VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO ANALÝZU OBSAHU DATABÁZE PRO ÚČELY TESTOVÁNÍ SOFTWARU

A TOOL FOR DATABASE CONTENT ANALYSIS FOR TESTING PURPOSES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **FRANTIŠEK KROPÁČ**
AUTHOR

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

BRNO 2017

Zadání bakalářské práce/19446/2016/xkropa06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Kropáč František**
Obor: Informační technologie
Téma: **Nástroj pro analýzu obsahu databáze pro účely testování softwaru**
A Tool for Database Content Analysis for Testing Purposes
Kategorie: Analýza a testování softwaru

Pokyny:

1. Nastudujte relační databáze a tvorbu náhodných testovacích dat.
2. Analyzujte požadavky pro testování systémů pracujících s relačními databázemi. Nastudujte projekt randb-gen pro tvorbu testovacích dat relační databáze.
3. Navrhněte platformu pro detekci strukturálních a sémantických omezení dat v databázi. Výstupem detekce bude sada omezení ve vstupním jazyce projektu randb-gen. Platforma bude modulární. Navrhněte aplikační rozhraní (API) pro jednotlivé detektory obsahu databáze.
4. Implementujte platformu jako knihovnu. Implementujte rozhraní k této knihovně pro příkazovou řádku (CLI program).
5. Ověřte správnost aplikačního rozhraní k imlementované knihovně. Ověřte funkcionalitu platofmy na demonstračním detektoru obsahu databáze.

Literatura:

- Želiar, D.: Nástroj pro generování obsahu databáze pro účely testování softwaru (randb-gen). Bakalářská práce, FIT VUT v Brně, 2016.
- M.S. Chen, J. Han, P.S. Yu: Data mining: an overview from a database perspective. Knowledge and data Engineering, IEEE Transactions on 8 (6), 866-883, 1996.
- 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.
- Domovská stránka generátoru dat, http://www.generatedata.com/

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrčka Aleš, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Akceptační testování aplikací před produkcí zahrnuje testování reálně vypadajících scénářů při používání aplikace. Tvorba testovacích dat pro aplikace využívající databázový systém je komplikovaná z důvodů specifikace omezení dat, která spadají do domény testované aplikace, a specifikace strukturálních omezení resp. vztahů mezi těmito daty. Tato práce se zabývá problematikou detekce datových vazeb v již existující relační databázi. Výsledkem je nástroj, který automaticky řídí a zprostředkovává detekci omezení v datech relační databáze. Výstupem detekce je váhově ohodnocené omezení dat, které reprezentují jak datový typ, tak vazbu mezi tabulkami a sloupci relační databáze. Tento výstup je pak možné použít pro generování náhodných testovacích dat, které budou reprezentovat vstupy pro reálně vypadající scénáře použití testované aplikace.

Abstract

Acceptance testing of applications before the production includes testing of scenarios resembling situations of real usage of the application. Creating the test data is complicated matter since the data are specified by restrictions concerning the domain of the tested application and the specifications of the structural restrictions and the relations between these data. This thesis focuses on the issues of detecting the data constraints in an already created relational database. The outcome of the thesis is a tool which automatically controls and mediates the detection of the data constraints in a relational database. The detection result is a weight rating of the data restrictions, which represents both the data type and the relation between tables, columns in relational database. These restrictions can be used to generate a random testing data which would represent inputs for seemingly realistic scenarios of the usage of the application.

Klíčová slova

analýza databáze, relační databáze, fuzzy testování, DBus, generování náhodných dat, C++

Keywords

database analysis, relational database, fuzzy testing, DBus, random data generation, C++

Citace

KROPÁČ, František. *Nástroj pro analýzu obsahu databáze pro účely testování softwaru.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Smrčka Aleš.

Nástroj pro analýzu obsahu databáze pro účely testování softwaru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Smrčky, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
František Kropáč
16. května 2017

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce Ing. Aleši Smrčkovi, Ph.D. za odborné vedení a cenné rady, které mi pomohli při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Testování softwaru je proces ověřující správnost, úplnost a kvalitu vyvíjeného programu. Obecně se tento proces skládá z nastavení prostředí, vykonání testu, ověření dosažených výsledků a vyčištění vedlejších účinků testu. Testování programů, které komunikují s databází, vyžadují nastavení vhodného prostředí pro spuštění testů, tedy nastavení vhodného testovacího obsahu databáze. Pro tvorbu nového obsahu testovací databáze existují dva přístupy:

- Tvorba náhodného obsahu databáze.
- Analýza již existující databáze a následná tvorba dat podle výsledků analýzy.

Existuje mnoho nástrojů pro vytvoření náhodného obsahu databáze, v nekomerční sféře však mají omezenou funkcionalitu, která se vztahuje např. na generování náhodných dat pouze nad jednou tabulkou. Takový nástroj je hned v několika směrech nevyhovující – snadno může dojít k porušení sémantických omezení dané tabulky, porušení vazby mezi tabulkami.

Existují také nástroje pro analýzu databáze, avšak tyto nástroje jsou silně komerční, kdy cena takového nástroje se pohybuje ve vyšších řádech desetitisíců korun. Tyto nástroje umožňují analyzovat databázi v rámci sloupce, tabulky nebo celé databáze, výsledky analýzy jsou pak nejčastěji interpretovány pomocí grafů.

1.1 Motivace

Obecně je kladen důraz na rychlou a kvalitní tvorbu testovacích dat, které se alespoň částečně blíží k datům reálným. V případě volby manuální tvorby testovacích dat má tento přístup několik aspektů:

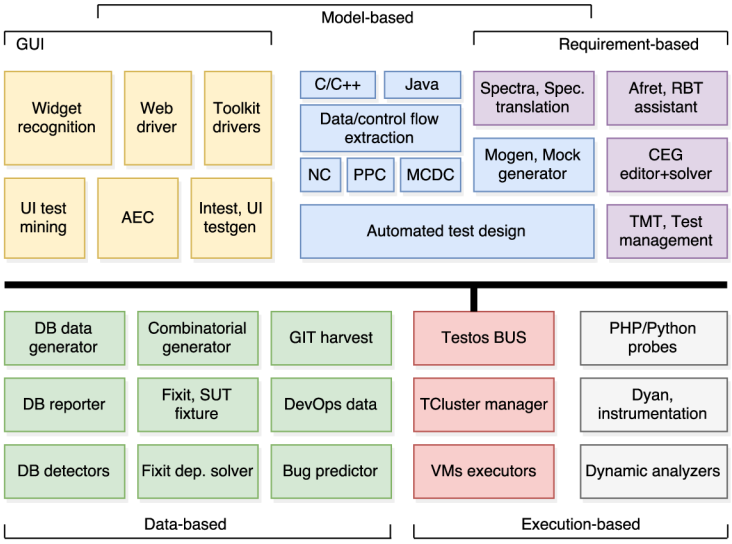
- Poměr stráveného času ku počtu vytvořených testovacích dat.
- Různorodost dat.

V případě že jsou testovací data tvořena jedním člověkem, tak je pro něj tato činnost zdlouhavá a nezáživná. Ve výsledku vytvořené testovací data nemusí dosahovat takové kvality, jako kdyby byly generovány náhodně.

Cílem této práce je navrhnout nástroj, který nabízí rychlou a kvalitní tvorbu testovacích dat. Požadavek na tento nástroj tedy je automatická analýza existující databáze, kdy se bude jednat o strukturální a sémantickou analýzu. Výsledkem této analýzy bude procentuální určení obsahu jednotlivých atributů dané tabulky v databázi.

Testos (Test Tool Set) [10] je projekt jehož hlavním cílem je vytvoření sady nástrojů podporující automatizované testování softwaru. Nástroje v platformě Testos (viz Obrázek 1.1) kombinují různé úrovně testování a lze je řadit do několika kategorií: testování založené na modelech (Model-based), testování založené na požadavcích (Requirement-based), testování grafického uživatelského rozhraní (GUI), testování založené na datech (Data-based) a dynamická analýza (Execution-based).

V aktuálním vývoji nástrojů pro testování založené na datech jsou nástroje pro snadnou tvorbu testovacích dat pro relační databáze: nástroj pro generování náhodných dat [9] a nástroje pro zjišťování logických vazeb mezi testovacími daty [7, 8].



Obrázek 1.1: Testos – jednotná sada testovacích nástrojů.

1.2 Vlastní přínos

Tato bakalářská práce úzce souvisí se souběžně vznikající bakalářskou prací Marka Ochodka[12]. V průběhu práce je mimojiné na tuto bakalářskou práci odkazováno. V případě že bude zmíněn nebo zmíněny detektory, tak je myšlena bakalářská práce Marka Ochodka.

Marek Ochodek ve své bakalářské práci popisuje návrh a implementaci sémantických detektorů obsahu databáze, řešení problematiky napojení na sběrnici DBus a připojení na databázi v programovacím jazyku Python.

V této práci je popsán návrh a implementace nástroje, který posílá požadavky na analýzu a ukládá výsledky analýz jednotlivých detektorů, dále je tento nástroj nazýván jako jádro.

Tato práce se zabývá řešením těchto problematik:

- Připojení programu na existující sběrnici DBus, případně vytvoření vlastní sběrnice.
- Správa detektorů a jejich závislostí.
- Správa výsledků jednotlivých analýz.
- Zajištění komunikace mezi jádrem a detektory.

Každá z problematik je dále podrobněji probrána v kapitole č. 3.

V průběhu práce je odkazováno na společnou práci, která zahrnuje návrh struktury zpráv, životní cyklus zprávy a řešení závislostí detektorů. Tato společná práce vznikala v průběhu pravidelných schůzek se společným vedoucím.

Text je rozdělen do několika kapitol. Kapitola 2 se zaměřuje na přehled existujících nástrojů a pohled na použité technologie. V kapitole 3 jsou pak popsány funkční požadavky nástroje a celkový architektonický a behaviorální návrh jádra nástroje. Dále pak kapitola 4 popisuje zajímavé implementační detaily. V kapitole 5 je zahrnuto ověření funkcionality nástroje. Zhodnocení, známé nedostatky nástroje a nástin dalšího pokračování projektu je sepsán v kapitole 6.

Kapitola 2

Konkurence a použité technologie

V této kapitole jsou představeny již existující projekty, u každého projektu je popsána základní funkcionalita, jeho výhody a nevýhody. Dále je zde představena použitá technologie při tvorbě nástroje.

2.1 Existující příbuzná řešení

Zde jsou popsány projekty, které řeší stejnou problematiku. Tato práce spadá do všeobecné problematiky tvorba testovacích dat. S ohledem na databáze se tato problematika dále dělí na analýzu databáze a na generování testovacích dat.

2.1.1 Analýza databáze

Aquafold [1]

Grafický nástroj pro analýzu obsahu databáze, umožňuje vytvářet, spravovat a graficky vizualizovat obsah databáze.

Výhody

- Analyzovaná databáze zobrazena pomocí dashboardů.
- Uživatel získá kompletní přehled nad databází.
- Široká škála podporovaných databází.

Nevýhody

- Komerční produkt, zkušební verze pouze na 14 dní.

ApexSQL [11]

Sada grafických nástrojů pro analýzu databáze – monitoring, obnova, záloha. Tato sada nástrojů zahrnuje i nástroj pro generování obsahu.

Výhody

- Komplexní a úplná sada nástrojů.
- Generování testovacího obsahu zvolených tabulek.

Nevýhody

- Komerční produkt.
- Vysoká cena.

Další nástroj, **Logi** [5], spadající do této kategorie nabízí jednoduché prezentování analyzovaných dat pomocí HTML[1].

2.1.2 Generování testovacích dat

Generatedata [4]

Grafický webový nástroj pro tvorbu testovacích dat. Hodnoty sloupců jsou specifikovány pomocí generátorů.

Výhody

- Výstupy jsou ve formě jazyku SQL, v programovacím jazyku nebo v jazyku pro serializaci.
- Široká škála datasetů.
- Možnost volby generování dat závislých na určité zemi.
- Podpora reference hodnot sloupce, které lze využít ve výrazech.

Nevýhody

- Komerční produkt
- Nekomerční verze neumožňuje přidání vlastních datasetů.
- Generování pouze nad jednou tabulkou.
- Uživatel musí ručně vytvořit schéma tabulky.

Mockaroo [6]

Grafický webový nástroj, který obsahuje několik generátorů a datasetů. Výstup nástroje je ve formátu různých jazyků pro serializaci dat.

Výhody

- Specifikace generátoru sloupce pomocí matematických formulí.
- Umožňuje procentuálně specifikovat kolik hodnot ve sloupci bude prázdných.

[1] HyperText Markup Language, značkovací jazyk používaný pro tvorbu webových stránek

- Vytvoření schématu tabulky pomocí CSV[2] hlavičky nebo pomocí analýzy příkazu `CREATE TABLE`[3] v SQL skriptu.
- Možnost tvorby testovacích dat nad více než jednou tabulkou.

Nevýhody

- Komerční produkt.
- Nekomerční verze umožňuje generování pouze 1000 řádků.

Databasetestdata [2]

Grafický webový nástroj obsahující několik základních generátorů a datasetů. Obsahuje několik předdefinovaných schémat tabulky.

Výhody

- Nekomerční produkt.
- Počet generovaných řádků není omezen.

Nevýhody

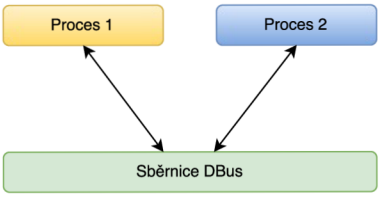
- Nástroj obsahuje pouze základní generátory.
- Dlouho trvající generování výsledků.

2.2 Použité technologie

2.2.1 Sběrnice DBus

Představení sběrnice DBus

Sběrnice DBus [3] vznikla v roce 2002 jako součást projektu *freedesktop.org*, aktuálně je podporována firmou *RedHat* a komunitou. Cílem tohoto projektu je snaha o standardizaci služeb linuxových desktopových prostředí. Jedná se o meziprocesový komunikační mechanismus, který je založen na bázi soketů.



Obrázek 2.1: Příklad meziprocesové komunikace při použití sběrnice DBus.

[2] Comma-separated values; jednoduchý souborový formát určený pro výměnu tabulkových dat
[3] Příkaz pro vytvoření nové tabulky v databázi

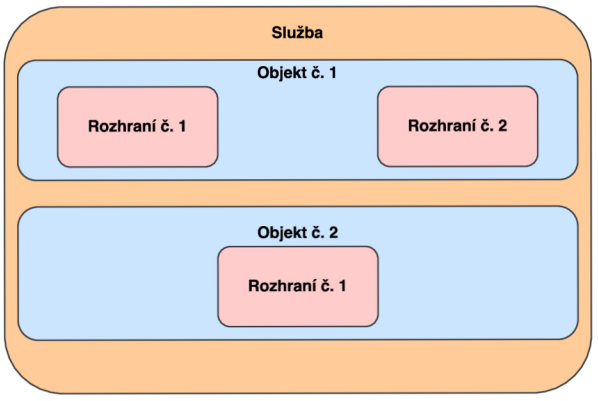
DBus sběrnice jsou dvojího druhu:

- Systemová sběrnice
  - Pouze jedna instance sběrnice pro všechny uživatele.
  - Vyhrazena pro systémové služby.
  - Použita pro nízkoúrovňové události jako je například připojení k síti nebo připojení USB zařízení atd.
- Sběrnice sezení
  - Každý uživatel má svou vlastní instanci.
  - Poskytuje služby uživatelským aplikacím.

Obecně se na sběrnici DBus pracuje s těmito prvky:

- Služby.
- Objekty.
- Rozhraní.
- Klienti – aplikace využívající DBus služby.

Jedna DBus služba obsahuje jeden nebo více objektů, které implementují jedno nebo více rozraní.



Obrázek 2.2: Struktura DBus služby.

Služby

Služba je kolekce objektů, které poskytují specifickou množinu funkcí. Po připojení služby na sběrnici aplikace obdrží unikátní jméno (např. `:1.312`), případně aplikace může požádat o přidělení čitelného jména [4] (např. `org.tests`).

[4] Anglicky **well-known name**

Objekty

Jednotlivé objekty jsou vázany na jednu službu, tyto objekty mohou být dynamicky přidávány, případně odebírány. Každý objekt je jednoznačně identifikovatelný pomocí cesty objektu[5] (např. `/` nebo `/org/testos/db/reporter`). Každý objekt musí implementovat jedno nebo více rozhraní.

Rozhraní

Rozhraní obsahuje jednotlivé členy daného objektu, mezi tyto členy patří vlastnosti, metody a signály. Každé rozhraní vlastní unikátní jméno v tečkové notaci (může připomínat jmenný prostor v programovacím jazyce Java [6]). DBus definuje několik standardních rozhraní, tyto rozhraní spadají do jmenného prostoru „org.freedesktop.DBus“.

Standanrdní rozhraní [7]

- **org.freedesktop.DBus.Introspectable** – Získání informací o objektu (jaké rozhraní, metody a signály jsou implementovány)
- **org.freedesktop.DBus.Peer** – Poskytuje metodu pro zjištění zda je připojená služba aktivní (ping)
- **org.freedesktop.DBus.Properties** – Změna hodnot atributů dané DBus služby
- **org.freedesktop.DBus.ObjectManager** – Přístup k zanoženým objektům v DBus službě

Komunikace

Pro komunikaci mezi jednotlivými procesy se využívají **metody** a **signály**. Jedná se o parametrizovatelné zprávy.

Metody mohou být přirovnány k metodám známým z objektově orientovaných programovacích jazyků. Metody mohou navracet hodnotu nebo objekt.

Signály jsou jednosměrné – nemají návratový typ, jedná se o všesměrné vyslání signálu. DBus služba, která chce daný signál přijímat, si nejprve musí požádat o registraci tohoto signálu. Signály slouží k jednosměrnému odeslání zprávy nebo k upozornění na změnu stavu objektu.

[5] Anglicky **object path**
[6] Více na https://www.javatpoint.com/interface-in-java
[7] Více na https://dbus.freedesktop.org/doc/dbus-specification.html#standard-interfaces

Příklad použití

Zde je uvedeno použití linuxových nástrojů, které byly použity při ověřování funkcionality nástroje, toto je popsáno v kapitole 5.

dbus-monitor

Tento nástroj slouží ke sledování provozu na sběrnici DBus. Na sledovaný provoz může být aplikován filtr.

```
dbus-monitor --session "type='signal', interface='testos.db.reporter.process'"
```

První argument (`--session`) specifikuje, na které sběrnici bude sledován provoz (tedy na sběrnici sezení), druhý argument specifikue filtr, který má být nad provozem aplikován. V tomto případě filtr specifikuje, že se mají zobrazovat pouze zprávy typu `signal`, které obsahují rozhraní `testos.db.reporter.process`.

dbus-send

Tento nástroj slouží k odesílání zpráv na sběrnici DBus. Vždy je nutné specifikovat cestu objektu, rozhraní a metodu, která má být zavolána.

```
dbus-send --session --type=signal /testos/db/reporter/core
                        testos.db.reporter.process.process_db
```

První argument specifikuje sběrnici, na které bude zpráva odeslána, druhý argument specifikuje typ zprávy. Další argumenty udávají cestu objektu, rozhraní a metodu. Rozhraní je od metody odděleno poslední tečkou. Tedy zde je rozhraní `testos.db.reporter.process` a volaná metoda je `process_db`.

dbus-daemon

Tento nástroj slouží k vytvoření vlastní sběrnice DBus. Umožňuje tvorbu jak systémové sběrnice, tak i sběrnice sezení.

```
dbus-daemon --session --print-address
```

První argument specifikuje typ vytvářené sběrnice, druhý argument specifikuje, že adresa nově vytvořené sběrnice má být zobrazena na standardní výstup – případně při specifikaci souborového deskriptoru je adresa sběrnice vypsána do něj.

Využití DBus sběrnice při návrhu nástroje

Při návrhu je využito sběrnice DBus s několika omezeními. V nástroji je implementována služba, která si vyžádá čitelné jméno *testos.db-reporter.core*, tato služba obsahuje jeden objekt, který je identifikován cestou */org/testos/db/reporter*, v tomto objektu je jedno rozhraní s názvem *process*. Toto rozhraní implementuje všechny metody, které jsou využity pro komunikaci mezi jádrem a detektory – standardní rozhraní spadající do jmenného prostoru „org.freedesktop.DBus“ nejsou implementována. Pro komunikaci mezi jádrem a detektory jsou využity zprávy typu `signal` a to z důvodu, že při počátečním návrhu se nepředpokládalo, že každý detektor bude na sběrnici DBus jednoznačně identifikovatelný pomocí jména detektoru.

Kapitola 3

Návrh nástroje

Tato kapitola se zabývá celkovým návrhem nástroje, nejprve jsou uvedeny funkční požadavky, dále pak architektonický a behaviorální návrh.

3.1 Funkční požadavky

Funkční požadavky jsou rozčleněny do několika sekcí, každá sekce se zabývá jednou hlavní problematikou.

3.1.1 Připojení na sběrnici DBus

Tento požadavek můžeme dekomponovat na tyto podproblémy:

- Tvorba vlastní sběrnice
- Získání adresy sběrnice

Tvorba vlastní sběrnice

Nástroj je spuštěn s parametrem, který specifikuje užití vlastní sběrnice. Vlastní sběrnice je vytvořena pomocí nástroje `dbus-daemon` a popřípadě je vypsána její adresa na standardní výstup.

Získání adresy sběrnice

V případě že má nástroj užít vlastní sběrnice, tak je adresa sběrnice získána již při vytváření.

V případě že nástroj má provést připojení na existující sběrnici sezení, tak je adresa sběrnice získána z proměnné prostředí [1] . V tomto objektu je uložena proměnná s názvem `DBUS_SESSION_BUS_ADDRESS`, kdy její hodnota obsahuje adresu sběrnice sezení.

[1] Anglicky environment variables; jedná se o objekt obsahující data, které jsou využívány jednou nebo více aplikacemi. Více na https://wiki.archlinux.org/index.php/environment_variables

3.1.2 Správa detektorů a jejich závislostí

Každý detektor má 0 až n závislostí[2]. Detektor který neobsahuje žádné závislosti je spustitelný vždy. V případě že detektor obsahuje závislosti, tak musí být nejprve splněny tyto závislosti - jádro musí být obeznámeno o výsledcích detektoru, na kterém je daný detektor závislý.

Je také nutno zajistit, aby detektor se splněnými závislostmi byl spouštěn pouze jednou s danou kombibací výsledků.

3.1.3 Správa výsledků jednotlivých analýz

Každému přijatému výsledku analýzy je přiděleno unikátní identifikační číslo, podle kterého je tento výsledek později dohledatelný. Dále je možné výsledek analýzy nalézt pomocí návratového typu výsledku a jména detektoru, který tento výsledek odeslal.

3.1.4 Komunikace mezi jádrem a detektory

Jako komunikační kanál je užita sběrnice DBus, připojení na sběrnici DBus je popsáno v sekci 3.1.1.

Pro komunikaci byl vytvořen nový komunikační protokol, který se sestává z parametrizovatelných zpráv. Každé nově vygenerované zprávě je přidělen unikátní identifikátor. Každá zpráva musí dodržovat životní cyklus zprávy.

3.1.5 Životní cyklus zprávy

Životní cyklus zprávy specifikuje možnosti zprávy v jednotlivých stavech. Dodržováním životního cyklu zprávy je zajištěno validní chování nástroje. Nástroj musí odolný vůči zprávám, které nedodržují životní cyklus.

[2]Závislostí detektoru rozumíme závislost na výsledcích jiného detektoru.

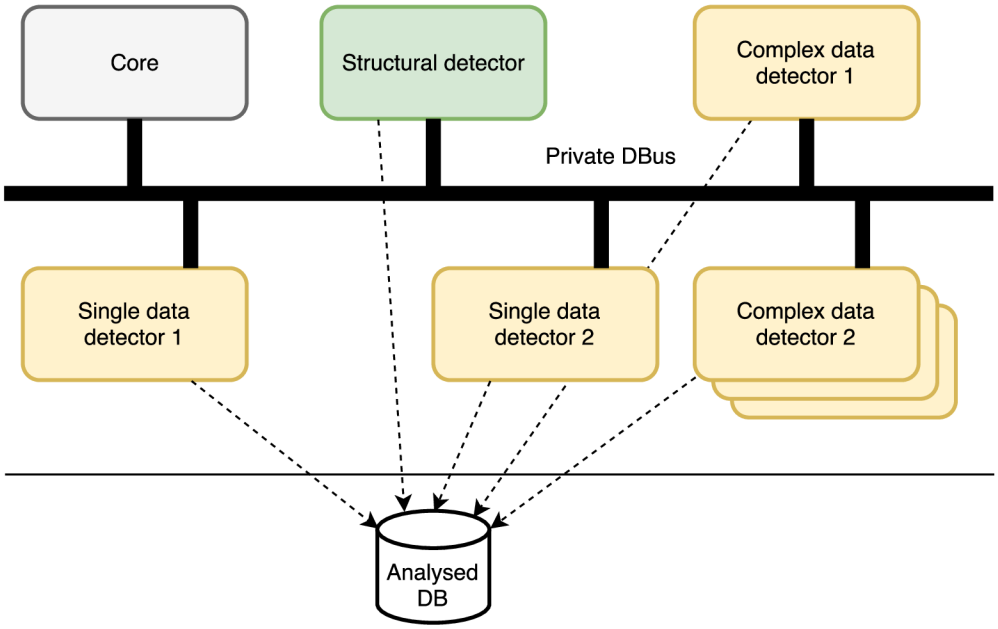
3.2 Popis návrhu

Tato sekce se věnuje návrhu nástroje, nejprve je popsán architektonický návrh, tedy architektura celého systému, hierarchie tříd v jádru a použitý komunikační protokol. Dále je zde popsán behaviorální návrh, tedy popis chování celého systému.

3.2.1 Architektonický návrh

Systém detektorů

Tento systém se sestává z jádra, sběrnice a jednoho nebo více databázových detektorů.



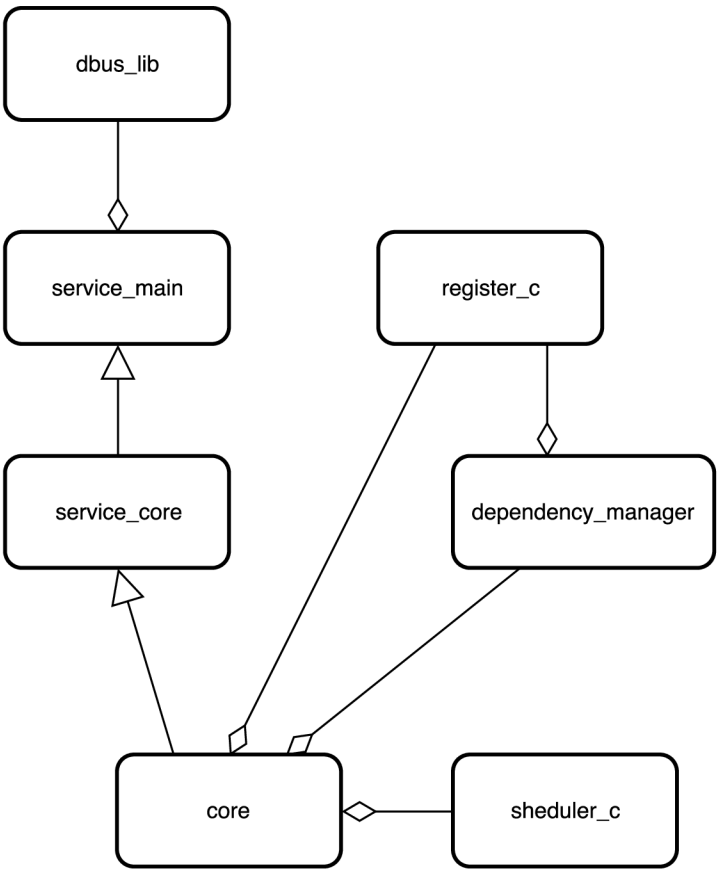
Obrázek 3.1: Architektura systému detektorů

Jádro spravuje DBus sběrnici sezení (případně vlastní sběrnici), spravuje klienty (detektory) připojené ke sběrnici, zajišťuje generování požadavků na detekci. Dále vyhodnocuje závislosti výsledků detekce a generuje tak další požadavky. Každý detektor si zajišťuje vlastní připojení k analyzované databázi.

Popis jednotlivých detektorů

Structural detector	Analýza strukturálních omezení databáze
Single data detector	Analýza jednoho sloupce v tabulce
Complex data detector	Analýza více sloupců v tabulce

Hierarchie tříd



Obrázek 3.2: Hierarchie tříd

Na nejnižší úrovni je použita třída `dbus_lib`, tato třída zajišťuje vytvoření vlastní sběrnice, připojení na sběrnici a komunikaci na sběrnici. Pro reprezentaci zpráv, které jsou přijímány a odesílany přes sběrnici je užita třída `message`. Tato třída specifikuje na jakém rozhraní a jaká metoda má být ze vzdálené služby zavolána a typ zprávy.

Abstraktní třída `service_main` představuje službu na sběrnici. Z této třídy dědí třída `service_core`, která upřesňuje, že se jedná o službu jádra.

Třída `core` je poděděna ze třídy `service_core`. Tato třída je stěžejní z celé hierarchie, neboť obstarává propojení jednotlivých částí jádra a ovládá další komponenty nezbytné pro správnou funkčnost jádra.
Třída `register_c` slouží k ukládání informací o detektorech a o jednotlivých výsledcích analýz. Tato třída je dále využívána ve třídě `dependency_manager`, kde je prováděno vyhodnocování závislostí jednotlivých detektorů a vytváření nových požadavků na detekci. Tyto požadavky jsou následně využity třídou `sheduler_c`, kde dochází k naplánování jejich odeslání detektorům.

Komunikační protokol

V této části jsou popsány jednotlivé zprávy z komunikačního protokolu. U každé zprávy je uveden odesílatel zprávy a jaký je důsledek přijetí zprávy.

Registrace detektoru

Název: `register_detector`
Odesílatel: Detektor

Tato zpráva slouží k obeznámení jádra o novém detektoru, který je připraven analyzovat databázi. Detektor po připojení sběrnici odesílá požadavek o registraci detektoru, očekává potvrzení od jádra o úspěšné registraci. Jádro si ukládá informace o detektoru a o jeho závislotech.

Název parametru	Popis parametru
`detector_name`	Název detektoru
`payload`	JSON[3] objekt obsahující závislosti jednotlivých návratových datových typů

Příklad závislosti detektoru ve formátu JSON

```
1  {
2    "dependencies": [{
3      "datatype": "org.testos.datatype1",
4        "datatype_dependency": ["org.testos.datatype0"]
5    },
6    {
7      "datatype": "org.testos.datatype3",
8      "datatype_dependency": ["org.testos.datatype4",
9          "org.testos.datatype5"
10     ]
11   }
12   ]
13 }
```

Na řádku č. 3 a č. 7 jsou definovány návratové datové typy detektoru, na řádku č. 4 a č. 8 jsou specifikovány závislosti daného návratového datového typu.
V tomto případě detektor navrací datový typ `org.testos.datatype1`, který je závislý na výsledcích detektoru, který navrací datový typ `org.testos.datatype0`.
Dále tento detektor navrací datový typ `org.testos.datatype3`, který je závislý na výsledcích detektorů, které navrací datový typ `org.testos.datatype4` a `org.testos.datatype5`.

Potvrzení registrace

Název: `register_ack`
Odesílatel: Jádro

Tato zpráva slouží pouze k potvrzení úspěšné registrace detektoru u jádra. Detektor po přijetí této zprávy může přijímat další zprávy od jádra.

Název parametru	Popis parametru
`detector_name`	Název detektoru

Zpráva obsahuje název detektoru, aby bylo možné jednoznačně určit příjemce zprávy.

[3] JavaScript Object Notation – způsob zápisu dat nezávislý na počítačové platformě

Odregistrace detektoru

Název: `unregister_request`
Odesílatel: Detektor

Tato zpráva slouží k obeznámení jádra o tom, že daný detektor přestává přijímat zprávy odeslané jádrem. Detektor ještě očekává odpověď, která potvrzuje odregistraci detektoru.

Název parametru	Popis parametru
`detector_name`	Název detektoru

Zpráva obsahuje název detektoru, který požaduje odregistraci.

Potvrzení odregistrace

Název: `unregister_ack`
Odesílatel: Jádro

Tato zpráva slouží pouze k potvrzení úspěšné odregistrace detektoru u jádra. Detektor po přijetí této zprávy může požádat o novou registraci. Jádro odregistrovanému detektoru neposílá žádné další zprávy.

Název parametru	Popis parametru
`detector_name`	Název detektoru

Zpráva obsahuje název detektoru, aby bylo možné jednoznačně určit příjemce zprávy.

Žádost o detekci

Název: `detect_request`
Odesílatel: Jádro

Touto zprávou jádro žádá specifický detektor o analýzu, předmět analýzy je specifikován v parametru s názvem `payload`.

Název parametru	Popis parametru
`msg_id`	Unikátní číslo požadavku
`recipient_id`	Název detektoru, který má být spuštěn
`payload`	Doplňující informace k požadavku – např. výsledky jiného detektoru

Jádro po odeslání požadavku očekává od daného detektoru potvrzení přijetí.

Odmítnutí žádosti o detekci

Název: `detect_request_nack`
Odesílatel: Detektor

V případě že detektor není schopen obsloužit daný požadavek, tak odpovídá touto zprávou.

Název parametru	Popis parametru
`msg_id`	Unikátní číslo požadavku
`detector_name`	Název detektoru

Zpráva obsahuje totožné `msg_id`, které bylo zasláno detektoru ve zprávě s požadavkem. Jádro po přijetí této zprávy dále s požadavkem nepracuje.

Potvrzení žádosti o detekci

Název: `detect_request_ack`
Odesílatel: Detektor

Detektor potvrzuje přijetí požadavku a začíná daný požadavek zpracovávat.

Název parametru	Popis parametru
`msg_id`	Unikátní číslo požadavku
`detector_name`	Název detektoru

Zpráva obsahuje totožné `msg_id`, které bylo zasláno detektoru ve zprávě s požadavkem. Jádro po přijetí této zprávy očekává výsledek, či výsledky daného detektoru.

Výsledek detekce

Název: `detect_request_result`
Odesílatel: Detektor

Detektor odesílá jádru výsledek nebo výsledky vlastní analýzy. Každý výsledek obsahuje návratový datový typ a jeho váhu – váha je definována na intervalu $< 0; 1 >$, kdy hodnota 0 značí nulový nález a hodnota 1 značí maximální nález. Další data jsou v nepovinných klíčích, tyto jsou závislá na konkrétním typu detekce.

Název parametru	Popis parametru
`msg_id`	Unikátní číslo požadavku
`detector_name`	Název detektoru
`payload`	Výsledek analýzy

Příklad výsledku strukturálního detektoru ve formátu JSON

```
{
  "weight": 1,
  "type": "org.testos.detectors.structural",
  "tables": {
   "Table1": [{
    "name": "column_name1",
    "type": "int",
    "null": false,
    "pk": false,
    "default": "",
    "other": ""
   }]
  }
}
```

Prvním klíčem je váha výsledku, následující klíč je návratový datový typ výsledku, tedy typ detektoru, který tento výsledek dodal. A další klíč již obsahuje výsledek analýzy strukturálního detektoru.

Ukončení sekvence výsledků detekce

Název: `detector_fin`
Odesílatel: Detektor

Detektor odesílá jádru informaci, že končí jeho sekvence výsledků.

Název parametru	Popis parametru
`msg_id`	Unikátní číslo požadavku
`detector_name`	Název detektoru

Žádost o aktuální stav detekce

Název: `current_status_request`
Odesílatel: Jádro

Jádro může kdykoliv poslat žádost detektoru o aktuální stav detekce.

Název parametru	Popis parametru
`msg_id`	Číslo požadavku, u kterého má být zjištěn aktuální stav
`recipient_id`	Název detektoru

Odpověď na aktuální stav detekce

Název: `current_status_result`
Odesílatel: Detektor

Detektor odpovídá na žádost o aktuální stav detekce.

Název parametru	Popis parametru
`msg_id`	Číslo požadavku, u kterého má být zjištěn aktuální stav
`detector_name`	Název detektoru
`payload`	Aktuální stav žádosti ve formátu JSON

Aktuální stav žádosti rozšiřuje formát zprávy `detect_request_result`, kdy jsou doplněny klíče `status` (definuje stav požadavku) a `progress` (aktuální procentuální stav zpracování).

Přerušení aktuální detekce

Název: `interrupt`
Odesílatel: Jádro

V případě že zpracování nějaké žádosti trvá detektoru neúnosnou dobu, tak jej jádro může ukončit.

Název parametru	Popis parametru
`msg_id`	Číslo požadavku, u kterého má být ukončeno zpracování
`recipient_id`	Název detektoru

Detektor ukončuje zpracování daného požadavku.

Informace o databázi

Název: `process_db`
Odesílatel: Uživatel

Pro spuštění analýzy databáze je nutné nástroji předat informace o databázi. To je provedeno pomocí zavolání DBus metody `process_db` za pomoci linuxového nástroje `dbus-send`.

Název parametru	Popis parametru
`connection_string`	Informace o databázi ve formátu JSON

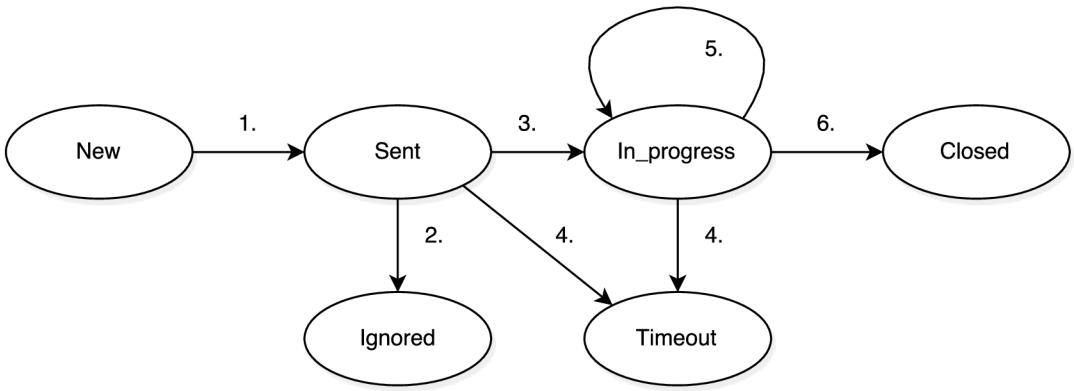
Struktura informací o databázi ve formátu JSON

```
1 {
2   "type": "mysql",
3   "host": "localhost",
4   "port": "3306",
5   "name": "db_name",
6   "user": "user_name",
7   "pass": "user_password",
8   "path": "/path/used/to/connect/to/sqlite/db"
9 }
```

3.2.2 Behaviorální návrh

Životní cyklus požadavku

Životní cyklus požadavku je definován na obrázku 3.3

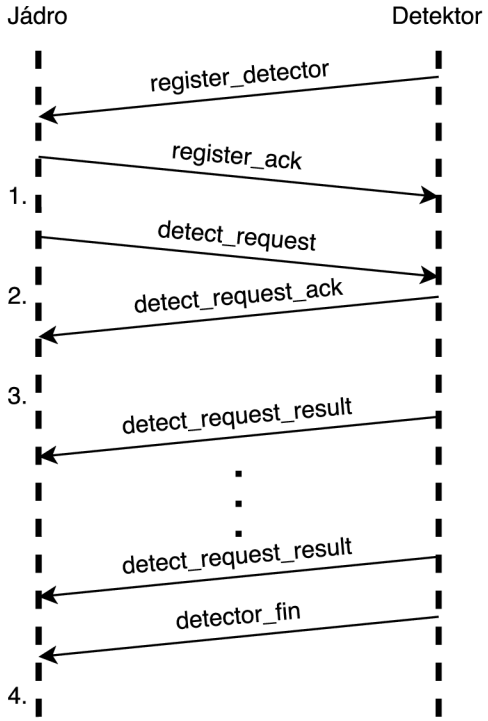


Obrázek 3.3: Stavy životního cyklu požadavku.

Popis jednotlivých stavů životního cyklu:

Stav	Popis stavu
New	Jádro vytvořilo nový požadavek, který je připraven na odeslání
Sent	Požadavek je odeslán pomocí sběrnice
Ignored	Požadavek je příjemcem ignorován
In_progress	Příjemce potvrdil příjem požadavku – požadavek je zpracováván
Closed	Příjemce ukončil zpracování požadavku
Timeout	Příjemce v daném časovém limitu neodeslal žádnou odpověď na daný požadavek

Ihned po vytvoření požadavku se nachází ve stavu *New*, v tomto stavu zůstává do té doby dokud není odeslán, například pomocí zprávy `detect_request`. Tímto přechází do stavu *Sent*, ve kterém setrvává do té doby dokud příjemce na daný požadavek neodpoví nebo nedojde k vypršení časového limitu. V případě že příjemce na daný požadavek odpoví záporně, tedy například zprávou `detect_request_nack`, tak požadavek přechází do stavu *Ignored*. Pokud se jádru nedostane odpovědi, tak požadavek přechází do stavu *Timeout*. Když příjemce na daný požadavek odpoví kladně, například zprávou `detect_request_ack`, tak požadavek přechází do stavu *In_progress*. V tomto stavu požadavek přetrvává do té doby, dokud příjemce odesílá odpovědi, tedy `detect_request_result`. Pokud dojde k vypršení časového limitu mezi přijatými odpověďmi, tak požadavek přechází do stavu *Timeout*. Když příjemce pro jádro nemá žádné další odpovědi, tak ukončuje sekvenci svých výsledků zprávou `detector_fin`, čímž požadavek přechází do stavu *Closed*. Případně se požadavek může dostat do stavu *Closed* také za pomoci zprávy `interrupt`, kdy dojde předčasnému ukončení zpracování požadavku.



Obrázek 3.4: Ukázka sekvence zpráv.

Popis jednotlivých bodů v diagramu

1. Požadavek se nachází ve stavu *New*.

2. Požadavek se nachází ve stavu *Sent*.
3. Požadavek se nachází ve stavu *In_progress*.
4. Požadavek se nachází ve stavu *Closed*.

Řešení závislostí detektorů

Řešení závislostí obsahuje následující fáze:

1. Kontrola výsledků analýz jednotlivých detektorů.
2. Aktualizace závislostí v případě nových výsledků.
3. Vytvoření kombinací výsledků analýz.
4. Tvorba požadavků z nových kombinací.

Kontrola výsledků analýz se provádí opakovaně, kontrolován je každý aktivní detektor [4]. V případě nalezení nových výsledků analýz, jsou aktualizovány záznamy závislostí detektorů, které mají závislost daného typu výsledku.

Následně dojde ke tvorbě kombinací výsledků a to pouze v případě, že daný detektor má splněny závislosti – tedy pro každý závislý typ detektoru lze nalézt alespoň jeden výsledek. Pokud detektor obsahuje více závislostí stejného typu, tak musí být zajištěna unikátnost této kombinace. Vytvořené kombinace jsou konfrontovány s již použitými kombinacemi, použité kombinace jsou tak odstraněny. Zbylé, unikátní, kombinace jsou připraveny ke tvorbě nových požadavků na zpracování. Tyto kombinace jsou také vloženy do použitých kombinací.

Tvorba požadavku spočívá ve sloučení výsledků závislých typů do jednoho celku, kdy tento celek je ve formátu JSON.

Příklad sloučených výsledků ve formátu JSON

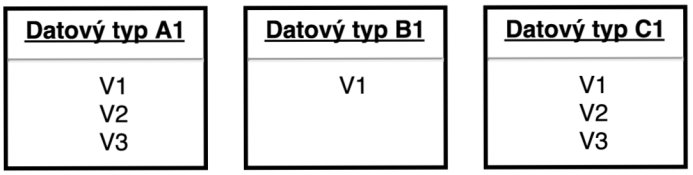
```
{
  "payloads": [{
    "detector_name": "detector1",
    "payload": {
      "type": "dbd.int",
      "weight": 85
    }
  }, {
    "detector_name": "detector1",
    "payload": {
      "type": "dbd.int",
      "weight": 90
    }
  }]
}
```

Klíč `payloads` obsahuje kolekci objektů, kdy každý objekt představuje jeden výsledek závislého typu.

Následně je vytvořen požadavek na zpracování, u kterého je specifikováno, kterému detektoru má být požadavek doručen.

[4] Detektory, které jsou registrovány u jádra.

Mějme detektor, který má závislost tří datových typů (tak jak je uvedeno na obrázku 3.5), každý závislý datový typ již obsahuje nějaké výsledky. Pokud detektor obsahuje více závislých datových typů, tak jádro musí zajistit, že detektoru budou poslány požadavky všech kombinací těchto výsledků, tedy kartézský součin všech množin výsledků závislých datových typů.



Obrázek 3.5: Příklad závislých datových typů.

Požadavky na detekci tedy budou poskládány z výsledků následovně:

1. A1.V1, B1.V1, C1.V1
2. A1.V1, B1.V1, C1.V2
3. A1.V1, B1.V1, C1.V3
4. A1.V2, B1.V1, C1.V1
5. A1.V2, B1.V1, C1.V2
6. A1.V2, B1.V1, C1.V3
7. A1.V3, B1.V1, C1.V1
8. A1.V3, B1.V1, C1.V2
9. A1.V3, B1.V1, C1.V3

Kapitola 4

Implementační detaily

Tato kapitola popisuje zajimavé implementační detaily a algoritmy použité při tvorbě nástroje.

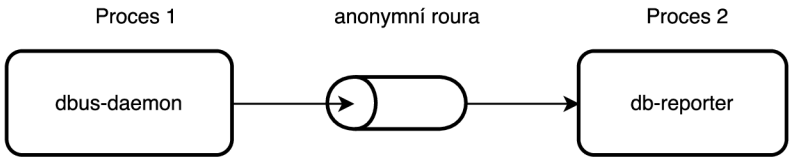
Nástroj je implementován v jazyce C++, rozhraní nástroje je implementováno formou příkazové řádky.

4.1 Použité knihovny

Kromě STL[1] knihoven jsou využity dvě externí knihovny.
Knihovna *JSON for Modern C++*[2] je využita pro práci s JSON objekty.
Knihovna *Super fast C++ logging library*[3] je využita pro logování stavu nástroje.

4.2 Tvorba vlastní sběrnice DBus

Vlastní sběrnice je vytvářena ve třídě `dbus_lib` v metodě `dbus_prepare`. Vytvoření je prováděno tak, že je nejprve vytvořena anonymní roura [4] (dále jen roura) mezi dvěma procesy.



Obrázek 4.1: Nákres meziprocesové komunikace za užití anonymní roury

Kdy *proces 1*, který je nalevo od roury, má přesměrovaný standardní výstup směrem do roury. *Proces 2*, který je napravo od roury, má přesměrovaný standardní vstup směrem z roury. Tedy *proces 2* čte na standardním vstupu data, která *proces 1* zapsal na standardní výstup.

[1] Standard template library; softwarová knihovna jazyka C++
[2] Dostupné z: https://github.com/nlohmann/json
[3] Dostupné z: https://github.com/gabime/spdlog
[4] Anglicky *anonymous pipe*; rozhraní umožňující komunikaci mezi procesy

Proces 1 je linuxový nástroj `dbus-daemon`, který vytváří sběrnici DBus a zobrazuje její adresu na standardní výstup (popřípadě ji posílá do specifikovaného souborového deskriptoru [5]).
Proces 2 je předmětný nástroj této bakalářské práce – `db-reporter`.

Pro spuštění linuxového nástroje `dbus-daemon` přímo ze zdrojového kódu je využito funkce `execl`.

```
int execl(const char *path, const char *arg, ...);
```

Podle konvence by měl první argument ukazovat na jméno souboru, který má být spuštěn. Další argumenty jsou argumenty programu, které musí být ukončeny NULL ukazatelem, který je přetypován na `char *`.

Proměnné použité v následující ukázce

```
1 enum{STDIN=0, STDOUT=1};
2 static char *const dbus_daemon_args[]={
3   const_cast<char *>("/usr/bin/env"),
4   const_cast<char *>("dbus-daemon"),
5   const_cast<char *>("--session"),
6   const_cast<char *>("--print-address="),
7   const_cast<char *>(static_cast<char *>(nullptr))
8 };
```

Každý prvek pole `dbus_daemon_args` musí být přetypován užitím funkce `const_cast<char *>`, neboť bez použití tohoto přetypování při překladu vzniká varování na nekompatibilitu datových typů.

Nástroj `dbus-daemon` je spouštěn v aktuálním prostředí za použití nástroje `/usr/bin/env`.

Zdrojový kód části funkce zajišťující vytvoření vlastní sběrnice

```
1  int fd[2], n_bytes;
2  pid_t childpid;
3  char buffer[80];
4
5  if(pipe(fd)!=0)
6      throw dbus_exception("Pipe failed", HERE);
7
8  if((childpid=fork())==-1)
9      throw dbus_exception("Fork failed", HERE);
10 if(childpid==0)
11 {
12     close(fd[STDIN]);
13     std::string addr_with_fd{dbus_daemon_args[3]+std::to_string(fd[STDOUT])
           };
14
15     if(execl(dbus_daemon_args[0],
16              dbus_daemon_args[0],
17              dbus_daemon_args[1],
18              dbus_daemon_args[2],
19              addr_with_fd.c_str(),
20              dbus_daemon_args[4])<0)
21         throw dbus_exception(strerror(errno), HERE);
22 }
23 else
24 {
25     close(fd[STDOUT]);
26     n_bytes=read(fd[STDIN], buffer, sizeof(buffer));
27     buffer[n_bytes]='\0';
```

[5] Anglicky *file descriptor*

```
28      dbus_address=buffer;
29      dbus_daemon_pid=childpid;
30      close(fd[STDIN]);
31  }
```

- Na řádcích 1 až 3 je deklarace proměnných.
  - Proměnná `fd` slouží pro uložení dvou souborových deskriptorů, kdy každý odkazuje na jeden konec roury.
  - Proměnná `n_bytes` slouží uložení návratové hodnoty funkce `read`, která navrací počet přečtených bajtů.
  - `childpid` slouží pro uložení návratové hodnoty funkce `fork`, tedy uložení PID [6] potomka.
  - `buffer` slouží pro uložení hodnot přečtených ze souborového deskriptoru.
- Řádky 5 až 9 ošetřují možné chybové stavy.
- Na řádcích 10 až 22 je proces potomka.
  - Řádek 13 – čtvrtý argument funkce `execl`, tedy argument `--print-address=` je konkatenován s číslem souborového deskriptoru, který představuje vstup do roury.
  - Řádky 15 až 20 provádí spuštění nástroje `env` se všemi argumenty, řadek 21 ošetřuje případnou chybu.
- Na řádcích 23 až 31 je proces rodiče.
  - Na řádku 26 je přečtena adresa nové sběrnice ze souborového deskriptoru do proměnné `buffer`, tato adresa je dále přesunuta do proměnné `dbus_address`, která si uchovává adresu sběrnice.

4.3 Práce s vlákny

V nástroji jsou využity vlákna z STL knihovny. Tyto vlákna jsou užita na dvou místech:

- Vlákno pro příjem zpráv ze sběrnice DBus.
- Vlákno pro odesílání nových požadavků a vyhodnocování závislostí.

V obou případech je potřeba využít synchronizačního prostředku pro zajištění výlučného přístupu ke kritické sekci programu. Toto je řešeno pomocí třídy `std::mutex` [7], kde je využita funkce `lock()` pro uzamknutí kritické sekce a funkce `unlock()` pro odemknutí.

Jako řídící proměnné jsou proměnné zapouzdřené ve třídě `std::atomic`, tato třída garantuje, že nedojde k časově závislé chybě nad daty [8].

Vytvoření vlákna

```
processing_thread=std::thread(core::process_thread, this);
```

Parametry objektu `std::thread` je statická funkce, kterou má vlákno provádět a parametry této funkce. Objekt navrací do proměnné `processing_thread` objekt vlákna.

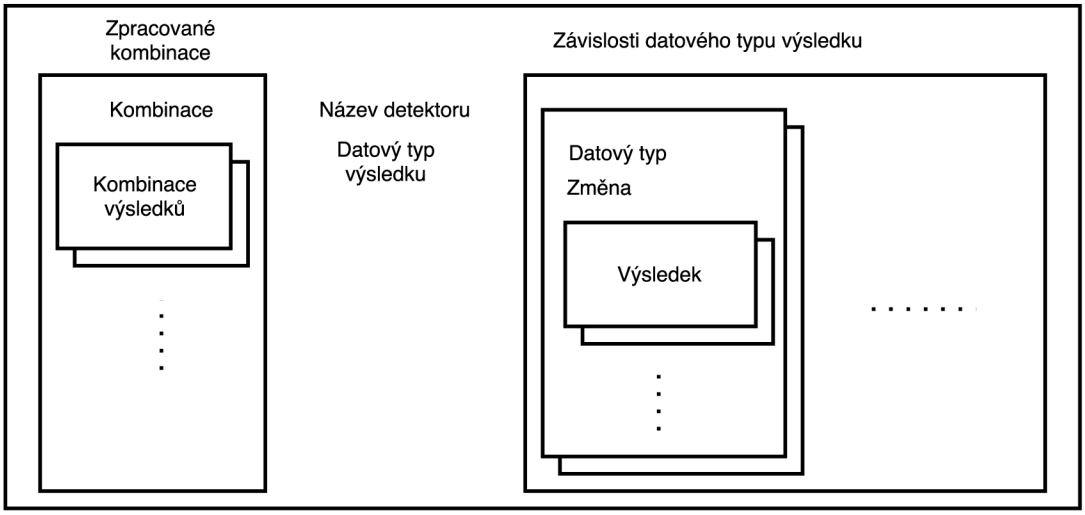
[6] Anglicky *process identifier*; číslo pod kterým je v jádře operačního systému jednoznačně evidován proces
[7] Více na http://www.cplusplus.com/reference/mutex/mutex/
[8] Anglicky *data race*

4.4 Řešení závislosti detektorů

Závislosti detektorů jsou řešeny ve třídě `dependency_manager`, která obsahuje referenci na třídu `register_c`, kde jsou uloženy výsledky. Tento přístup byl zvolen zejména kvůli zjednodušení práce s těmito třídami. Třída `dependency_manager` zajišťuje správu závislostí detektorů, správu zpracovaných kombinací výsledků, jejich aktualizaci a tvorbu nových požadavků na detekci.

Závislosti jsou ukládany v komplexní datové struktuře, která nejenže uchovává jednotlivé závislosti, ale také již zpracované kombinace výsledků.

Tato kompexní datová struktura je použita pro každou unikátní trojici *název detektoru*, *datový typ výsledku* a *závislosti datového typu výsledku*. Tedy v případě že detektor navrací dva datové typy výsledku, tak jsou vytvořeny dvě odpovídající komplexní datové struktury.



Obrázek 4.2: Nákres komplexní datové struktury.

Metoda zajišťující aktualizování výsledků v komplexní datové struktuře

```
1  void dependency_manager::update_dependencies()
2  {
3    for(auto &item: dependency)
4    {
5      for(auto &depend_datatype: item.type_dependency)
6      {
7        auto result_it=register_ptr->result_pool.find(depend_datatype.type);
8        if(result_it!=register_ptr->result_pool.end())
9        {
10         auto result_size=depend_datatype.type_result.combinations.size();
11         for(auto result=result_it->second.begin();
12           result!=result_it->second.end();
13           ++result)
14         {
15           if(std::find(
16             depend_datatype.type_result.combinations.begin(),
17             depend_datatype.type_result.combinations.end(),
18             result->get())
19             ==depend_datatype.type_result.combinations.end())
20           {
21             depend_datatype.type_result.combinations.push_back(
```

```
22                  result->get());
23              }
24            }
25            if(result_size!=depend_datatype.type_result.combinations.size())
26              depend_datatype.changed=true;
27            else
28              depend_datatype.changed=false;
29          }
30        }
31      }
32  }
```

- Na řádcích 3 až 5 začínají cykly, nejprve přes komplexní datové struktury a následně přes závislosti datového typu výsledku.
- Na řádcích 7 a 8 je hledání všech výsledků daného datového typu (`depend_datatype.type`). Pokud výsledek není nalezen, tak se přechází na hledaní dalšího závislého datového typu.
- Na řádku 10 je uložen počet nalezených výsledků pro následné vyhodnocení změny v komplexní datové struktuře.
- Na řádcích 11 až 13 je začátek cyklu přes všechny nalezené výsledky daného datového typu.
- Na řádcích 15 až 19 probíhá ověření, zda už je daný výsledek uložen v komplexní datové struktuře.
- V případě že se jedná o nový výsledek (není nalezen v komplexní datové struktuře), tak je vložen do komplexní datové struktury – řádky 21 a 22.
- Řádek 25 obsahuje kontrolu změny počtu uložených výsledků v komplexní datové struktuře.
- Pokud byl přidán výsledek, tak se přechází na řádek 26, kde je nastaven příznak změny.
- Pokud nebyl přidán výsledek, tak se přechází na řádek 28, kde je zrušen příznak změny.

4.5 Použití nástroje

Nástroj lze přeložit pomocí nástroje `make`, po překladu je vytvořen v aktuální složce spustitelný soubor `reporter`. Tento soubor je možné spustit z příkazové řádky s parametry, které specifikují následující činnost nástroje.

Parametry příkazové řádky

`-h`	Zobrazení nápovědy nástroje
`-d`	Výpis adresy sběrnice na standardní výstup
`-s`	Připojení nástroje na sběrnici sezení
`-v=NUM`	Nastavení úrovně logování, 1 – nejnižší úroveň, 3 – nejvyšší úroveň

Pokud je program spuštěn bez argumentů příkazové řádky, tak dochází k tomu, že si nástroj vytváří vlastní sběrnici, na kterou se následně připojí.

Kapitola 5

Ověření funkcionality nástroje

Pro ověření funkcionality nátroje byly vytvořeny testovací sady, které ověřují funkční požadavky na nástroj.

5.1 Testovací sada č. 1

Tato sada testů slouží k otestování funkčního požadavku uvedeného zde: 3.1.1. Jedná se o testy, které ověřují správné připojení na sběrnice DBus, a to jak na sběrnici sezení, tak i na sběrnici, která je nástrojem vytvořena.

Spuštěním testů dojde ke spuštění samotného nástroje, v případě připojení na sběrnici sezení je nástroj spuštěn s parametrem `-s`. V dalším testu, ověření připojení na vlastní sběrnici, je nástroj spuštěn s parametrem `-d`.

Ověření zda je nástroj připojen na sběrnici je provedeno pomocí nástroje `dbus-send`.

5.2 Testovací sada č. 2

Tato sada testů ověřuje komunikaci jádra s detektory 3.1.4 a dodržování životního cyklu požadavku 3.1.5.

Nejprve je otestována běžná sekvence zpráv, tím je ověřena komunikace jádra s okolím. Dále jsou testovány jednotlivé stavy životního cyklu a zejména pak stavy, u kterých dochází k vypršení časového limitu.

5.3 Jednotkový test

Tento test se zabývá kritickou částí nástroje, kterou je správa závislostí detektorů. Jedná se o jednotkový test třídy `dependency_manager`. Tento test tedy ověřuje správné zpracování závislostí detektorů 3.1.2 a správné ukládání výsledků jednotlivých analýz 3.1.3.

Aby test mohl být spuštěn je potřeba nejprve vložit testovací výsledky do třídy pro uchovávání výsledků analýz. Dále pak po každé fázi testu jsou vloženy další testovací výsledky, aby mohlo být otestováno i další generování požadavků.

Ověření správnosti jednotkového testu je provedeno pomocí porovnání s předpokládanými požadavky, které mají být vygenerovány.

Kapitola 6

Závěr

Cílem bakalářské práce bylo navrhout a implementovat jádro nástroje pro automatickou analýzu existující databáze. Nástroj je schopen sám iniciovat požadavky na detektory, ukládat výsledky analýz a řešit závislosti detektorů. V rámci této práce je užita sběrnice DBus jako komunikační kanál mezi jádrem a jednotlivými detektory. Jádro umožňuje připojení na sběrnici sezení, případně na sběrnici jím vytvořenou a komunikovat tak s detektory. Navržené hierarchické členění tříd umožňuje snadné vytvoření detektoru nebo případně i dalšího jádra, za použití již implementovaných tříd.

Je nutno podotknout, že se jedná o první nástroj tohoto druhu, který se zabývá výhradně analýzou existující databáze pro účely testování. Existující projekty se spíše zaměřují na visualizaci obsahu databáze.

6.1 Známé nedostatky

Nástroj je schopen spouštět analýzu pouze nad jednou databází. Je možné nástroji předat i více než jednu databázi, ale druhá databáze nebude nikdy zpracována.
Pro budoucí rozšiřování projektu je také nutné vytvořit podrobnou dokumentaci programového řešení.

Nástroj nyní pracuje s DBus signály, neboť na začátku vývoje nástroje nebylo plánováno, že detektor bude možné unikátně nalézt na sběrnici.

6.2 Možnosti pokračování projektu

Nástroj může po analýze celé databáze vytvořit HTML stránky, které budou sloužit jako prezentace výsledku analýzy databáze. Bude také umožňovat zobrazení výsledků analýz jednotlivých detektorů.

Přepracovat jádro i detektory pro komunikaci pomocí DBus metod, čímž by mohlo dojít k redukci parametrů u zpráv. Doplnit jádro a detektory o standardní rozhraní sběrnice DBus.

Literatura

[1] AquaFold - Tools for Software & Database Professionals. [online]. [cit. 2017-05-06]. Dostupné z: http://www.aquafold.com/.

[2] *Database test data generator - Fill your database with random test data!* [online]. [cit. 2017-05-06]. Dostupné z: http://www.databasetestdata.com/.

[3] *dbus*. [online]. [cit. 2017-05-06]. Dostupné z: https://www.freedesktop.org/wiki/Software/dbus/.

[4] *generatedata.com*. [online]. [cit. 2017-05-06]. Dostupné z: https://www.generatedata.com/.

[5] *Logi DevNet*. [online]. [cit. 2017-05-06]. Dostupné z: http://devnet.logianalytics.com/.

[6] *Mockaroo - Random Data Generator | CSV / JSON / SQL / Excel*. [online]. [cit. 2017-05-06]. Dostupné z: https://www.mockaroo.com/.

[7] *Nástroj pro analýzu obsahu databáze pro účely testování softwaru*. [online]. [cit. 2017-05-06]. Dostupné z: https://pajda.fit.vutbr.cz/testos/db-reporter.

[8] *Nástroj pro analýzu obsahu databáze pro účely testování softwaru*. [online]. [cit. 2017-05-06]. Dostupné z: https://pajda.fit.vutbr.cz/testos/db-detectors.

[9] *Nástroj pro generování obsahu databáze pro účely testování softwaru*. [online]. [cit. 2017-05-06]. Dostupné z: https://dspace.vutbr.cz/handle/11012/62169.

[10] *Skupina Testos. Domovská stránka projektu Testos*. [online]. FIT VUT v Brně. 2017 [cit. 2017-05-06]. Dostupné z: http://testos.org.

[11] *SQL Server tools for DBAs and developers | ApexSQL*. [online]. [cit. 2017-05-06]. Dostupné z: http://www.apexsql.com/.

[12] Ochodek, M.: *Nástroj pro analýzu obsahu databáze pro účely testování softwaru*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.

Přílohy

Příloha A

Obsah CD

Adresářová struktura CD

- *doc* – Dokumentace nástroje.
- *lib* – Externí knihovny.
- *src* – Zdrojové soubory.
- *test* – Testovací sady.
- *text* – Text práce.

A.1 Návod na instalaci

Nástroj je závislý na balíčku `libdbus-1-dev`, který lze nainstalovat pomocí příkazu `apt install libdbus-1-dev`. Dále nástroj obsahuje závislost na externích knihovnách pro práci s JSON formátem a pro logování stavu nástroje.

Stažení knihovny pro práci s JSON formátem:

```
cd lib/
git clone https://github.com/nlohmann/json.git
```

Stažení knihovny pro logování stavu:

```
cd lib/
git clone https://github.com/gabime/spdlog.git logging
```

Poté je možné nástroj přeložit pomocí příkazu `make`. Spuštění automatizovaných testů je docíleno příkazem `make tests`.