácia je dostupná na https://dataster.testos.org.

Konečné ciele práce Predtým, než sa práca dostane ku špecifickým aspektom problematiky generovania testovacích dát, je nápomocné stanoviť celkové a čiastkové ciele, ktoré tak tvoria základné myšlienkové línie pre nadväzujúce kapitoly. Nasledujúci zoznam popisuje najpodstatnejšie ciele, o ktoré sa práca usiluje:

(1) **Vyhodnotenie** a prehľad o štandardne používaných metódach a technikách *automatického generovania testovacích dát* vzhľadom k zvolenému *kritériu adekvátnosti* v oblasti dátami riadeného testovania.

(2) **Zjednotenie** kolektívnej snahy prác a projektov z platformy *Testos* o vytvorenie uceleného riešenia pre automatizáciu analýzy a generovania štruktúrovaných dát.

(2) **Návrh** systematického prístupu, konkrétnych stratégií a techník pre generovanie dát so sémantickým príznakom *správ v informačnom systéme.*

(3) **Vytvorenie** prakticky použiteľného *nástroja* na spracovanie a analýzu štruktúrovaných dát a najmä *automatické generovanie testovacích dát.*

(4) **Overenie** vytvoreného riešenia z pohľadu splnenia navrhnutej špecifikácie, efektívnosti a výpočetnej zložitosti, a v konečnom dôsledku aj použiteľnosti v praxi.

Neformálne by sa teda dalo zhodnotiť, že finálnym výstupným cieľom tejto diplomovej práce je vytvorenie implementácie pre nástroj, ktorý by dokázal demonštrovať navrhnuté techniky a stratégie automatizovaného generovania testovacích dát. K tomu, aby bolo možné takýto nástroj vhodne navrhnúť a následne implementovať, je však vopred potrebné vhodne definovať samotnú problematiku generovania dát a s tým spojené výzvy.

Výzvy pri práci na projekte Vyššie popísané ciele so sebou prinášajú istú mieru nejednoznačnosti a otvárajú tak priestor pre možné problémy pri tvorbe riešenia. Pred začiatkom práce bolo identifikovaných niekoľko najvýznamnejších problémov, ktoré bude práca v nasledujúcich kapitolách musieť relevantne riešiť:

- **Problematika generovania štruktúrovaných dát** – Generovanie dát sa na prvý pohľad javí ako veľmi abstraktný koncept. Prináša potrebu zodpovedať viaceré čiastkové podproblémy ako: (1) Čo si predstaviť pod štruktúrovanými dátami? (2) Aké sú kritéria a obmedzenia pre tvorbu nových dát? (3) Koľko možných (a pre testovanie potrebných) variant umelých dát je možné (a nutné) vygenerovať?
- **Efektivita generovania dát** – Výpočetná zložitosť sa pravdepodobne výrazne odlišuje podľa použitej metódy generovania, typu a štruktúry dát. Je preto potrebné zistiť, ktoré metódy generovania sú z pohľadu zložitosti efektívne a ktoré metódy a techniky sú vhodné pre prípad tvorby štruktúrovaných dát v komunikácii informačného systému.
- **Reprodukcia komunikácie informačného systému** – Ak má byť imitovaná komunikácia informačného systému do čo najväčšej miery, mal by byť pravdepodobne napodobnený aj časový rozmer prenosu správ v systéme. Prvým možným prístupom je tieto časové momenty jednoducho replikovať alebo navrhnúť metódu pre napodobnenie časového rozloženia prenosu správ s určitou mierou náhodnej odchýlky.

- **Zachovanie podobnosti voči reálnym dátam** – Tvorba podobných dát je na prvý pohľad ťažko prístupná myšlienka. Čo znamená, aby dátové štruktúry mali *podobnú* štruktúru a sémantiku? Ako možno vyjadriť, že štruktúry majú *rovnakú* štruktúru a sémantiku? Preto bude potrebné vhodne definovať podobnosť a zabezpečiť, aby ju nástroj pri generovaní dodržiaval.
- **Kritéria pre generovanie** – Vzhľadom na zamýšľané využitie pre účely testovania, je už teraz možné predpokladať určité stanovené kritéria adekvátnosti pri testovaní. Aké kritéria sa však môžu vzťahovať na generovanie testovacích dát? Čo ak medzi požiadavky testovania patrí aj kombinačné kritérium? Mali by sa testovacie dáta pri generovaní systematicky kombinovať?
- **Integrácia kolektívnej snahy projektov Testos** – Keďže sa v minulosti zaoberali podobnou problematikou už aj iné projekty platformy Testos, bude potrebné vyhodnotiť, ktoré z nástrojov je možné aktívne využiť a ktoré nie. Prípadne taktiež môžu slúžiť ako počiatočná inšpirácia pri návrhu nového riešenia. Hlavná otázka však je, ako zjednotiť viacero nezávislých projektov do jedného použiteľného celku.

1.2 Dosiahnuté výsledky a prínosy práce

Výsledkom tejto práce sú nasledujúce prínosy do problematiky automatického generovania testovacích dát:

- **Uvedenie do problematiky a súhrnný prehľad o metódach generovania testovacích dát:** Práca ponúka teoretické podklady pre pochopenie problematiky a motivácie generovať testovacie dáta automatizovane oproti pracnej a časovo náročnej manuálnej tvorbe dát. Ďalej taktiež poskytuje širokospektrálny prehľad o rôznych prístupoch a zaužívaných stratégiach generovania dát. Súčasne sú v práci uvedené niektoré vybrané súvisiace projekty a existujúce nástroje podobného zamerania.
- **Formálna definícia problému generovania dát:** Práca formalizuje problém generovania ako výpočetný problém a uvádza jeho vlastnosti, výpočetnú zložitosť a možné algoritmické prístupy k riešeniu daného problému.
- **Analýza komunikácie informačného systému:** Práca popisuje prostredie informačného systému ako zdroj zmysluplných štruktúrovaných dát. Tieto dáta sú typicky prenášané pomocou správ v rámci sieťovej komunikácie a pre efektívne spracovanie je potrebné dáta zo záznamov komunikácie (*communication logs*) extrahovať a abstrahovať do vhodne definovaného modelu komunikácie (*communication model*).
- **Zjednotená abstrakcia pre serializované formáty dát:** Medzi hlavné dátové objekty, s ktorými sa v práci pracuje, patria serializované formáty štruktúrovaných dát ako `JSON`, `XML` a podobne. Pre ich zjednotenie je zavedený formát abstraktného stromu (*Abstract Tree*), ktorý vhodne vystihuje stromovú štruktúru, nezávisle od zvoleného serializovaného formátu.
- **Štrukturálna a sémantická analýza stromových dát:** Súčasťou vytvoreného riešenia je aktívne využitie nástrojovej sady platformy *Testos*. Ide o nástroje *ts-reporter*, *s-detectors*, a *combine*, ktoré sú efektívne zapojené do vnútorného procesu nástroja

ISAGEN. Vďaka týmto nástrojom je možné rozsiahlo sémanticky a štrukturálne analyzovať vstupné dáta a generovať dáta s presnejšiou mierou podobnosti.

- **Syntéza stromových štruktúr:** Výstupom práce je nástroj ISAGEN, ktorého hlavnou funkciou je syntéza stromových štruktúr podľa špecifických obmedzení definovaných charakteristikou abstraktného stromu. Nástroj je tak schopný replikovať alebo reprodukovať podobné stromové dáta iba na základe vzorky reálnych dát.
- **Generovanie T-wise kombinácií:** Nástroj dokáže vzhľadom na zadaný celočíselný parameter T splniť kombinačné kritérium pre všetky nejasné sémantické vlastnosti alebo variantné cesty v *abstraktnom strome*. Je tak možné generovať dáta, ktoré efektívne kombinujú všetky varianty, ktoré z pohľadu analýzy nebolo možné deterministicky označiť.
- **Generovanie náhodných atomických hodnôt:** Medzi sekundárne schopnosti generátora ISAGEN patrí generovanie typovo podobných atomických hodnôt uložených v stromových štruktúrach. Nástroj poskytuje sadu čiastkových generátorov, zameraných na tvorbu umelých dát spadajúcich do určitej typovej domény.
- **Generovanie sémanticky podobných atomických hodnôt:** Generátor sa primárne snaží dodržať zachovanie podobnosti voči reálnym dátam. Vďaka dôkladnej analýze sémantických vlastností dát je potom generátor schopný vytvoriť nové umelé dáta s rovnakou sémantikou.
- **Generovanie správ v podobnom časovom usporiadaní:** Generátor na vyššej úrovni generuje celé postupnosti správ, imitujúc tak komunikáciu reálneho informačného systému. Súčasťou tejto aktivity je napodobnenie pravdepodobnostného rozloženia výskytu správ a ich vzájomných časových rozostupov.
- **Využitie pri testovaní:** Vytvorený nástroj berie do úvahy praktické využitie pri testovaní, a tak súčasťou každej žiadosti o generovanie dát je špecifikácia testovacieho kritéria, ktoré by sa malo zohľadniť. Dáta je tak možné generovať jednorázovo, náhodne alebo vyčerpávajúcim prístupom vygenerovať všetky varianty. Umožnené je taktiež zadefinovať menej striktné kritéria a generovať napríklad len všetky kombinácie kategorických párov variabilných zložiek dát.
- **Vyhodnotenie výsledného riešenia:** Práca v závere sebakriticky hodnotí vytvorené riešenie a poskytuje možné návrhy pre zlepšenie efektivity alebo možných zlepšení stratégie generovania.

1.3 Organizácia

Zvyšok tejto diplomovej práce je tématicky členený na osem častí, ktoré na seba plynule nadväzujú, od načrtnutia základných teoretických podkladov, cez definovanie špecifických problémov, až po navrhnutie, implementovanie a vyhodnotenie praktického riešenia. Každá kapitola má svoj osobitný a odlišný význam, no ako celok vytvárajú postupnú dokumentáciu postupu pre pochopenie problematiky generovania štruktúrovaných dát a vytvorenie softvérového riešenia, ktoré ho realizuje. Kapitola 2 obsahuje úvodný teoretický materiál poskytujúci súhrnný prehľad o automatickom generovaní testovacích dát, od uvedenia kontextu dátami riadeného testovania softvéru, automatizácie testovania, až po konkrétne metódy syntézy dát a príklady existujúcich populárnych moderných nástrojov alebo výskumných projektov, ktoré riešia podobnú problematiku. Kapitola 3 špecifikuje konkrétne aspekty prostredia a podobu dát, ktoré sú záujmom práve tejto práce. Kapitola predstavuje informačné systémy ako centrálne prostredie a zdroj vzorových dát, serializačný formát `JSON` ako primárny formát predpokladanej dátovej štruktúry a uvádza abstraktný dátový strom ako zvolenú abstrakciu pre prácu s dátami. Kapitola 4 využíva získané poznatky v predchádzajúcich dvoch teoreticky orientovaných kapitolách na vytvorenie vhodnej formálnej definície samotného problému generovania, od syntézy stromových štruktúr až po splniteľnosť obmedzení pre atomické hodnoty či časové usporiadanie pri generovaní správ ako abstrakcie pre moment prenosu dátovej informácie v informačnom systéme. Kapitola 5 analyzuje a špecifikuje požiadavky pre softvérový projekt ISAGEN, na základe čoho sú pre projekt navrhnuté základné koncepty, abstrakcie, stratégie a algoritmy pre generovanie a na záver kapitoly aj celý komplexný návrh architektúry a popis činnosti komponentov nástroja. Kapitola 6 dopĺňuje a reaguje na predchádzajúcu kapitolu realizáciou návrhu pre nástroj. Popisuje fázu praktického vývoja a s ním spojené významné problémy, ktoré bolo potrebné riešiť. V kapitole je taktiež popísaná technická charakteristika softvérového projektu, použité technológie a prehľad o najdôležitejších komponentoch zdrojového kódu. Kapitola taktiež popisuje proces testovania vytvoreného riešenia, použité typy testov a meraní. Na záver je poskytnuté aj vyhodnotenie testovania, a teda aj výslednej kvality vytvoreného riešenia. Kapitola 7 uzatvára prácu zhrnutím dosiahnutých výsledkov a zhodnotením finálneho stavu projektu ISAGEN. V závere kapitoly a aj celej práce sú obsiahnuté vízie o prípadných možnostiach budúceho rozšírenia projektu a ďalšieho nadväzujúceho vývoja.

Kapitola 2

Automatické generovanie testovacích dát

Cieľom tejto kapitoly je poskytnúť základné teoretické poznatky pre uvedenie čitateľa do kontextu problematiky tejto práce – *automatizácie generovania testovacích dát*. Začiatok kapitoly sa zaoberá testovaním a automatizáciou testovania vo všeobecnosti. Nasledujúca sekcia venuje špeciálnu pozornosť dátami riadenému testovaniu, jeho hlavným zložkám a aktivitám, ktoré je možné v rámci procesu testovania automatizovať. Záver kapitoly je úzko zameraný na problematiku generovania dát, prezentuje výber najčastejšie používaných prístupov ku generovaniu a existujúcich riešení v tejto oblasti.

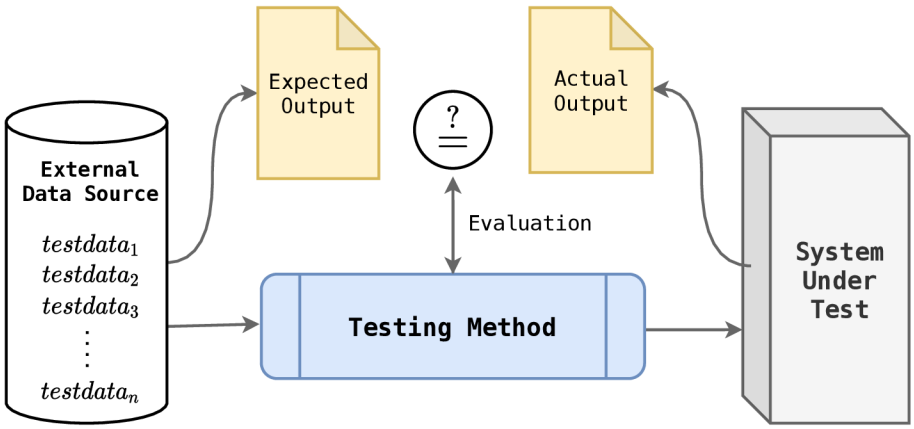
2.1 Testovanie riadené dátami

Zmyslom tejto sekcie je poskytnúť prehľad o základných princípoch testovania a priblížiť najzásadnejšie koncepty a pojmy používané v kontexte dátami riadeného testovania ako hlavného kategorického rámca tejto práce.

Testovanie softvéru [17, 31, 12, 6] Súčasťou každého netriviálneho softvérového projektu je aj potenciálna a vysoko pravdepodobná existencia chýb v softvéri. Zavedenie chýb môže mať rôzne príčiny, od nevedomého pochybenia programátorom pri vývoji programového kódu, cez nejasne formulované alebo nesprávne pochopené požiadavky v špecifikácii programu. Nezvratný je ale fakt, že chyby v softvéri s veľkou pravedpodobnosťou budú aj po veľmi ostražitom a dôslednom vývoji a ich úplné vylúčenie nie je možné. Odpoveďou a intuitívnym riešením na tento na prvý pohľad neriešiteľný problém, je snaha o minimalizáciu výskytu chýb v podobe testovania. *Testovanie softvéru* je možné popísať ako proces spúšťania programu s cieľom odhalenia chýb. Ide o veľmi dôležitý aspekt pri vývoji softvéru, ak je treba zabezpečiť určitú mieru spolahlivosti. *Spolahlivosť softvéru* je možné definovať ako pravdepodobnosť toho, že softvér nespôsobí zlyhanie alebo chybu vo vymedzenom čase pri špecifikovaných podmienkach spustenia. Základnú jednotku v procese testovania tvoria testovacie prípady. *Testovací prípad* (Test case) je špecifická množina vstupných testovacích dát spolu s očakávanými výsledkami a určitým testovacím cieľom, ako napríklad vykonanie určitej vlastnosti alebo funkcionality programu, alebo overenie splnenia špecifickej požiadavky. Jednoducho sa dá povedať, že ide o kombináciu vstupných hodnôt pre spúštaný

program, ktorých snahou je overiť konkrétne správanie testovaného systému. *Testovacia sada* (Test set) je množina testovacích prípadov zoskupených za účelom testovania toho istého programu. Najdôležitejšími zložkami testovacích prípadov sú vhodne zvolené dáta. Tieto dáta sa nazývajú *testovacie dáta*, a je ich možné chápať ako konkrétne vstupy a obmedzenia súvisiace s určitým testovacím prípadom. Snahou testera je zvyčajne výber takých testovacích dát, ktoré by priichprocese testovania maximalizovali množstvo odhalených unikátných chýb. Konkrétna podoba testovacích dát sa odlišuje podľa prostredia, testovaného programu a jeho rozhrania. Dáta môžu mať komplexnú štruktúru, zloženú z viacerých úrovní a typov alebo predstavovať iba jednoduché atomické hodnoty. Dáta taktiež možno klasifikovať podľa ich sémantiky alebo kombinácie zastúpených typov.

Testovanie riadené dátami [31, 41] Dátami riadené testovanie alebo taktiež testovanie založené na dátach je prístup k testovaniu, pri ktorom jednotlivé testovacie prípady využívajú externé dátové zdroje, nad ktorými sa opakovane vykonávajú testovacie behy. Dôležitou vlastnosťou tohto prístupu je, že jeden test má typicky viacero množín hodnôt testovacích dát. Hlavnou výhodou je možnosť testovacie dáta spravovať nezávisle od testovacieho prípadu a jeho vnútornej logiky. Tvorba dát tak môže byť delegovaná na iné procesy a nie je nutné, aby spolu logika a dáta testovacieho prípadu úzko súviseli. Dátami riadené testovanie je často spájané s black-box testingom a spolu predstavujú dôležitú stratégiu testovania, pri ktorej je program vnímaný ako čierna skrinka, a vnútorná funkcionalita a štruktúra programu je pre testovanie irelevantná. Namiesto toho je kladený dôraz na objavovanie okolností, pri ktorých sa program správa odlišne od jeho špecifikácie. Testovacie dáta nie je teda možné odvodiť od vnútornej štruktúry programu, je však možné na tento účel využiť špecifikáciu programu alebo sa snažiť napodobniť vzorku reálnych dát. Práve imitácia reálnych dát je hlavným zvoleným prístupom uvažovaným v tejto práci.



Obr. 2.1: Konceptuálne znázornenie dátami riadeného testovania.

Prístupy a techniky testovania [31] Testovanie je veľmi rozsiahle odvetvie a je možné ho dodatočne rozdeliť na množstvo špecifických disciplín s odlišnými prístupmi a technikami. Techniky testovania je možné rozdeliť podľa zamerania, stratégie alebo metód, ktoré sa pri tomto procese používajú. Celková stratégia testovania sa typicky rozdeľuje na dve intuitívne kategórie podľa toho, či testovanie prebieha so znalosťami o vnútornom fungovaní programu, alebo bez nich.

- **Black-box testing** – Prístup, pri ktorom je testovaný systém vnímaný výhradne z externej perspektívy a bez znalosti vnútorného zdrojového kódu programu. Testuje sa funkcionalita programu vzhľadom k jeho špecifikácii, interné správanie programu a štruktúra kódu nie je pre testera známa. S *black-box testovaním* je prirodzene spájaný prístup *dátami riadeného testovania*, kde v procese testovania hrajú centrálnu rolu testovacie dáta (*test data*). Nevýhodou je realita toho, že pre zaručené odhalenie všetkých chýb v programe je potrebné splnenie tzv. vyčerpávajúceho testovania vstupov (*exhaustive input testing*), ktoré vyžaduje vytvorenie testovacieho prípadu pre každý možný vstup programu. Takéto kritérium nie je pre prevažnú väčšinu programov možné striktne dodržať a v praxi sa tým pádom typicky na miesto toho stáva hlavným cieľm maximalizovať množstvo odhalených chýb konečným množstvom testovacích prípadov. Z toho plynie, že dôležitú úlohu pri *black-box testovaní* zohráva najmä správna voľba stratégie pre tvorbu testovacích prípadov.
- **White-box testing** – Alternatívny prístup, pri ktorom je testerovi umožnené analyzovať vnútornú logiku a štruktúru zdrojového kódu programu. Testovacie dáta je možné odvodiť a vytvoriť práve na základe vnútornej štruktúry programu, a na rozdiel od vyčerpávajúceho testovania vstupov typického pre black-box testovanie, je pri white-box testovaní obdobným kritériom tzv. vyčerpávajúce testovanie ciest programu (*exhaustive path testing*), pri ktorom je snahou testovania pokryť všetky možné cesty riadenia pri spustení testovaného programu. Podobne ako pri kritériu pokrytia všetkých vstupov pri black-box testovaní, je takéto kritérium vo väčšine prípadov nemožné dodržať, keďže množstvo teoreticky dosiahnuteľných ciest v komplexných programoch je neprakticky masívne. Nevýhodou taktiež je, že aj kompletné testovanie programu, pri ktorom by sa pokryli všetky možné cesty riadenia, nemusí byť postačujúce pre odhalenie chybného správania programu, ak sa neberie do úvahy aj jeho špecifikácia.

2.2 Kritéria testovacích stratégií

Keďže vyčerpávajúce prístupy k testovaniu v obidvoch prípadoch (black-box a white-box) nie sú prakticky realizovateľné, bude pre kvalitné testovanie potrebné stanoviť alternatívnu mieru adekvátnosti testov. Medzi základné otázky v oblasti testovania patrí práve určenie toho, čo sa dá považovať za dostatočne adekvátne.

Kritérium adekvátnosti [47] Kritérium adekvátnosti v kontexte softvérového testovania je možné chápať v dvoch rozdielnych, no úzko súvisiacich líniach. V prvom rade je možné *kritérium adekvátnosti* (def. 2.2.1) definovať ako určité *pravidlo pre zastavenie* procesu testovania a rozhodujúce vyhodnotenie toho, či prebehlo dostatočné množstvo testov. Druhá, určitým spôsobom doplňujúca definícia (def. 2.2.2) *kritéria adekvátnosti*, je z pohľadu *metriky*, kde adekvátnosť predstavuje kvantifikovateľnú (často percentuálnu) mieru priradenú k testovacej sade. Na rozdiel od prvej definície je tak kvalita testovacej sady popísaná sofistikovanejším spôsobom.

Definícia 2.2.1 (**Kritérium adekvátnosti (zastavovacie pravidlo)** [47])**.** *Kritérium adekvátnosti* C je funkcia:

$$C : P \times S \times T \longrightarrow \{true, false\}$$

kde p predstavuje program, s predstavuje špecifikáciu pre softvér a t testovaciu sadu. Mapovanie $C(p, s, t) = true$ znamená, že sada t je adekvátna pre testovanie programu p voči špecifikácii s podľa kritéria C. V opačnom prípade je sada t neadekvátna.

Definícia 2.2.2 (**Kritérium adekvátnosti (metrika)** [47])**.** Kritérium adekvátnosti C je funkcia:

$$C : P \times S \times T \longrightarrow [0, 1]$$

kde p predstavuje program, s predstavuje špecifikáciu softvéru a t testovaciu sadu. Mapovanie $C(p, s, t) = r$ znamená, že adekvátnosť testovania programu p pomocou sady t vzhľadom k špecifikácii s je stupňa (na úrovni) r podľa kritéria C. Čím väčšie je reálne číslo r, tým viac je testovanie adekvátne.

Tieto definície kritéria adekvátnosti tvoria významný základ pre každú metódu a stratégiu testovania. Najdôležitejšiou vlastnosťou kritéria adekvátnosti je, že priamo vytvára špecifikáciu požiadaviek pre testovanie a tak rozhoduje, ktoré testovacie prípady vyhovujú vyplývajúcim požiadavkám. Kritéria adekvátnosti je možné klasifikovať na dve hlavné kategórie podľa zdroja informácií použitých na ich vytvorenie:

1. *Kritéria založené na špecifikácii* vyjadrujú pokrytie funkcionality vychádzajúcej zo špecifikácie požiadaviek programu.
2. *Kritéria založené na programe* popisujú pokrytie vzhľadom k rozsiahlosti vykonania vnútorných komponentov programu, ako sú výrazy, vetvy alebo cesty vnútorného riadenia v programe.

Okrem toho existuje aj tzv. *kombinované kritérium*, ktoré využíva súčasne špecifikáciu a aj samotný program. Pre kontext tejto práce je pravdepodobne najbližším typom kritéria tzv. *kritérium založené na rozhraní* (interface-based criteria), ktoré špecifikuje požiadavky na testovanie výhradne na základe typu a rozsahu vstupných dát programu bez špecifikácie alebo znalosti o vnútornom fungovaní programu [47].

Kritéria pokrytia [47, 12] Jednoduchým a intuitívnym riešením je zavedenie metriky určujúcej, aké množstvo možného správania programu bolo testami pokryté. Kvantifikovať pokrytie správania programu nie je však jednoduché, a preto boli do súčasnosti predstavené mnohé iné *kritéria pokrytia* (*coverage criteria*). Ich hlavným zmyslom je zodpovedať otázku adekvátnosti testov a poskytnúť jednoznačnú kvantifikovateľnú metriku, ktorá popisuje, či testovanie vzhľadom ku kritériu bolo dostatočné, a pokiaľ áno, tak či je možné testovanie v danom momente ukončiť a vyhodnotiť. Využitie kritérií pokrytia môže navyše súčasne slúžiť ako kritérium pri tvorbe testov a usmerniť testera pri ich návrhu. Príkladmi typicky používaných kritérií pre pokrytie sú:

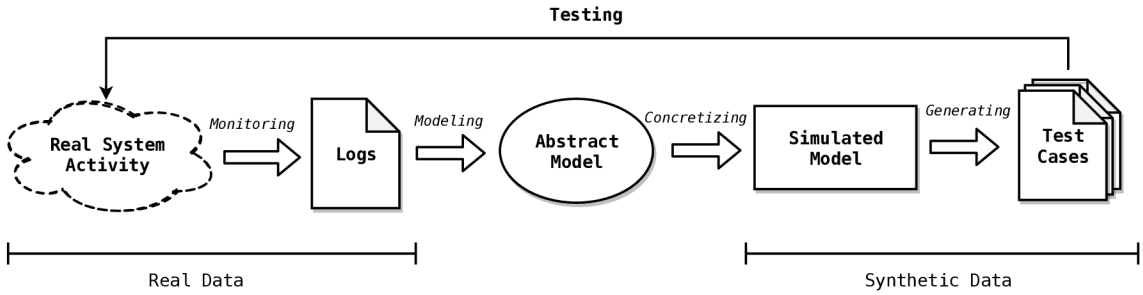
- Pokrytie výrazov (statement coverage)
- Pokrytie vetiev (branch coverage)
- Pokrytie podmienok (condition coverage)
- Pokrytie riadkov kódu (line coverage)
- Pokrytie ciest riadenia (path coverage)

Automatizácia testovania

Alternatívne sa *testovanie* dá chápať aj ako sekvencia interakcií s programom popretkávaných s vyhodnoteniami oproti očakávaným hodnotám [3]. Takáto definícia mierne navádza na automatizáciu celého procesu. Dobré a kvalitné testovanie však vyžaduje interaktívny kognitívny proces. Je pravda, že niektoré kroky pri testovaní sa často opakujú, a je teda rozumné ich v prípade nadmerného opakovania automatizovať. Na druhej strane sú mnohé interakcie a problémy zložité, nejasné a časovo premenné. Automatizáciu je najlepšie aplikovať len na úzku časť testovania, ktoré je zbytočne redundantné, no nie na celý proces. *Manuálne testovanie* je proces, ktorý sa dokáže jednoducho adaptovať a zvládnuť aj komplexné aspekty testovania [3]. Najväčšou výhodou manuálneho testovania oproti automatickému je neobmedzená variabilita pri tvorbe testovacích prípadov. Pojem *automatizácia testovania* je sám o sebe nejednoznačný a môže v kontexte testovania pomenovávať viac čiastkových aktivít. Je preto potrebné špecifikovať o automatizáciu akej aktivity konkrétne ide. Ako bolo spomenuté vyššie, kompletná automatizácia nie je nutne želaným cieľom, no čiastková môže byť pre prácu testera užitočná. Čiastkové úlohy pri procese automatického testovania je možné rozdeliť nasledovne:

- **Generovanie testovacieho prípadu.**
- **Generovanie testovacích dát.**
- **Spustenie testovacieho behu.**

V zvyšku tejto práce bude hlavným predmetom záujmu práve *generovanie testovacích dát* ako samostatná čiastková úloha pri testovaní. Automatická tvorba testovacích prípadov a ich prípadné automatické spúšťanie nie je záujmom tejto práce a predpokladá sa buď manuálne alebo iným nástrojom vykonávané automatické testovanie. Pre lepšie pochopenie celkového kontextu a zmyslu automatizácie generovania testovacích dát spoločne s doplnkovými aktivitami pri testovaní môže poslúžiť ilustrácia procesu na obrázku 2.2.



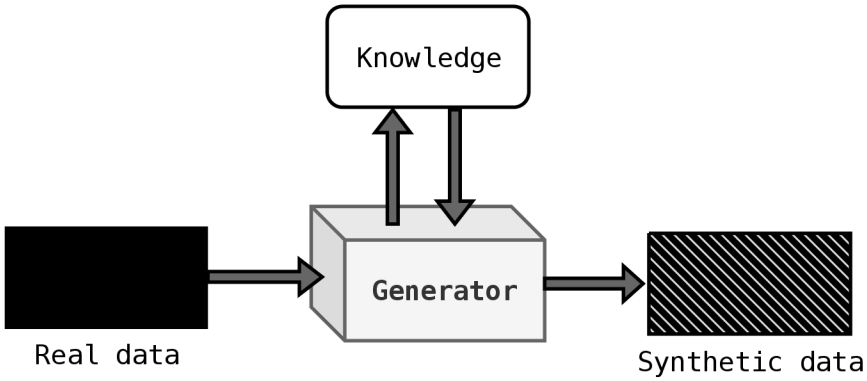
Obr. 2.2: Proces *automatického generovania testovacích dát* na základe vzorových vstupných dát testovaného systému a ich následné využitie pri testovaní toho istého systému.

2.3 Metódy generovania testovacích dát

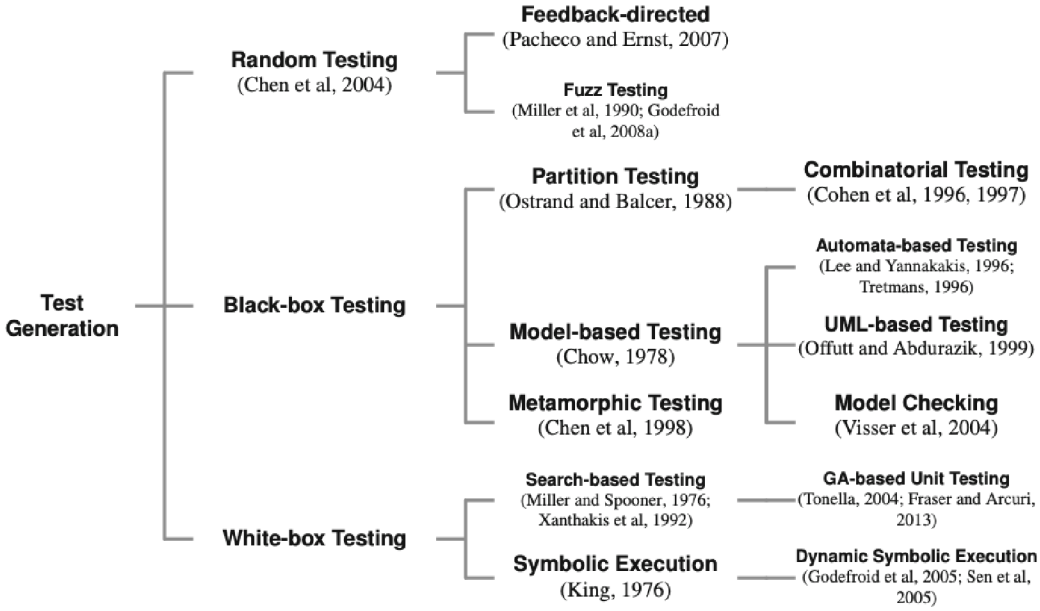
Medzi najobtiažnejšie problémy testovacieho procesu patrí tvorba testovacích dát, ktoré súčasne splnia vybrané testovacie kritéria. Manuálna tvorba testovacích dát je časovo náročná a pracná aktivita, ktorú, ak je to možné, je užitočné automatizovať. Rozsiahlu automatizáciu tohto procesu umožňujú takzvané *generátory testovacích dát.*

Generátor testovacích dát [13, 22, 11] *Generátor testovacích dát* je nástroj, ktorého zmyslom je generovať dáta a pomáhať pri tvorbe testovacích dát. Z pohľadu automatizácie je najviac hodnotné automatizovať proces generovania dát, keďže táto činnosť môže byť pracná a časovo náročná. Generátory dát je možné klasifikovať ako systémy schopné produkovať umelé dátové sady typicky prostredníctvom dvojfázového procesu, kde prvým krokom je vytvorenie modelu pre dáta a druhým krokom je syntéza umelých dát. Predpokladom pre vytvorenie generátora dát je špecifikácia toho, aké dáta je potrebné generovať. Generátor by mal taktiež prihliadať na určité zvolené kritéria, podľa ktorých by mali byť dáta vytvorené. Tieto kritéria sa odlišujú podľa toho, aká testovacia stratégia bola pri testovaní vybraná, alebo aké sú požiadavky na testovanie.

Generátory dát je možné rozdeliť napríklad aj podľa úrovne intervencie od užívateľa (*plne automatické*, *polo-automatické* alebo *manuálne*) alebo typu jadra pre generovanie (*algoritmické*, *matematické SAT/SMT solvery*, *simulátory*, atď.).



Obr. 2.3: Konceptuálne zobrazenie funkcie generátora dát.



Obr. 2.4: Hierarchické rozdelenie typov generovania testov [Prevzaté z [12]].

Techniky generovania dát

Konkrétne metódy a techniky pre generovanie testovacích dát a testovacích prípadov sa odlišujú podľa toho, aká je celková stratégia a možnosti testovania. Pri tvorbe testovacích dát a testovacích prípadov je možné využiť informácie o programe počas jeho behu, zdrojový kód programu, špecifikáciu a popis fungovania alebo charakteristiku vstupných a výstupných dát, prípadne kombináciu všetkých spomenutých informácií o programe [2]. Medzi základné techniky patria:

- **Naivné generovanie** [29] – Generovanie založené na priamom, no naivnom a vyčerpávavom vygenerovaní všetkých hodnôt stavového priestoru a potom postupné alebo náhodne vyberanie z nich. Táto metóda je však veľmi neefektívna z pohľadu výpočetnej zložitosti problému generovania. Dá sa povedať, že je vo všeobecnosti pri dynamických štruktúrach minimálne *NP-ťažký*.
- **Náhodné testovanie (Random Testing)** [39, 6] Základnou, a určitým spôsobom na prvý pohľad naivnou technikou, je generovanie dát náhodne ako súčasť náhodného testovania, pri ktorom je snahou generovať všetky vstupy náhodne, nehľadiac na vnútornú štruktúru programu, kombinačné vlastnosti vstupov alebo inú charakteristiku vstupov, ktorá by generovanie riadila. Keďže sú ale dáta generované náhodne a nie systematicky, prirodzene tak vznikne množstvo zbytočne opakovaných variant, čím sa táto technika stáva neefektívnou z pohľadu optimálnosti. Na druhej strane existujú rozšírenia a varianty tejto techniky, ako je napr. *adaptívne náhodne generovanie*, ktoré oproti rovnomernému rozloženiu pravdepodobnosti využívanému pri klasickom náhodnom testovaní, upravuje výber z náhodných hodnôt tak, že vždy po vykonaní

jedného náhodného testovacieho prípadu vyberie ďalší testovací prípad podľa najviac "vzdialených" dát prípadu z množiny tzv. kandidátnych testovacích prípadov. Vzdialnosťou hodnôt je v základnej podobe myslená Euklidovská vzdialenosť medzi pozíciami vstupov všetkých vstupných domén. Viac informácií o tomto prístupe testovania poskytuje publikácia [6]. Náhodné generovanie testovacích dát je veľmi jednoduchý a ľahko realizovateľný prístup, ktorý má napriek svojej neefektivite značné využitie. Avšak ak je cieľom pri testovaní dosiahnuť vytvorenie čo najoptimálnejšej testovacej sady, náhodné testovanie je z tohto hľadiska zlým výberom.

- **Symbolické spustenie (Symbolic Execution)** [2, 20] – White-box technika symbolického spustenia je založená na myšlienke nahradenia konkrétnych vstupných dát programu "symbolickými hodnotami", a pri vykonávaní programového kódu sa postupne akumulujú obmedzenia nad týmito symbolickými hodnotami v booleovských formulách označujúcich obmedzenia pre danú cestu v programe. Pri každom rozvetvení podľa podmieneného výrazu v programe sa vytvorí nová cesta v strome symbolického spustenia s iným predpisom pre obmedzenia danej cesty v programe. Technika symbolického spustenia je tak veľmi úzko spojená s kritériami pre pokrytie zrojového kódu. Pre získanie konkrétnych hodnôt, ktoré by pokryli danú časť programu, je potrebné vyriešiť formulu obmedzení nad cestou v programe. Práve to však robí z tejto techniky výpočetne náročnú úlohu, najmä ak sa jedná o program so štrukturálne komplexnými vstupmi.

- **Testovanie založené na vyhľadávaní (Search-based testing)** [2] – Techniky založené na vyhľadávaní využívajú optimalizačné a vyhľadávacie algoritmy pre vyhľadanie takých testovacích dát, ktoré maximulizujú dosiahnutie testovacích požiadaviek a zároveň minimalizujú náročnosť a cenu pri ich tvorbe a celkovo testovaní. Proces generovania testovacích dát pomocou tejto techniky využíva typicky algoritmus založený na vyhľadávaní spoločne s tzv. *fitness* funkciou, ktorá vhodne reprezentuje aktuálny cieľ testovania a slúži na riadenie a poskytovanie spätnej väzby do procesu vyhľadávania dát. Z toho dôvodu je táto technika veľmi vhodnou možnosťou pre automatizáciu generovania testovacích dát.

- **Kombinatorické testovanie (Combinatorial Testing)** [24, 26, 2] – Medzi významné prístupy k testovaniu a tvorbe testovacích dát patrí aj tzv. *kombinatorické testovanie*, ktorého princip spočíva vo vytvorení testovacích sád výberom podmnožiny zo všetkých kombinácií dát pre vstupné parametre programu. Pre testovaný systém, ktorý má n parametrov, kde každý parameter má d možných hodnôt, je celkový počet kombinácií d^n. Vo väčšine prípadov je však prakticky takmer nemožné vyčerpávajúco otestovať všetky kombinácie, vzhľadom na exponenciálnu výpočetnú zložitosť procesu tvorby takýchto kombinácií. Z toho dôvodu existujú stratégie kombinatorického testovania, ktorých snahou je vytvoriť menšiu, no dostatočnú podmnožinu kombinácií pre testovacie prípady. Jednou takou stratégiou je *T-wise*/*T-way* testovanie, ktorého ústredná myšlienka je vytvoriť všetky možné kombinácie pre t parametrov a nie nutne všetkých n (t.j. $t <= n$). Jedinou podmienkou je, aby pre každú kombináciu hodnôt t parametrov existoval aspoň jeden testovací prípad, ktorý by ich pokryl. Hodnota t sa taktiež nazýva ako *sila pokrytia*, a je typicky zvolená ako malé číslo, oproti teoreticky o mnoho väčšiemu počtu všetkých parametrov. Vďaka stratégii *T-wise* testovania je možné výrazne zredukovať celkový počet testovacích prípadov. Základným prípadom *T-wise* testovania je *Pair-wise* testovanie (t.j. $t = 2$, ktoré je zamerané na vytvore-

nie všetkých párových kombinácií medzi parametrami. Vychádza tak z predpokladu, že pre odhalenie chýb v programe je často potrebná interakcia medzi dvoma parametrami, avšak chyby môžu byť spôsobené aj interakciou viac ako dvoch parametrov. Pri *Pair-wise* testovaní sa typicky používa na tvorbu testovacích prípadov stratégia *In-Parameter-Order* (IPO), publikácia [26] však rozširuje túto stratégiu do generickej podoby, vďaka ktorej je možné algoritmicky tvoriť kombinácie vzhľadom na ľubovoľne veľkú silu pokrytia t.

Okrem toho existuje množstvo iných techník a prístupov ako generovanie testovacích prípadov na základe modelov (Model-based test case generation), mutačné testovanie, fuzzing, testovanie založené na špecifikácii alebo testovanie založené na genetických algoritmoch.

2.4 Existujúce riešenia a súvisiace projekty

Vo výskumnom ale aj komerčnom prostredí existuje v súčasnosti niekoľko ukážkových softvérových projektov, služieb a prístupov umožňujúcich automatické generovanie dát pre rôzne účely:

- **Korat** [29, 9] – Nástroj KORAT pre generovanie využíva metódu imperatívnych predikátov, vyplývajúcich zo špecifikácie testovaného programu. Ide o nástroj, ktorý umožňuje testovanie realizovať ako black-box ale aj white-box stratégiu a technika generovania spočíva vo vyčerpávajúcom generovaní všetkých neizomorfných (izomornými štruktúrami sa v kontexte tohto nástroja myslí hlavne vlastnosť, že určité rôzne dáta vyvolajú rovnaké správanie programu, *izomorfizmus* je teda v tom, ako dané dáta vníma program a ako na ne reaguje) instancií dátových štruktúr až do určitej hraničnej veľkosti. Imperatívne predikáty, vyplývajúce zo špecifikácie programu, alebo vnútorných konštrukcií v programe je možné zapísať v notácii programovacieho jazyka Java a nástroj vytvorí sadu testovacích vstupov spĺňujúcich zadané predikáty.

  Bounded exhaustive testing je prístup automatického generovavania využívajúci techniky založené na *imperatívnych predikátoch*, ktoré generujú všetky testovacie dáta až po určitú hranicu, ktorá spĺňa množinu obmedzení definovaných pomocou imperatívneho predikátu.

- **Mockaroo**[1] – Mockaroo je voľne dostupný webový nástroj, ktorý sa primárne zaoberá generovaním dát pre relačné databázy. Jeho hlavnou výhodou je poskytnutie veľkého množstva sémantických typov, pre ktoré sú k dispozícii masívne datasety, a je teda možné pri správnej špecifikácii zadania pre generovanie vytvoriť veľmi realistické dáta, minimálne z pohľadu obsadenia sémantických hodnôt.

- **Tonic**[2] – Medzi populárne komerčné služby patrí nástroj Tonic, ktorý sa špecializuje okrem iného aj na maskovanie, subsetting a automatickú detekciu sémantických vlastností z existujúcich produkčných databáz.

- **json-generator**[3]**.** – Medzi jednoduchšie voľne dostupné generátory dát patrí napríklad webový nástroj JSON-GENERATOR umožňujúci generovanie JSON dát s ohľadom

[1] Webová služba *Mockaroo* je voľne dostupná na https://www.mockaroo.com/.
[2] Webová služba *Tonic.ai* je komerčne dostupná na https://www.tonic.ai/.
[3] Nástroj je dostupný na https://www.json-generator.com/

na špecifikované obmedzenia, zvolené sémantické typy alebo pravidlá definované v rámci jedného dokumentu.

Vyhodnotenie

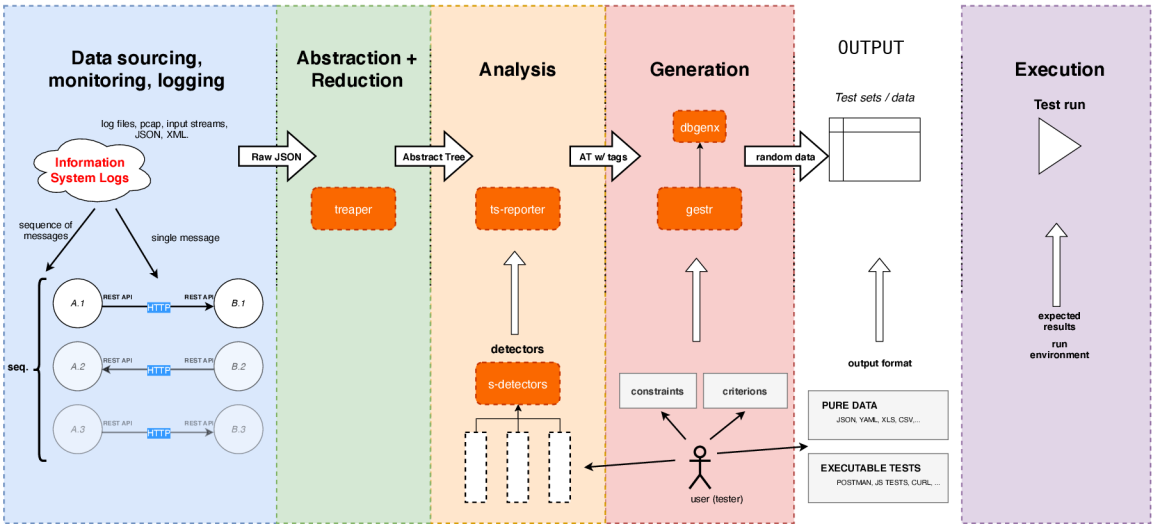
Všetky spomenuté nástroje sa dajú charakterizovať spoločne ako generátory, ktoré berú do úvahy manuálne špecifikovanú podobu dát, ktoré je potrebné generovať. Význačným rozdielom tejto práce oproti projektom spomenutým v predošlej sekcii, je zameranie sa na spracovanie reálnych dát, ktoré pre účely generovania máme k dispozícii, a len na základe ich podoby vytvorenie vhodného modelu a následne reprodukcie. Hlavným rozdielom teda je, že o vnútornom fungovaní komponentov informačného systému v kontexte tejto práce nepredpokladáme nič a jediným zdrojom pre tvorbu nových dát je vzorka reálnych dát. Druhým rozdielom je, že všetky spomenuté generátory sú úzko zamerané na tvorbu buď štrukturálnych alebo relačných dát, ale v žiadnom prípade sa neberie do úvahy kontext komunikácie a výmeny správ. Jednou z hlavných motivácií v tejto práci je vytvorenie riešenia, ktoré by súčasne kombinovalo generovanie štruktúrovaných dát, ale taktiež reprodukciu vysokoabastraktnej komunikácie v informačnom systéme.

Testos projekty

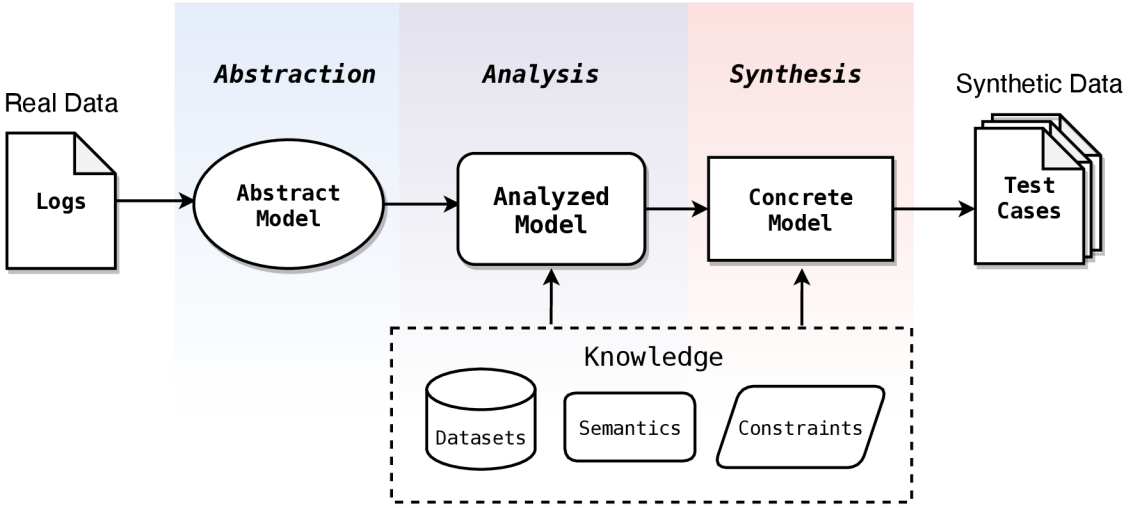
V rámci platformy Testos boli do súčasnosti vyvíjané viaceré projekty, ktoré sa priamo zaoberali generovaním testovacích dát alebo tvorili súvisiace komponenty pre generátor. Projekty TS-REPORTER, TREAPER, S-DETECTORS a GESTR sa spoločne podieľali na systematickej analýze stromových dát s cieľom vytvoriť vhodný abstraktný model popisujúci charakteristické vlastnosti dát ako je ich štruktúra a sémantika, pre eventuálne generovanie nových umelých dát rovnakého alebo podobného typu. Projekty **dbgenx**, **dataster** a **datasterx** sa zaoberali generovaním testovacích dát pre relačné databázy na základe kombinácie schémy, sady obmedzení a datových sád.

- **ts-reporter**[4]: Hlavnou funkciou projektu je obsluha žiadostí na analýzu sady stromových dát s dynamickým využitím detektorov v korektnom poradí.
- **s-detectors**[5]: Projekt predstavuje sadu špecializovaných detektorov, vďaka ktorým je možné v stromových dátach vyhodnotiť sémantiku a dátové typy primitívnych hodnôt.
- **treaper**[6]: Projekt vytvára komplexné riešenie pre úlohy analýzy a redukcie nad stromovými dátami.
- **dbgenx**: Generátor dát pre relačné databázy s využítim *SMT* solvera *Z3*.
- **dataster**: Informačný systém a frontend pre generátor DBGENX.
- **gestr**: Generátor stromových šruktúr.
- **combine**: Nástroj *Combine* umožňuje generovať kombinácie testovacích dát podľa *T-wise* kombinačného kritéria.

[4] https://pajda.fit.vutbr.cz/testos/ts-reporter
[5] https://pajda.fit.vutbr.cz/testos/s-detector
[6] https://pajda.fit.vutbr.cz/testos/treaper



Obr. 2.5: Počiatočný konceptuálny návrh automatizovaného generovania testovacích dát s využitím nástrojov TREAPER, TS-REPORTER, S-DETECTORS, GESTR a DBGENX, ktoré sa spoločne usilovali o vytvorenie kolektívneho riešenia



Obr. 2.6: Ústredným záujmom tejto práce je proces tvorby umelých dát na základe vlastností reálnych vzoriek dát z komunikačných záznamov. Generovanie tak musí prebiehať len na základe vhodne vytvoreného abstraktného a analyzovaného modelu.

Kapitola 3

Štruktúrované dáta v komunikácii informačného systému

Táto kapitola uvádza kontext *informačného systému* ako hlavný uvažovaný kontext projektu, v ktorom dáta možno jednak získať a analyzovať, no taktiež ich vytvárať a používať pre účely testovania. Sekcia 3.1 popisuje správy informačného systému na najvyššej konceptuálnej úrovni. Sekcia 3.2 popisuje konkrétne serializačné formáty používané v informačných systémoch v rámci komunikácie. Sekcia 3.3 uvádza možnosti pre vhodnú a jednotnú abstrakciu dát pre všetky serializačné formáty.

Štruktúrované dáta Komplexné štruktúrované dáta predstavujú jeden zo základných stavebných blokov efektívnej výmeny informácií medzi softvérovými systémami. Je vďaka nim možné reprezentovať a prenášať stav z jedného systému do druhého alebo sofistikovane špecifikovať žiadosti pre službu, ukladať dokumenty a iné dáta v digitálnej podobe. Štruktúrované dáta budú v kontexte tejto práce chápané ako akékoľvek dáta, ktoré nie sú atomické, majú hierarchickú alebo inak tvarovanú štruktúru a obsahujú nejaký konečný počet primitívnych hodnôt alebo podštruktúr. Vo svojej podstate je ich možné chápať aj ako dynamické štruktúry, kde ich tvar je daný prepojením iných, menších podštruktúr. Na začiatok je potrebné štruktúrované dáta odlíšiť od primitívnych dát. Primitívne dáta môžu nadobúdať konkrétne hodnoty z nejakej domény. Komplexné štruktúrované objekty je však oveľa náročnejšie priradiť k nejakej konkrétnej doméne. Najvýraznejšou vlastnosťou je ich tvar, no taktiež je dôležité aj obsadenie hodnôt, z ktorých sa komplexná štruktúra skladá.

3.1 Správy v informačnom systéme

Informačný systém je možné chápať aj ako systém, ktorého hlavnými komponentmi sú webové služby. *Webová služba* je softvérový systém navrhnutý na komunikáciu medzi viacerými strojmi na sieti. S webovými službami typicky komunikujú iné systémy pomocou HTTP protokolu na aplikačnej vrstve zasielaním správ obsahujúcich štruktúrované dáta v určitom serializovanom formáte (`XML`, `JSON`) a iné webové štandardy [15].

Message Oriented Model [15] Ak sa na komunikáciu v informačnom systéme pozeráme z hľadiska výmeny správ medzi webovými službami, tak je možné komunikáciu reprezentovať pomocou modelu správ *Message Oriented Model*. Tento model sa zameriava na správy, štruktúru správ, transport správ bez akéhokoľvek modelovania ich významu a dôležitosti. Jednoducho povedané ide o model orientovaný striktne na samotné subjekty komunikácie. *Message Oriented Model* je vhodným základom pre reprezentáciu komunikácie v tomto projekte, avšak v praxi sa často používajú modely orientované na služby, a nie na správy ako je to v tomto modeli. Najtypickejšími modelmi architektúr pre komunikáciu v prostredí informačných systémov sú protokoly *REST API* a *SOAP*.

REST (*Representational State Transfer*) [15, 7] je jednoduchý protokol vytvorený pre vzájomnú komunikáciu webových služieb prostredníctvom jednotnej a preddefinovanej sady operácií na prístup a manipuláciu webových zdrojov. Typicky sú dáta posielané v serializačnom formáte *JSON* a komunikácia je realizovaná cez aplikačný protokol `HTTP`.

SOAP (*Simple Object Access Protocol*) [16] Protokol *SOAP* slúži na výmenu štruktúrovaných dát v distribuovanom prostredí alebo vzdialené spustenie procesov prostredníctvom sieťového rozhrania. Protokol pre serializáciu štruktúrovaných dát v komunikácii využíva textový formát XML a pre prenos dát je možné použiť väčšinu typicky používaných aplikačných a transportných protokolov ako sú `HTTP`, `SMTP`, `TCP` alebo `UDP`). Základnými vlastnosťami protokolu *SOAP* je neutralita (v kontexte použitia ľubovoľného protokolu pre prenos dát) a nezávislosť od paradigmatu alebo technológie implementácie pri jeho využití.

3.2 Serializované formáty štruktúrovaných dát

V praxi sú typicky na účely ukladania štruktúrovaných dát používané formáty `XML`, `JSON` a podobné štandardizované formáty pre ukladanie komplexných dát. Pre hlbšie pochopenie problematiky generovania štruktúrovaných dát sa však táto práca v tejto kapitole zameria aj na viac abstraktnú reprezentáciu štruktúrovaných dát pomocou matematických štruktúr ako sú grafy a stromy, ktoré intuitívne zachytávajú štrukturálne vlastnosti vyššie spomenutých serializačných formátov.

Existuje množstvo štandardizovaných formátov pre štruktúrované dáta, vhodné pre rôzne špecifické prostredia a spôsoby používania. Pre kontext tejto práce je však dôležité len prostredie informačných systémov a serializačné formáty používané v komunikácii webových služieb. Takýmito formátmi sú najtypickejšie `XML`, `JSON`, `YAML` a niekoľko ďalších. Zvyšok tejto sekcie popisuje práve tieto formáty.

JSON (*JavaScript Object Notation*) [10, 5] `JSON` je syntaktický formát serializovanej reprezentácie pre štruktúrované dáta. Syntax vychádza z jazykových literálov pre objekty v jazyku *JavaScript* podľa štandardu *ECMA (ECMAScript Programming Language Standard, Third Edition)*. Pomocou formátu JSON je možné reprezentovať štyri základné primitívne dátové typy: textový reťazec (`string`), číslo (`number`), booleovsky výraz (`bool`) a prázdny výraz (`null`). Okrem toho je možné vyjadriť dva štruktúrované typy: objekt (`object`) a pole (`array`). Objekt je neusporiadaná kolekcia dvojíc kľúč-hodnota, kde názov

kľúča je textový reťazec a hodnota je jeden z primitívnych alebo štrukturálnych typov. Pole je zoradená postupnosť hodnôt, ktoré patria medzi primitívne alebo štrukturálne typy.

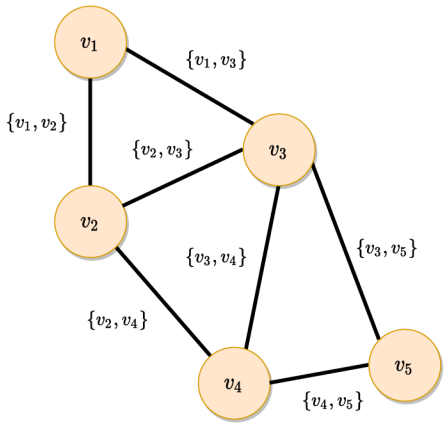
XML (*Extensible Markup Language*) [43] `XML` je značkovací jazyk pre dokumenty typicky využívaný v komunikácii medzi webovými službami a slúži podobne ako `JSON` na výmenu štruktúrovaných dát medzi dvoma entitami. Syntax `XML` je typická svojími značkami, ktoré typicky v pároch uzatvárajú vždy jeden logický celok. Štruktúra XML dokumentu topologicky vytvára stromovú štruktúru, kde je možné jednoznačne odlíšiť hierarchické usporiadanie jednotlivých zložiek dokumentu.

3.3 Abstrakcia štruktúrovaných dát

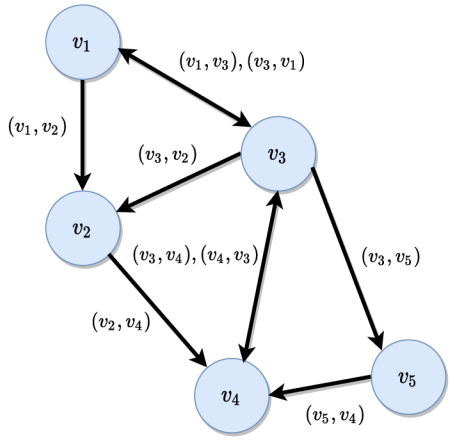
Serializačné formáty slúžia ako vhodný mechanizmus pre konzistentný prenos a reprezentáciu komplikovaných dátových štruktúr, avšak ich konkrétna implementačná podoba sa navzájom odlišuje. Aby bolo možné o dátach uvažovať jednotnejšie, je nutné aby boli konkrétne serializačné formáty abstrahované na jednotnejší formát. Táta sekcia postupne popisuje rôzne alternatívne prístupy k tomu, ako komplexné dynamické šturktúrované dáta reprezentovať a formalizuje niekoľko matematických a programových štruktúr, s ktorými sa v zvyšnej časti práce bude manipulovať.

Grafy Prvou zvolenou matematickou štruktúrou, ktorá vhodne abstrahuje vlastnosti dynamických štruktúr, je *graf*, vzhľadom k tomu, že štruktúrované dáta typicky ukladajú hodnoty, ktoré majú navzájom jasne definované vzťahy.

Definícia 3.3.1 (Graf [27]). Graf $\mathcal{G}$ je dvojica $\mathcal{G} = (V, E)$, kde V je množina vrcholov a $E \subseteq V \times V$ je množina hrán. Graf nazývame **neorientovaný**, ak jeho hrany nie sú orientované $\forall (i,j) \in E, (i,j) = (j,i) \in E$, inak graf nazývame **orientovaný** .



Obr. 3.1: Neorientovaný graf.



Obr. 3.2: Orientovaný graf.

Stromy Matematická štruktúra *stromu* vhodnejšie reprezentuje skutočnú podobu štruktúrovaných dát, v ktorých je typické hierarchické usporiadanie. Oproti grafu presnejšie reflektuje štruktúru formátov `JSON` alebo `XML`, a taktiež vyjadruje prirodzenú formu dokumentov, webových stránok (DOM modelu), atď. Okrem toho poskytuje štruktúra stromu značné zefektívnenie operácií z pohľadu výpočetnej zložitosti na ich štruktúre oproti grafom, najmä vďaka neexistujúcicm cyklom.

Definícia 3.3.2 (**Strom** [21])**.** Strom S je súvislý graf, v ktorom nie sú žiadne cykly.

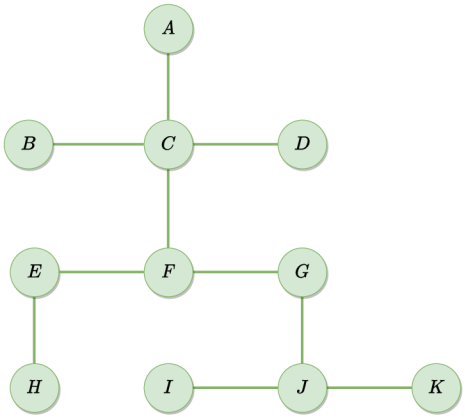
Definícia 3.3.3 (**Orientovaný (koreňový) strom** [21])**.** Orientovaný strom, niekedy taktiež nazývaný ako koreňový strom, je *orientovaný graf* so špecifikovaným vrcholom R, a platí nasledovné:

a) Každý vrchol $V \neq R$ je počiatočným vrcholom práve jednej hrany, označovanej $e[V]$.

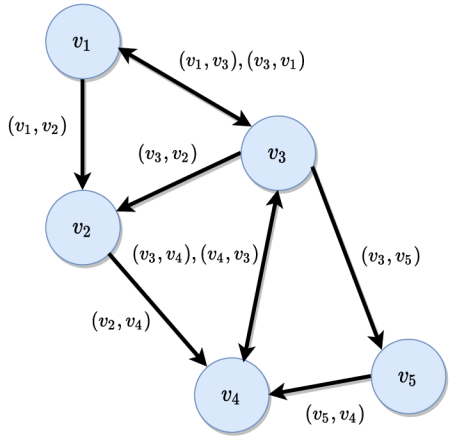
b) R nie je počiatočným vrcholom žiadnej hrany.

c) R je koreň stromu a pre každý vrchol $V \neq R$ existuje unikátna orientovaná cesta z V do R.

Komplexné dátové štruktúry bývajú veľmi často reprezentované a modelované ako koreňové stromy. Štruktúru je totiž možné definovať ako koreňový strom, kde uzly reprezentujú objekty a hrany reprezentujú atribúty [29].



Obr. 3.3: Jednoduchý strom.

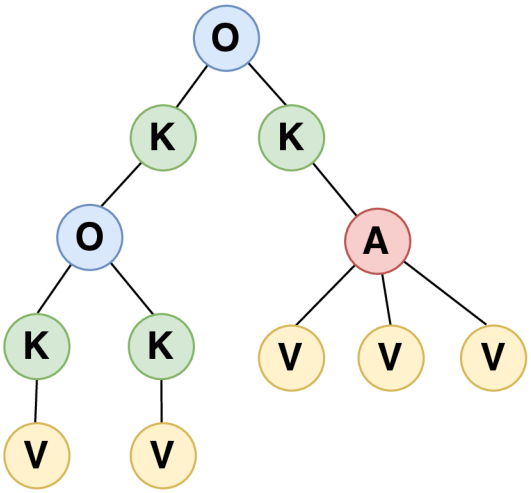


Obr. 3.4: Orientovaný graf.

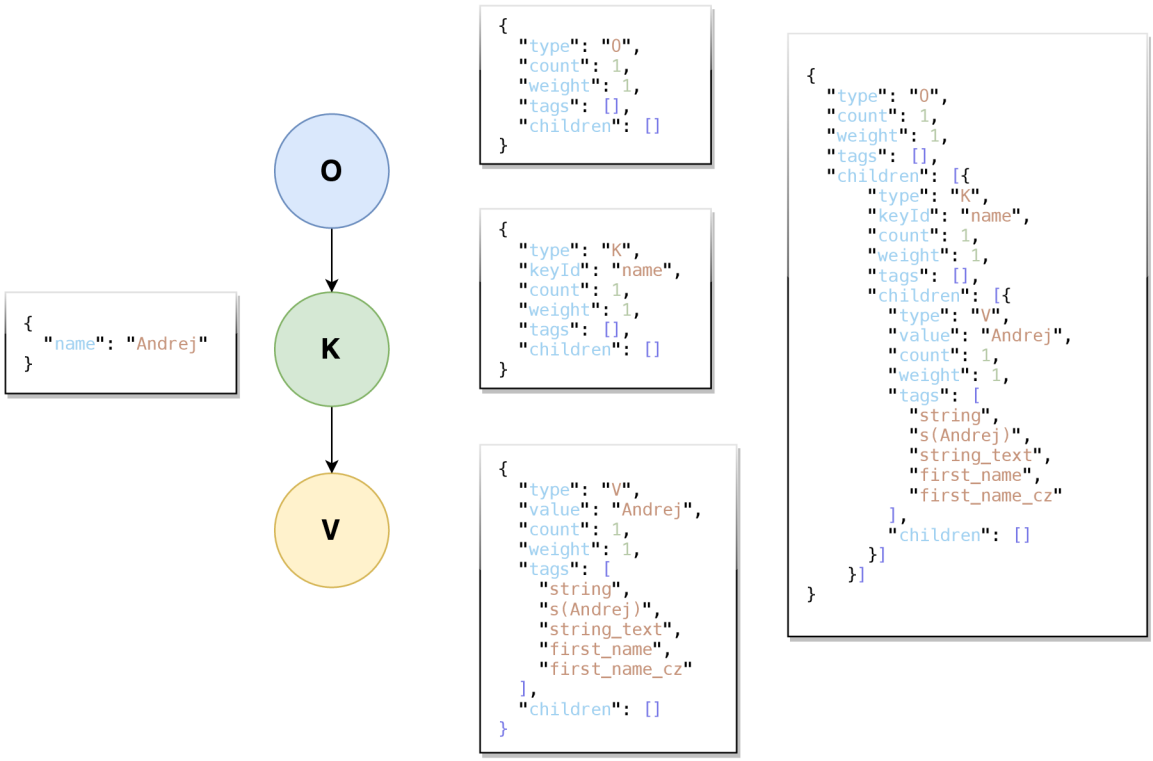
Abstraktný strom Keďže štruktúrované dáta môžu mať v serializovanej podobe odlišnú syntax a formát, bola v rámci projektov platformy *Testos* zavedená dátová štruktúra *Abstraktného stromu*, najmä pre účely zjednotenia. Navyše poskytuje abstraktný strom možnosti pre pridanie meta dát o sémantických a štrukturálnych vlastnostiach pôvodnej dátovej štruktúry.

Definícia 3.3.4 (**Abstraktný (dátový) strom** (AT) [34])**.** Stromová štruktúra AT je založená na jednoduchom koreňovom strome. Pre všetky uzly ukladá nasledujúce metadáta.

- **Typ uzla** – reprezentuje štruktúru uzlov a jeho potomokov a existujú dve kategórie typov.
  - **Štrukturálny typ uzla** – reprezentuje špecifickú datovú štruktúru nasledujúcich typov.
    - **A** – usporiadané pole/zoznam.
    - **O** – množina dvojíc kľúč-hodnota, kde kľúč je unikátny identifikátor a reprezentuje ho typ štrukturálneho uzlu **K** a hodnota predstavuje potomka - uzol ľubovoľného typu).
    - **K** – reprezentuje unikátny identifikátor. Jeho potomkom je uzol v kontexte hodnoty nejakého objektu.
    - **V** – primitívna atomická hodnota bez nasledujúcich potomkov.
  - **Špeciálne typy uzlov** – reprezentujú dynamické uzly, ktoré do stromu zavádzajú možnosti variability a slúžia na súčasné vyjadrenie vlastností viacerých stromov redukovaných do jednej spoločnej abstrakcie. Sú definované dva nasledujúce typy.
    - **R** – variantný uzol predstavuje všetky varianty hodnôt na danom mieste v strome pochádzajúcich zo vzoriek rôznych stromov.
    - **V** – virtuálny uzol predstavuje bránu medzi reálnym a virtuálnym priestorom jeho rodičovského uzla. Reálnym priestorom je podstrom z pohľadu rodičovského uzla, ktorý bol reálne v stromových dátach zaznamenaný a virtuálny priestor predstavuje podstrom vychádzajúci z tohto virtuálneho uzla a ukladá dodatočné metadáta a štrukturálne varianty v podstrome.
- **Váha** – vyjadruje mieru istotu pri detekcii.
- **Početnosť** – vyjadruje počet reálnych vzoriek, v ktorých sa konkrétny uzol vyskytol.
- **Značky** – označujú sémantické a iné informácie, ktoré boli odhalené detektormi.



Obr. 3.5: Príklad abstraktného stromu.



Obr. 3.6: Rozklad abstraktného stromu reprezentujúceho jednoduchý JSON objekt.

Kapitola 4

Problém generovania štruktúrovaných dát

Táto kapitola popisuje *problém generovania dát* ako *výpočetný problém*. Pre vhodnú formálnu definíciu problému generovania sú využité poznatky o predmetných dátových štruktúrach (najmä *stromových štruktúrach*) z kapitoly 3 spoločne s *kritériami adekvátnosti* z kapitoly 2 riadiace samotný proces generovania. Sekcia 4.1 konkretizuje pojem generovania dát na jednotlivých úrovniach v kontexte generovania štruktúrovaných dát pre účely testovania. Do úvahy je brané najmä prostredie komunikácie informačného systému, ako bolo uvedené v kapitole 3, no taktiež sa nahliada na problém generovania vo všeobecnosti, bez ohľadu na konkrétny kontext. Nasledujúca sekcia 4.1 uvádza konkrétne prístupy k syntéze stromových štruktúr, ich charakteristiku, výpočetnú zložitosť, no taktiež aj kritéria adekvátnosti spojené s tvorbou stromových štruktúr pri snahe testovať rozhranie informačného systému. Tieto prístupy vychádzajú zo špecifikácie pre testovanie postavenej na *kritériach založených na rozhraní* (*interface-based criteria*) vzhľadom k znalosti toho, akú majú reálne vstupné dáta typicky podobu. Zvolené kritéria adekvátnosti dopĺňa aj *kombinačné kritérium*, ktoré tvorí jednu z hlavných stratégií pre generovanie dostatočného množstva a rôznorodosti umelých testovacích dát. Na záver v sekcii 4.3 je popísané generovanie správ v podobe časových rád, ktoré súčasne možno chápať ako abstrakciu pre generovanie celého rámca testovacej sady.

4.1 Syntéza štruktúrovaných dát

Na proces generovania dát je možné nahliadať ako na výpočetný problém, v ktorom je nutné vhodným spôsobom vytvoriť takú štruktúru, tvar a obsadenie hodnôt, aby boli splnené všetky potrebné obmedzenia a kritéria, podľa ktorých je generovnaie riadené. Tento proces teda už zo svojej povahy potrebuje určité obmedzenia, inak by generovanie nemalo zmysel, a v druhom rade potrebuje metódu, akou vytvárame konkrétne inštancie štruktúry, pre ktoré dané obmedzenia platia.

Keďže strom je dynamická dátová štruktúra, nie je možné aplikovať jednoduché numerické alebo logické obmedzenia takým spôsobom, ako pri atomických hodnotách. Dôležitou zložkou v reprezentácii dynamických štruktúr sú ukazovatele. Problém generovania dynamickej štruktúry stromu je možné redukovať na problém riešenia *obmedzení pre ukazovatele*. Ty-

pickou technikou pre riešenie tohto problému je využitie tzv. *symbolických hodnôt*, čo robí v dôsledku z problému nerozhodnuteľný problém [46].

Pri generovaní stromových dát je možné problém rozdeliť na dve časti:

1. Generovanie **štruktúry stromu**.
2. Generovanie konkrétnych **primitívnych hodnôt** pre premenné v uzloch stromu.

Syntéza a transformácie štruktúry stromu Syntetické dáta by mali byť *podobné* reálnym dátam. Čo to ale znamená? Podobnosť v tomto kontexte značí jednak podobnosť tvaru štruktúry, ale aj obsadenie konkrétnych hodnôt v uzloch. Najjednoduchším prípadom je, ak existuje jedna dátová štruktúra, pre ktorú je potrebné vytvoriť novú štruktúru s identickým tvarom.

Izomorfizmus [14] Najjednoduchším prípadom vo všeobecnosti je, ak existuje jedna dátová štruktúra, pre ktorú je potrebné vytvoriť novú štruktúru s identickým tvarom. Toto mapovanie sa z matematického hladiska nazýva *štruktúru-zachovávajúce mapovanie*, a ak pre dve štruktúry $\mathcal{A}$ a $\mathcal{B}$ platí, že medzi nimi existuje *štruktúru-zachovávajúce mapovanie*, tak tieto štruktúry opisujeme ako *izomorfické* [14]. Naopak, ak medzi dvoma štruktúrami neexistuje mapovanie, ktoré by zachovávalo tvar, tak tieto štruktúry opisujeme navzájom ako *neizomorfické*. Pri testovaní izomorfizmu pri orientovaných ale aj neorientovaných stromoch (bez koreňového uzlu), je na rozdiel od problému pre overenie izomorfizmu pri grafoch (ktorý je považovaný za NP-ťažký) problém overenia izomorfizmu pri orientovaných aj neorientovaných stromoch riešiteľný iba v lineárnom čase – $O(n)$ [18].

Definícia 4.1.1 (Izomorfizmus na orientovaných stromoch [14, 40]**).** Izomorfizmus dvoch orientovaných (koreňových) stromov $T_1(V_1, E_1, r_1)$ a $T_2(V_2, E_2, r_2)$ je bijektívne a štruktúru zachovávajúce mapovanie medzi množinami vrcholov $\varphi : V_1 \longrightarrow V_2$, pre ktoré platí:

$$\forall u, v \in V_1 : (u, v) \in E_1 \iff (\varphi(u), \varphi(v)) \in E_2 \wedge \varphi(r_1) = r_2$$

4.2 Generovanie atomických hodnôt

Pre tvorbu atomických, a teda už viac nedeliteľných hodnôt v štruktúrovaných dátach, je potrebné splniť obmedzenie pre ich typ, logické a numerické vlastnosti a prípadne sémantiku. Na rozdiel od syntézy štruktúr nie je nutné zaoberať sa dynamickým aspektom štruktúry atomických hodnôt. Na najnižšej úrovni je problém, v ktoróm je potrebné určité dosadenie hodnôt, možné chápať ako *SAT problém*. Problém je však taktiež možné vnímať ako *SMT problém*, vďaka čomu je možné na konkrétne pravidlá pre dosadenie hodnôt do uzlov využívať vysokoabstraktné pojmy a miesto booleovských výrazov je možné používať predikáty a rozličné datové typy. Atomické hodnoty možno abstrahovať ako premenné, ktoré môžu nadobúdať hodnoty z určitého a jasne definovaného definičného oboru. Vďaka tomu je ich tvorba výrazne jednoduchšia oproti syntéze štruktúr, keďže ak je známy typ, sémantika a obmedzenia pre danú atomickú hodnotu, tak je zároveň známa aj konečná množina hodnôt, ktoré môže premenná nadobudnúť. Zložitejším problémom je generovanie kombinácie

viacerých atomických hodnôt. V jednoduchom prípade však ide iba o splnenie obmedzení čo sa týka typu, prípadne sémantiky. Ťahším prípadom je, ak je súčasťou generovania aj nejaké dodatočné kritérium pre kombinačné vlastnosti. Generovanie atomických hodnôt je možné riešiť ako *SMT* problém, avšak pri generovaní štruktúry stromu, je problém oveľa náročnejší.

Problém splniteľnosti obmedzení (CSP) [30, 46, 13, 28] Problém generovania testovacích dát je možné taktiež klasifikovať ako *CSP (Constriant Satisfaction Problem)*, ktorý je možné formulovať ako množinu premenných pre ktoré platí, že ich hodnoty spĺňajú určitý počet obmedzení. Pre vyriešenie CSP problému je základným princípom systematické prehľadávanie, ktoré garantuje presný výsledok.

Definícia 4.2.1 (**CSP** [37])**.** Problém CSP na konečných definičných oboroch je definovaný trojicou $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ kde:

- $\mathcal{X} = \{x_1, \cdots, x_n\}$ je množina premenných problému.
- $\mathcal{D} = \{\mathcal{D}_1, \cdots, \mathcal{D}_n\}$ je množina definičných oborov premenných tak, že platí $x_k \in \mathcal{D}_k$ pre všetky $k \in [1; n]$.
- $\mathcal{C} = \{\mathcal{C}_1, \cdots, \mathcal{C}_m\}$ je množina obmedzení, kde obmedzenie $\mathcal{C}_i = (\mathcal{X}_i, \mathcal{R}_i)$ je definované množinou $\mathcal{X}_i = \{x_{i_1}, \cdots, x_{i_k}\}$ premenných a reláciou $\mathcal{R}_i \subset \mathcal{D}_{i_1} \times \cdots \times \mathcal{D}_{i_k}$, ktorá definuje množinu hodnôt povolených súčasne pre premenné množiny $\mathcal{D}_{i_k}$

4.3 Generovanie správ v časovej rade

Správy v informačnom systéme je možné chápať aj ako udalosti, ktoré sa vyskytujú v určitej časovej postupnosti. Pri ich vytváraní je možné zohľadniť pôvodné časové rozloženie alebo naopak vygenerovať nové, buď náhodne alebo s podobným rozložením.

Jedným možným prístupom, ako generovať pravdepodobnostné časové rozloženie, je využitie *Poissonovského procesu*. Ten je možné využiť v dvoch podobách. Jednak ako homogénny Poissonovský proces, kde sa priemerná miera výskytu udalosti drží konštantne na tej istej hodnote počas celého časového úseku, alebo nehomogénny Poissonovský proces, ktorý povoľuje, aby sa miera výskytu udalostí v priebehu času dynamicky menila vzhľadom na určitý predpis funkcie [25].

Vyhodnotenie z pohľadu využitia v projekte

Pre kontext tejto práce je možné na problém generovania nahliadať trochu jednoduchšie. Vzhľadom na to, že úlohou projektu je v princípe napodobniť štruktúry a istým spôsobom iba reprodukovať už existujúci model, a nie výrazne mutovať alebo vyčerpávajúcim spôsobom vytvárať množstvo iných teoreticky možných štruktúr, nie je nutné pokladať problém generovania z praktického hladiska za zložitý.

Kapitola 5

ISAGEN – Automatický generátor testovacích dát

Táto kapitola sa zaoberá vyhodnotením zadania z pohľadu analýzy požiadaviek projektu a plynulo prechádza do fázy návrhu. Sekcia 5.1 progresívne znižuje mieru abstrakcie, a od veľmi generických pojmov postupne prechádza až k detailnej špecifikácii požiadaviek. V sekcii 5.2 sú jasne definované dôležité pojmy a koncepty, ktoré budú v projekte často využívané, a objasnené málo jasné požiadavky a obmedzenia. Dodatočne je vytvorený aj základný návrh pre vnútorné fungovanie nástroja, rozdelenie na najdôležitejšie fázy a komponenty. Sekcia 5.2 poskytuje dôkladný návrh pre generovanie na viacerých úrovniach a ponúka niekoľko možností pre zvolenie stratégie generovania na každej úrovni. V sekcii 5.3 je vytvorený návrh pre implementáciu nástroja, jednak z pohľadu systémovej architektury, ale taktiež aj vnútornej štruktúry aplikácie, najdôležitejších čiastkových komponentov a ich funkcie.

5.1 Analýza požiadaviek

Cieľom tejto sekcie je jasne definovať zadanie a odkryť na prvý pohľad nejasné požiadavky, ktoré pre projekt vyplývajú zo zadania. Táto úloha je v tejto fáze jednoduchšia vďaka formálne definovanému teoretickému základu v kapitolách 2, 3 a 4.

Zodpovedanie základných otázok o projekte Predtým, ako sa definujú špecifické požiadavky projektu, je vhodné načrtnúť základne charakteristiky neformálne. Základnú predstavu o vlastnostiach poskytne zodpovedanie otázok o veľkosti, rámci a prostredí projektu. Vďaka tomu je možné získať intuitívne pochopenie motivácie a cieľov pre tento projekt.

- **Kto bude nástroj primárne využívať?**
  Softvéroví testeri a vývojári.
- **Aká je hlavná funkcia nástroja?**
  Generovanie štruktúrovaných dát a reprodukcia správ podobných *reálnej komunikácii informačného systému.*
- **Ako sa bude nástroj používať?**
  Nástroj bude možné používať priamo cez prostredie príkazového riadku ako konzolovú

aplikáciu. Možným rozšírením by pre nástroj mohlo byť aj GUI, napríklad vo forme webovej aplikácie alebo obalenie v podobe HTTP webovej služby.

- **Aké sú očakávané vstupy pre nástroj?**
  K dispozícii budú komunikačné logy zachytávajúce vysokoúrovňovú HTTP komunikáciu medzi uzlami informačného systému. Pre jednoduchú demonštráciu sa predpokladá serializačný formát `JSON` a štruktúra udalostí v komunikačných logoch by mala obsahovať údaj o *čase*, *zdrojovom* a *cieľovom identifikátore* komunikačného uzla a samotné {emphdáta prenášané v tele správy. V rámci tejto práce a implementácie projektu nie je uvažovaný serializačný formát `XML` ani iné špecifické štruktúry pre dáta komunikácie vzhľadom na krátky obmedzený časový rámec projektu. Ak by sa preukázalo, že myšlienka za nástrojom je bez problémov realizovateľná, doimplementovanie podpory pre iné serializačné formáty pre prenos dát by malo byť triviálne.

- **Aké množstvo užívateľov bude nástroj využívať?**
  V jeden moment bude nástroj využívaný vždy maximálne iba jedným užívateľom alebo jedným procesom. Nástroj bude chápaný ako bezpamäťový nástroj na jednorázové vykonávanie úloh bez ukladania stavu alebo čakania na žiadosti od užívateľa.

- **Aké množstvo a veľkosť dát bude nástroj spracovávať?**
  Odhadom budú vstupy pre nástroj v rozsahu jednotiek kilobajtov až desiatok megabajtov.

- **Aké množstvo žiadostí bude nástroj v daný moment očakávať?**
  Nástroj je uvažovaný tak, že bude v jeden moment používaný vždy iba výhradne jedným používateľom s jednou žiadosťou. Tieto žiadosti budú komplexné a rozsahovo veľké, čiže nástroj sa bude prioritne používať s malou frekvenciou, sekvenčne a s veľkým objemom dát jednorázovo.

- **Aká je tolerovateľná doba čakania na dokončenie úlohy zadanej nástroju?**
  Tým, že nástroj bude pracovať s veľkým objemom dát v rámci jednej úlohy, je tolerovateľné čakanie v horizonte sekúnd až jednotiek minút. Prirodzene sa pri väčších objemoch a intenzitách kombinovania dát môže očakávať aj dlhšie čakanie.

Špecifikácia požiadaviek

Tabuľka 5.1 sumarizuje všetky spomenuté požiadavky v jednotnom formáte. Každá požiadavka je jasne definovaná štvoricou údajov: identifikátorom, popisom, kategóriou a prípadnou závislosťou k iným požiadavkám.

Existujú dve kategórie požiadaviek:

- **Funkcionálne požiadavky** – označované pomocou **F**.
- **Nefunkcionálne požiadavky** – označované pomocou **NF**.

Id.	Popis	Kategória	Závislosť
REQ.1	Nástroj by mal generovať dáta *podobné* reálnym vzorovým dátam z komunikácie informačného systému.	F	-
REQ.2	Nástroj by mal na vstupe prijímať aspoň jeden zo serializačných formátov `JSON`/`XML`.	F	-
REQ.3	Nástroj by mal vedieť spracovať a modelovať dáta komunikačných logov z prostredia informačného systému.	F	-
REQ.4	Nástroj by mal vedieť generovať kombinácie sémanticky alebo štrukturálne nejednoznačných zložiek dát s ľubovoľne silným faktorom `t-strength`.	F	-
REQ.5	Nástroj by mal vedieť prijímať vstupné dáta zo súborového systému.	F	-
REQ.6	Nástroj by mal vedieť prijímať vstupné dáta zo `STDIN`	F	-
REQ.7	Nástroj by mal vedieť zapisovať výstupné dáta do súborového systému.	F	-
REQ.8	Nástroj by mal vedieť zapisovať výstupné dáta na `STDOUT`	F	-
REQ.9	Nástroj by mal vedieť generovať štrukturálne izomorfné štruktúry.	F	-
REQ.10	Nástroj by mal vedieť generovať postupnosť správ komunikácie informačného systému s rovnakými časovými rozostupmi ako v zdrojovej komunikácii.	F	-
REQ.11	Nástroj by mal vedieť generovať postupnosť správ komunikácie informačného systému s podobnými časovými rozostupmi ako v zdrojovej komunikácii.	F	-
REQ.12	Nástroj by mal vedieť generovať postupnosť správ komunikácie informačného systému s náhodnými časovými rozostupmi.	F	-
REQ.13	Nástroj by mal vedieť generovať postupnosť správ, v ktorej sa zachová sémantický typ správ a hodnôt v správach.	F	-
REQ.14	Nástroj by mal vedieť poskytovať priebežné výstupy každej fázy spracovania.	F	-
REQ.15	Všetky fázy a moduly nástroja by mali byť modulárne a rozšíriteľné.	NF	-
REQ.16	Nástroj by mal poskytnúť minimálne generátory pre všetky *značky* využívané nástrojom TS-REPORTER	F	-

Tabuľka 5.1: Špecifikácia požiadaviek projektu.

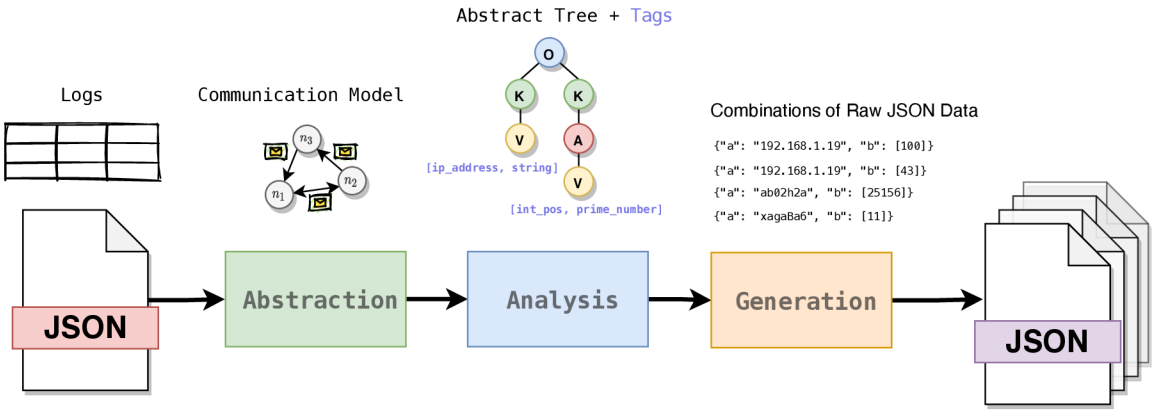
5.2 Základná koncepcia a definícia pojmov

Prvým krokom pri tvorbe návrhu je vytvorenie vysoko abstraktného konceptu. Nástroj by mal vykonávať množstvo malých čiastkových úloh, a je preto vhodné začať s návrhom systematicky odhora postupne smerom dole. Na najvyššej úrovni sa dá fungovanie nástroja rozdeliť na tri fázy, ktoré je možné vykonávať nezávisle od seba. Práve tieto fázy by tak mohli tvoriť základné stavebné bloky architektúry nástroja.

Abstrakcia Transformácia reálnych štruktúrovnaých dát do abstraktného modelu. Tento proces by mal byť súčasťou nástroja najmä preto, že je v pláne aktívne využiť iné nástroje platformy *Testos*, ktoré využívajú zjednotený formát pre štruktúrované stromové dáta, ktorým je formát *Abstraktného stromu*. Ďalej je špecifickou abstrakciou pre tento projekt snaha abstrahovať komunikáciu vychádzajúcu z logov komunikácie do prirodzenejšej podoby. Abstrakcia rozširuje reálne dáta o konštrukcie, do ktorých je neskôr možné zapisovať metadáta a ďalšie pomocné údaje pre ďalšiu prácu a transformácie nad dátami.

Analýza Sémantická a štrukturálna analýza abstrahovaných dát. V nástroji sú uvažované dve úrovne analýzy, vertikálna a horizontálna. Vertikálna analýza sa zaoberá sémantickými a štrukturálnymi vlastnosťami vždy iba jedného stromu, kde prechádza jeho štruktúru odhora dole (vertikálne). Horizontálna analýza sa zaoberá súvislosťami medzi stromami alebo uzlami stromov naprieč celou stopou v časovej rade, čiže analyzuje stromy navzájom horizontálne. Výstupom analýzy by mali byť abstraktné dáta obohatené o metadáta o ich sémantike a štrukturálnych vlastnostiach.

Generovanie Syntéza nových umelých dát na základe abstraktného modelu. Najdôležitejšou a ústrednou časťou celého nástroja je proces generovania. Generovanie by malo prebiehať na viacerých úrovniach, kde medzi najdôležitejšie patrí syntéza štruktúry stromu a následne naplnenie štruktúry sémanticky validnými hodnotami. Okrem toho by malo generovanie brať do úvahy kombinačné a iné kritéria pre generovanie.



Obr. 5.1: Konceptuálne znázornenie vnútorného procesu nástroja *ISAGEN*.

Hlavné komponenty nástroja

Táto sekcia popisuje najdôležitejšie navrhnuté komponenty nástroja *ISAGEN* spoločne s popisom vnútorného procesu fungovania pre celý nástroj. Výsledná implementácia sa môže sčasti odlišovať.

1. **Abstractor** – Komponenta zabezpečujúca abstrakciu vstupných reálnych dát do podoby buď abstraktného stromu, stopy alebo komunikačného modelu. Tieto formáty predstavujú abstraktný model, ktorý je možné rozširovať o rôzne metadáta a efektívnejšie transformovať pre účely analýzy a následne generovania.
2. **Analyzer** – Komponenta zabezpečujúca priebeh vertikálnej a horizontálnej analýzy nad stromovými dátami. Abstraktné stromy rozširuje o značky a ďalšie sémantické a štrukturálne metadáta.
3. **Generator** – Komponenta zabezpečujúca priebeh generovania a syntézy nových umelých dát. Generátor by mal byť schopný pracovať na viacerých úrovniach a jednak tvoriť komplexné štruktúry, ale taktiež sémanticky validné hodnoty uložené v štruktúrach.

Okrem toho však existuje ešte množstvo okolitých procesov, ktoré zabezpečujú automatický chod celého nástroja, čítanie vstupu alebo export dát z nástroja von. Jednotlivé vysoko úrovňové fázy je taktiež možné rozdeliť na ešte špecifickejšie etapy. Fázy automatického generovania dát je možné vymenovať nasledovne:

1. **Príprava kontextu** – Načítanie a spracovanie vstupu a parametrov pre spustenie programu.
2. **Abstrakcia**
   (a) Získavanie znalostí z komunikačného logu a vytvorenie komunikačného modelu.
   (b) Abstrakcia dát uložených v správach do podoby *abstraktného stromu*.
3. **Analýza**
   (a) Vertikálna analýza dát pomocou nástroja *ts-reporter*.
   (b) Horizontálna analýza dát.
4. **Generovanie**
   (a) Generovanie komunikačného modelu.
   (b) Generovanie stôp (časových postupností správ).
   (c) Generovanie kombinácií variantných ciest a nejednoznačných značiek v abstraktných stromoch.
   (d) Generovanie podštruktúr v stromoch (objektov a polí).
   (e) Generovanie atomických hodnôt.
5. **Príprava výstupu** – Export testovacích dát alebo opakované generovanie iných variant.

Vstupy programu – Komunikačné logy

Hlavným uvažovaným vstupom pre nástroj bude záznam o komunikácii viacerých uzlov informačného systému. Komunikačný log je možné chápať ako usporiadanú časovú sériu ukladajúcu časovú značku, zdrojovú IP adresu, cieľovú IP adresu a dáta správy.

- `Timestamp` – Časová značka reprezentujúca konkrétny časový moment.
- `Src` – IP adresa odosielateľa v rámci výmeny správy.
- `Dst` – IP adresa prijímateľa v rámci výmeny správy.
- `Payload` – Telo správy obsahujúce serializované dáta vo formáte `JSON`.

Timestamp	Src	Dst	Payload
2021-05-21T02:39:27-02:00	172.42.69.121	172.42.69.122	{...}
2021-05-21T02:39:53-02:00	172.42.69.122	172.42.69.121	{...}
2021-05-21T02:40:02-02:00	172.42.69.123	172.42.69.121	{...}
2021-05-21T02:40:15-02:00	172.42.69.121	172.42.69.123	{...}

Tabuľka 5.2: Príklad Komunikačného logu, v ktorom sú uložené 4 správy reprezentujúce komunikáciu 3 uzlov. Dáta správy sú pre jednoduchosť skryté, avšak je ich možné chápať ako akékoľvek JSON dáta uložené ako textové reťazce.

Definícia základných pojmov

Pre intuitívne chápanie súvislostí a vnútorných procesov sú pre návrh a zvyšné sekcie tejto kapitoly stanovené nasledujúce pojmy a abstrakcie:

- **Communication Log** – Komunikačný log alebo záznam komunikácie vo forme textového súboru, udržiavajúceho formát tabuľky. Riadky v logu predstavujú konkrétne momenty výmeny dát medzi uzlami v informačnom systéme a majú presne stanovený formát zložený zo štyroch atribútov: časová značka (`Timestamp`), zdrojová a cieľová IP adresa uzla (`Src`, `Dst`) a dáta prenášané medzi uzlami (`Pata`).
- **Communication Model** – Model komunikácie založený na štruktúre grafu, ktorý vhodne reprezentuje reálny systém, v ktorom sú uzly (`Nodes`) jeho komponenty a prepojenia predstavujú komunikačné kanály (`Channels`). Výmena dát je reprezentovaná pomocou udalostí (`Events`), ktoré sú združované do skupín podľa kanála, nazývaných stopy (`Traces`).
- **Node** – Uzol komunikačného modelu reprezentujúci uzol informačného systému. Má jednoznačný identifikátor a s ostatnými uzlami ho spája množina kanálov.
- **Channel** – Kanál komunikácie spájajúci dva rozdielne uzly v komunikačnom modeli. Každý kanál má definovaný zdrojový a cieľový uzol a ich vzájomnú orientáciu. V komunikačnom modeli predstavuje kanál hranu v grafe.

- **Trace** – Stopa udalostí na danom kanáli, ktorá ukladá kolekciu udalostí v usporiadanom zozname. Udalosti sú usporiadané logicky podľa ich časovej značky a vyjadrujú jednosmerný prúd dát medzi dvoma uzlami.
- **Event** – Udalosť ako abstrakcia a obal pre zasielané dáta v komunikácii. Udalosť zapúzdruje samotné dáta (`payload`) a k nim súvisiace metadáta (napr. `eventType`), ktoré vznikajú počas procesu analýzy.
- **Message** – Správa v kontexte tohto projektu je zameniteľným pojmom s udalosťou, znamená teda úplne to isté. Udalosť dáva prenášaným dátam abstraktný zmysel a navádza na možné rozšírenie nástroja v budúcnosti o spracovávanie a generovanie dát iných ako tých, ktoré sú získané výhradne z komunikácie informačného systému.
- **Payload** – Telo správy/udalosti ukladajúce priamo dáta komunikácie, ktoré sú už deserializované z čisto textovej serializovanej podoby formátu *JSON*. Dáta je možné v tomto kontexte chápať a označovať ako *stromy*, keďže práve tie reprezentujú komplexnú štruktúru dát uložených v tele správy.
- **Similarity** – Podobnosť medzi dvomi stromovými štruktúrami. Jednak v zmysle generovania syntetických dát na základe reálnych dát s tým, že sa zachová určitá miera podobnosti, no taktiež aj podobnosť v zmysle odhalenia podobných stromových štruktúr v rámci jednej stopy. Miera podobnosti je závislá na štruktúre a sémantike hodnôt v uzloch stromu.
- **Equivalence** – Maximálne striktná podobnosť medzi dvomi stromovými štruktúrami. V tomto prípade sa ale odlišuje ekvivalencia štruktúry a štruktúry spolu s hodnotami uzlov. Ekvivalencia štruktúry dvoch stromov je v podstate izomorfizmom medzi stromami, pre kontext návrhu je dôležité, že ekvivalenciou sa rozumie zhoda štruktúry zdrojového a cieľového stromu a sémantika hodnôt nie je nutnou požiadavkou.
- **Vertical Analysis** – Vertikálna analýza vyhodnocuje stromové dáta v izolácii pomocou sémantickej a štrukturálnej analýzy uzlov. Výstupom vertikálnej analýzy je *abstraktný strom* (AT), ktorý o každom uzle z pôvodných reálnych dát vytvára komplexnejší obraz s metadátami.
- **Horizontal Analysis** – Horizontálna analýza sa zaoberá vzťahmi, vzormi a kompozičnými vlastnosťami naprieč celou stopou. Analyzuje stromové dáta a uzly na najnižšej úrovni ale naprieč širokým rámcom. Snahou horizontálnej analýzy je vyhladať súvisiace hodnoty a podštruktúry v sekvencii stromov a vhodne označiť tieto súvislosti tak, aby sa zohľadnili pri generovaní.
- **Generation** – Fáza procesu generovania, v ktorej prebieha konkretizácia štruktúry a uzlov z abstraktného stromu do konkrétneho nového stromu.
- **Abstract Tree** – Formát pre ukladanie abstraktných stromových štruktúr s komplexným popisom vlastností o uzloch, metadátami, značkami a virtuálnymi uzlami, ktoré umožnujú v jednom strome udržiavať viaceré varianty toho istého typu stromu. Formát bol navrhnutý v rámci platformy *Testos* ako zjednotený formát pre výmenu abstraktných stromových dát medzi viacerými nástrojmi, ktoré s nimi pracujú.
- **Real Data** – Reálne dáta z produkčného prostredia informačného systému. Všetky hodnoty majú konkrétny tvar, nie sú o nich známe žiadne sémantické vlastnosti ani ich

dátové typy. Dáta tak reprezentujú jediné znalosti o vstupných dátach informačného systému.

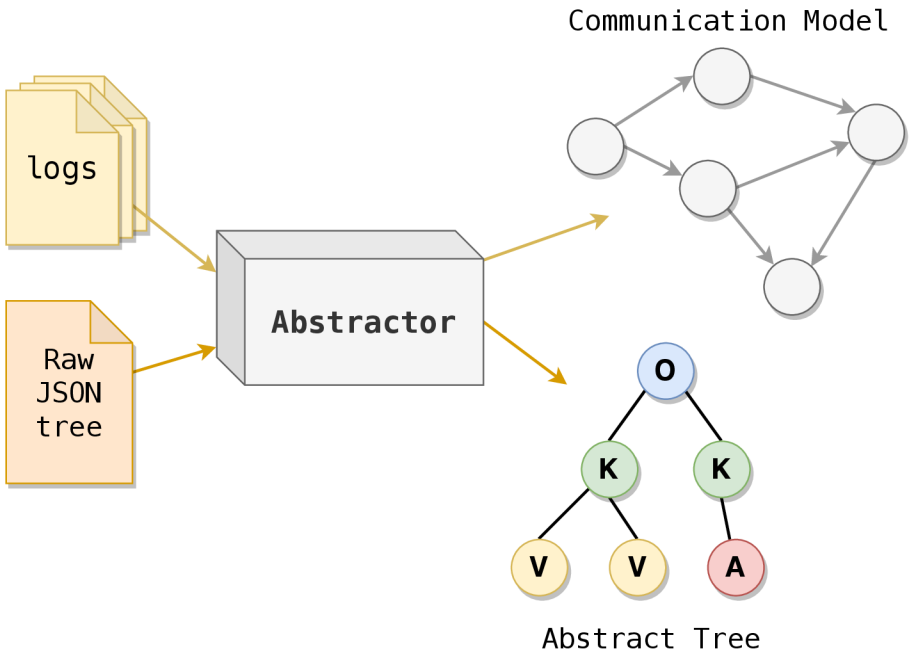
- **Synthetic Data** – Výstupné dáta generátora, ktoré vychádzajú zo znalostí o reálnych dátach, no ich konkrétny tvar a výber hodnôt sa môže odlišovať.

Abstrakcia

Nástroj na začiatku na vstup dostane sekvenčný popis komunikácie uzlov informačného systému, v ktorom sa môžu správy navzájom popretkávať. Oveľa vhodnejšie by bolo tieto správy vhodným spôsobom kategorizovať a zoskupiť len tie, ktoré spolu súvisia. Dobre zvolený model by dokázal správy v komunikácii systematicky združovať podľa toho, akých uzlov sa priamo týkajú, akým smerom a v aký čas boli poslané. Z reálneho systému je poskytnutý log komunikácie medzi uzlami vo forme tabuľky. Praktickejšie je však reprezentovať komunikáciu v podobe blízkej topológii reálneho systému.

Abstrakcia bude realizovaná na dvoch hlavných úrovniach:

- Na úrovni komunikačného modelu.
- Na úrovni stromových dát.



Obr. 5.2: Konceptuálne znázornenie fungovania modulu `Abstractor`.

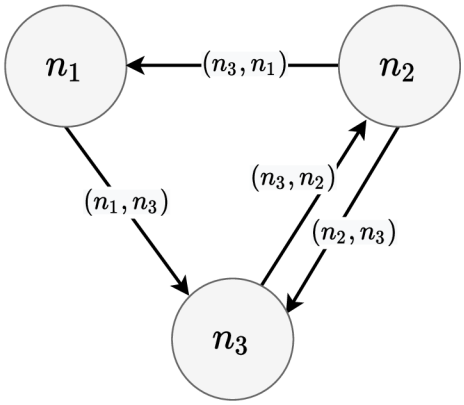
Modelovanie komunikácie informačného systému Pre modelovanie komunikácie v informačnom systéme bude využitá štruktúra založená na matematickej štruktúre *orientovanom grafe*. Okrem toho však štruktúru grafu rozširuje o dodatočné objekty, množiny a funkcie. Inšpiráciou pre modelovanie komunikácie v tomto projekte je najmä ob-

lasť dolovania procesov, kde je tvorba určitého modelu, typicky modelu procesov, na základe logu udalostí veľmi častým problémom [1, 19, 4]. Pri tvorbe komunikačného modelu ako prvá prebehne transformácia z tabuľkovej reprezentácie záznamov udalostí z komunikácie (`Log Events`) na model komunikácie, kde je každý uzol informačného systému modelovaný ako uzol grafu. Pre každú dvojicu uzlov, medzi ktorými bola zachytená aspoň jedna výmena správy, je vytvorená nová hrana v grafe, predstavujúca *komunikačný kanál* (channel). Pre každý *komunikačný kanál* (channel) je zhromaždená kolekcia udalostí (správ) zaslaných v danom smere, ďalej označovaných ako *stopy* (traces). Udalosti v stopách sú zoradené podľa času. Konceptuálne proces modelovania vyjadruje obrázok **??** a formálnu definícu komunikačného modelu poskytuje definícia 5.2.1.

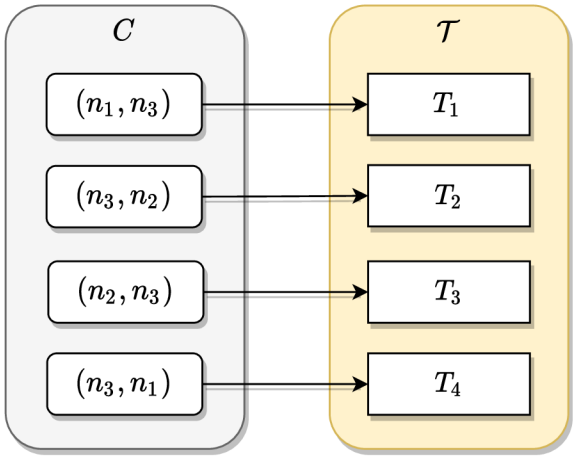
Definícia 5.2.1 (**Model komunikácie**)**.** Korektné zachytenie komunikácie v informačnom systéme vyžaduje zvolenie vhodnej topologickej štruktúry, ktorá by odrážala relačné vlastnosti komunikácie. Štruktúra $M = (N, C, \mathcal{T}, t)$ predstavuje *model komunikácie* na báze **orientovaného grafu**. Komponenty štruktúry M sú definované nasledovne:

- **N** je konečná množina komunikačných uzlov, ktoré je možné chápať taktiež ako vrcholy grafu. Komunikačný uzol je identifikovateľný pomocou jedinečného reťazca, typicky vo forme IP adresy.
- $\mathbf{C} \subseteq \{(x, y) : (x, y) \in N^2 \ \wedge \ x \neq y\}$ je konečná množina komunikačných kanálov, ktoré je možné chápať taktiež ako orientované hrany grafu. Kanál je vždy zložený z usporiadanej dvojice dvoch komunikačných uzlov, kde prvá hodnota predstavuje zdrojový komunikačný uzol a druhá hodnota cieľový komunikačný uzol.
- $\mathcal{T} = \{T : \exists c \in C \wedge t(c) = T\}$ je konečná množina stôp.
  - Stopa T predstavuje dátovú štruktúru popisujúcu skupinu správ medzi dvoma komunikačnými uzlami. Stopa má nasledujúce atribúty:
    - `Src` $\in N$ je zdrojový komunikačný uzol.
    - `Dst` $\in N$ je cieľový komunikačný uzol.
    - `Events` je mapovanie časových momentov na abstraktné stromy reprezentujúce dáta určitej správy (`DateTime` $\rightarrow$ `AT`).
    - `Payloads` je konečná postupnosť abstraktných stromov reprezentujúcich dáta pre všetky správy v danej stope.
- $t : C \rightarrow \mathcal{T}$ je bijektívne mapovanie, ktoré priraďuje vždy práve jednu stopu $T \in \mathcal{T}$ ku každému komunikačnému kanálu $c \in C$. Podmienkou pre toto mapovanie je, aby priradená stopa mala zdrojový (`Src`) a cieľový (`Dst`) komunikačný uzol zhodný s dvojicou uzlov v rámci asociovaného komunikačného kanálu. Formálne tak musí platiť, že pre stopu $T \in \mathcal{T}$ a komunikačný kanál $(n_1, n_2) \in C$ platí rovnosť `T.Src` $= n_1$ a `T.Dst` $= n_2$ a existuje mapovanie $t : (n_1, n_2) \mapsto T$.

Stopy (*Traces*) v komunikačnom modeli slúžia na vytvorenie abstrakcie pre prenos dát medzi dvoma komunikačnými uzlami, kde okrem toho, že obsahuje samotné dáta správy, tak obsahuje tiež dodatočné metadáta ohľadom kontextu komunikácie. Dáta správy sú v rámci stopy už deserializované a abstrahované do podoby *abstraktného stromu* (`AT`), ktorý



Obr. 5.3: Štruktúra *modelu komunikácie* vychádza z matematickej štruktúry *orientovaného grafu*. Jednotlivé uzly medzi sebou vytvárajú spojenia na základe toho, odkiaľ kam bola zaslaná jedna alebo viac správ. Správy môžu byť medzi dvoma uzlami posielané v obidvoch smeroch, z toho dôvodu model zohľadňuje odlišnosť smerov komunikácie.

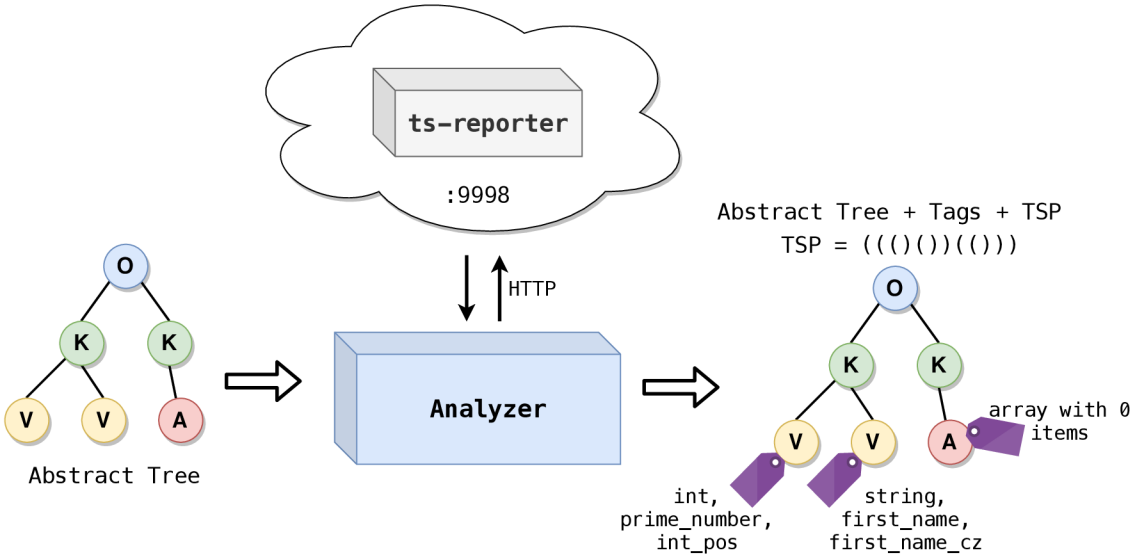


Obr. 5.4: Mapovanie t priraďuje každému komunikačnému kanálu z množiny C práve jednu stopu z množiny $\mathcal{T}$, vďaka čomu je možné zoskupiť relevantné správy na úrovni komunikácie vždy iba dvoch uzlov.

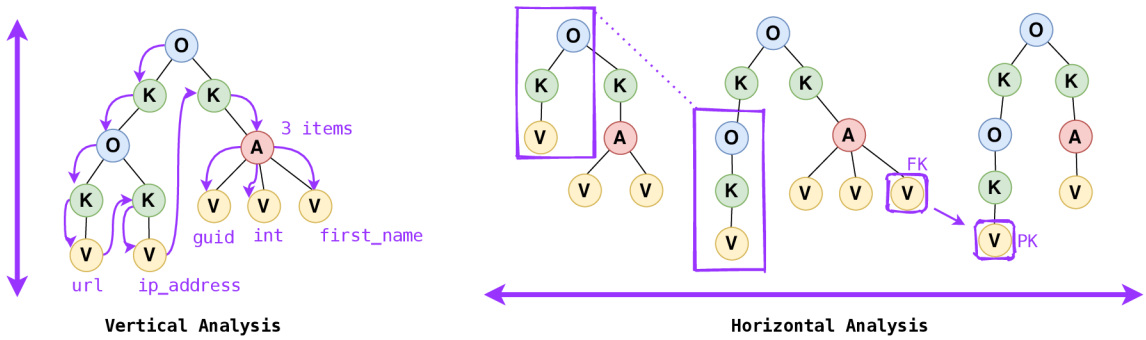
však sám o sebe neudržuje žiadne iné informácie ohľadom komunikácie ako je napríklad časová značka momentu, kedy bola správa odoslaná. Práve preto udržuje stopa v atribúte `Events` mapovanie všetkých zaznamenaných časových momentov udalostí, ktoré v rámci komunikácie dvoch komunikačných uzlov nastali a priraďuje im konkrétny abstraktný strom, obsahujúci dáta správy. Pre úlohy nad dátami, ktoré časový rozmer nepotrebujú využívať je dostupný atribút `Payloads`, ktorý udržuje abstraktné stromy v jednoduchej konečnej postupnosti bez časovej informácie.

Analýza

Hlavným zmyslom analýzy je spracovanie už abstrahovaných vstupných dát a vyhodnotenie sémantických a štrukturálnych vlastností. Vo výsledku tak analýza rozšíri vstupné dáta o dodatočné metadáta, popisujúce ich charakteristiku a vlastnosti, ktoré bude neskôr možné pri generovaní využiť pre presnejšie napodobnenie pôvodných dát. Analýza je zložená z dvoch hlavných častí. Prvou je *vertikálna analýza*, ktorá "vertikálne" prechádza stromové dáta strom po strome a s využitím špecializovaných detektorov odhaľuje vlastnosti dát v stromových štruktúrach. Druhou je *horizontálna analýza*, ktorá berie do úvahy celý rámec stopy a stromové dáta analyzuje a hľadá medzi nimi súvislosti naprieč celou stopou.



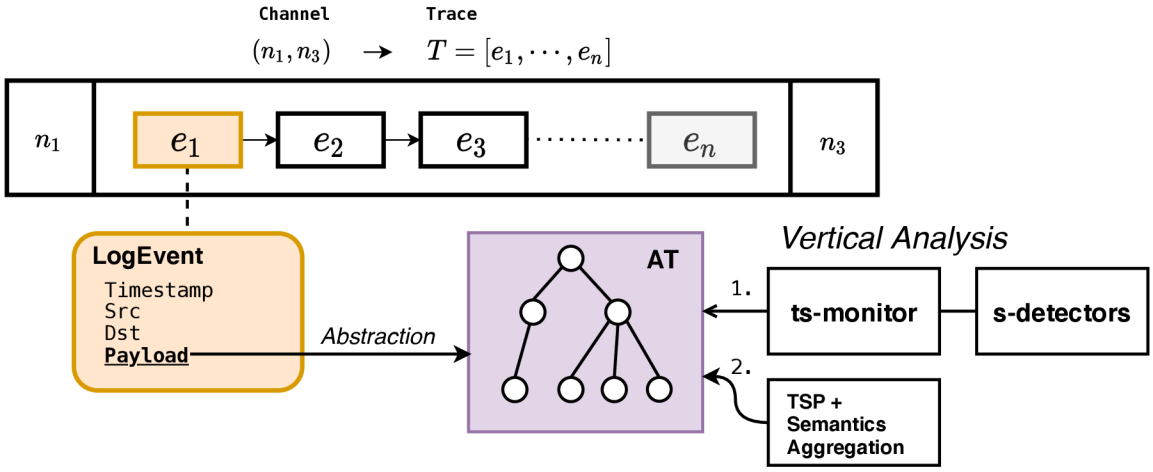
Obr. 5.5: Konceptuálne znázornenie fungovania modulu `Analyzer`.



Obr. 5.6: Vertikálna a horizontálna analýza stromových dát.

Vertikálna analýza Úlohou vertikálnej analýzy je vyhodnotiť konkrétne reálne vzorky štruktúrovaných dát z pohľadu ich štruktúry, typu hodnôt a sémantiky, variability hodnôt a podštruktúr, a vytvorenie abstraktnej štruktúry, ktorá by dostatočne reprezentovala všetky podstatné vlastnosti reálnych dát. Motiváciou pre vytvorenie takejto abstrakcie je hlavne

príprava na fázu generovania, ktorá po analýze nasleduje. Počas generovania sa abstraktná forma dát využije ako predpis na konkretizáciu špecifických syntetických dát. Výsledkom vertikálnej analýzy je opäť model stopy (Trace), no teraz už každá udalosť ukladá abstraktný strom (AT) s dodatočnými metadátami o jeho vlastnostiach.



Obr. 5.7: Konceptuálne znázornenie procesu analýzy nad udalosťami v stope.

1. **Sémantická analýza abstraktného stromu** – Pre detekciu sémantiky dát v strome bude využitý nástroj TS-REPORTER. Postupným priechodom cez komunikačný model a jeho stopy sa sekvenčne bude úloha detekcie delegovať na spracovanie nástrojom TS-REPORTER. Nástroj v súčasnosti obsahuje sadu základných detektorov, ktoré je možné jednoducho v budúcnosti rozšíriť, a tak zvýšiť schopnosti analýzy.
2. **Štrukturálna analýza abstraktného stromu** – Pre jednoznačné zaznamenanie štruktúry stromu je zavedená *charakteristika stromu* označovaná ďalej ako `TSP` (Tree Structure Property). Tvorí ju kombinácia reťazca jednoznačne identifikujúceho štruktúru stromu a množina zjednotenia všetkých značiek vychádzajúcich zo sémantickej analýzy. Reťazec opisujúci štruktúru je vytvorený algoritmom *AHU* (6.2) a je možné ho neskôr spolu s celou charakteristikou využiť pri horizontálnej analýze na detekciu podštruktúr alebo podobných stromov. Ďalej je vďaka tomuto reťazcu možné napríklad overiť jednoduchým spôsobom izomorfizmus, a teda štrukturálnu ekvivalenosť dvoch stromových štruktúr alebo aj ich vzájomnú mieru podobnosti. Týmto prístupom sa zaoberá zvyšná časť tejto sekcie.

Horizontálna analýza Ďalším krokom je tzv. horizontálna analýza, ktorej úlohou je odhalenie vzťahov medzi udalosťami a ich dátami v rámci jednej stopy. Cieľom je odhaliť skryté prepojenia, referencie, vzťahy alebo jednoducho vzory a časté postupnosti pri odosielaní správ. Popísať sa dajú nasledujúce tri hlavné ulohy horizontálnej analýzy:

- **Odhalenie vzorov v postupnosti udalostí** Ak medzi udalosťami v rámci jednej stopy existuje nejaký často opakový vzor, napr. 2 udalosti po sebe zvyknú nasledovať $A \rightarrow B$, mal by byť tento vzor zachytený a pri generovaní náhodných dát zohľadnený. Na druhej strane by mal mať užívateľ možnosť zvoliť identické reflektovanie postupnosti z reálneho prostredia, kedy odhaľovanie vzorov postupnosti zmysel nemá alebo

naopak úplne náhodné generovanie postupnosti správ, kde taktiež zmysel modelovania vzorov postupnosti nie je.

- **Hľadanie vzťahov medzi uzlami na úrovni stopy** Medzi stromovými dátami môžu existovať závislosti a vzájomné referencie na tie isté hodnoty dát. Jedným z cieľov horizontálnej analýzy by malo byť práve odhalenie takýchto súvislostí medzi uzlami na nízkej úrovni, ale v širokom rámci stopy. Príkladom takýchto vzťahov sú napríklad *many-to-one* vzťahy, dvojice primárneho a cudzieho kľúča alebo časté odkazovanie na tie isté unikátne hodnoty Naivným prístupom pre odhalenie identických uzlov naprieč celou stopou, a stromami v nej by mohlo byť kompletné prehľadanie priestoru a všetkých uzlov v každom strome v jednej stope. Tento prístup by však bolo potrebné opakovať pre každý jeden uzol. Ak sa predpokladá spracovanie iba v rámci jednej stopy, tak pri celkovo n uzloch na vstupe by bolo nutné v najhoršom prípade n krát prehľadať všetky zvyšné uzly okrem uzlov v rámci toho istého stromu, ich počet označíme m. Zložitosť tohto problému by teda naivným prehľadávaním bola $\mathcal{O}(n * (n - m)) \approx \mathcal{O}(n^2)$, ak hovoríme iba o odhalení ekvivalentných uzlov.
- **Detekcia podstromov v iných stromoch** V sekvencii stromových dát môžu vzhľadom na neobmedzené šturkturálne vlastnosti stromov vzniknúť situácie, kde jeden konkrétny strom alebo typ stromu je komponentom iného stromu. Horizontálna analýza by mala odhaliť tieto kompozičné alebo rekurzívne vzťahy. Najzložitejším problémom horizontálnej analýzy je odhalenie rovnakých podštruktúr naprieč všetkými stromami v celej stope. Problematikou sa zaoberá publikácia[1], ktorá poskytuje algoritmus na odhaľovanie podstromovov v jednom strome. Tento problém je vo všeobecnosti možné riešit ako $\mathcal{O}(n)$ (lineárne) zložitý problém pre orientované stromy bez značiek v uzloch a ako $\mathcal{O}(n * log(n))$ problém pre orientované stromy so značkami v uzloch [8].

V návrhu horizontálnej analýzy v tejto práci je však uvažované vyhľadávanie podstromov v celej sekvencii stromov v stope. Aby bolo možné využiť prístupy podobné tým, ktoré sú spomenuté vyššie, museli by sa všetky stromy v stope integrovať pod jeden koreňový strom. Avšak problém sa stáva zložitejším, ak počítame s tým, že reálne dáta budú mať značnú mieru variability aj medzi jedným a tým istým typom správy. Preto je nutné do problému vyhľadávania podstromov začleniť aj určitú mieru tolerantnosti voči nepresnosti.

Výstup analýzy Priebežným aj koncovým výsledkom čiastkových krokov procesu analýzy je *Abstraktný strom*, ktorý v sebe ukladá metadáta o pôvodnych stromových dátach. Uzlom priraďuje podľa ich štruktúry typ, prípadne značky, váhu a početnosť v prípade, že abstraktný strom vznikol redukciou viacerých vzoriek. Vo vyvíjanom nástroji bude *Abstraktný strom* reprezentovaný vo formáte *JSON*, podobne ako v nástrojoch TREAPER a TS-REPORTER.

[1] https://www.sciencedirect.com/science/article/abs/pii/S0020019012002505

Generovanie

Proces generovania je možné navrhnúť na viacerých úrovniach podľa toho, či je predmetom generovania celá datová štruktúra, podštruktúra alebo atomická hodnota. Jednotlivé *úrovne generovania* môžu byť navzájom závislé, ak existuje kritérium, ktoré by ich logicky spájalo. Z praktického hľadiska každá úroveň generovania využíva iný prístup a metódy. Je teda vhodné začať hovoriť o odlišných *stratégiach generovania* podľa toho, na akej úrovni generovanie prebieha, a s akými datovými štruktúrami pracuje. Generovanie možno rozdeliť podľa toho, na akých štruktúrach pracuje na nasledujúce úrovne:

1. **Generovanie komunikačného modelu** – Replikácia celej štruktúry modelu alebo podmnožiny komunikačného systému, v ktorom by mali byť opäť zastúpené tie isté komunikačné uzly a komunikačné kanály, avšak s novými, umelými dátami komunikácie.

2. **Generovanie stopy/sekvencie správ** – Reprodukcia sekvencie správ s časovým usporiadaním, ktoré zohľadňuje časovú distribúciu výskytu udalostí na stope zachytenej v reálnom systéme. Pri generovaní sekvencie správ je hlavnou predmetnou štruktúrou abstraktný strom, ktorý predstavuje vždy konkrétny abstraktný typ správy. Proces generovania prechádza stopu sekvenčne a vzhľadom na to, aké temporálne kritérium bolo zvolené, vhodne upravuje časové značky správ.

3. **Generovanie stromových štruktúr** – Syntéza umelých stromových dát, kde základom syntézy je transformácia *abstraktného stromu* na konkrétny dátový strom v serializovanej podobe. Štruktúra by mala byť podobná pôvodným reálnym dátam, no zároveň by sa mali brať do úvahy kombinačné kritéria a pre účely testovania vytvorenie adekvátneho množstva kombinácií nejednoznačných zložiek stromových štruktúr (t.j. značiek a variantných uzlov).

4. **Generovanie podštruktúr v stromoch** – Syntéza základných podštruktúr *abstraktného stromu* ako sú objekty a polia, pričom sa berie ohľad na systematické pokrytie podľa zadaného kritéria pre danú podštruktúru a mení sa oproti reálnym dátam počet položiek.

5. **Generovanie atomických hodnôt** – Generovanie konkrétnych atomických hodnôt podľa typu (*string*, *int*, *float*, *bool* alebo null) a sémantického kritéria podľa priradenej značky (tagu) v danom hodnotovom uzle. Hodnoty môžu byť generované náhodne (t.j. podľa náhodného kritéria).

Generovanie komunikačného modelu

Na najvyššej úrovni je podľa požiadavky REQ.X potrebné vytvárať správy podobné komunikácii reálneho informačného systému. Pre tento účel bol zavedený *komunikačný model*, ktorý slúži na modelovanie štruktúry informačného systému podľa zachytenej komunikácie. Pri tvorbe nových správ podobných modelovanému systému je na úrovni celého modelu postačujúce jednoducho celý model replikovať alebo vyčleniť podmnožinu komunikačných uzlov alebo kanálov, ktoré budú kostrou pre ďalšie fázy generovania.

Stratégie generovania komunikačného modelu

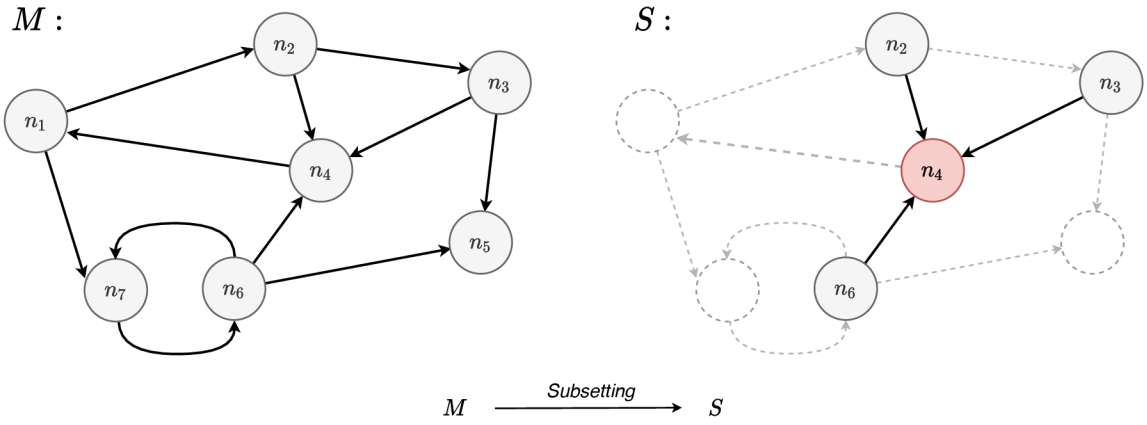
Pri generovaní komunikačného modelu bude nástroj *ISAGEN* podporovať dve stratégie. Prvá jednoducho komunikačný model *replikuje* ako celok podľa originálne zaznamenanej komunikácie a druhá *vyberá menšiu časť* komunikačného modelu vzhľadom na výber cieľového uzlu. Druhá stratégia má opodstatnenie najmä pre situáciu, ak je záujmom testovania iba jedno konkrétne rozhranie v informačnom systéme (t.j. iba jeden komunikačný uzol) a relevantná je tak komunikácia v smere k danému uzlu. Vďaka tomu je možné redukovať objem dát, ktoré je potrebné vygenerovať, a tak znížiť výpočetnú dobu pre proces generovania.

a) **Úplná replikácia (*Full replication*)** – Pri úplnej replikácii možno hovoriť o štruktúru zachovávajúcej (izomorfnej) transformácii originálneho komunikačného modelu M_1 na nový komunikačný model M_2. Ak medzi množinami komunikačných uzlov a komunikačných kanálov obidvoch modelov existuje bijektívne mapovanie zachovávajúce štruktúru, tak je možné označiť tieto modely za navzájom izomorfné ($M_1 \cong M_2$).

$$M_1 = (N, C, \mathcal{T}, t), M_2 = (N', C', \mathcal{T}', t')$$

$$M_1 \cong M_2 \implies \exists f \colon N \leftrightarrow N' : (n_1, n_2) \in C \iff (f(n_1), f(n_2)) \in C'$$
$$\wedge\ t(n_1, n_2) \in \mathcal{T} \iff t'(f(n_1), f(n_2)) \in \mathcal{T}'$$

Úplnú replikáciu je možné prakticky zrealizovať veľmi intuitívnym prístupom, a to jednoducho využitím vytvoreného modelu bez akejkoľvek zmeny.



Obr. 5.8: Znázornenie transformácie pomocou *subsettingu*.

b) **Výber časti (*Subsetting*)** – Pri výbere časti komunikačného modelu tak, aby boli zachované iba tie komunikačné uzly, kanály a stopy, ktoré sú orientované voči jednému cieľovému uzlu, je transformáciu z komunikačného modelu M_1 možné vyjadriť ako vytvorenie "podgrafu" S, pre ktorý platí, že všetky hrany sú orientované voči zvolenému cieľovému uzlu τ.

$$M = (N, C, \mathcal{T}, t),\ S = (N_S, C_S, \mathcal{T}_S, t|_{C_S}),\ \tau \in N$$

$$C_S = \{(n_1, n_2) : n_1, n_2 \in N \wedge n_2 = \tau\}$$

$$N_S = \{n \in N : \exists m \in N : (n, m) \in C_S \vee (m, n) \in C_S\}$$

$$\mathcal{T}_S = \{T \in \mathcal{T} : \exists (n_1, n_2) \in C_S : t|_{C_S}((n_1, n_2)) = T\}$$

Výber časti modelu je nutné zrealizovať pomocou transformácie, pri ktorej bude odfiltrovaná nepotrebná časť modelu. Proces transformácie popisuje algoritmus 5.1.

Algoritmus 5.1: `SUBSET-MODEL`

Input: $M = (N, C, \mathcal{T}, t), \tau \in N$
Output: $S = (N_S, C_S, \mathcal{T}_S, t|_{C_S})$

$N_S \leftarrow \varnothing$, $C_S \leftarrow \varnothing$, $\mathcal{T}_S \leftarrow \varnothing$;
for $(n_1, n_2) \in C$ **do**
  if $n_2 = \tau$ **then**
    $C_S \leftarrow C_S \cup (n_1, n_2)$;
    $N_S \leftarrow N_S \cup \{n_1, n_2\}$;
    $\mathcal{T}_S \leftarrow \mathcal{T}_S \cup t((n_1, n_2))$;
  end
end
$t|_{C_S} \leftarrow \{c \mapsto t(c) : c \in C_S \wedge t(c) \in \mathcal{T}_S\}$;
return S;

Generovanie stopy/sekvencie správ

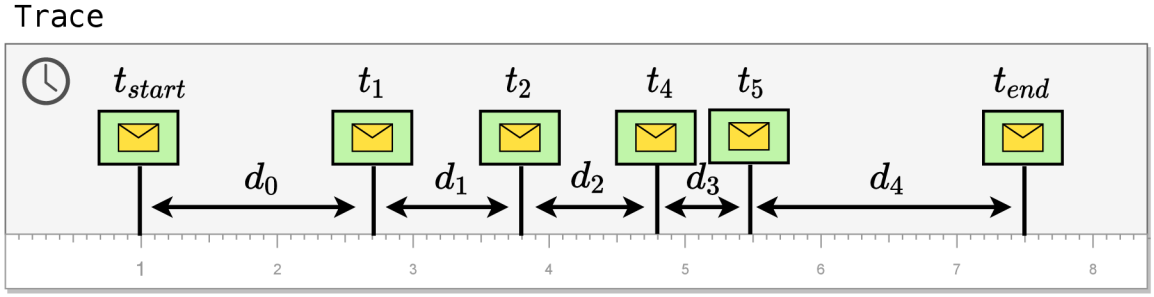
Na úrovni *stopy*, čiže časovo usporiadanej sekvencie správ, je podľa požiadavky REQ.Y potrebné vytvárať nové dáta tak, aby boli zastúpené všetky rôzne typy správ, ktoré boli zachytené. V tomto zmysle sa teda dá hovoriť o *replikácii abstraktných stromov*. Ďalej je potrebné brať do úvahy aj požiadavku REQ.Z, ktorá vyžaduje, aby mali generované správy v rámci jednej stopy medzi sebou *podobné časové rozostupy*. Z tohto hľadiska ide o reprodukciu časových rozostupov medzi správami s podobným alebo identickým rozložením pravdepodobnosti ich výskytu.

Stratégie generovania stopy/sekvencie správ

Pri generovaní stôp budú využité dva typy stratégií. Prvá bude replikovať časové rozostupy podľa pôvodnej komunikácie. Druhá využije pravdepodobnostné rozloženie časových rozostupov medzi všetkými správami v rámci jednej stopy, a na základe tohto rozloženia vytvorí novú podobnú postupnosť. Obidve stratégie zachovajú logickú postupnosť správ a ich typ daný abstrakciou abstraktného stromu.

a) **Replikovanie pôvodných časových rozostupov** – Zachovajú sa všetky správy v stope spolu s ich pôvodnými časovými rozostupmi.

b) **Reprodukcia časového rozloženia rozostupov** – Zachovajú sa všetky správy v stope, ale ich časové rozostupy sa vygenerujú odznova podľa zachyteného rozloženia časových rozostupov v pôvodnej komunikácii. Tvorbu nových časovaných rozostupov bude riadiť náhodný *Poissonovský proces* na základe priemernej dĺžky časového rozostupu medzi správami v stope. Existujú dve varianty pre implementáciu Poissonovského procesu, buď ako homogénny alebo nehomogénny.



Obr. 5.9: Znázornenie časového rozmeru na stope.

Generovanie stromových štruktúr

Samotné správy v stope sú zastúpené v podobe *abstraktných stromov*, ktoré tak predstavujú jedinečný typ správy. Pri syntéze nových umelých stromových štruktúr je potrebné vykonať transformáciu z *abstraktného stromu* na konkrétne stromové dáta, avšak abstraktné stromy majú typicky nejednoznačnú povahu. Hodnotové uzly (typu V) môžu obsahovať aj viac ako jednu sémantickú značku pre hodnotové uzly, ak bola detekcia nejednoznačná alebo jednoducho pre danú hodnotu boli počas analýzy odhalené súčasne sémantické ako aj typové vlastnosti. Ďalšou vlastnosťou abstraktných stromov je, že ich štruktúra môže obsahovať variantné uzly, a tak aj rôzne vetvy pre konkrétnu realizáciu stromovej štrukúry.

Ak abstraktný strom obsahuje aj variantné uzly, vzniká pri generovaní otázka, ktorý z variantných uzlov zvoliť ako vzor pre tvorbu nových dát. Nedeterministický charakter tohto problému je potrebné zohľadniť aj v stratégiach pre generovanie stromových štruktúr. V prípade, že *abstraktný strom* má jednoduchú štruktúru, čiže neobsahuje žiadne virtuálne alebo variantné uzly, je postačujúci jednoduchý deterministický priechod *abstraktným stromom* zhora-dole, a postupné replikovanie hierarchie uzlov s podštruktúrami a hodnotami tak ako sa v *abstraktnom strome* vyskytujú. Komplikovanejším prípadom je transformácia zložitého *abstraktného stromu*, v ktorom sa vyskytujú variantné a virtuálne uzly, ktoré je podľa požiadavky `REQ.4` potrebné zohladniť a systematicky generovať všetky alebo kritériom stanovené kombinácie variantných ciest v strome. Alternatívne sa dá k transformácii zložitého abstraktného stromu pristupovať aj metódou náhodného výberu, kedy sa vyberie vždy náhodná varianta bez ohľadu na pokrytie všetkých kombinácií.

Druhým aspektom je priradenie viac ako jednej sémantickej značky hodnotovému uzlu, kde podobne vzniká nejednoznačnosť pri rozhodovaní o tom, ktorá zo značiek by mala byť v konečnom prípade použitá ako autoritatívna pri výbere stratégie generovania konkrétnej hodnoty. Stratégie pre generovanie stromových štruktúr by tak mali taktiež zohľadniť aj výber značiek buď jednoznačným deterministickým výberom, náhodným výberom alebo systematickým kombinovaním všetkých, alebo kritériom daných kombinácií značiek.

Pre zohľadnenie variantných ciest stromu a nejednoznačného výberu spomedzi značiek je potrebné rozdeliť stratégie pre generovanie stromových štruktúr na dve kategórie. Prvá kategória je zameraná na generovanie, kombinovanie a výber variantných uzlov a ciest v strome. Druhá sa zaoberá priraďovaním a kombinovaním značiek pre hodnotové uzly. Stratégie z obidvoch kategórií je možné navzájom kombinovať. Riadiacim mechanizmom pre každú stratégiu je postupný prechod abstraktným stromom zhora-dole, od koreňového uzlu až po všetky listové uzly.

Stratégie pre generovanie variantných uzlov:

- **Deterministický výber variantnej cesty** – Ak nie je potrebné pokryť všetky varianty štruktúry abstraktného stromu, tak najjednoduchšiou stratégiou pri výbere spomedzi variantných uzlov je výber vždy len jednej varianty podľa určitého pozičného predpokladu. Pre jednoduchosť je možné napríklad predpokladať výber vždy prvého nasledovníka v zozname nasledovníkov každého variantného uzla. Dôležitou vlastnosťou je, že bude výsledkom stratégie vždy iba jedna varianta.

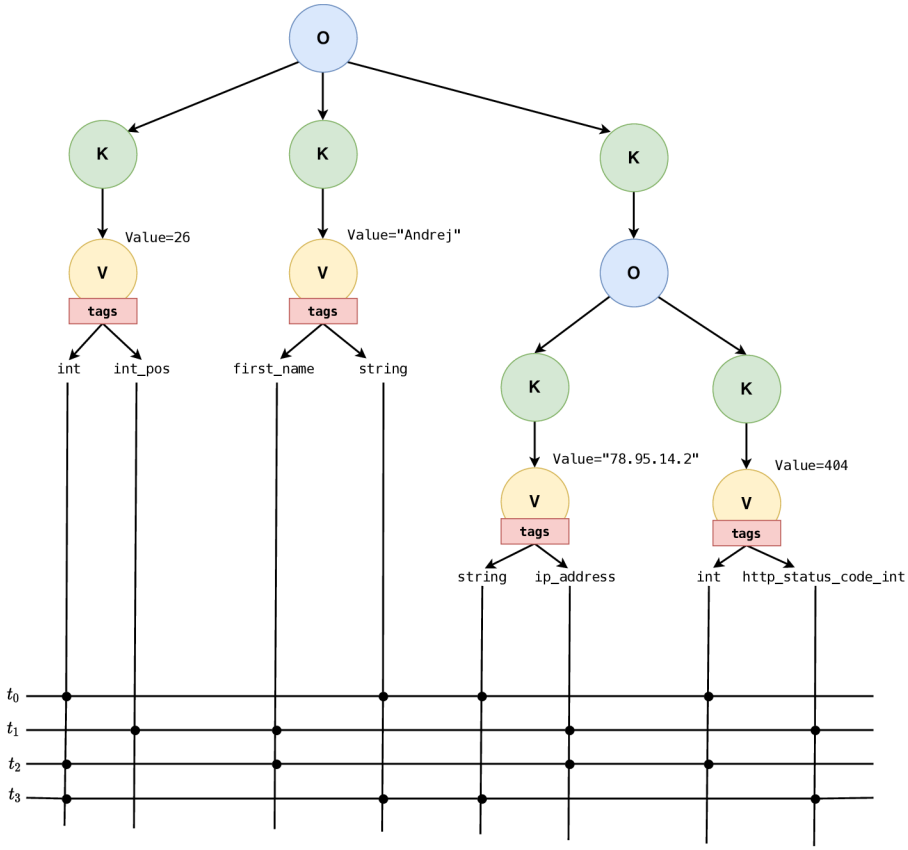
- **Náhodný výber variantnej cesty** – Podobný postup ako predošlá stratégia, avšak pri výbere z nasledovníkov variantného uzla, sa vyberie nasledovník náhodným výberom. Vo výsledku sa pre jeden strom vytvorí len jedna varianta.
- **All-Combinations pokrytie všetkých variantných ciest** – Stratégia je založená na kritériu pre pokrytie všetkých kombinácií parametrov, kde parameter je chápaný ako variantný uzol, kde pre splnenie kritéria je potrebné vytvoriť všetky kombinácie výberu nasledovníkov pre všetky variantné uzly v strome.
- **T-wise pokrytie kombinácií variantných ciest** – Stratégia využíva kritérium pre pokrytie kombinácií ciest s využitím techniky t-wise, ktoré stanovuje silu pre tvorbu kombinácií t-tic. Nie je tak potrebné kombinovať úplne všetky varianty ale len všetky t-tice variant.

Stratégie pre generovanie značiek:

- **Deterministický výber značky** – Ak nie je potrebné pokryť všetky varianty značiek priradených k hodnotovým uzlom, tak najjednoduchšou stratégiou pre výber značiek je jednoduchý výber vždy prvej podporovanej značky. Stratégia bude iteratívne prechádzať zoznam značiek, a ak narazí na značku, pre ktorú je definovaný generátor, tak sa vyberie práve táto varianta.
- **All-Combinations pokrytie všetkých značiek** – Stratégia je založená na kritériu pre pokrytie všetkých kombinácií parametrov, kde parameter je chápaný ako značka a pre splnenie kritéria je potrebné vytvoriť všetky kombinácie značiek všetkých hodnotových uzlov.
- **T-wise pokrytie kombinácií značiek** – Stratégia využíva kritérium pre pokrytie kombinácií značiek s využitím techniky t-wise, ktoré stanovuje silu pre tvorbu kombinácií t-tic. Nie je tak potrebné kombinovať úplne všetky varianty, ale len všetky t-tice variant.

Generovanie podštruktúr v stromoch

Polia a objekty tvoria základné podštruktúry v *abstraktných stromoch* a ich syntéza otvára priestor na viacero možností, ako tieto podštruktúry generovať. V prvom rade je možné zhodnotiť, že uzly typu `O` a `A`, čiže uzly reprezentujúce objekty a polia, majú dynamické množstvo nasledovníkov. Pri objektoch, sa jedná vždy iba o uzly typu `K`, ktoré ďalej nadväzujú na buď ďalšiu podštruktúru alebo uzol `V` s hodnotou. Pri poliach môže byť nasledovník priamo ďalšia podštruktúra alebo uzol `V` s hodnotou. V istom zmysle sú teda obidve podštruktúry podobné, a to tým, že obsahujú premenné množstvo nasledovných uzlov. Pri generovovaní by sa malo zohľadniť aj množstvo položiek v poliach a objektoch. Na druhej strane je potrebné poznamenať, že ak zmeníme počet položiek, či už v poliach alebo v objektoch, tak tým meníme samotnú štruktúru stromu. Preto by malo byť umožnené variovať počet nasledovníkov v poliach a objektoch podľa uváženia užívateľa. Samotná zmena počtu má taktiež viacero podôb, počet položiek môže byť zmenšený, zväčšený alebo vynulovaný. V neposlednom rade by mali byť tieto varianty zmien nad poliami/objektmi možné kombinovať a generovať podštruktúry vzhľadom k zadanému kritériu.



Obr. 5.10: Tvorba testovacích prípadov na základe kombinácií značiek v hodnotových uzloch. Na obrázku sú zobrazené len prvé 4 testovacie prípady.

Stratégie pre generovanie objektov:

a) **Zachovanie pôvodnej štruktúry objektu** – Keďže objekty sú istým spôsobom piliermi pre stanovenie štruktúry stromu, je vzhľadom k snahe generovať podobné dáta vhodné poskytnúť možnosť ich štruktúru pri generovaní plnohodnotne zachovať. V prípade zvolenia tejto stratégie prebehne izomorfná transformácia, pri ktorej sa všetky nasledovné kľúčové uzly objektového uzla zachovajú v pôvodnej podobe.

b) **Náhodné vynechanie nasledovníkov objektu** – Alternatívne je však možné štruktúru objektov do určitej miery redukovať. Z toho dôvodu je navrhnutá stratégia pre generovanie objektov tak, že sa pri zostupe stromom určití nasledovníci vynechajú. Pri výbere toho, ktorí nasledovníci sa vynechajú, rozhoduje náhodný výber, pri ktorom sa spomedzi nasledovníkov objektu vyberie náhodny počet uzlov, ktoré sa zachovajú a zvyšné sa pri generovaní budú ignorovať.

c) **Maskovanie kľúčov** – Doplnkovou stratégiou pri generovaní objektov je možnosť zvoliť názvy kľúčových uzlov, ktoré by sa do výsledných dát nemali dostať. Pre tento účel je využité filtrovanie, ktoré všetky kľúčové uzly so zvoleným názvom pri generovaní bude ignorovať.

Stratégie pre generovanie polí:

a) **Replikovanie počtu položiek** – Použije sa presná predloha *abstraktného stromu* a počet položiek sa tak zachová.

b) **Náhodny počet položiek** – Počet položiek v poli sa upraví na náhodnú hodnotu avšak v blízkom rozsahu od pôvodného počtu.

c) **Systematické kombinovanie hraničných hodnôt** – Ďalšiou možnou stratégiou pri generovaní polí je vytvorenie charakteristických blokov pre počet položiek a vytvorenie všetkých kombinácií pre tieto bloky. Základné bloky predstavujú prípady, keď dané pole obsahuje nula, jednu a viac ako jednu položku.

Generovanie atomických hodnôt

Poslednou fázou je priechod cez už konkretizovaný strom a pre každý uzol s hodnotou sa podľa identickej pozície v štruktúre *abstraktného stromu* vygeneruje hodnota pomocou priradeného generátora. Generátor sa k hodnotovému uzlu priradí na základe *značky* (tag) a v prípade, že žiadne značky uzol neobsahuje, bude použitá pôvodná hodnota uložená v danom uzle.

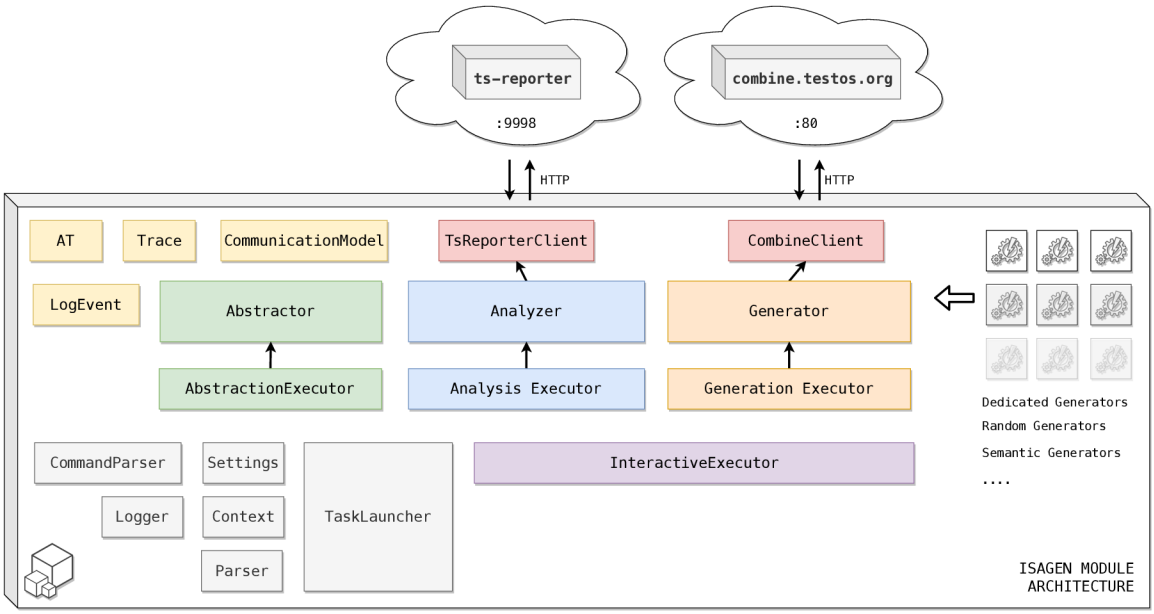
Konkrétne navrhnuté a priamo implementované generátory sú popísané v kapitole 6. Jednoducho je však možné povedať, že generátory sa dajú rozdeliť podľa toho, aký typ dát generujú. Ak majú dáta určitý sémantický príznak, bude využitá kategorická skupina sémantických generátorov. Pre jednoduché len typovo špecifické dáta, budú použité náhodne generátory. Pre značky reprezentujúce konštantné hodnoty, ktoré majú len jednu legálnu hodnotu, budú použité tzv. imperatívne, čiže príkazové generátory, ktorých úlohou je jednoducho vytvoriť inštanciu konštantnej hodnoty. Pre parametrické značky v tvare `s(reťazec)`, kde `reťazec` predstavuje konkrétnu hodnotu, ktorá by mala byť vygenerovaná, budú použité tzv. parametrické generátory. Okrem toho bude nástroj implementovať aj generátor pre regulárne výrazy.

Výstupy programu – Testovacie dáta

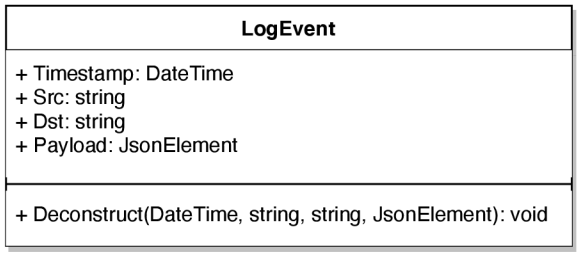
Výstupmi programu budú *testovacie dáta*, respektíve testovacie prípady, s ktorými bude možné uskutočniť testovacie behy. Ich množstvo závisí od vstupných dát – záznamov reálnej komunikácie, nastaveniach pre generovanie a hlavne zvolené kombinačné kritérium. Dáta budú v základnej podobe programu exportované do podoby serializačného formátu `JSON`.

5.3 Návrh celkovej architektúry a komponentov

Táto sekcia popisuje návrh architektúry pre nástroj ISAGEN spoločne s návrhom vnútornej štruktúry a rozhraní komponentov programového riešenia pre nástroj. Diagramy tried predstavujú len prehľad o vzťahoch jednotlivých súvisiacich tried. Ich zaznačené vzťahy preto nevyjadrujú skutočnú aritu alebo iné vzťahové vlastnosti medzi triedami. Viac o fungovaní nástroja, jeho spustení je možné nájsť v repozitári projektu ISAGEN[2].

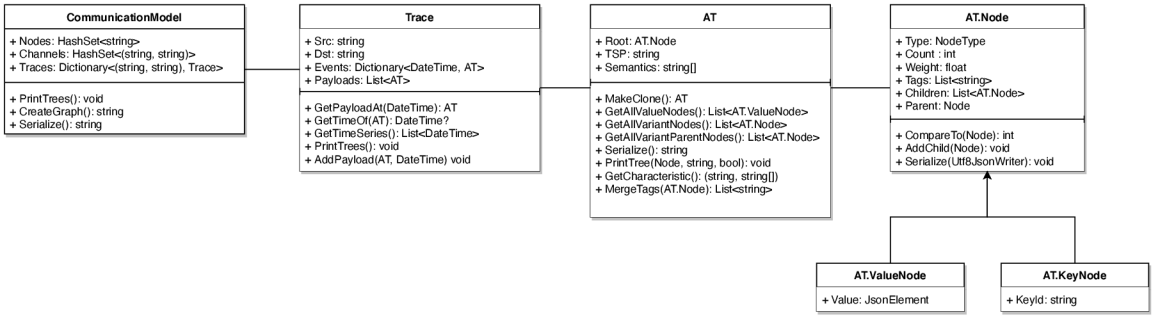


Obr. 5.11: Návrh architektúry pre nástroj ISAGEN.

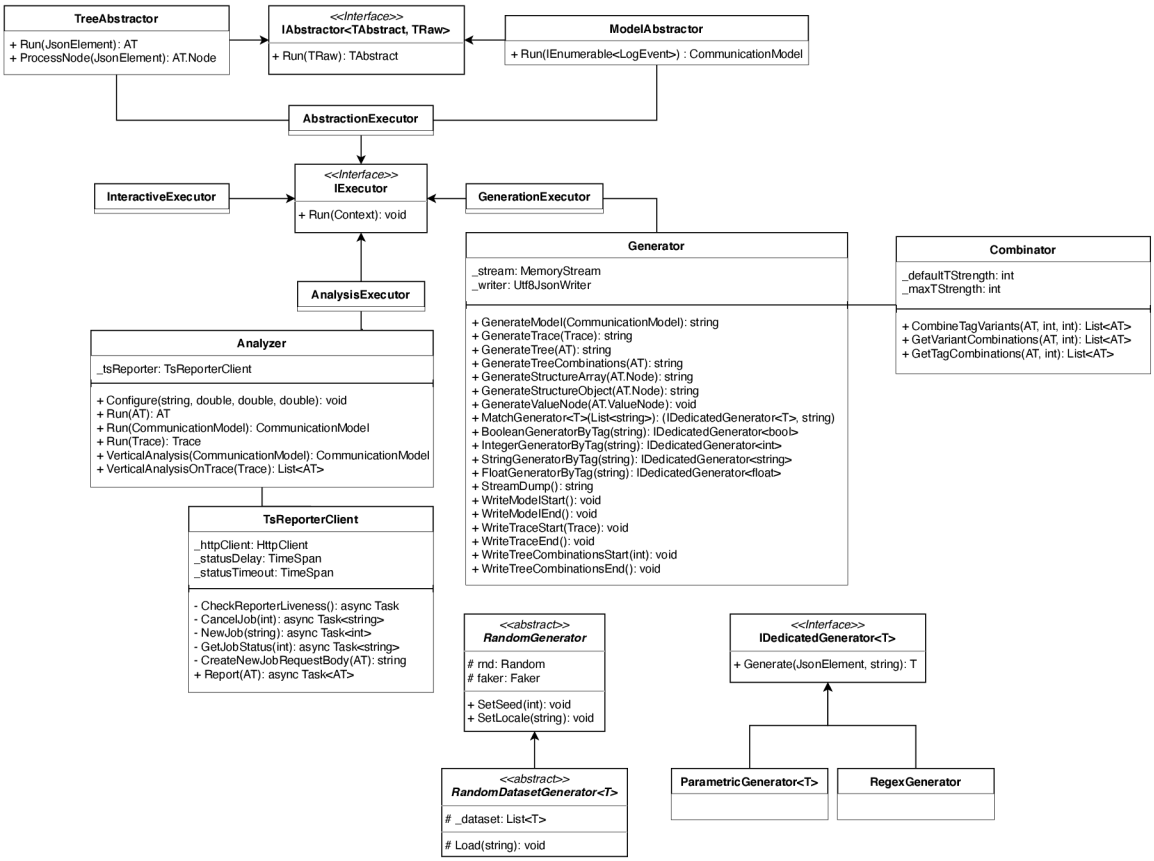


Obr. 5.12: Rozhranie triedy `LogEvent` reprezentujúcej jednu modelovanú udalosť z komunikačného logu. Základnými atribútmi sú časový moment, zdrojový a cieľový identifikátor komunikačných uzlov a samotné dáta prenášané v komunikácii zabalené ako JSON dáta.

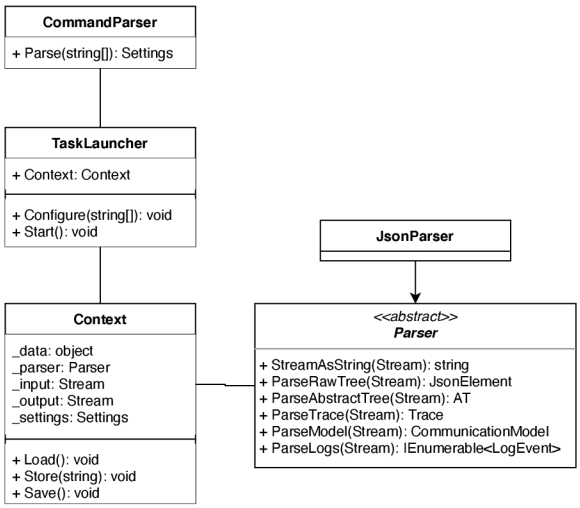
[2]Detailný popis interných častí projektu spoločne s návodom na spustenie sa nachádza v dokumente README v repozitári projektu dostupnom na: https://pajda.fit.vutbr.cz/testos/isagen.



Obr. 5.13: Diagram tried pre základné modelové reprezentácie dátových štruktúr používaných v nástroji. Diagram popisuje komunikačný model, stopu udalostí, abstraktný strom, uzol abstraktného stromu a dva špecializované typy pre uzol abstraktného stromu: hodnotový uzol a kľúčový uzol.



Obr. 5.14: Diagram tried obsahujúci najdôležitejšie triedy pre exekutívnu činnosť nástroja. Centrálnym komponentom je exekútor zvoleného typu implementujúci rozhranie `IExecutor`. Exekútori následne využívajú potrebné komponenty na spracovanie dát pre vykonanie abstrakcie, analýzy a generovania.



Obr. 5.15: Diagram tried popisujúci triedy dôležité pre obsluhu vedľajších činností nástroja, ako je spracovanie argumentov a vstupu, udržiavanie kontextu pre generovnaie, čítanie zo vstupu a zapisovanie na výstup.

Kapitola 6

ISAGEN – Implementácia, testovanie a vyhodnotenie

Zmyslom tejto kapitoly je popísať implementačné vlastnosti nástroja ISAGEN a jeho vnútorných komponentov. Sekcia 6.1 popisuje najdôležitejšie technológie, ktoré boli použité pri vývoji nástroja spoločne s dôvodmi ich výberu. Sekcia 6.2 popisuje najdôležitejšie časti programu, typicky riešiace významný problém pre úspešné fungovanie nástroja. V sekcii sú vymenované najdôležitejšie algoritmy programu a myšlienky, na ktorých sú založené. Záver kapitoly popisuje záverečné vyhodnotenie výsledného projektu ISAGEN z pohľadu splnenia požiadaviek, výkonnosti a efektivity, no taktiež poskytuje náhľad na budúcnosť projektu, možné zlepšenia a cesty, ktorými by sa projekt mohol uberať aj mimo rámca tejto práce.

6.1 Použité technológie

Pre vytvorenie prakticky použiteľného riešenia boli brané do úvahy viaceré faktory vyplývajúce z požiadaviek projektu a kontextu pre vývoj a finálne nasadenie. Od začiatku vývoja bolo nutným predpokladom zabezpečenie bezproblémovej integrácie v rámci platformy Testos a umožnenie hladkého priebehu vývoja v budúcnosti. V priebehu vývoja vznikla aj okrem iného požiadavka na integráciu nástroja ts-reporter spolu s nástrojom ISAGEN do jedného spoločného systému. Okrajovým faktorom bolo taktiež využitie takej implementačnej technológie, ktorá by zabezpečila výkonný beh programu aj pri komplexných a veľkých vstupoch.

.NET a C#

Ako hlavný implementačný programovací jazyk bol zvolený jazyk C# a vývojová platforma .NET[1]. Objektovo orientované konštrukcie jazyka umožňujú vhodne abstrahovať navrhnuté modely a práca s nimi je vo veľkej miere moduláry, a tak použiteľná pre budúce rozšírenie.

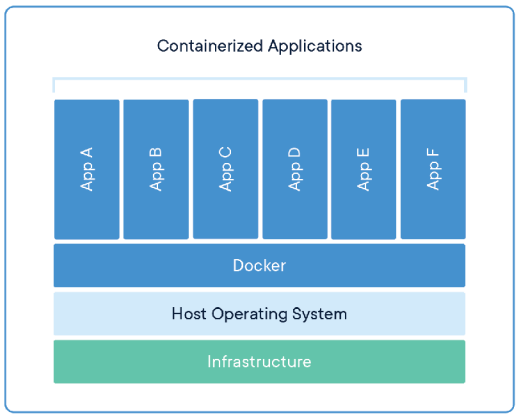
Knižnica jazyka C# poskytuje taktiež veľmi široké možnosti práce s dátovými štruktúrami vo formáte JSON pomocou knižnice `System.Text.Json`. Za spomenutie stojí minimálne

[1] https://dotnet.microsoft.com/

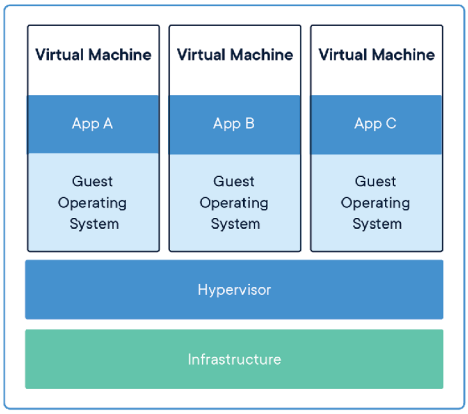
sofistikované rozhranie pre serializáciu a deserializáciu JSON reťazcov z/na interne definované abstraktné dátové štruktúry. Toto rozhranie umožňuje navyše aj možnosť dodefinovať vlastnú implementáciu pre špecifické zložité prípady tzv. *polymorfnej deserializácie*, kedy je JSON reťazec transformovaný (deserializovaný) na triedy v jazyku C#, ktoré medzi sebou hierarchicky dedia. Okrem toho boli v implementácii využité knižnice `Bogus/Faker`[2] pre uľahčenie algoritmického generovania náhodných atomických hodnôt. Pre vytvorenie generátora regulárnych výrazov bola zas využitá knižnica `Fare`[3] ako C# verzia knižnice `Xeger`[4]

Docker

Pre súčasné automatizované nasadenie nástrojov ISAGEN a TS-REPORTER bola v projekte využitá kontajnerizácia technológiou *Docker*[5], ktorá umožňuje nezávisle od seba nasadiť obidva projekty do preddefinovaného prostredia. Docker funguje na príncipe kontajnerizácie, kde kontajnery zdieľajú nielen fyzické prostriedky, ale taktiež priamo prostriedky hostitelského operačného systému a taktiež dostupné knižnice, proti tradičným virtuálnym strojom na príncipe *Hypervisor-based VM*, ktoré poskytujú abstrakciu len na úrovni hardware.



Obr. 6.1: Kontajnerizácia.



Obr. 6.2: Virtuálny stroj.

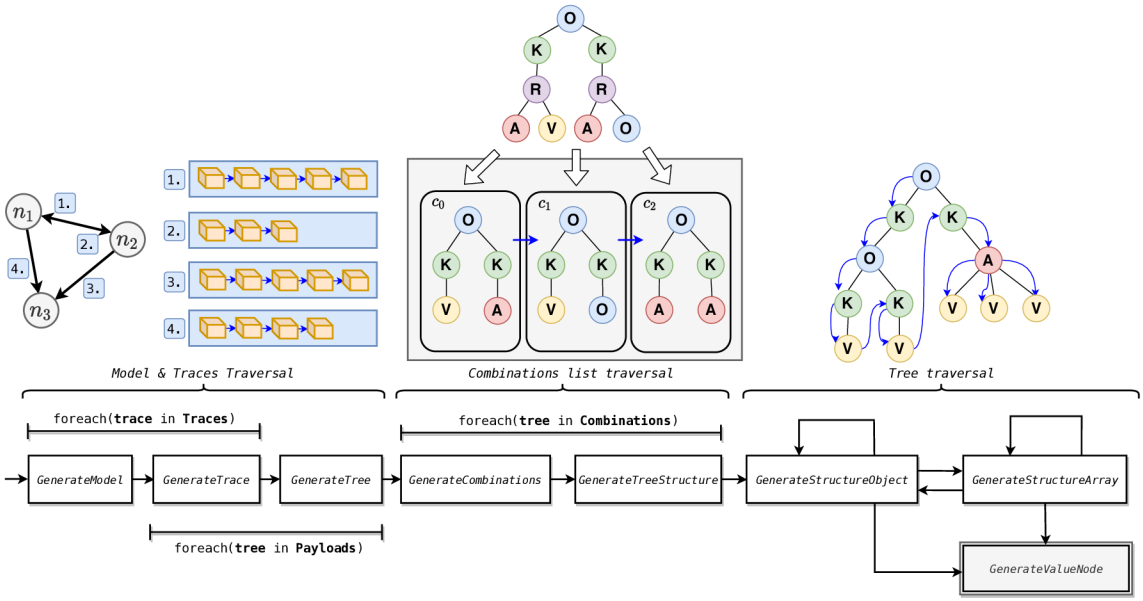
Obrázky 6.1 a 6.2 sú prevzaté z [https://docs.docker.com/].

[2]Repozitár je dostupný na: https://github.com/bchavez/Bogus.
[3]Repozitár je dostupný na https://github.com/moodmosaic/Fare.
[4]Knižnica je dostupná na https://code.google.com/archive/p/xeger/.
[5]https://www.docker.com/

6.2 Najdôležitejšie problémy a algoritmy

Táto sekcia popisuje najdôležitejšie komponenty nástroja, algoritmy a myšlienky, ktoré boli pre ich vznik potrebné a popis základných princípov fungovania.

Proces generovania dát Pre zabezpečenie plynulého chodu v procese generovania bolo potrebné rozložiť problém na niekoľko menších častí. Generovanie začne na úrovni komunikačného modelu, následne zostupuje na úroveň stopy (každej stopy v modeli), ďalej sa prechádzajú všetky stromy v stope, pre každý strom sa spustí proces tvorby kombinácií vzhľadom na zvolené kritérium, a pre každú kombináciu sa na záver postupným prechodom cez abstraktný strom generujú konkrétne hodnoty uzlov. Celý proces je znázornený na obrázku 6.3. Priebeh procesu má iteratívne prehlbujúci a na záver rekurzívny charakter. Pri generovaní konkrétnych hodnôt stromom sa rekurzívna hierarchia volaní zastaví až v momente, keď narazí na hodnotový uzol alebo akýkoľvek uzol, ktorý nemá ďalších nasledovníkov.



Obr. 6.3: Proces generovania dát.

Testovanie izomorfizmu a miery podobnosti Medzi výstupy analýzy patrí aj charakteristika stromu, ktorá obsahuje reťazec `TSP` (Tree Structure Property), popisujúci jednoznačným spôsobom abstraktnú štruktúru stromu. Dôležitou vlastnosťou `TSP` je, že nezáleží pri tvorbe reťazca na tom, z akých typov je strom zložený alebo aké hodnoty ukladá, ale iba na čistom tvare. Pre vytvorenie reťazca `TSP` je využitá metóda algoritmu `AHU` [40], ktorý je založený na princípe rekurzívneho pomenovávania uzlov stromu hierarchicky tak, aby každý uzol stromu mal pomenovanie reprezentujúce celú jeho históriu nasledovníkov postupne až smerom hore ku koreňu, a zároveň aby výsledný reťazec jednoznačne pomenovával tvar celého stromu. Pre overenie izomorfného vzťahu medzi štruktúrami dvoch stromov je k dispozícii dvojica algoritmov 6.1 a 6.2. Prvý slúži na rekurzívne pridelenie kanonického tvaru

pomenovania pre každý uzol v strome a druhý na overenie izomorfizmu dvoch stromov. Varianta implementovaná v tomto projekte má kvadratickú výpočetnú zložitosť $O(|V|^2)$ kde V predstavuje množinu uzlov v strome. Algoritmus je však možné vylepšiť na variantu, ktorá by dokázala overovať izomorfnú vlastnosť dvoch stromov aj v lineárnom čase $O(|V|)$, tento problém však bol jeden z vedľajších problémov tejto práce, a preto bola zvolená pre demonštráciu len jednoduchšia varianta. Viac o `AHU` algoritme a vylepšení popisujú publikácie [40, 18]. V prípade, že ale nie sú dva stromy úplne štrukturálne identické, no napriek tomu by sa o nich dalo povedať, že sú veľmi podobné, by algoritmus ako `AHU`, ktorý jednoducho vyhodnotí, či dva stromy sú alebo nie sú izomorfné, rozhodol odmietnutím. Práve preto je v projekte zavedená alternatívna miera pre vyhodnotenie podobnosti dvoch stromových štruktúr. Opäť sa využíva reťazec vytvorený algoritmom `ASSIGN-CANONICAL-NAMES`, no pre vyhodnotenie podobnosti je miesto algoritmu `AHU-TREE-ISOMORPHISM` zavedená metrika vzdialenosti reťazcov pomocou tzv. *Levenshteinovej vzdialenosti* (a taktiež Levenshteinovho pomeru). *Levenshteinová vzdialnosť* (taktiež aj vzdialenosť úprav) vyjadruje minimálne množstvo potrebných úprav pre transformáciu jedného reťazca na druhý (t.j. aby boli rovnaké). Pri transformácii sú povolené úpravy vo forme vloženia, zmazania a zámeny znaku [32].

Algoritmus 6.1: `ASSIGN-CANONICAL-NAMES` (Algoritmus prevzatý z [40])

```
   Input: v
   Result: Name is assigned to the node v
 1 if v is a leaf then
 2 |  name(v) ← '10';
 3 else
 4 |  for all children w in v do
 5 |  |  ASSIGN-CANONICAL-NAMES(w);
 6 |  end
 7 end
 8 Sort the names of the children of v;
 9 Concatenate the names of all children of v to temp;
10 name(v) ← 1temp0;
11 return;
```

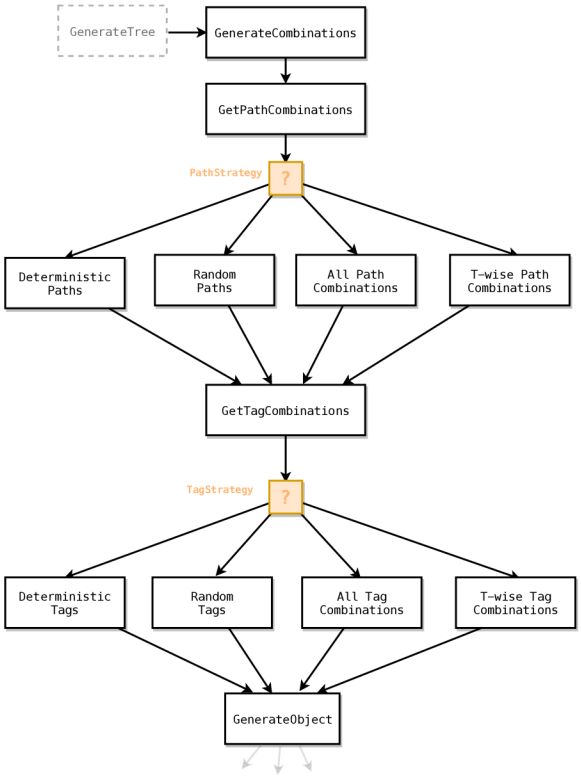
Algoritmus 6.2: `AHU-TREE-ISOMORPHISM` (Algoritmus prevzatý z [40])

```
   Input: two trees (T1, T2)
   Output: boolean {True, False}
 1 r1 ← root(T1);
 2 r2 ← root(T2);
 3 ASSIGN-CANONICAL-NAMES(r1);
 4 ASSIGN-CANONICAL-NAMES(r2);
 5 if name(r1) = name(r2) then
 6 |  return True;
 7 else
 8 |  return False;
 9 end
```

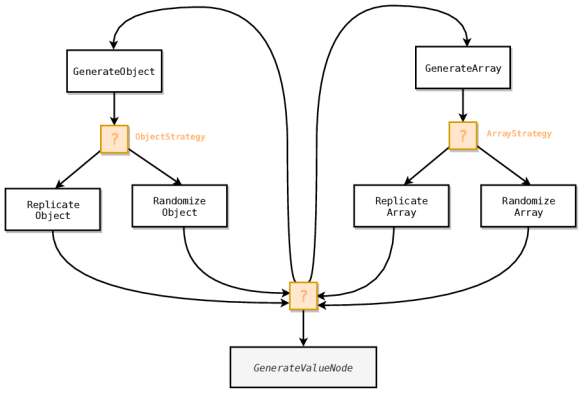
Kombinovanie Medzi najhlavnejšie sekundárne funkcie generátora patrí možnosť kombinovať variantné alebo nejednoznačné zložky *abstraktných stromov*. Variantné zložky *abstraktného stromu* sú jeho variantné cesty, zaznačené pomocou variantných R uzlov. Sémanticky predstavujú možné variácie štruktúry stromu a využívajú sa na zjednodušenie zápisu pre podobné stromy, ktoré majú malé odlišnosti. Nejednoznačné zložky abstraktného stromu sú napríklad situácie, keď je uzlu s hodnotou priradených niekoľko sémanticky odlišných značiek (napríklad ak má úzol súčasne sémantickú značku prvočíslo *prime_number* a kód krajiny *state_code_int*). Pre tieto situácie umožňuje kombinátor navzájom kombinovať všetky možné varianty a vytvoriť tak rozsiahlu množinu variant pre syntetické testovacie dáta.

Funkcia kombinátora je rozdelená na dve metódy. Prvá metóda kombinuje sémantické značky (`GetTagCombinations`) a druhá kombinuje variantné cesty (`GetVariantPaths`). Existuje však aj tretia možnosť, ktorá sekvenčne aplikuje kombinovanie variantných ciest a následne na výslednú kolekciu kombinácií aplikuje ešte dodatočne aj kombinátor značiek. Algoritmus pre kombinovanie vzhľadom ku kombinačnému kritériu vychádza z *T-wise* kritéria pre pokrytie vstupov. Celočíselný parameter T určuje, aký má byť stupeň kombinovania, respektíve koľko najpočetnejších vstupných parametrov by malo byť kompletne navzájom skombinovaných. Pre jednoduchosť bude uvedené vysvetlenie predpokladať metódu kombinovania značiek, od metódy kombinovania variantných ciest sa výrazne neodlišuje. Každý abstraktný strom má v svojich listových hodnotových uzloch uložene hodnoty, ktorým bola v predchádzajúcej fáze analýzy priradená značka, popisujúca ich sémantiku. Značiek môže byť pre každý hodnotový uzol priradených viac. V procese generovania je abstraktný strom premieňaný na konkrétny (JSON) strom a informáciu o jeho sémantických vlastnostiach je potrebné premeniť v konkrétnu instanciu hodnoty.

Ak sa na syntézu jedného stromu nahliada ako na tvorbu jedného testovacieho prípadu, dalo by sa zhodnotiť, že testovací prípad zastrešuje toľko vstupných parametrov, koľko má strom hodnôt. Ak ešte nie je k dispozícii konkrétny vygenerovaný strom, obdobne sa dá povedať, že počet hodnotových V uzlov v abstraktnom strome odráža počet vstupných parametrov pre tvorbu jedného testovacieho prípadu. Na rozdiel od štandardného kombinačného testovania však nie je záujmom vytvárať kombinácie konkrétnych hodnôt pre každý parameter, ale miesto toho kombinovať sémantické kategórie, definované značkami v hodnotových uzloch.



Obr. 6.4: Rozhodovací strom pre výber kombinačných stratégií v procese generovania.



Obr. 6.5: Rozhodovací strom pre výber stratégií pri generovaní podštruktúr v strome.

Využitie nástroja Combine Pre realizáciu T-wise kombinovania značiek alebo variantných ciest v programe je využitý externý nástroj COMBINE vytvorený v platforme *Testos* pre vykonávanie kombinatorického testovania na základe princípu t-wise, kombinujúceho všetky t-tice medzi parametrami. Pri využití v nástroji ISAGEN je parametrom myslená množina tzv. *špekulatívnych hodnotových uzlov*, čo sú klasické hodnotové uzly *abstraktného stromu* (typu V) avšak také, ktoré súčasne spĺňajú podmienku, že počet značiek (tagov) v uzle je väčší alebo rovný dvom. Taktiež platí, že samotné kombinovanie môže prebehnúť vždy nad aspoň dvoma parametrami, čiže inak povedané, aby bolo kombinovanie možné uskutočniť, musí mať strom aspoň dva špekulatívne hodnotové uzly a každý takýto uzol musí mať aspoň dve značky. Kombinovanie má hlavne zmysel ako prostriedok pre *systematickú determináciu* procesu výberu z množiny značiek priradených jednému uzlu. Obdobne to platí aj pre variantné cesty a ich tvorbu kombinácií.

V_1	V_2	V_3	V_4
"string" "ip_address"	"int" "int_pos"	"float" "float_neg"	"city" "string" "s(Brno)"

Tabuľka 6.1: Parametrami sú značky priradené hodnotovým uzlom (V_1, V_2, V_3, V_4).

T-max kombinátor (pôvodná verzia) V priebehu práce na projekte bolo jedno z menej vo výsledku relevantných rozhodnutí vytvorenie samostatného riešenia pre kombinovanie parametrov stromu. Spätne toto rozhodnutie možno brať ako istým spôsobom naivné, vzhľadom k už dlhodobo existujúcemu riešeniu v podobe nástroja COMBINE.

Pozitívným dôsledkom tejto snahy však bolo plnohodnotné pochopenie rozdielu medzi *t-wise* testovaním a obyčajným kombinovaním parametrov. Pôvodný kombinátor totiž v svojej základnej podobe poskytoval, dá sa povedať, pseudo-t-wise kombinovanie v tom zmysle, že kombinoval parametre na základe určitého parametru T, ale takým spôsobom, že sa kombinovalo vždy iba T najväčších parametrov. T-wise testovanie však vyžaduje, aby boli testované všetky t-tice hodnôt zo všetkých parametrov, čo je v konečnom dôsledku úplne odlišná charakteristika kombinácií.

Pri návrhu pôvodného kombinátora sa problém javil ako jednoduchší ako v skutočnosti je. Nech $\mathcal{P}$ je množina N parametrov, kde *parameter* je množina *značiek* hodnotového uzla v abstraktnom strome $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$. Rozsah parametru, čiže všetky jeho možnosti značiek, bude označený ako $R(P_i)$ pre nejaký parameter z $\mathcal{P}$. Ďalej veľkosť alebo

V_1		V_2		V_3		V_4	
tag	**value**	**tag**	**value**	**tag**	**value**	**tag**	**value**
"string"	"absafac"	"int"	232	"float"	2.4242421	"city"	*"Tokyo"*
"ip_address"	"124.32.23.10"	ïnt_pos"	120	"float_neg"	-59.234151	"city"	"Berlin"
'string"	"bb2ag"	"int_pos"	795	"float"	6.0024214	"string"	"xAAAAg"
"ip_address"	"97.5.16.109"	"int"	2121515	"float_neg"	-3.72311	"string"	"b"
"string"	"cc02aaaagg"	"int"	-1295912	"float_neg"	-108.2424	"s(Brno)"	"Brno"
"ip_address"	"78.102.20.15"	"int_pos"	1302	"float"	0.512131	"s(Brno)"	"Brno"

Tabuľka 6.2: Vytvorené kombinácie pri sile $t = 2$.

počet možností parametra P_i je $|R(P_i)| = n_i$. Ak by bolo potrebné vytvoriť všetky možné kombinácie značiek z parametrov, ich počet je možné vyjadriť ako produkt $\prod n_i$.

$$\mathcal{P} = \{P_1, P_2, \cdots, P_N\}, N = |\mathcal{P}|,$$
$$\mathcal{P}_s =< P_i >: i < j \implies |Range(P_i)| < |Range(P_j)|$$
$$X(T) \subset \mathcal{P}_s = \{P_{N-1}, P_{N-2}, \ldots P_{N-T}\}$$
$$\mathcal{A} \subset \mathcal{P}_s : P_i \in \mathcal{P}_s : i < n$$
$$\forall n \in (1, N) : index(P_n) = n / \prod A\% |P_n|$$

V prípade, že by v strome bolo napríklad 5 hodnotových uzlov, každý so 4 značkami, celkový počet kombinácií by bol $4^5 = 1024$ len pre jeden abstraktný strom. Pri jednotlivých stromoch s podobným rozsahom hodnotových uzlov a značiek je to ešte relatívne spracovateľné množstvo kombinácií, ale vzhľadom k tomu, že primárnou funkciou nástroja je prijímanie rozsiahlych logov komunikácie, kde sa očakáva sekvencia stromových dát často s počtom hodnotových uzlov ďaleko rozsiahlejším ako 5 uzlov, nie je generovanie všetkých teoreticky možných kombinácií značiek nutne želaným cieľom. Z toho dôvodu je možné zaviesť stupeň pre kombinovanie T, ktorý množstvo kombinácií výrazne redukuje, a napriek tomu efektívne kombinuje najviac početné parametre.

Problémom tejto metódy je, že vyberie vždy len všetky kombinácie daných T najväčších parametrov, ale nie všetky t-tice všetkých parametrov. V tomto bol základný omyl pri návrhu metódy a chyba sa odhalila až pri testovaní. Na prvý pohľad totiž aj pri základných manuálnych testoch pôsobilo, že táto metóda generuje predsa len všetky kombinácie t-tic. Išlo však len o výnimočný prípad, keď bola kombinácia zvolených parametrov navzájom veľkostne výrazne rozdielna a tak implicitne prebehlo aj kombinovanie všetkých t-tic. Pre komplexnejšie prípady to však neplatilo, a pri porovnaní napríklad s nástrojom COMBINE, bol počet vytvorených kombinácií opakovane menší. Z toho dôvodu bola metóda nahradená využitím externého nástroja COMBINE, ktorý vykonáva kombinovanie zaručene kvalitnejšie.

6.3 Dedikované generátory

Nástroj ISAGEN vo výslednej podobe obsahuje niekoľko kategórií generických generátorov pre atomické hodnoty. Tieto generátory sú v projekte označované ako *dedikované generátory* a kategoricky ich možno rozdeliť na *imperatívne*, *parametrické*, *náhodné* a *sémantické* generátory a špeciálnym typom generátora je *generátor regulárnych výrazov*.

- **Imperatívne generátory** – Generátory zabezpečujúce generovanie konštantných hodnôt.
- **Parametrické generátory** – Generátory zabezpečujúce replikovanie v značkách explicitne zaznačených parametrických hodnôt.
- **Náhodné generátory** – Generátory, ktoré náhodne vyberajú hodnoty z domény daného typu.
- **Sémantické generátory** – Generátory, ktoré nahodne vyberajú hodnoty z danej sémantickej kategórie pričom využívajú staticky definované datasety alebo obmedzenia pre formát a rozsah hodnôt.
- **Generátory regulárnych výrazov** – Generátory, ktoré využívajú predpis v podobe regulárneho výrazu na vytvorenie reťazcov, ktoré dané regulárne výrazy spĺňajú.

Imperatívne generátory Imperatívne generátory sú najjednoduchšie generátory nástroja, slúžia na jednoduché vytvorenie hodnoty daného typu, pre ktoré existuje iba jedna možná hodnota, ktorá by spĺňala obmedzenie dané výrazom (napr. celé číslo s hodnotou nula, pravdivý booleovský výraz, ...).

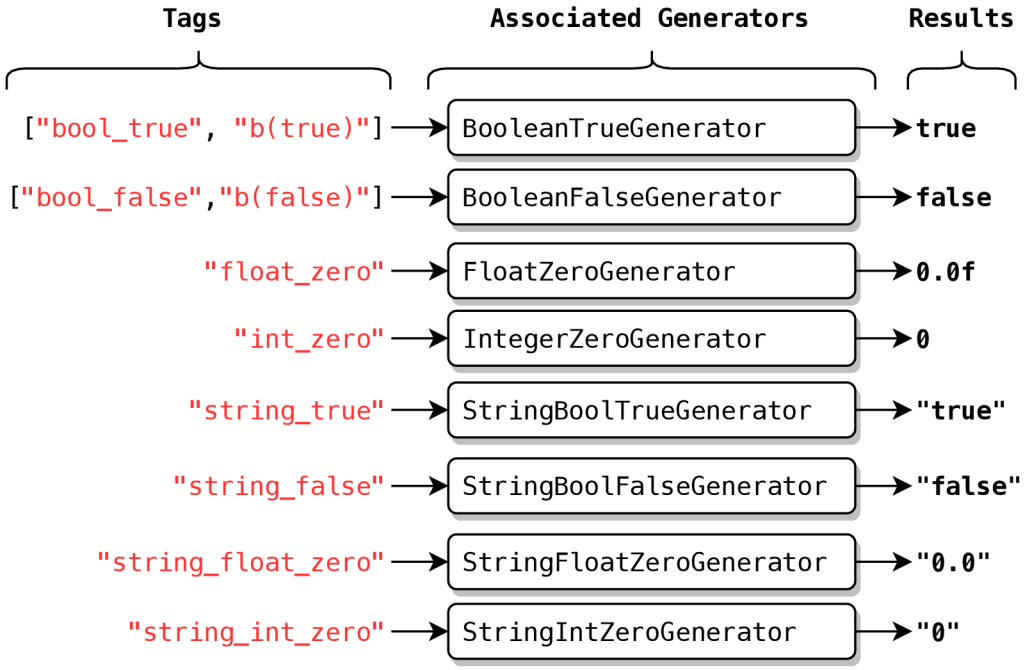
Parametrické generátory Parametrické generátory slúžia na replikovanie pôvodnej hodnoty a umožňujú reágovať na takzvané parametrické značky, ktoré vždy pre daný typ hodnoty umožňujú doplniť konkrétnu hodnotu, ktorá by sa mala replikovať do zátvoriek v reťazci značky.

Náhodné generátory Náhodné generátory sú zamerané na sémanticky nevýznamné generovanie hodnôt, ktoré jednoducho spadajú do jednoduchých typových kategórii alebo podmnožín definičného oboru pre dané typy. Patria sem generátory celých a desatiných čísiel, reťazcov a booleovských typov pre ktoré platí určité jednoduché obmedzenie (napr. celé číslo musí byť záporné).

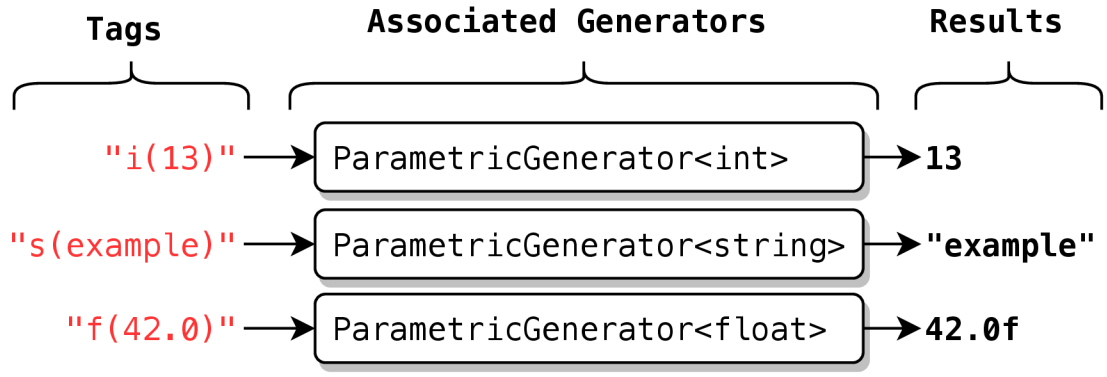
Sémantické generátory Sémantické generátory sú špecializované dedikované generátory, ktorých náplňou je vytvárať hodnoty, ktoré patria do určitej sémantickej kategórie. Kategória je daná parametrom `tag`, ktorý jasne a deterministicky priraďuje konkrétny sémantický generátor, ktorý typicky pracuje dvoma spôsobmi. V prvom rade, ak je sémantická kategória reprezentovaná určitým konečným externým datasetom, kde hodnoty patriace do kategórie definuje ich explicitné vymenovanie, tak sémantický generátor vyberie hodnotu z datasetu náhodným výberom. Alternatívne, ak sa jedná o nekonečnú sémantickú kategóriu (napr. prvočíslo) alebo o konečnú ale algoritmicky definovanú sémantickú kategóriu (napr.

IP adresa) tak generátor spustí algoritmus, ktorý špecifickým spôsobom podľa staticky definovaných obmedzení vytvorí novú náhodnú hodnotu.

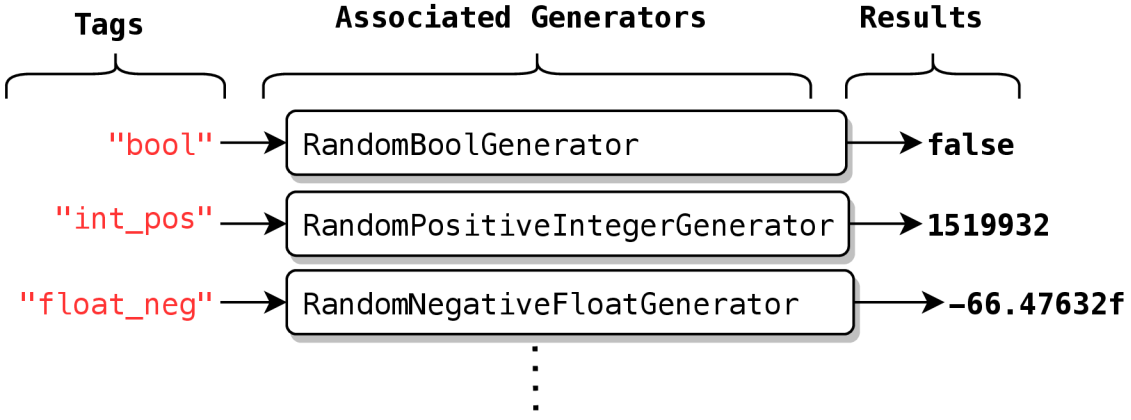
Generátor regulárnych výrazov Špeciálnym typom dedikovaného generátoru je generátor založený na regulárnych výrazoch, ktorý využíva parameter `tag` s hodnotou `regex` na vytvorenie reťazca spĺňajúceho regulárny výraz uložený v pôvodnej hodnote práve spracovávaného uzlu.



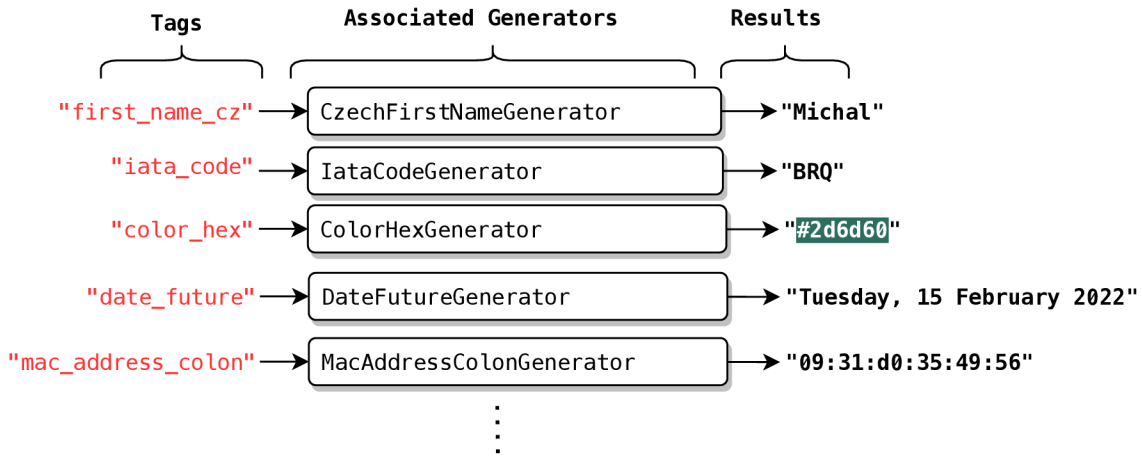
Obr. 6.6: Sada *imperatívnych generátorov* podľa zadanej značky vytvorí konštantne stanovenú hodnotu. Jedná sa o najjednoduchší typ dedikovaných generátorov pre jednotvarné hodnoty všetkých typov.



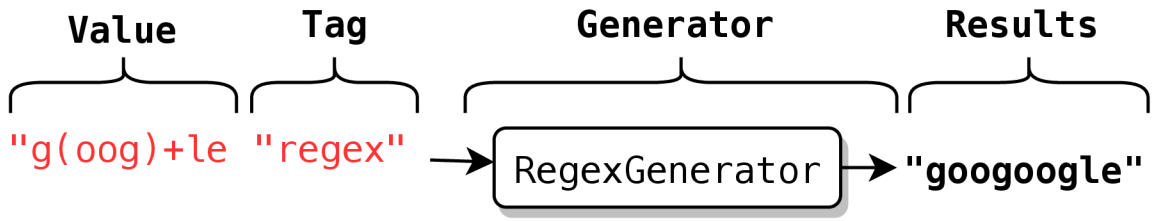
Obr. 6.7: Trojica *parametrických generátorov* priradených podľa značky vo formáte reťazca so skratkou pre typ celého čísla, desatinného čisla a reťazca (`i`, `s` a `f`) a parametrom medzi zátvorkami replikuje vždy hodnotu danú parametrom.



Obr. 6.8: Sada *náhodných generátorov* podľa zadanej značky generuje náhodné hodnoty pre všetky základné typy a charakteristiky. Nástroj celkovo poskytuje 26 náhodných generátorov.



Obr. 6.9: Sada *sémantických generátorov* priradených podľa relevantnej značky poskytuje možnosť generovať hodnoty s určitým špecifickým významom. K dispozícii je v nástroji až 48 rôznych sémantických generátorov, ktoré pre svoju funkcionalitu využívajú buď externé datasety alebo syntaktické pravidlá pre generovanie určitého sémantického typu.



Obr. 6.10: Generátor *regulárnych výrazov* je priradený k značke `regex` a na základe hodnoty reťazca uloženého v hodnotovom uzle vytvorí taký náhodný reťazec, ktorý bude spĺňať obmedzenie dané vzorom regulárneho výrazu v zadanom reťazci.

6.4 Testovanie a experimentálne vyhodnotenie nástroja

Počas vývojového procesu bol nástroj testovaný na viacerých úrovniach. Na najnižšej úrovni boli testované jednotlivé najmenšie komponentny programu ako sú triedy a metódy v nich. Tento typ testov zabezpečili jednotkové testy, umiestnené v projekte `isagen.UnitTests`. Pre integráciu viacerých komponentov nástroja, sekvenčné spracovanie dát a komunikáciu s externou službou `ts-reporter` boli zavedené integračné testy, obsiahnuté v projekte `isagen.IntegrationTests`.

Jednotkové testy

Implementáciu dopĺňa sada jednotkových testov, ktoré testujú všetky podstatné funkcie a moduly programu samostatne. Medzi dôležité testy patria napríklad jednotkové testy pre testovanie funkcionality *dedikovaných generátorov*, aby generátory spoľahlivo generovali sémantický korektné a presné dáta. Množstvo testov by sa v nadväzujúcej práci na projekte dalo značne rozšíriť, no v aktuálnej podobe pokrýva najdôležitejšie časti aplikácie a vo výsledku je vďaka nim možné zhrnúť, že všetky generátory a iné dôležité časti programu, fungujú podľa špecifikácie správne.

Integračné testy

Integračné testy boli vzhľadom na obmedzený časový rámec projektu vynechané. Charakter projektu sa sústavne menil a integračné testy boli vyhodnotené ako aktuálne menej závažný aspekt pre dokončenie projektu. Výstupom tejto práce mal byť určitým spôsobom demonštratívny projekt, ktorý v prípade, že preukáže svoju použiteľnosť, bude určite vhodné doplniť o dôkladné testovanie a integráciu s inými nástrojmi. Druhým dôvodom, prečo na integračné testy nebol kladený dôraz, je jednak premenlivosť dostupnosti služieb `ts-reporter` a `combine`, ktoré nemusia byť vždy dostupné pre testovanie. Čo sa týka integrácie modulov interne v rámci nástroja, prípadne spolupráca s lokálnym súborovým systémom, charakter aplikácie už v svojom základe všetky tieto aktivity dostatočne overuje v svojej činnosti a pri rôznorodých zlyhaniach chyby zachytáva ako výnimky a užívateľovi/vývojarovi vypisuje.

Experimentálne vyhodnotenie výkonnosti

Medzi najzásadnejšie výzvy na začiatku práce patrilo vytvorenie nástroja, ktorý by dokázal generovať dáta efektívne. V závere práce je preto vhodné pomocou výkonnostných testov zhodnotiť, ako sa nástroj správa pri odlišných veľkostiach a vlastnostiach vstupných dát.

Najzásadnejším faktorom pri generovaní dát je z pohľadu výpočetnej zložitosti kombinačný faktor. Vzhľadom na to, že sa dáta primárne pri generovaní jednoducho reprodukujú na základe predlohy v podobe abstraktného stromu, nie je výpočetná zložitosť tejto časti generačného procesu významne náročná a vzhľadom na veľkosť vstupu má na väčšine úrovní lineárnu zložitosť ($O(n)$). Avšak pri generovaní kombinácií stromov buď na základe kombinácie značiek (`tags`) alebo kombinácie variantných ciest v strome, sa výpočetná zložitosť výrazne zvyšuje.

t-strength	# tags	*1 node*				*2 nodes*				*3 nodes*			
		Mean	**Min**	**Max**	**Relative**	**Mean**	**Min**	**Max**	**Relative**	**Mean**	**Min**	**Max**	**Relative**
$t = 1$	1	649.5 ± 25.0	636.1	719.8	1.00	878.5 ± 20.6	855.1	927.7	1.35 ± 0.06	888.8 ± 17.8	873.2	932.8	1.37 ± 0.06
	2	892.5 ± 33.2	858.1	953.9	1.37 ± 0.07	1445.8 ± 144.8	1377.9	1851.6	2.23 ± 0.24	1438.5 ± 53.9	1382.5	1562.9	2.21 ± 0.12
	3	925.8 ± 27.7	881.9	981.8	1.43 ± 0.07	1418.8 ± 50.4	1370.3	1513.6	2.18 ± 0.11	1408.1 ± 24.5	1380.8	1459.8	2.17 ± 0.09
	4	799.7 ± 8.8	790.5	818.5	1.23 ± 0.05	1516.6 ± 80.3	1436.1	1679.9	2.34 ± 0.15	1530.7 ± 76.4	1455.3	1665.7	2.36 ± 0.15
	5	886.8 ± 26.8	861.1	935.5	1.37 ± 0.07	1488.9 ± 23.6	1464.4	1535.5	2.29 ± 0.10	1454.8 ± 53.1	1406.9	1578.0	2.24 ± 0.12
	6	886.0 ± 7.5	875.5	897.0	1.36 ± 0.05	1449.5 ± 35.7	1413.0	1536.6	2.23 ± 0.10	1502.0 ± 18.5	1487.1	1549.6	2.31 ± 0.09
	7	888.6 ± 18.2	873.0	924.4	1.37 ± 0.06	1481.4 ± 37.1	1452.6	1565.8	2.28 ± 0.10	1494.1 ± 105.5	1430.4	1787.8	2.30 ± 0.18
	8	884.4 ± 7.4	875.7	898.1	1.36 ± 0.05	1495.0 ± 30.0	1471.0	1564.3	2.30 ± 0.10	1550.9 ± 32.4	1505.1	1616.2	2.39 ± 0.10
	9	905.6 ± 41.6	875.7	989.1	1.39 ± 0.08	1556.6 ± 139.5	1473.5	1928.5	2.40 ± 0.23	1468.6 ± 23.4	1446.9	1518.5	2.26 ± 0.09
$t = 2$	1	657.4 ± 14.4	644.1	684.6	1.01 ± 0.04	874.5 ± 12.7	857.9	895.1	1.35 ± 0.06	909.5 ± 27.4	878.8	961.9	1.40 ± 0.07
	2	874.3 ± 19.1	857.7	920.8	1.35 ± 0.06	1396.6 ± 29.3	1372.5	1452.9	2.15 ± 0.09	1417.8 ± 49.4	1383.8	1546.2	2.18 ± 0.11
	3	880.2 ± 10.2	873.6	904.9	1.36 ± 0.05	1409.1 ± 57.4	1370.6	1541.1	2.17 ± 0.12	1432.3 ± 33.0	1397.1	1485.8	2.21 ± 0.10
	4	823.4 ± 24.9	793.1	871.2	1.27 ± 0.06	1572.8 ± 70.3	1502.5	1712.8	2.42 ± 0.14	1598.2 ± 53.6	1523.9	1727.3	2.46 ± 0.13
	5	872.2 ± 10.3	861.7	892.4	1.34 ± 0.05	1506.6 ± 58.1	1452.0	1614.5	2.32 ± 0.13	1655.8 ± 23.6	1596.2	1686.4	2.55 ± 0.10
	6	885.2 ± 9.4	869.5	899.3	1.36 ± 0.05	1678.3 ± 68.1	1591.1	1766.7	2.58 ± 0.14	1750.4 ± 94.5	1643.8	1926.0	2.70 ± 0.18
	7	888.0 ± 15.7	873.8	926.9	1.37 ± 0.06	1800.5 ± 98.5	1644.7	1980.7	2.77 ± 0.19	1702.8 ± 52.3	1663.5	1842.2	2.62 ± 0.13
	8	961.2 ± 100.2	884.1	1211.8	1.48 ± 0.16	1713.9 ± 69.2	1671.2	1900.1	2.64 ± 0.15	1971.3 ± 73.5	1884.9	2146.3	3.04 ± 0.16
	9	879.3 ± 7.6	870.7	893.2	1.35 ± 0.05	2093.4 ± 178.3	1953.3	2482.8	3.22 ± 0.30	2199.3 ± 78.0	2126.4	2381.1	3.39 ± 0.18
$t = 3$	1	671.4 ± 21.0	642.2	710.7	1.03 ± 0.05	878.7 ± 22.5	854.1	925.8	1.35 ± 0.06	884.3 ± 7.8	873.1	895.9	1.36 ± 0.05
	2	866.2 ± 14.1	851.4	895.9	1.33 ± 0.06	1393.7 ± 27.5	1354.4	1440.2	2.15 ± 0.09	1440.0 ± 33.1	1400.8	1491.0	2.22 ± 0.10
	3	880.4 ± 6.2	873.0	889.7	1.36 ± 0.05	1390.5 ± 17.9	1372.2	1433.0	2.14 ± 0.09	1468.8 ± 30.2	1433.5	1522.9	2.26 ± 0.10
	4	806.7 ± 16.8	791.0	840.9	1.24 ± 0.05	1539.8 ± 32.5	1509.3	1599.7	2.37 ± 0.10	1969.1 ± 27.9	1930.1	2004.6	3.03 ± 0.12
	5	900.0 ± 13.0	884.7	919.3	1.39 ± 0.06	1536.9 ± 70.6	1444.7	1669.1	2.37 ± 0.14	2487.5 ± 76.6	2371.1	2587.9	3.83 ± 0.19
	6	879.7 ± 6.6	869.3	890.5	1.35 ± 0.05	1600.7 ± 40.8	1569.3	1707.1	2.46 ± 0.11	2645.7 ± 51.8	2587.1	2750.9	4.07 ± 0.18
	7	901.4 ± 18.6	888.4	951.6	1.39 ± 0.06	1662.3 ± 38.5	1628.2	1720.8	2.56 ± 0.11	3240.1 ± 123.1	3053.0	3502.7	4.99 ± 0.27
	8	897.5 ± 13.2	877.2	922.7	1.38 ± 0.06	1702.2 ± 31.2	1659.3	1762.2	2.62 ± 0.11	5063.5 ± 65.5	4979.6	5179.5	7.80 ± 0.32
	9	888.0 ± 15.8	873.8	923.1	1.37 ± 0.06	1964.1 ± 34.3	1912.2	2013.7	3.02 ± 0.13	6948.9 ± 120.7	6743.0	7142.9	10.70 ± 0.45

Tabuľka 6.3: Testovanie výkonnosti implementácie nástroja ISAGEN vzhľadom na počet uzlov v strome, počet značiek v každom uzle a zvolenej sile pri t-wise kombinovaní. Všetky časy sú uvedené v milisekundách. Testovanie bolo vykonané kombináciou manuálneho skriptu a utility `hyperfine`(https://github.com/sharkdp/hyperfine).

6.5 Zhodnotenie a budúcnosť projektu

Nástroj ISAGEN v aktuálnej podobe poskytuje rozsiahle možnosti pre generovanie náhodných hodnôt pre štruktúrované dáta spĺňajúce obmedzenia dané charakteristikou reálnych vzoriek dát z komunikácie informačného systému. Okrem toho sa generovanie dokáže riadiť sadou kombinačných kritérií, berie sa do úvahy štruktúra a sémantika dát. Ďalej dokáže nástroj generovať postupnosti správ, kde sa reprodukuje rozloženie časových rozostupov medzi správami podľa pôvodneho rozloženia.

Nástroj je navrhnutý modulárne. Jednak z pohľadu spustenia, jednotlivé fázy nástroja je možné spúšťať samostatne a nezávisle od seba, na vstupe dokáže nástroj akcepovať všetky aj inými fázami priebežne vytvárané formáty dát. Spustenia nástroja v iných módoch je teda možné navzájom reťaziť a používať vždy iba to, čo je potrebné. Na druhej strane nástroj umožňuje aj spustenie interaktívneho módu, ktorý umožňuje spustiť všetky čiastkové fázy pre kompletný proces od abstrakcie cez analýzu, až po výsledne generovanie.

Nástroj v súčasnom stave obsahuje množstvo nedokončených, no perspektívne navrhnutých komponentov a rozhraní, ktoré sa žiaľ do výsledného riešenia v časovom rámci práce nestihli zakomponovať. Medzi hlavné nedostatky patrí nedokončenie horizontálnej analýzy, ktorá bola na začiatku práce uvažovaná ako jedna z metód na odhalenie komplexných vzťahov medzi stromovými dátami naprieč stopou. Rozhranie a komponenty sú v projekte pripravené, avšak samotná implementácia dokončená nie je. Autor projektu má však záujem na prácu eventuálne nadviazať a riešenie rozšíriť aj o možnosti horizontálnej analýzy.

Využiteľnosť nástroja závisi od motivácie vývojára alebo testera, ktorý by mal nástroj využívať. Okrem zreteľnej možnosti využiť nástroj pri testovaní informačných systémov, je

ďalšou zaujímavou možnosťou využitia tvorba tzv. *digitálneho dvojčaťa* ([38]). Nástroj ISA-GEN vzhľadom na to, že aktívne využíva a analyzuje reálnu komunikáciu z informačného systému, je vhodným adeptom na mechanizmus, ktorý by dokázal čiastočne simulovať alebo napodobňovať typickú komunikáciu v informačnom systéme, a nepriamo tak simulovať procesy v uzloch informačného systému napriek nevedomosti o ich vnútornom fungovaní.

Možné rozšírenia v budúcnosti

- Obmedzenia založené na *TCTL/PLTL logike*.
- Odhaľovanie *vzorov* a *kauzality* v správach na úrovni stopy.
- Vizualizácia dátových štruktúr (najmä štruktúry abstraktného stormu).
- Vytvorenie užívateľského rozhrania pre interaktívny mód používania.
- Dynamické načítavanie datasetov užívateľom a eventuálne schopnosť zadefinovať vlastné značky a k nim asociované datasety.
- Využitie paralelizmu pri analýze aj generovaní.
- Výraznejšie využitie mechanizmov pre analýzu izomorfných vlastností, odhaľovanie podobnosti medzi stromami a ich zoskupovanie, klasifikácia pre jednoduchšie spracovanie a komplexnejšie možnosti tvorby podobných dát.
- Podpora pre ďalšie serializačné formáty (`XML`, `YAML`, atď.),.
- Podpora pre ďalšie štrukturálne formáty – Zaujímavým vstupom pre nástroj by mohli byť sieťové `PCAP` log súbory v generickejšej forme ako sú jednoduché `JSON` dáta.
- `HTTP API` pre nástroj.

Kapitola 7

Záver

Na záver práce môžem zhodnotiť, že sa podarilo úspešne naštudovať, zhodnotiť a poskytnúť prehľad o problematike automatického generovania testovacích dát a konceptuálne navrhnúť prakticky použiteľné riešenie v podobe automatického generátora testovacích dát – ISAGEN. Generovanie testovacích dát sa na prvý pohľad javí ako veľmi zložitý problém, ktorý je náročné rozobrať na menšie čiastkové podproblémy. Na začiatku práce v kapitole 2, sa postupne klasifikovali a definovali *metódy generovania testovacích dát* a ich najdôležitejšie vlastnosti. Hlavným poznatkom kapitoly 3 bolo, že *testovacie dáta* mávajú v kontexte *informačných systémov* často komplexnú štruktúrovanú podobu, ktorú najlepšie zachytáva štruktúra stromu. Na základe toho potom vznikla potreba podrobnejšie definovať tieto štruktúrované objekty, prakticky vo forme serializačných formátov (`JSON`), ale taktiež aj formálne ako matematické abstraktné štruktúry, *grafy* a *stromy*, o ktorých sa dá lepšie a precíznejšie diskutovať a rozhodovať. Práve tomu sa venovala kapitola 3, po čom nasledovalo stanovenie a analýza problému *generovania štruktúrovaných dát* v kapitole 4. V kapitole prebehlo porovnanie s triedou problémov *CSP* a na záver kapitoly boli navrhnuté niektoré metódy ako ku generovaniu možno pristupovať. Kapitola 5 sa zaoberala vyhodnotením zadania projektu, kontextom, rámcom a požiadavkami pre softvérové riešenie, nástroj ISAGEN, ktorý je na konci tejto práce jej hlavným výstupom. Na základe špecifikácie bol vytvorený konceptuálny návrh všetkého, čo by mal nástroj v svojom vnútri vykonávať. Kapitola 6 zameraná na implementáciu a experimentálne vyhodnotenie projektu popisuje najdôležitejšie problémy, algoritmy a ich následné zhodnotenie a overenie testovaním a výkonnostnými meraniami.

Na konci práce je možné zhodnotiť, že sa prevažnú väčšinu požiadaviek podarilo splniť. Hlavným nedostatkom je nekompletná fáza analýzy, ktorá oproti prvotnému návrhu, ktorý uvažoval aj o horizontálnej analýze, realizuje žiaľ iba vertikálnu analýzu. Horizontálna analýza sa ukázala byť výrazne zložitejším problémom ako sa na začiatku projektu zdalo. Súčasťou nadväzujúceho plánu práce je teda aj dokončenie návrhu horizontálnej analýzy. Najväčšou výzvou v tomto projekte bolo rozobratie problému na niekoľko úrovní abstrakcie, čo sa na záver značne podarilo. Oproti iným nástrojom a riešeniam, zaoberajúcimi sa taktiež automatizáciou generovania testovacích dát, sa tento projekt odlišuje najmä v tom, že o vnútornom fungovaní testovaných systémov nie je známe nič a jediné informácie, ktoré sú k dispozícii, sú fragmenty komunikácie medzi uzlami na úrovni protokolu na výmenu správ medzi webovými službami. Špecifickou vlastnosťou tohto projektu je totiž tvorba

testovacích dát pre informačné systémy a generovanie nielen samotných dát, ale aj celého kontextu a komunikačnej sekvencie správ.

Medzi najdôležitejšie dosiahnuté výsledky tejto práce taktiež patrí úspešné integrovanie kolektívnej snahy viacerých projektov v rámci platformy *Testos*. Podarilo sa spojiť funkcionalitu nástrojov TS-REPORTER a COMBINE do jedného spoločného riešenia, ktoré spolu s vlastnými mechanizmami pre abstrakciu a generovanie poskytuje kompletný zreťazený proces pre automatické generovanie testovacíc dát.

Literatúra

[1] Aalst, W. van der, Weijters, T. a Maruster, L. Workflow mining: discovering process models from event logs. zv. 16, č. 9, s. 1128–1142. DOI: 10.1109/TKDE.2004.47. ISSN 1041-4347. Dostupné z: http://ieeexplore.ieee.org/document/1316839/.

[2] Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B. et al. An orchestrated survey of methodologies for automated software test case generation. zv. 86, č. 8, s. 1978–2001. DOI: 10.1016/j.jss.2013.02.061. ISSN 01641212. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0164121213000563.

[3] Bach, J. Test Automation Snake Oil. s. 6. Navštívené 19.02.2021. Dostupné z: https://www.satisfice.com/download/test-automation-snake-oil.

[4] Beschastnikh, I., Brun, Y., Ernst, M. D. a Krishnamurthy, A. Inferring models of concurrent systems from logs of their behavior with CSight. In: *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. ACM Press, s. 468–479. DOI: 10.1145/2568225.2568246. ISBN 978-1-4503-2756-5. Dostupné z: http://dl.acm.org/citation.cfm?doid=2568225.2568246.

[5] Bray, T. *The JavaScript Object Notation (JSON) Data Interchange Format* [Internet Requests for Comments]. STD 90. RFC Editor, December 2017. Dostupné z: https://tools.ietf.org/html/rfc7159.html.

[6] Chen, T. Y., Kuo, F.-C., Merkel, R. G. a Tse, T. Adaptive Random Testing: The ART of test case diversity. zv. 83, č. 1, s. 60–66. DOI: 10.1016/j.jss.2009.02.022. ISSN 01641212. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0164121209000405.

[7] Chen, X., Ji, Z., Fan, Y. a Zhan, Y. Restful API Architecture Based on Laravel Framework. zv. 910, s. 012016. DOI: 10.1088/1742-6596/910/1/012016. ISSN 1742-6588, 1742-6596. Dostupné z: https://iopscience.iop.org/article/10.1088/1742-6596/910/1/012016.

[8] Christou, M., Crochemore, M., Flouri, T., Iliopoulos, C. S., Janoušek, J. et al. Computing all subtree repeats in ordered trees. zv. 112, č. 24, s. 958–962. DOI: 10.1016/j.ipl.2012.09.001. ISSN 00200190. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0020019012002505.

[9] Dini, N., Yelen, C., Gligoric, M. a Khurshid, S. Extension-Aware Automated Testing Based on Imperative Predicates. In: *2019 12th IEEE Conference on Software*

Testing, Validation and Verification (ICST). IEEE, s. 25–36. DOI: 10.1109/ICST.2019.00013. ISBN 978-1-72811-736-2. Dostupné z: https://ieeexplore.ieee.org/document/8730211/.

[10] ECMA International. *Standard ECMA-262 - ECMAScript Language Specification*. 5.1. June 2011. Dostupné z: http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[11] Fernandes, E. C., Santos, L. I. dos, Camatti, J. A., Brown, L. a Borsato, M. Flexible Production Data Generator for Manufacturing Companies. zv. 51, s. 1478–1484. 30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2021). DOI: 10.1016/j.promfg.2020.10.205. ISSN 2351-9789. Dostupné z: http://www.sciencedirect.com/science/article/pii/S2351978920320680.

[12] Fraser, G. a Rojas, J. M. Software Testing. In: Cha, S., Taylor, R. N. a Kang, K., ed. *Handbook of Software Engineering*. Springer International Publishing, s. 123–192. DOI: 10.1007/978-3-030-00262-6_4. ISBN 978-3-030-00261-9 978-3-030-00262-6. Dostupné z: http://link.springer.com/10.1007/978-3-030-00262-6_4.

[13] Gotlieb, A., Botella, B. a Rueher, M. Automatic test data generation using constraint solving techniques. zv. 23, č. 2, s. 53–62. DOI: 10.1145/271775.271790. ISSN 0163-5948. Dostupné z: https://dl.acm.org/doi/10.1145/271775.271790.

[14] Gowers, T., Barrow Green, J., Leader, I. a University, P., ed. *The Princeton companion to mathematics*. Princeton University Press. ISBN 978-0-691-11880-2.

[15] Group, W. W. Web Services Architecture. s. 98. Dostupné z: https://www.w3.org/TR/ws-arch/wsa.pdf.

[16] Gudgin, M., Hadley, M. a Mendelsohn, N. *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*. W3C Recommendation. W3C, apríl 2007. Dostupné z: https://www.w3.org/TR/2007/REC-soap12-part2-20070427/.

[17] Hetzel, W. C. a Hetzel, B. *The complete guide to software testing*. 2. ed., 7. pr. Wiley-QED. ISBN 978-0-471-56567-3. OCLC: 248309285.

[18] Hopcroft, J. E. a UIIman, J. D. Alfred V. Aho Bell Laboratories. s. 479.

[19] Janes, A. Test Case Generation and Prioritization: A Process-Mining Approach. In: *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, s. 38–39. DOI: 10.1109/ICSTW.2017.11. ISBN 978-1-5090-6676-6. Dostupné z: http://ieeexplore.ieee.org/document/7899028/.

[20] King, J. C. A new approach to program testing. zv. 10, č. 6, s. 228–233. DOI: 10.1145/390016.808444. ISSN 0362-1340, 1558-1160. Dostupné z: https://dl.acm.org/doi/10.1145/390016.808444.

[21] Knuth, D. E. *The art of computer programming*. 3rd ed. Addison-Wesley. ISBN 978-0-201-89683-1 978-0-201-89684-8 978-0-201-89685-5.

[22] Korel, B. Automated software test data generation. zv. 16, č. 8, s. 870–879. DOI: 10.1109/32.57624. ISSN 00985589. Dostupné z: http://ieeexplore.ieee.org/document/57624/.

[23] Kotyz, J. *Nástroj pro tvorbu obsahu databáze pro účely testování software*. Brno, CZ, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/18066/.

[24] Kuhn, D. R., Kacker, R. N. a Lei, Y. *Practical combinatorial testing*. NIST SP 800-142. National Institute of Standards and Technology. NIST SP 800–142 s. Edition: 0. Dostupné z: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-142.pdf.

[25] Leemis, L. M. Nonparametric Estimation of the Cumulative Intensity Function for a Nonhomogeneous Poisson Process. zv. 37, č. 7, s. 886–900. DOI: 10.1287/mnsc.37.7.886. ISSN 0025-1909, 1526-5501. Dostupné z: http://pubsonline.informs.org/doi/abs/10.1287/mnsc.37.7.886.

[26] Lei, Y., Kacker, R., Kuhn, D. R., Okun, V. a Lawrence, J. IPOG: A General Strategy for T-Way Software Testing. In: *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*. IEEE, s. 549–556. DOI: 10.1109/ECBS.2007.47. ISBN 978-0-7695-2772-7. Dostupné z: http://ieeexplore.ieee.org/document/4148973/.

[27] Lerouge, J., Le Bodic, P., Héroux, P. a Adam, S. GEM++: A Tool for Solving Substitution-Tolerant Subgraph Isomorphism. In: Liu, C.-L., Luo, B., Kropatsch, W. G. a Cheng, J., ed. *Graph-Based Representations in Pattern Recognition*. Springer International Publishing, sv. 9069, s. 128–137. DOI: 10.1007/978-3-319-18224-7_13. ISBN 978-3-319-18223-0 978-3-319-18224-7. Series Title: Lecture Notes in Computer Science. Dostupné z: http://link.springer.com/10.1007/978-3-319-18224-7_13.

[28] Ma, F. Constraint solving techniques for software testing and analysis. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*. ACM Press, sv. 2, s. 417. DOI: 10.1145/1810295.1810407. ISBN 978-1-60558-719-6. Dostupné z: http://portal.acm.org/citation.cfm?doid=1810295.1810407.

[29] Marinov, D. *Automatic Testing of Software with Structurally Complex Inputs*. 2004. PhD dissertation. Massachusetts Institute of Technology.

[30] Meudec, C. ATGen: automatic test data generation using constraint logic programming and symbolic execution. zv. 11, č. 2, s. 81–96. DOI: 10.1002/stvr.225. ISSN 0960-0833, 1099-1689. Dostupné z: http://doi.wiley.com/10.1002/stvr.225.

[31] Myers, G. J., Sandler, C. a Badgett, T. *The art of software testing*. 3rd ed. John Wiley & Sons. ISBN 978-1-118-03196-4 978-1-118-13313-2 978-1-118-13314-9. OCLC: ocn728656684.

[32] Navarro, G. A guided tour to approximate string matching. zv. 33, č. 1, s. 31–88. DOI: 10.1145/375360.375365. ISSN 0360-0300, 1557-7341. Dostupné z: https://dl.acm.org/doi/10.1145/375360.375365.

[33] Naňo, A. *Frontend pro generátor testovacích dat.* Brno, CZ, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/21378/.

[34] Nováček, P. *Automatizovaná detekce závislostí datových struktur.* Brno, CZ, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22816/.

[35] Oháňka, M. *Automatizovaná detekce datových typů ve strukturách.* Brno, CZ, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/21837/.

[36] Olšák, O. *Generování strukturovaných testovacích dat.* Brno, CZ, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/21539/.

[37] Saad, B., Herrmann, F., Lanuel, Y. a Tamisier, T. DDCSP: Constraint Satisfaction Problem Used for Rule-Based System. In: *2010 International Conference on Data Storage and Data Engineering.* IEEE, s. 240–244. DOI: 10.1109/DSDE.2010.51. ISBN 978-1-4244-5678-9. Dostupné z: http://ieeexplore.ieee.org/document/5452578/.

[38] Sacks, R., Brilakis, I., Pikas, E., Xie, H. S. a Girolami, M. Construction with digital twin information systems. *Data-Centric Engineering.* Cambridge University Press. 2020, zv. 1, s. e14. DOI: 10.1017/dce.2020.16.

[39] Sahoo, R. R. a Ray, M. Metaheuristic Techniques for Test Case Generation: A Review. zv. 11, č. 1, s. 158–171. DOI: 10.4018/JITR.2018010110. ISSN 1938-7857, 1938-7865. Dostupné z: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/JITR.2018010110.

[40] Smal, A. Explanation for 'Tree isomorphism' talk. s. 10. Navštívené 08.05.2021. Dostupné z: https://logic.pdmi.ras.ru/~smal/files/smal_jass08.pdf.

[41] Steffens, A., Lichter, H. a Moscher, M. Towards Data-driven Continuous Compliance Testing. 2018, s. 7.

[42] Testos, S. *Domovská stránka projektu Testos* [http://testos.org]. 2018. Navštívené: 27.4.2021.

[43] Thompson, H. a Lilley, C. *XML Media Types* [Internet Requests for Comments]. RFC 7303. RFC Editor, July 2014.

[44] Červinka, R. *Asistent pro generování testovacích scénářů.* Brno, CZ, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/20303/.

[45] Želiar, D. *Automatizovaná syntéza stromových struktur z reálných dat.* Brno, CZ, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22217/.

[46] Zhao, R., Li, Z. a Wang, Q. Test Generation for Programs with Binary Tree Structure as Input. zv. 25, č. 7, s. 1129–1151. DOI: 10.1142/S0218194015500205. ISSN 0218-1940, 1793-6403. Dostupné z: https://www.worldscientific.com/doi/abs/10.1142/S0218194015500205.

[47] Zhu, H., Hall, P. A. V. a May, J. H. R. Software unit test coverage and adequacy. zv. 29, č. 4, s. 366–427. DOI: 10.1145/267580.267590. ISSN 0360-0300, 1557-7341. Dostupné z: https://dl.acm.org/doi/10.1145/267580.267590.