Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Gráfmintaillesztő rendszerek tesztelése automatikus gráfgenerátorokkal

TDK dolgozat

Készítette:

Bekő Mária

Konzulens:

Semeráth Oszkár

2018

Tartalomjegyzék

Kivonat

Napjainkban az adatokat többféle formátumban is tárolják. Ezek közé tartoznak a gráfadatbázisok, ahol csomópontok reprezentálják az entitásokat és az élek az entitások közötti kapcsolatokat. Az adatstruktúrához illeszkedve többféle gráflekérdező nyelv jött létre, amelyek képesek komplex struktúrák felírására.

A gráfmintaillesztő rendszerek tesztelése azonban komoly kihívást jelent, főképp automatizált megoldásokban nem bővelkedünk. A legnagyobb kihívást ebben az esetben a változatos modellek és lekérdezések automatikus és szisztematikus előállítása jelenti, melyek tesztbemenetként szolgálnak. Továbbá, gráfadatbázisok teljesítménymérését is nagyban segítené az automatikusan előállított modellkészlet.

Dolgozatom célja hogy ezekre a problémákra megoldást találjak. Kutatásom során megmutatom, hogy egy automatikusan előállított diverz modell halmazzal, amelynek modelljei lekérdezésként értelmezhetőek egy gráfmintaillesztő rendszerben (pl.: VIATRA vagy Neo4j), hogyan lehetséges az adott gráfmintaillesztő rendszer tesztelése.

Munkám során fejlett logikai következtetők alkalmazásával állítok elő modelleket, melyek diverzitását szomszédsági formákkal (neighborhood shape-ek) biztosítom. A logikai következtetők eredményeit lekérdezésekként, és adatbázisok tartalmaként egyaránt értelmezhetjük, amelyek eredményei különböző megvalósításokkal összehasonlíthatóvá válnak. A megoldásomat egy esettanulmány keretében prezentálom.

Ezzel a módszerrel lehetővé válik, nagyobb megbízhatóságú gráfmintaillesztő rendszerek fejlesztése olcsóbban. Illetve egy ekkora modell halmaz különböző gráfmintaillesztő rendszerekben lekérdezésekre fordítva és a válaszidőket lemérve teljesítménymérésekre is használható.

Abstract

Nowadays the data is stored in multiple formats. One of this is the graph database, where entities are represented as graph nodes and the relations between entities as edges. Utilizing rich data structure, a variety of graph querying languages have been created in order to query complex structures.

Testing of graph query engines is a challenging task, especially in an automated way. The greatest challenge in this case is the automatic and systematic creation of a diverse set of models and queries, which serve as test inputs. In addition, the development of performance benchmarks for graph databases would be greatly aided.

The purpose of my thesis is to find solutions to these problems. In my thesis, I will show an approach to automatically generate a diverse set of models, that can be interpreted as queries of the graph query engine under test (e.g. VIATRA model query language or Neo4j graph database queries). Additionally, I propose a testing process.

In the course of my work, I produce models with the help of state-of-the-art logic solvers (SAT solvers and Graph Solvers), and use neighborhood shapes to ensure their diversity and create effective equivalence partitioning. The results of the logic solvers are interpreted as queries and as databases content, and the result of query evaluation can be compared to other implementations. I illustrate my solution in a case study.

This method makes it possible to develop more reliable graph query engines at a lower cost. And such a set of models translated to multiple graph querying languages can be used in those graph query engines for performance measurements by measuring response times.

1. fejezet

Bevezetés

Kontextus. Napjainkban ....

Problémafelvetés. Azonban ...

Célkitűzés. Dolgozatom célja ...

Kontribúció. Dolgozatomban bemutatok ...

Hozzáadott érték. Ezáltal ...

Dolgozat felépítése.A második fejezetben bemutatom a dolgozat megértéséhez szükséges háttérismereteket. blabla ...

2. fejezet

Előismeretek

2.1. Esettanulmány

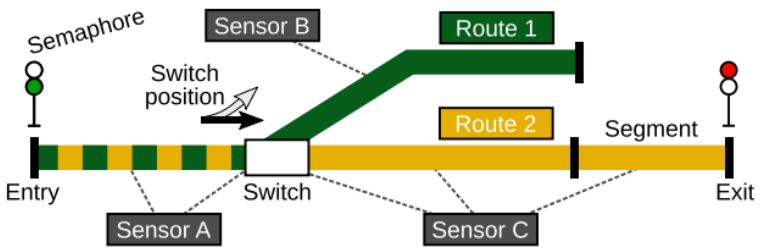
A dolgozatban elért eredményeket a Train Benchmark [11] esettanulmány segítségével fogom bemutatni. Ez a benchmark azért jött létre, hogy össze tudjuk hasonlítani különböző gráflekérdező rendszerek teljesítményét, főleg időigény és memória felhasználás szempontjából. Ehhez többek között definiálja a vasúti rendszer metamodelljét. Munkám során ezt a metamodellt felhasználva generáltam lekérdezéseimet, ezért bemutatom, hogy milyen elemekből áll.

A 2.1-es ábrán látható egy a trainbenchmark metamodelljére alapuló részlet. Ebben a kontextusban egy vasúti útvonal nem más mint szegmensek és váltók sorozata, illetve a belépést és a kilépést egy-egy szemafor jelzi. Ahhoz hogy biztonságos legyen a közlekedés szükség van szenzorokra, amelyek monitorozzák a különböző szegmensek és váltók kihasználtságát. Egy útvonal definiálásához, a felsorolt elemeken kívül a váltók adott útvonalhoz tartozó pozícióját is el kell tárolnunk. Egy útvonal akkor aktív, ha a rendszerben a specifikációjának megfelelően állnak a váltók.

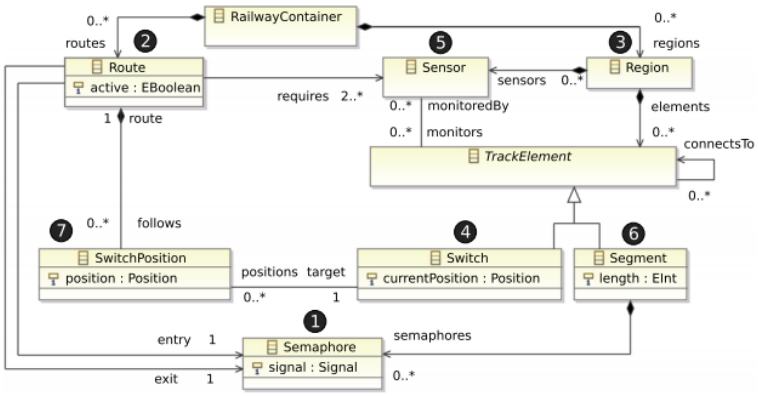
A metamodellezés egy tecknika arra, hogy definiájunk modellező nyelveket, ahol a metamodell specifikálja a nyelv szintaktikáját. A trainbenchmark metamodellje a 2.2 -es ábrán látható.

2.2. Gráflekérdező rendszerek

A gráfok intuitív formalizációt nyújtanak modellezési szempontból arra, hogy úgy írhassuk le a világot ahogy az ember gondolkozik róla. Tehát mint dolgok (csomópontok) és köztük lévő kapcsolatok (élek) [6]. A property gráf adatmodell kiterjeszti a gráfokat úgy, hogy címkéket/típusokat illetve tulajdonságokat ad a csúcsoknak és az éleknek. A gráf adatbázisok alkalmasak tulajdonság gráfok tárolására, és az abban lévő adatok lekérdezésére



2.1. ábra. Vasúti útvonal részlet (forrás: [11])



2.2. ábra. A train benchmark metamodellje.(forrás: [11])

komplex gráf minták használatával. Ilyen rendszerek például a Neo4j [1], OrientDB [3] és a SparkSee [5].

Ahhoz hogy jobban megérthessük mi is ez az adatmodell a 2.3 -es ábrán látható egy példa.

2.2.1. Neo4j

A Neo4J egy populáris NoSQL property gráf adatbázis és a Cypher lekérdező nyelvet kínálja lekérdezések írására. A Cypher egy magas szintű deklaratív lekérdező nyelv és mivel le van választava a lekérdező rendszerről, ezért az képes a Cypher nyelven írt lekérdezések optimalizálására. A Cypher szintaxisa olyan gráf minták megírását teszi lehetővé, amelyeknek megértése nagyon egyszerű.

```
MATCH (tr:Train)-[:ON]->(seg:Segment)
RETURN tr, seg
```

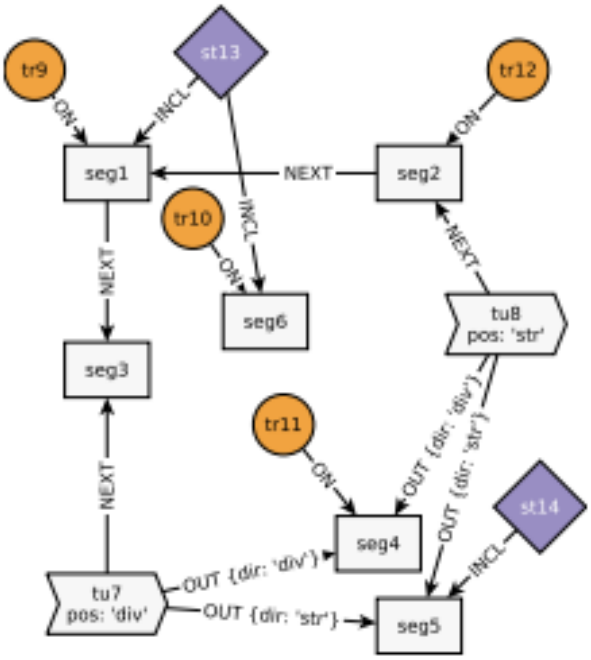
A fenti példán egy olyan lekérdezést láthatunk, ami az összes olyan vonat, szegmens párral tér vissza, ahol az adott vonat rajta van az adott szegmensen.

2.3. Modellezés és metamodellezés

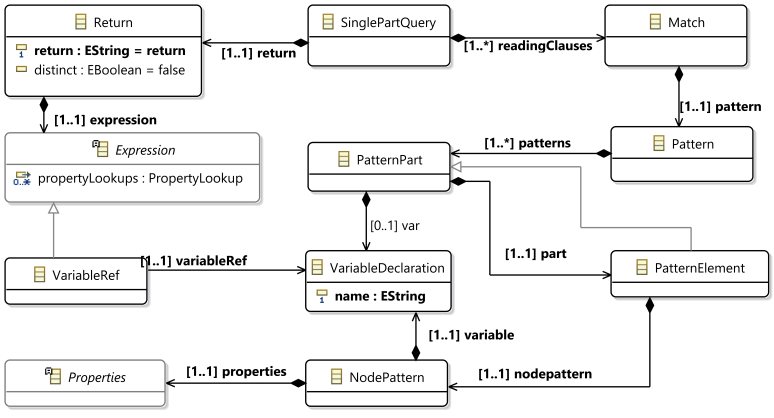
A metamodellek definiálják a legfontosabb fogalmakat, relációkat és attribútumokat a cél domainen , hogy specifikálják modellek alap struktúráját [9]. Dolgozatomban az Eclipse Modeling Framework (EMF) [4] -öt használtam metamodellezésre.

2.3.1. Cypher query-k metamodellje

A 2.4 -ös ábrán látható a korábban említett Cypher nyelv egyszerűsített metamodellje. A SinglePartQuery elem reprezentálja a modell gyökerét. Egy ilyen elem két részből áll : Egy Match és egy Return elemből. A Match elem minták összességéből áll (Pattern), amelyek részekre (PatternPart) bonthatóak. Egy ilyen részben pedig vagy tartalmaz változó deklarációt, vagy egy belső részt tartalmaz ami taltalmaz változó deklarációt. (Variable Declaration). Ezáltal az összes változót a Match elemen belül deklaráljuk. A Return elem pedig egy kifejezést (Expression) tartalmaz, amelyben mindenképp szerepel egy változó referencia (VariableRef) is, így összekötve egymással a Match és a Return elemet.



2.3. ábra. Property gráf példa(forrás: [6])



2.4. ábra. Cypher metamodell

2.3.2. xText

Az Xtext keretrendszer programozási nyelvek és domain-specifikus nyelvek fejlesztésére készül. Az Xtext egy erős nyelvtani szabályokkal rendelkező nyelvet használ az egyedi nyelvtanok definiálására. Ezáltal egyszerre biztosít parszolót, linkelőt, helyesírásellenőrzőt és fordítót. És a felhasználó határozhatja meg a nyelvének célformátumát is.Ahhoz, hogy a Cypher nyelven megírt lekérdezéseket értelmezni lehessen a 2.4-ös metamodellen megismert elemek szintjén egy XText [2] keretrendszerben íródott nyelvtanra van szükség. A dolgozatomban a slizaa[10] által készített nyelvtanra építettem.

```
SinglePartQuery:
(readingClauses+=ReadingClause)* return=Return ;

Return:
(return='RETURN' distinct?='DISTINCT'? body=ReturnBody);
```

A fenti ábrán a SinglePartQuery elem Xtext nyelvtana látható. Azt mondja ki, hogy amikor egy ilyen elem készül akkor összerak egy vagy több readingClause-t (Match elem absztrakt ősosztálya), és egy returnt. Alatta pedig a Return elem Xtext nyelvtana következik, ami azt mondja ki hogy a return elemet úgy kell sorosítani, hogy ”RETURN DISTINCT(ezt csak akkor kell odaírni ha ezt a property-t igazra állítottuk) kifejezés”. Tehát itt határozza meg hogy a Return elem úgy néz ki mint a **??** lékérdezésen.

2.3.3. Viatra jólformáltsági kényszerek

Az Eclipse VIATRA keretrendszer [8] egy model és adat transzformáló eszköz. Specifikusan olyan eseményvezérelt és reaktív transzformációkra fókuszál, amelyek a modell változása közben történnek. A legnagyobb előnye, hogy lezárja az absztrakciós lyukakat. A lekérdezés generálás során arra használom, hogy a metamodell alapján nem meghatározott mégis betartandó kényszereket meghatározzam.

2.4. Gráfgenerálás

Munkám alapját mégis leginkább gráfok generálása képezi. A Viatra Solver [12] egy korszerű nyílt forráskódú szoftver keretrendszer amely képes diverz szakterület-specifikus gráf modellek automatikus szintézisére, melyek teszt készletként használhatóak gráf alapú modellező eszközök szisztematikus tesztelése során. Bemenetként a megoldó a tesztelni kívánt modellező eszköz specifikációját használja fel metamodell formátumban az Eclipse Modeling Framework-öt használva, jólformáltsági kényszerek egy halmazát a Viatra keretrendszer használatával, és opcionálisan egy példánymodell részletet. Kimenetként pedig diverz gráfok egy halmazát generálja. Minden kimeneti gráf megfelel a metamodell specifikációinak és kielégíti az összes jólformáltsági kényszert. Struktúrájukban pedig különböznek egymástől biztosítva ezzel tesztkészlet diverzitását. Én ezt a keretrendszert használom.

3. fejezet

Áttekintés

3.1. Funkcionális áttekintés

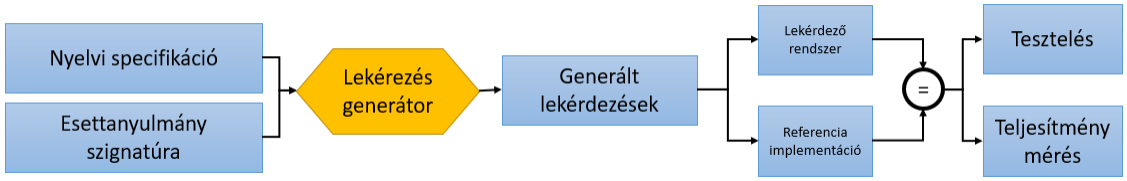
Munkám célja hogy mutasson egy olyan megközelítést, amelynek segítségével lehetséges gráfmintaillesztő rendszerek tesztelése. Az elképzelést a 3.1 ábrán mutatom be.

Az ötlet lényege az, hogy a tesztelni kívánt rendszer nyelvi specifikációjának és egy esettanulmány szignatúrájának (ezalatt egy olyan property gráf adatmodell alapú adatbázis szignatúrájára gondolok amely az adott lekérdező rendszert használja) bemenetként való felhasználásával, a kimeneten szöveges és a gráfmintaillesztő rendszer nyelvén íródott lekérdezéseket kapjak.

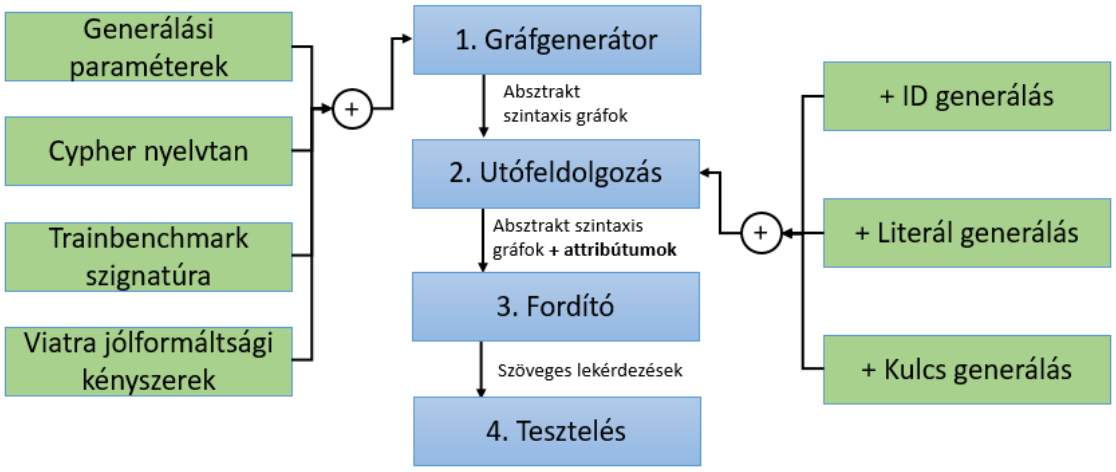
Amint rendelkeznék egy tesztkészletnyi ilyen lekérdezéssel azokat tudnám futtatni azon az adatbázison amelynek szignatúráját esettanulmányként választottam. Ha az ezzel a módszerrel generált lekérdezésekre adott válaszok válaszidejeit összehasonlítjuk több rendszeren tudnánk teljesítményben tesztelni azokat. Illetve ha a generált lekérdezések helyes eredményével is rendelkeznénk akkor ellenőrizni tudnánk azt is hogy a teszt alatt álló lekérdező rendszer hogyan funkcionál, helyes válaszokat ad-e? Ha elég bonyolultak a lekérdezések, akkor az is lehetséges, hogy lesznek olyanok amelyek az egyes lekérdező motorokon nem míg másikakon működnek, így azt is tesztelni tudnánk hogy mekkora az egyes lekérdező motorok funkcionális lefedettsége.

Felmerül a kérdés, hogy mégis miért lesz ez jobb, mintha írnánk a lekérdezéseket magunk. Azért, mert a generálás segítségével ki tudunk törni az emberi sematikus gondolkozásból, és olyan lekérdezéseket tudunk készíteni amelyek számításokkal bizonyítottan különböző ekvivalencia osztályba tartoznak. //TODO mi az? Illetve nem korlátoz minket az sem, hogy a teszteléshez írt lekérdezésekből túl kevés van, mert a generátor segítségével megadott számú minta, akár óriási tesztkészlet előállítható.

A megközelítésemet egy Neo4j [1] property gráf adatbázison mutatom be, amely a Train benchmark [11] által használt szignatúrával van felszerelve. A lekérdezéseket a Neo4j által kifejlesztet Cypher [7] nyelven generálom, a slizaa [10] által készített openCypher nyelvi specifikáció felhasználásával.



3.1. ábra. Az elképzelés funkcionális áttekintése



3.2. ábra. A lekérdezés generálási folyamat áttekintése

3.2. Lekérdezés generálási folyamat felépítése

A folyamat felépítését a 3.2 -es ábra segítségével ismertetem.

1. **Gráfgenerátor** A gráf generálást a Viatra Solver keretrendszer segítségével végzem. Ehhez sok különböző bemenetet kell megadnom.

   (a) **Nyelvi specifikáció**: A Cypher nyelv specifikációját tartalmazó metamodell a nyelv egészére kiterjed. Így lekérdezéseken kívül sok egyéb műveletet is definiál, mint például létrehozás, törlés. Olyan elemeket is tartalmaz amelyek csak bonyolítják a lekérdezéseket, hogy felhasználó centrikusabban adhassák vissza a tartalmat, például a visszatérési referencia átnevezése, az adatok csökkenő sorrendbe rendezése. Ahhoz, hogy egyszerű lekérdezéseket generáljak nem szükséges ezt a hatalmas metamodellt feldolgozni, viszont egyértelműen meg kell határozni egy olyan részmodelljét, amelyből hiánytalanul előáll az egyszerű lekérdezések nyelvi specifikációja.

   (b) **Kényszerek**:Azonban vannak olyan szabályok amelyeket a metamodell nem tud kifejezni, betartásuk nélkül viszont a generált példánygráfok nem értelmezhetőek Cypher nyelvű lekérdezésekként. Például annak meghatározása, hogy milyen változókra lehet, és milyenekre nem lehet hivatkozni a visszatérési értékben. Ezeket a szabályokat jólformáltsági kényszerekkel tartatom be.

   (c) **Konfigurációs paraméterek**: A generátor működéséhez elengedhetetlen a saját nyelvén íródott konfigurációs fájl. Itt határozható meg, hogy milyen megoldóval működjön a generálás, hogy hányat használjon az egyes elemekből a generálás során, hogy mekkora példányokat generáljon stb.

   (d) **Esettanumány szignatúra** : A generált példánygráfok változók nélkül jönnek létre. Ahhoz, hogy egy értelmes adatbázison végezhessük el őket, fontos hogy fel legyenek fegyverezve annak az adatbázisnak a címkéivel, típusaival. Ezért hát össze készítettem több halmaznyi szót, ami a Train benchmark által használt adatbázis címkéiből áll.

2. **Utófeldolgozás** : Az általam generált gráfokban a változóknak nem adok értéket. Megtehetném, hogy a generálás során kitöltöm őket, de csak úgy, hogy a gráfgenerátor a generált szavakat különbségekként kezelje két példánygráf között. A nagyobb diverzitás elérésének érdekében döntöttem úgy hogy üresen hagyom az értékeket. Az

utófeldolgozás során az Esettanulmány szignatúra szavaival töltöm fel az addig még csonka példánygráfokat.

3. **Fordító** : Az utófeldolgozás során sorosíthatóvá vált példány gráfokat a Cypher nyelv XText nyelven íródott nyelvtanának segítségével szöveges lekérdezésekké alakítom.

4. fejezet

Gráflekérdezések automatikus generálása

4.1. Lekérdezések generálása

4.1.1. Nyelv minimalizásása

4.1.2. Jólformáltsági kényszerek betartása

Ahhoz, hogy a nyelvtan alapján olyan modelleket tudjunk generálni, amelyeket vissza tudunk majd fordítani szintaktikailag helyes lekérdezésekre, ahhoz kényszereket kell felállítanunk, amelyek betartására kötelezzük a modellgenerátort.

Én a következő kényszereket írtam fel: `pattern x(s:State) State(s);`, ahol a `s` változó jelöli a...

```
pattern goodReferenceToVariable(q : SinglePartQuery, match : Match, vari : VariableDeclaration,  variRef : VariableRe
  SinglePartQuery.readingClauses(q ,match);
  Match.^pattern.patterns.^var(match, vari);
  SinglePartQuery.^return.body.returnItems.items.expression(q , variRef);
  VariableRef.variableRef(variRef, vari);
}

@Constraint(severity = "error", key ={variRef}, message ="error")
pattern notGoodReferenceToVari(variRef : VariableRef){
  neg find goodReferenceToVariable(_, _, vari ,variRef );
  VariableRef.variableRef(variRef,vari);
}
```

A fenti két kényszer azt biztosítja, hogy a lekérdezésnek, amit generálunk mindenképpen meglegyen a két fő része. A match részben határozzuk meg azt a bizonyos mintát, amit keresünk, a return részben pedig visszatérünk az erre a mintára illeszkedő gráfrészletekkel.

Ezen belül ki kell kössük szintén, hogy a két fő rész ne csak csonkként vagy helytelenül generálódjon, hanem az egész elengedhetetlen részfa megszülessen.

Ezért van szükség a return ágon az alábbi kényszerekre:

4.1.3. Diverzitás biztosítása

4.2. Automatizálás

5. fejezet

Értékelés

5.1. Benchmark eredmények

5.2. Feature fedés eredmények

5.3. Diverzitás mérése

6. fejezet

Összefoglalás

6.1. Elméleti eredmények

6.2. Gyakorlati eredmények

6.3. Jövőbeli tervek

Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Irodalomjegyzék

[1] 2018 October 16 – 2018 October 15: The neo4j graph platform – the #1 platform for connected data. URL `https://neo4j.com/`.

[2] Sven Efftinge – Miro Spoenemann: Why xtext?
URL `https://www.eclipse.org/Xtext/#feature-overview`.

[3] Graph database | multi-model database. URL `https://orientdb.com/`.

[4] Richard Gronback: Eclipse modeling framework (emf).
URL `https://www.eclipse.org/modeling/emf/`.

[5] High-performance human solutions for extreme data.
URL `http://sparsity-technologies.com/#sparksee`.

[6] József Marton – Gábor Szárnyas – Márton Búr: Model-driven engineering of an opencypher engine: Using graph queries to compile graph queries. In *International SDL Forum* (konferenciaanyag). 2017, Springer, 80–98. p.

[7] Neo4j's graph query language: An introduction to cypher.
URL `https://neo4j.com/developer/cypher-query-language/`.

[8] Scalable reactive model transformations. URL `https://www.eclipse.org/viatra/`.

[9] Oszkár Semeráth – Ágnes Barta – Ákos Horváth – Zoltán Szatmári – Dániel Varró: Formal validation of domain-specific languages with derived features and well-formedness constraints. *Software & Systems Modeling*, 16. évf. (2017) 2. sz., 357–392. p.

[10] Slizaa: slizaa/slizaa-opencypher-xtext, 2018. Aug.
URL `https://github.com/slizaa/slizaa-opencypher-xtext`.

[11] Gábor Szárnyas – Benedek Izsó – István Ráth – Dániel Varró: The train benchmark: cross-technology performance evaluation of continuous model queries. *Software & Systems Modeling*, 17. évf. (2018) 4. sz., 1365–1393. p.

[12] Viatra: viatra/viatra-generator, 2018. Oct.
URL `https://github.com/viatra/VIATRA-Generator`.