VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Transformavimo operatoriai pusiau struktūrizuotų (XML) duomenų bazių užklausų kalbose

Transformation operators in semi-structured (XML) database query languages

Magistro baigiamasis darbas

Atliko:	Jonas Šiuparis	(parašas)
Darbo vadovas:	Lek. Arūnas Janeliūnas	(parašas)
Recenzentas:	Lek. dr. Arūnas Stočkus	(parašas)

Vilnius – 2011

Santrauka

Magistro darbo tema – Transformavimo operatoriai pusiau struktūrizuotų (XML) duomenų bazių užklausų kalbose. Yra nustatoma tyrimui naudojama užklausų kalba – XQuery. Detaliai aprašoma informacija susijusi su užklausų kalba ir informacija, kuri bus naudojama XQuery užklausų kalbos transformavimo operatorių (šiame darbe – funkcijų) kūrime, bei tyrime. Pasinaudojama formaliu modeliu pasiūlytu „Université de Toulon et du Var“ universiteto tyrimo grupės [BMM03]. Yra sukuriamos keturios funkcijos : *transf_insert*, *transf_delete*, *transf_replace*, *transf_move*. Toliau yra sukuriamas XML dokumentas, kuris bus naudojamas tyrime t.y. palyginime tarp sukurtų XQuery funkcijų, bei XSLT transformacijų. Tyrimas yra atliekamas lyginant kiekvieną sukurtą funkciją su XSLT transformacijomis po du kartus. Pirmą kartą yra lyginama transformacija su elementu, o antrą kartą su elementu ir požymiu. Antrame tyrime yra naudojamos automatiškai sugeneruotos XML struktūros, kurios skiriasi medžio ilgiu, gyliu ir galutiniu dokumento dydžiu. Yra lyginamos transformacijos su elementu ir požymiu šiem dokumentam. Galiausiai yra pristatomos išvados ir rezultatai, kuriuose yra pasiūloma kaip ir kada galima būtų naudoti naujai sukurtas funkcijas ir kada galima būtų naudoti XSLT transformacijas.

Raktiniai žodžiai:

- XML
- Užklausų kalba
- XQuery
- XSLT
- Transformavimo operatoriai
- Transformacija
- Transformavimo funkcijos

Summary

This document is about the transformation operators in semi-structured (XML) database query languages. XQuery is chosen as a query language to be used for research. A detailed description of the information related to the query language and information that will be used for development and investigation of XQuery transformation operators (in this work – functions) is provided. The formal model of transformation operators that was created by „Université de Toulon et du Var“ university research group [BMM03] is used in this work. We create four functions (named: *transf_insert*, *transf_delete*, *transf_move*, *transf_replace)* and an XML document. The document is used in the comparison between new XQuery functions and the XSLT transformation. The comparison is executed twice for each of the functions: first, for elements only and then for the same elements along with their attributes. The second part of the research is performed on different XML documents that are automatically generated and their structure differs in tree length, tree depth and resulting size of the document. Finally, conclusions and results are being introduced. It is concluded in which situations which technology should be preferred to use – either XQuery or XSLT transformations.

Key words:

- XML
- Query language
- XQuery
- XSLT
- Transformation operators
- Transformation
- Transformation functions

Turinys

1. Įvadas

Šiuo metu egzistuoja nemažai užklausų kalbų pusiau struktūrizuotiems duomenims, tokių kaip XML-QL, XQuery, LOREL ir kt. Nors yra tam tikras apibrėžimas [El01], ką privalo turėti užklausų kalba, ir daugelis šių kalbų turi panašumų, bet kaip dažniausiai būna, jos turi ir tam tikrų skirtumų. Šio darbo nagrinėjamas objektas – transformavimo operatoriai (transformacijos) – taip pat jų turi. Dažnai pasitaiko, kad tie skirtumai tampa diskusijų objektu, ir vienaip ar kitaip tampa problema, kurią reikia išspręsti, standartizuoti, kad nebeliktų diskusijų.

Dažniausiai susiduriama su šiais diskusijų apie transformavimo operatorius objektais:

1. Nėra ribos užklausoje tarp rezultato suskaičiavimo (kaip SELECT tipo sąlyga SQL kalboje) ir suskaičiuoto rezultato transformavimo (Dokumento struktūros pakeitimo. Plačiau apie tai skyriuje „Transformacija ir Rekonstrukcija“).
2. Dažniausiai tada transformavimas vyksta kaip užklausos sudedamoji dalis.
3. Tam tikro bendro ar „standartinio“ transformavimo operatoriaus neegzistavimas (Pavyzdžiui norime perkelti tam tikrą dokumento struktūrą arba grafo šaką į kitą struktūrą, tačiau neegzistuoja tiesioginė tokia funkcija. Tada reikia ieškoti kitų būdų veiksmui atlikti.)
4. Užklausų kalba XQuery vykdo rezultato suskaičiavimą kaip transformaciją. Taip pat egzistuoja XSLT (XSL Transformacijos skirtos XML dokumentam), kuri yra transformacijų kalba ir naudoja tą patį duomenų modelį kaip ir XQuery. Kokias transformacijas vykdyti XQuery ir kokias XSLT?
5. Neatskyrus užklausoje paieškos dalies nuo rezultatų transformavimo, dažnai, iš pradžių yra transformuojamas visas dokumentas (t.y. taikoma rekonstrukcija; rekonstrukcija naudojama tada, kai yra kuriamas komponentas (dokumentas), panaudojant prieš tai buvusį komponentą, kaip bazę kūrimui), o tik po to jame ieškomi reikalingi duomenys. O visos struktūros rekonstrukcija yra neefektyvi.

Iš šių diskusijų objektų ir atsirado tam tikri uždaviniai, kurie yra sprendžiami darbe.

1.1. Darbo tikslai

Pagrindinis šio darbo tikslas yra: pusiau struktūrizuotų duomenų užklausų kalboje XQuery atskirti duomenų paieškos dalį nuo duomenų – rezultato transformavimo ir pastarajam sukurti universalius kalboje trūkstamus operatorius.

Šiam tikslui pasiekti toliau darbe sprendžiami šie uždaviniai :

1. Detaliai aprašoma kalba XQuery ir jos galimybės.
2. Nustatomi reikalingi (trūkstami) transformacijos operatoriai, kuriuos nagrinėsime.
3. Atliekama teorinė, bei praktinė esamų operatorių analizė.
4. Neegzistuojančių operatorių teorinis apibrėžimas.
5. Neegzistuojančių operatorių realizavimas programavimo kalba, naudojantis sukurtu teoriniu lygmeniu.
6. Sukurtų operatorių bandymams sukuriamos medžio, grafo tipo duomenų struktūros ir paverčiamos į tinkamai suformuotus XML dokumentus.
7. Apskaičiuojamas sukurtų transformacijos operatorių efektyvumas, lyginant užklausų atlikimo laiką, pasinaudojant sukurtomis duomenų struktūromis.

Kadangi XQuery yra aprašyta kaip vienintelė W3C rekomendacija XML užklausų kalboms, todėl ji ir buvo pasirinkta šiam tyrime. Be to ši užklausų kalba jau dabar yra plačiai naudojama.

Pavyzdžiai:

- XQuery yra naudojama MarkLogic nestruktūrizuotų duomenų bazėse. MarkLogic naudoja Springer, Oxford University Press, JAV vyriausybinės organizacijos.
- Data Direct XQuery™ iš DataDirect Technologies. Tai produktas, kuris grąžina suvienodintus duomenis tarp XML ir reliacinių duomenų bazių.
- XQJ – tai XQuery API Java. Ji leidžia programuotojams naudoti XQuery, XML duomenų apdorojimui ir integravimui, su pilnu Java Standard Edition (Java SE) ir Java Enterprise Edition (Java EE) platformų palaikymu. XQJ leidžia Java programai prisijungti prie XML duomenų šaltinių, apdoroti juos XQuery užklausų kalba ir gražinti XML.

XQuery užklausų kalboje trūkstamos transformavimo funkcijos atitinka transformavimo operatorius pasiūlytus „Université de Toulon et du Var“ universiteto tyrimo grupės [BMM03]. Jame siūlomi *insert*, *delete*, *replace-with* ir *move* transformavimo operatoriai. Tad konkretizuoti mokslinio darbo uždaviniai :

1. Realizuoti šių naujų transformavimo funkcijų sukūrimą XQuery kalboje:
   - *local:transf_insert* – įterps nustatytą šaką ar medį į XML dokumentą, nurodytoje vietoje. Vieta yra XPath reiškinys, kuris nurodys kelią iki įterpimo vietos.
   - *local:transf_delete* – ištrins pasirinktą XML dokumento šaką ar medį. Trynimo objektas nurodomas XPath reiškiniu.
   - *local:transf_replace* – pakeis nustatytą XML šaką ar medį nauja šaka.
   - *local:transf_move* – perkels nustatytą XML šaką ar medį į kitą medžio vietą. Kitas medis yra XPath reiškinys, kuris nurodys kelią iki medžio viršūnės.
2. Sukurti XML medį, kurį naudosime transformacijų palyginimuose.
3. Palyginti sukurtas transformavimo funkcijas XQuery kalboje ir transformacijas XSLT kalboje. Pasinaudosime vienodu XML medžiu.

Šie sprendimai galėtų būti pritaikomi visose sistemose, kurių užklausos yra atliekamos XML dokumentam XQuery užklausų kalba. Pavyzdžiui:. XQJ pagalba Java programa prisijungia prie tam tikro XML medžio, o vėliau apdoroja jį sukurtomis transformavimo funkcijomis ir tada grąžina XML medį.

1.2. Darbo struktūros aprašas

1. Skyriuje „2. Aktualumas“ yra aprašomos iškeltos problemos susijusios su XQuery kalba, ir trumpai jos aprašomos.
2. Skyriuje „3. Literatūros apžvalga“ supažindiname su tyrimu susijusiomis sąvokomis:
   - Pusiau stuktūrizuoti duomenys
   - XML duomenys
   - Transformacija ir Rekonstrukcija
   - Užklausų kalba
   - XQuery užklausų kalba ir jos detalus aprašas
3. Skyriuje „4. Siekiami rezultatai“ yra aprašomi siekiami principiniai ir pagalbiniai magistro darbo rezultatai.

4. Skyriuje „5. Tinkamai suformuoto dokumento kūrimas“ yra trumpai aprašoma XML dokumento struktūra.
5. Skyriuje „6. XQuery transformavimo operatorių kūrimas“ yra detaliai aprašoma, kaip bus kuriamos transformavimo funkcijos, pateikiami jų iškvietimo pavyzdžiai.
6. Skyriuje „7. XSLT ir XQuery transformacijų palyginimas“ yra palyginamos analogiškos transformacijos tarp XSLT ir naujų XQuery funkcijų, panaudojant vieną XML dokumentą. Transformacijos vykdomos visoms naujoms XQuery funkcijoms. Taip pat yra palyginamos transformacijos skirtingo dydžio XML dokumentam. Lyginama pagal atlikimo laiką, dokumento ilgį, gylį, bei dydį.
7. Skyriuje „8. Išvados“ pateikiamos išvados apie gautus rezultatus.

2. Darbo aktualumas

Kaip minėta skyriuje „1. Įvadas“ dažnai susiduriama su tam tikrais diskusijų apie transformavimo operatorius objektais. Vieni iš galimų sprendimų jiem buvo pasiūlyti „Université de Toulon et du Var“ universiteto tyrimo grupės moksliniame straipsnyje [BMM03]. Jame pasiūlyta XQuery kalbą praplėsti transformavimo operatoriais. Pagal straipsnio autorius, taip bus atsisakoma visiškos dokumento rekonstrukcijos. Transformavimo operatoriai, kurie siūlomi yra pritaikyti medžiui ir suformuos naują medį, kuriame, kai kurios, šakos su medžiais bus įterptos, pakeistos, ištrintos ir perkeltos. Siūlomi operatoriai: insert, replace-with, delete, move.

Realizavus šiuos semantinius sprendimus, būtų atsisakyta visos dokumento rekonstrukcijos, koncentruojantis tik ties modifikuojamomis medžio dalimis. Taip pat būtų atskirta transformacija nuo rezultato. Šiuo metu sudėtingi reiškiniai yra sunkiai struktūriškai suprantami ir operatoriai (šiame darbe funkcijos), padėtų lengviau juos išreikšti. Tai ypač aktualu užklausų ar duomenų bazių kūrėjams, nes reikalaujant greitų sprendimų iškyla poreikis turėti aiškiai struktūriškai suprantamą užklausą, kad būtų galima greitai suprasti ir įvykdyti užduotis.

Kitas diskusijų objektas – ryšys su XSLT. XSLT yra transformacijų kalba, o XQuery užklausų kalba, tad pasirinkimas, kurią naudoti vykdyti transformacijoms turėtų būti aiškus, tačiau XSLT atliekant sudėtingas transformacijas taip pat yra sunkiai suprantama. Tad sukurtos XQuery transformavimo funkcijos leistų lengviau išreikšti sudėtingas transformacijas.

3. Literatūros apžvalga

Šioje dalyje apžvelgsime aktualią informaciją ir literatūrą, kuri yra panaudota arba susijusi su mokslo tiriamuoju darbu. Apžvalgą sudarys :

- Pusiau struktūrizuoti duomenys
- XML duomenų struktūra
- Duomenų transformacija ir rekonstrukcija
- Užklausų kalbos

3.1. Pusiau struktūrizuoti duomenys

Praeito dešimtmečio pabaigoje atsirado apibrėžimas, kuris apibūdina pusiau struktūrizuotus duomenis. Pusiau struktūrizuoti duomenys yra duomenys, kurie neturi reguliarios ir statinės (nekintamos) struktūros, tokios, kokios egzistuoja reliacinėse (sąryšinėse) duomenų bazėse, bet kurių schema yra dinaminė. Kartais yra vadinama be-scheminiu (schemaless) duomenų modeliu [El01]. Šiuose duomenyse informacija, kuri yra dažniausiai laikoma schemoje, yra laikoma pačiuose duomenyse. Kai kuriuose šaltiniuose jie dar įvardijami, kaip patys save apibūdinančiais duomenimis (self-describing) [B97]. Dažniausiai jie sudaro gairių ar kitų žymių rinkinį, tam, kad atskirti semantinius elementus, įrašų hierarchija ir laukus su elementais.

Šie duomenys tapo svarbia tyrinėjimų tema dėl:

1. Yra duomenų šaltinių, tokių kaip internetas, kuris negali būti apribotas kažkokia vienalype schema.
2. Labai norima turėti ypatingai lankstų formatą duomenų apsikeitimam (data exchange) tarp nesugretinamų duomenų bazių.
3. Dėl naršymo (duomenų žiūrėjimo) tikslų[1].

[1] Vartotojas negali parašyti užklausos nežinodamas duomenų bazės schemos. Be to, schemos gali turėti neaiškią terminologiją ir neaiškią, logiškai nesuprantamą konstrukciją. Tad būtų labai naudinga rašyti užklausas be visos schemos žinojimo. Pavyzdžiui: Kur duomenų bazėje yra eilutė su „XString“ įrašu? Ar yra duomenų bazėje sveikųjų skaičių didesnių už X? Kokie objektai duomenų bazėje turi atributą, kurio pavadinimas prasideda „abc“?

Tokie klausimai negali būti atsakyti standartinėmis sąryšinėmis ir objektų orientuotomis užklausų kalbomis.

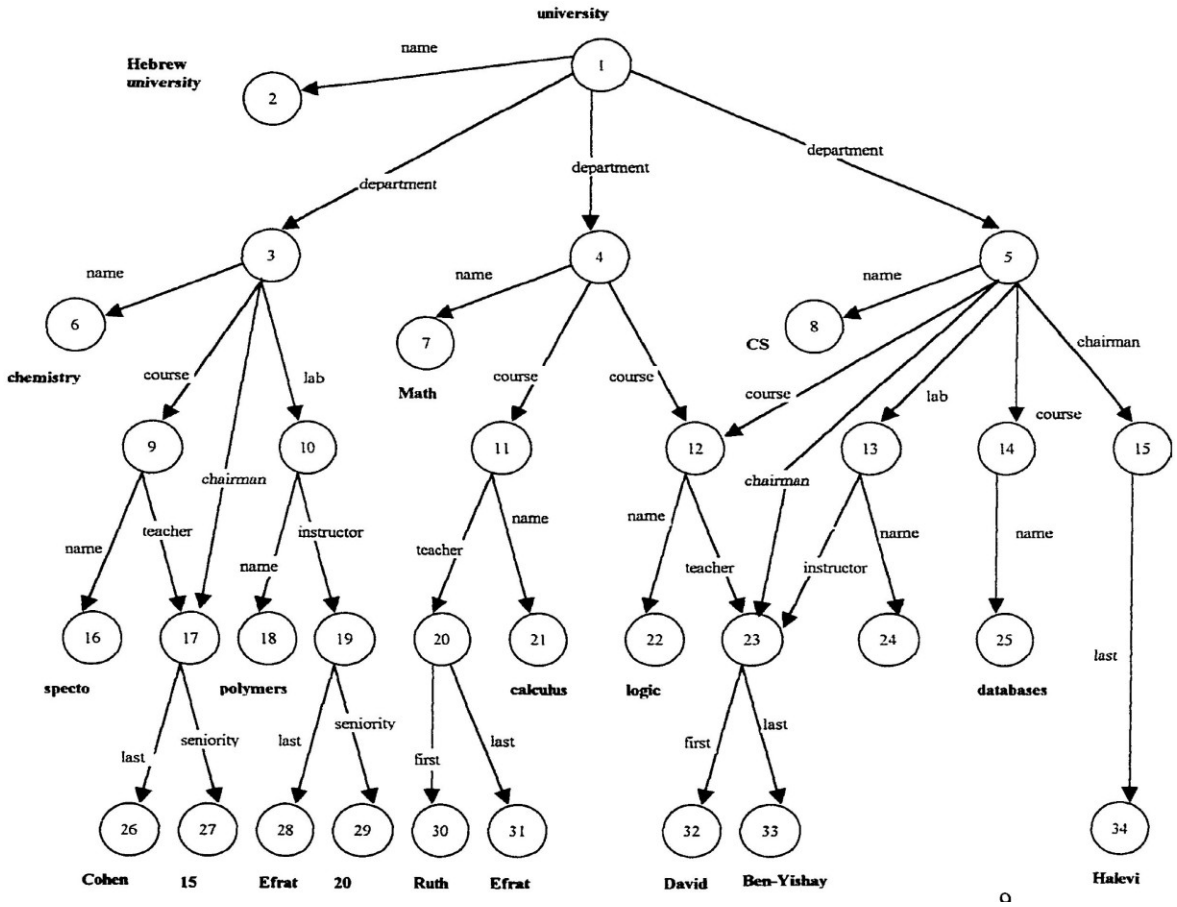
Pavyzdys pusiau struktūrizuotų duomenų panaudojimui : Jei tiriamas objektas būtų orų prognozė internete, tai egzistuojančiuose puslapiuose ar duomenų bazėse egzistuojančių „laukų“ kiekis skirtųsi. Netgi tame pačiame puslapyje prognozės „laukai“ gali skirtis nuo pasirinktos vietovės (pvz. Kalnuotose vietovėse gali būti informacija apie sniego dangos, ledo storį, o tropiniuose tokios informacijos nebus). Tačiau pusiau strukūrizuoti duomenys, neturėtų būti tapatinami tik su internetu. Jie yra randami ir failų sistemose, elektroninėse pašto sistemose ir kt. vietose.

Dažniausiai pusiau struktūrizuoti duomenys naudojami tada, kai atsiranda poreikis kelioms, ar daugiau, heterogeninėm schemoms (Pvz. Keliose heterogeniniuose šaltiniuose yra neatitikimai tarp įvairių duomenų: kai kuri informacija neegzistuoja viename šaltinyje; atributas gali turėti vieną reikšmę viename šaltinyje ir kitą kitame; arba kai kurios esybės gali būti atvaizduotos skirtingų tipų, skirtinguose šaltiniuose). Tuose atvejuose pusiau struktūrizuoti duomenys atsiranda todėl, kad integruojami objektai dažnai remiasi dinamine (kartais ir prieštaringa) informacija iš sudėtinių šaltinių, verčiančių integratorių išfiltruoti, sujungti, ar praleisti tam tikrus „laukus“, kai vykdomas integravimas.

Pagrindiniai privalumai naudojant tokį duomenų modelį (schemaless), kur duomenys ir „metaduomenys“ yra sumaišyti, yra duomenų heterogeniškumas ir duomenų integravimas, kas yra bendri bruožai duomenų naudojamų internete.

Bendra idėja atvaizduoti pusiau struktūrizuotus duomenis yra grafo arba medžio pavidalo struktūra. Net jei bus leidžiami ciklai duomenyse, tai paprastai tokius grafus vadinsime medžiais. Grafiškai duomenys yra atvaizduojami, kaip viršūnės grafe, kurios sujungiamos briaunomis (šakomis).

Pirmame paveiksle panaudota struktūra iš Hebrajų Universiteto duomenų bazės (1 Pav.)



1 Pav. Hebrajų Universiteto duomenų bazės struktūra

3.2. XML

W3C XML kalbą rekomenduoja kaip bendros paskirties duomenų struktūrų, bei jų turinio aprašomąją kalbą. Jos paskirtis užtikrinti lengvesnį duomenų keitimąsi tarp skirtingo tipo sistemų, dažniausiai sujungtų internetu. Pagrindinis XML kalbos vienetas yra elementas. Elementas visada turi vardą ir, be jo, gali turėti [W3C]:

- norimą skaičių atributų. Atributas turi savo vardą bei reikšmę.
- kitus (dukterinius) šio elemento viduje esančius elementus.
- su elementu susijusį tekstą.

Ši kalba dėl savo paprastumo ir išplitimo tapo viena iš populiariausių šiuolaikinių duomenų keitimosi technologijų. XML suteikia tam tikrų privalumų prieš reliacines duomenų bazių sistemas.

Duomenų formato perskaitymo paprastumas leidžia lengvai naudoti ją. Ji yra nepririšta prie jokios „taikomosios programos“, sistemos. Jos struktūra yra lanksti, tad ja galima vykdyti įvairaus lygio reikalavimus duomenų laikymui ar konfigūracijai. Pavyzdys XML dokumento:

```
<bib>
   <book year="1995">
       <title>  Database Systems </title>
      <author> <lastname> Date </lastname> </author>
      <publisher> Addison-Wesley  </publisher>
   </book>
   <book year="1998">
      <title> Foundation for Object/Relational Databases </title>
      <author> <lastname> Date </lastname> </author>
      <author> <lastname> Darwen </lastname> </author>
      <ISBN> <number> 01-23-456 </number >  </ISBN>
   </book>
 </bib>
```

3.3. Transformacija ir Rekonstrukcija

Rekonstrukcija naudojama tada, kai yra kuriamas komponentas (dokumentas), panaudojant prieš tai buvusį komponentą, kaip bazę kūrimui. Tai reiškia, kad bus rekonstruotas dokumentas su visais elementais. Didžiausias nepatogumas iškyla tada, kai norint rekonstruoti dokumentą, reikia rekonstruoti taip pat visus jo elementus, nors dažnai užtektų rekonstruoti vieno ar kelių. Šioj situacijoj gelbėja transformacija, kuri padeda išvengti rekonstrukcijos tų elementų, kurie išlieka nepakitę. Tam reikalingi transformacijos operatoriai pavyzdžiui: *pakeisti* ar *ištrinti* ar kt. Todėl, bet kokiai transformacijai vartotojas turi turėti žinias apie pirminę struktūrą ir aiškiai apibrėžti galutinio dokumento struktūrą. Ne visos užklausų kalbos turi tokius transformacijos operatorius.

3.4. Užklausų kalba

Bendru atveju užklausų kalba yra kalba, skirta vykdyti užklausas duomenų bazėse ir informacinėse sistemose.

Yra nemažai pusiau struktūrizuotų duomenų užklausų kalbų tokių kaip LOREL [AQMWW97], XML-QL [JMZ00] ir kt. Yra panašumų tarp sintaksės ir semantikos, bet kiekvienas prisideda vis prie kito aspekto pusiau struktūrizuotų užklausų kalbų kūrime. Jie visi naudoja panašią sintaksę į SQL (select-from-where) bet jų semantika šiek tiek skirasi [El01].

Vienas iš pagrindinių bruožų pusiau struktūrizuotų užklausų kalbų yra gebėjimas pasiekti sutartą gylį duomenų grafe. Tai atlikti užklausų kalbai reikia panaudoti, tam tikra forma, kelio reiškinio sąvoką (reiškinį) (path expression). David Maier [Mai86] apibrėžė bruožus panašius visoms pusiau struktūrizuotų duomenų užklausų kalbom:

- Reguliarus kelio reiškinys (path expression) – Pusiau struktūrizuotų duomenų modelis yra atvaizduotas kaip grafas, kurio viršūnės yra duomenys, kuriuos galima užklausti. Be to užklausų kalba turi galėti gauti informaciją patalpintą, bet kuriame duomenų bazės grafo lygyje. Reguliarūs reiškiniai su detalizuotais simbolių (wildcard) bruožais yra svarbūs komponentai bet kurios užklausų kalbos, kuri susiduria su duomenų bazės grafo atvaizdavimu. Pavyzdžiui :. Užklausos rašytojas yra neužtikrintas, kokia yra telefono numerio žymė (bet žino, kad tai arba numeris arba telefonas) ir nežino tikslaus kelio iki darbuotojo miesto. Sekanti užklausa yra pavyzdys (priklauso nuo užklausų kalbos), kaip gražinti telefono numerį visiems darbuotojams, kurie turi „Vilnius“ elementą. Užklausoje simbolis „*“ reiškia bet kokią seką 0 ir daugiau elementų.

```
SELECT <newemp> e.(phone|number) </newemp>
FROM employees.emp e
WHERE e.* = "Vilnius"
```

- Reikšminga galia (expressive power) – Kitas svarbus užklausų kalbos bruožas yra gebėjimas palaikyti įvairias SQL ar RA (Reliacinės Algebros) operacijas. Kadangi dauguma duomenų bazių šiais laikais yra paremtos reliaciniu modeliu ir šios duomenų bazės gali būti atvaizduotos, kaip pusiau struktūrizuotų duomenų grafas, tai pusiau struktūrizuotų užklausų kalbų sėkmė priklausys ir nuo sugebėjimo vykdyti įprastas SQL operacijas, ypač, JOIN operacijas.

- Tiksli semantika – Kai kiekvienos užklausų kalbos operacijos prasmė jau yra apibrėžta, tada atsiranda kitas svarbus aspektas – Transformacija ir Optimizavimas. Užklausų kalba turi

leisti vartotojui išreikšti tą pačią užklausą skirtingais būdais. Toks ekvivalentumas yra svarbus bruožas efektyviai duomenų bazių valdymo sistemai, kuri atlieka protingas optimizacijas.

- Kompozicija – Reliacinėje duomenų bazėje užklausos išeitis taip pat yra duomenų bazė, kuriai galima taip pat vykdyti užklausą. Būtų patogu ir pageidautina, kad šis bruožas būtų išlaikytas ir pusiau struktūrizuotų duomenų užklausų kalboje. Kai išeitis iš pusiau struktūrizuotų duomenų užklausų kalbos gali būti panaudota kitoje užklausoje, tai tada galima bus atlikti sudėtingesnes užklausų kalbos operacijas.

- Prasminga struktūra - Pusiau struktūrizuota duomenų užklausų kalba turi galėti išnaudoti duomenų bazės schemą. Kaip pavadinimas sako, pusiau struktūrizuoti duomenys yra dalinai struktūrizuoti. Kadangi visiškai arba teisingai apibrėžta schema neegzistuoja pusiau struktūrizuotuose duomenyse, užklausų kalba tūrėtų atsižvelgti į bendrą duomenų struktūrą ir galėti panaudoti šią informaciją optimizuojant savo paieškos mechanizmus

- Programos valdymas – bet kokia pusiau struktūrizuotų duomenų užklausų kalba turi turėti paprastą ir aiškią, „daugiažodę“ kalbą. Užklausų kalba, kuri palaiko daug operacijų, leidžia geresnį užklausos optimizavimą.

3.4.1. XQuery užklausų kalba

XQuery yra suprojektuota taip, kad atitiktų reikalavimus aprašytus W3C XML Query darbo grupės [CFMR05]. Ji sukurta kaip nedidelė, lengvai įvykdoma kalba, kurioje užklausos yra trumpos ir lengvai suprantamos. XQuery yra išvesta iš Quilt XML užklausų kalbos, kuri taip pat buvo sukurta, remiantis kitomis užklausų kalbomis. Iš XPath ir XQL kalbų buvo išvesta XQuery kelio reiškinių sintaksė, tinkama hierarchiniams dokumentams. Taip pat tam tikri kiti funkcionalumai buvo išvesti iš kitų kalbų, tokių kaip : XML-QL, OQL, Lorel, YATL ir SQL. Kaip ir OQL XQuery yra funkcinė kalba, kurioje užklausa yra atvaizduojama kaip reiškinys. XQuery palaiko kelius tipus reiškinių, struktūrų ir užklausos išvaizda gali labai skirtis priklausomai kokio tipo reiškinį

naudosime. Įvairios XQuery reiškinių formos gali būti įdėtinės (nested) su pilnu funkcionalumu. Duomenų modelis sugeba modeliuoti XML dokumentus, ir gerai suformuotus dokumento fragmentus, eilę dokumentų arba eilę dokumentų fragmentų. Pavyzdys duomenų modelio yra surūšiuota viršūnių eilė, kuri kiekviena gali turėti įdėtines viršūnių eiles. Nėra skirtumo tarp vienos viršūnės ir eilės viršūnių, kurios ilgis yra vienas. Taip pat, eilės yra visada surūšiuotos. Pagrindinės XQuery reiškinių formos:

- Kelio reiškiniai
- Elemento konstruktoriai
- FLWR (For-Let-Where-Return) reiškiniai
- Reiškiniai apimantys operatorius ir funkcijas
- Sąlygos reiškiniai
- Kiekybiniai reiškiniai
- Reiškiniai, kurie testuoja ar keičia duomenų tipus

XQuery užklausos vykdomos XML dokumentui, o gaunamas rezultatas yra XML dokumento fragmentas. Šį fragmentą galima toliau naudoti užklausose. Toks bruožas yra vadinamas kompozicija. Pagrindinė XQuery standarto dalis yra XPath kelio kalba arba kaip dažniausiai vartojama – XPath kelio reiškinys, kuris yra W3C standartas. XPath reiškiniai nurodo kelią nuo dokumento viršūnės iki nurodyto elemento. XQuery reiškiniai gali suteikti papildomus apribojimus viršūnėm, surastiem XPath reiškiniams.

Kai kurie vartotojai XQuery naudoja XML dokumentų manipuliavimui, pavyzdžiui, transformuojant žinutes, šiuo atveju XML dokumentus, tarp sistemų. Tais atvejais XQuery konkuruoja su XSLT.

3.4.2. XQuery užklausos ir sintaksė

Keletas pagrindinių sintaksės taisyklių:

- XQuery skiria didžiąsias ir mažąsias raides (*case-sensitive*)
- XQuery elementai, atributai ir kintamieji turi būti galiojantys XML vardai
- XQuery eilutės reikšmė gali būti viengubose arba dvigubose kabutėse

- XQuery kintamieji yra apibrėžti simboliu „$“ prieš vardą pvz:. $universitetas
- XQuery komentarai yra apibrėžti „:“ simboliais. Pvz:. :Universitetas:
- XQuery taip pat palaiko sąlygos reiškinius: *if-then-else*

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="CHILDREN")
  then <child>{data($x/title)}</child>
  else <adult>{data($x/title)}</adult>
```

- XQuery palaiko lyginimo operatorius : =, !=, <, <=, >, >=

```
$bookstore//book/@q > 10
```

- XQuery palaiko lyginimo operatorius : eq, ne, lt, le, gt, ge

```
$bookstore//book/@q gt 10
```

Užklausų tipai:

- Paprasčiausia XQuery užklausos forma gali būti XPath reiškinys

```
document("FileSystem.xml")/filesystem/drive[@letter="C"]//file
```

- Pagrindinis XQuery „variklis“ yra FLWR (**F**or-**L**et-**W**here-**R**eturn) reiškiniai

```
for $myNode in document("FileSystem.xml")//folder
where $myNode/@name="Quake"
return $myNode
```

- FLWR yra SQL SELECT tipo užklausos. Tai šiame kontekste FLWR reiškia:
    - For – Sąlyga nurodo kartojimą XML viršūnių sąrašo ir yra atitikmuo FROM sąlygos SQL kalboje. XML viršūnės yra nurodomos per XPath reiškinį. Pavyzdžiui, jei mes

norime pereiti visus <folder> elementus, mes naudosi XPath reiškinio dokumentą („FileSystem.xml")//folder

- Where – Sąlyga sudaro reiškinį, kuris gražina loginę reikšmę, taip kaip ir WHERE sąlyga SQL kalboje. Kiekviena XML viršūnė XML viršūnių sąraše for sąlygoje yra įvertinama where sąlygos reiškinio. Tie, kurie atitinka loginę reiškę TRUE, yra grąžinami, kurie ne – praleidžiami
- Return – Return sąlyga grąžina gautą FLRW reiškinio turinį ir yra SELECT sąlygos atitikmuo SQL kalboje.

```
for $myNode in document("FileSystem.xml")//folder
where $myNode/@name="Quake"
return $myNode
```

- Įdėtinės užklausos

XQuery leidžia įdėtines užklausas, kur rezultatas vienos užklausos gali būti siunčiamas į kitą. Reikia turėti omenyje, kad XPath reiškiniai taip pat gali būti užklausa.

Žemiau esančiame pavyzdyje FOR sąlyga suranda visas unikalias *word* viršūnes su atributu *label*, tada RETURN sąlyga konstruoja *word* elementus su ta pačia atributo *label* reikšme. Šių sukonstruotų *word* elementų vaikai bus rezultatas kito XPath reiškinio, kuris randa *phoneme* elementus, kurie yra vaikai *word* elementų, surastų prieš tai buvusio reiškinio.

```
FOR $wl IN
distinct-values(//word/@label)
RETURN
<word label={$wl}>
$wl/parent::word/phoneme
</word>
```

3.4.3. XQuery reguliarūs kelio reiškiniai

Kaip minėjome anksčiau XQuery naudoja standartizuotą W3C XPath kelio reiškinių modelį [Cl99b] . XPath yra septynių tipų viršūnės: elementai, atributai, tekstai, vardų (namespace),

komentarų, vykdomų instrukcijų, ir dokumentų viršūnės. Su XML dokumentais elgiamasi kaip su viršūnių medžiu. Pirmasis (aukščiausias) elementas medyje yra šakninis elementas.
Dažniausiai naudojami kelio reiškiniai :

Reiškinys	**Aprašymas**
viršūnėsvardas	Išrenka visus pasirinktos viršūnės vaikus
/	Renka nuo šakninės viršūnės
//	Išrenka viršūnes dokumente nuo dabartinės viršūnės, kuri atitinka pasirinkimą.
.	Išrenka dabartinę viršūnę
..	Išrenka tėvą dabartinės viršūnės
@	Išrenka atributus

Pavyzdžiai kelio reiškinių:

Kelio reiškinys	**Rezultatas**
Viršūnė1	Išrenka visus vaikus viršūnė1 elemento.
/Viršūnė1	Išrenka šakninę viršūnę
Viršūnė0/Viršūnė1	Išrenka visus viršūnė1 elementus, kurie yra vaikai viršūnė0
//Viršūnė3	Išrenka visus viršūnė3 elementus nepaisant, kur jie yra dokumente.
Viršūnė1//Viršūnė3	Išrenka visus viršūnė3 elementus, kurie yra palikuonys viršūnė1, nepaisant to, kur jie yra po viršūnė1elemento.
//@viršūnėnode	Išrenka visus atributus kurie vadinasi viršūnėnode

Panaudojame skyriuje „Priedai“ esančią *catalog.xml* struktūrą ir pritaikome keletą XPath reiškinių:

- //artist arba catalog/cd/artist

```
<artist>U2</artist>
<artist>Led Zeppelin</artist>
<artist>Rush</artist>
<artist>Billy Joel</artist>
<artist>Led Zeppelin</artist>
<artist>Jimi Hendrix</artist>
```

Predikatai yra naudojami surasti tam tikrą viršūnę arba viršūnę, kuri turi tam tikrą reikšmę. Predikatai yra visada apskliausti laužtiniais skliaustais „[]“.

Kelio reiškinys	**Rezultatas**
/Viršūnė/Viršūnė5[1]	Išrenka pirmą viršūnė5 elementą, kuris yra vaikas viršūnė elemento.
/Viršūnė/Viršūnė5[last()]	Išrenka paskutinį viršūnė5 elementą, kuris yra vaikas viršūnė elemento
/Viršūnė/Viršūnė5[position()<3]	Išrenka pirmus du viršūnė5 elementus, kurie yra vaikai viršūnė elemento
//viršūnė3[@viršūnėnode]	Išrenka visus viršūnė3 elementus, kurie turi atributą viršūnėnode
//viršūnė3[@viršūnėnode=‘3‘]	Išrenka visus viršūnė3 elementus, kurių atributo viršūnėnode reikšmė lygi 3.
/viršūnė3/viršūnė3a[viršūnėnode >3]/viršūnė4	Išrenka visus viršūnė4 elementus iš viršūnė3a elemento, kuris yra vaikas viršūnė3 elemento, ir kurio viršūnėnode reikšmė didesnė už 3.

Kaip ir ankstesnėse užklausų kalbose XQuery Xpath kelio reiškiniai turi pakaitos simbolius:

- * - atitinka, bet kokią elemento viršūnę.
- @* - atitinka, bet kokią atributo viršūnę.
- Node() – atitinka, bet kokią viršūnę, bet kokio tipo
- | - atitinka AND loginę operaciją kelio reiškiniui t.y. galima sujungti du kelio reiškinius į vieną.
- Taip pat galima į kelio reiškinį įterpti paprastus XPath operatorius : +,-,= ir t.t.

Panaudojame skyriuje „Priedai“ esančią *catalog.xml* struktūrą ir pritaikome keletą XPath reiškinių:

- //cd[@=“602498678299“]

```
<cd upc="602498678299">
     <artist>U2</artist>
     <title>How to Dismantle an Atomic Bomb</title>
     <price>13.98</price>
```

```
    <label>Interscope Records</label>
    <date>2004-11-23</date>
</cd>
```

- //artist[.="Led Zeppelin"]

```
<artist>Led Zeppelin</artist>
```

3.4.4. XQuery transformacija

Kaip ir XML-QL XQuery palaiko transformaciją kaip dalį užklausų kalbos. Transformacija yra atliekama, priskiriant kintamuosius WHERE sąlygoje ir naudojant šiuos kintamuosius FOR sąlygoje, sukuriant naują medį. Taip pat galimas kūrimas panaudojant įdėtines užklausas.

Tačiau čia vyksta daug diskusijų ar tai turėtų būti vadinama transformacija, ir taip yra dėl kelių priežasčių:

- XQuery palaiko XSLT (naudoja tą patį duomenų modulį). XSLT - tai kalba, aprašanti XML dokumento transformaciją į kitokios struktūros XML dokumentą. XSLT naudoja XML sintaksę. Tad diskusijos kyla, dėl to, kad XQuery transformacijas atlieka kaip užklausa ir yra atveju, kai ji negali atlikti transformacijos, o XSLT gali. Tad kyla klausimas, kodėl XSLT nenaudoti tik transformacijom, o XQuery – tik užklausom.
- XQuery W3C standarte įvardinta, kaip vykdanti transformacijas.

4. Darbe siekiami rezultatai

Principinis šio magistro darbo siekiamas rezultatas yra:

XQuery transformavimo operatorių sukūrimas. Sukurti transformavimo operatoriai padeda lengviau suprasti užklausas, kadangi jiems yra sukurti operatoriai (*transform*), atlikti transformacijas nenaudojant dokumento rekonstrukcijos.
Kuriami šie keturi operatoriai:

- *local:transf_insert* – įterps nustatytą šaką ar medį į XML dokumentą, nurodytoje vietoje. Vieta yra XPath reiškinys, kuris nurodys kelią iki įterpimo vietos.
- *local:transf_delete* – ištrins pasirinktą XML dokumento šaką ar medį. Trynimo objektas nurodomas XPath reiškiniu.
- *local:transf_replace* – pakeis nustatytą XML šaką ar medį nauja šaka.
- *local:transf_move* – perkels nustatytą XML šaką ar medį į kitą medžio vietą. Kitas medis yra XPath reiškinys, kuris nurodys kelią iki medžio viršūnės.

Sukūrus operatorius, siekiami šie pagalbiniai magistro darbo rezultatai:

- XQuery ir XSLT palyginimui kuriamas tinkamai suformuotas XML dokumentas – Šis medžio dokumentas yra sukuriamas pagal formalų modelį ir yra naudojamas kitose dvejose užduotyse. Naudojama <*catalog*> medžio formos duomenų struktūra.
- XQuery ir XSLT palyginimas. Sukurti transformavimo operatoriai XQuery kalboje yra palyginami su XSLT transformacijom. Lyginami veiksmai : perkėlimas, įterpimas, ištrynimas, išėmimas. Gaunamas rezultatas yra palyginimas, kada ir kokiuose veiksmuose, kuri užklausų kalba veikia greičiau.

5. Tinkamai suformuoto dokumento kūrimas

Formalus modelis, pagal kurį kursime XML dokumentą, kuris pagal W3C standartus [XML98] yra vadinamas kaip tinkamai suformuotas XML dokumentas. Jis turi atitikti šiuos reikalavimus:

- Jis turi apimti visas XML savybes : atributus, elementus, tekstinius duomenis, tvarkingą seką ir kt.
- Jis turi būti su baigtinio aukščio struktūra
- Jis turi būti išreikštas tam tikru schemos formalizmu pvz:. DTD ar XML Schema
- Jis turi leisti naudoti XPath kelio reiškinius.

Pirmame tyrime „Funkcijų funkcionalumo tyrimas“, mes naudosime tinkamai suformuoto dokumento struktūrą, kad galėtume gauti tvarkingus rezultatus, kad galėtumėme panaudoti XPath reiškinį su elemento požymiais (savybėmis) : atributus, tekstas.

6. XQuery transformavimo operatorių kūrimas

Transformavimo operatoriai turi atitikti standartines transformacijas. Standartinės siekiamos transformacijos yra :

- Trynimas (*local:transf_delete*) – Medžio ar jos šakos ištrynimas XML dokumente ar jo šakoje.
- Pakeitimas (*local:transf_replace*) – Medžio ar jos šakos pakeitimas kitu XML dokumentu ar jo šaka.
- Perkėlimas (*local:transf_move*) – Medžio ar jos šakos perkėlimas iš vienos XML dokumento vietos į kitą XML dokumento vietą.
- Įterpimas (*local:transf_insert*) – Medžio ar jos šakos įterpimas į nustatytą XML dokumento vietą.

Transformacijos yra pasiekiamos sukuriant transformavimo operatorius. Toliau aprašysime transformavimo operatorius ir kokias problemas, susijusias su jais, nagrinėsime ir jų realizavimą.

Iškeltos problemos:

1. Ar sukurti transformacijos operatoriai atlieka veiksmus efektyviau už XSLT (XSL Transformacijos skirtos XML dokumentam) transformacijas.
2. Supaprastinti medžio transformavimo reiškinius. Šiuo metu atliekant sudėtingą reiškinį pvz:. Reiškinys susidedantis iš n aibės reiškinių (kur, tarkim n = 6), yra sunkiai suprantamas. Pasinaudojant transformavimo operatoriais jis tampa suprantamesnis, kadangi reiškinys prasidėtų operatoriumi ir būtų aišku, koks veiksmas planuojamas pvz :. `Insert(),Replace()`

Pasiūlyti sprendimai:

1. Sukurti specialius transformavimo operatorius. Transformavimo operatoriai yra pritaikyti medžiui ir sukurs naują medį, kuriame šakos bus įterptos, perkeltos, pakeistos ar ištrintos. Kadangi jie konstruoja naujus medžius, tai transformacijos paklus tom pačiom taisyklėms kaip ir konstruktoriai t.y. Konstruktorius sukuria naują mazgą : kiekvienam reiškinio grąžintam mazgui, nauja mazgo kopija yra konstruojama ir kiekviena mazgo kopija turi naują mazgo struktūrą. Planuojami sukurti transformavimo operatoriai:
    - *Insert*
    - *Replace-with*
    - *Delete*
    - *Move*
2. Pasinaudojus tinkamai suformuotu XML dokumentu palyginti, XQuery su transformacijos operatoriais ir XSLT transformacijų, pasirinktų užduočių įvykdymo laiką. Pasinaudosime straipsniais [Le01] [Mc08], kuriuose lyginamos XQuery ir XSLT kalbos. Bei XSLT visų transformacijų aprašu [Cl99a].
3. Parodysime, jog naujų operatorių naudojimas leidžia supaprastinti transformavimo reiškinius.

6.1. *Insert* operatoriaus kūrimas

6.1.1. Formalus „*insert*“ aprašymas

Apibrėžiamas operatoriaus veiksmas: Šis operatorius padaro kopiją medžio, į kurį įterpsime naują medį. Įterpimo vieta yra aprašoma kelio nuoroda. Kelias šiuo atveju bus XPath, o kelią

sudarysime pagal aprašytas taisykles [Cl99b] [CSD00]. Medis (šiuo atveju jau XML dokumentas) yra sudaromas pagal taisykles aprašytas „5. Tinkamai suformuoto XML dokumento kūrimas“ skyriuje. Pasinaudosime „Université de Toulon et du Var“ universiteto tyrimo grupės [BMM03] pasiūlytu sprendimu t.y kaip turėtų atrodyti *insert* operatorius ir kokius operatorius jis pats turės:

```
insert q' (into | preceding | following) p in q
```

Taigi matome kaip ji turi atrodyti. Taip pat žinome tai, kad XQuery užklausų kalboje yra galimybė kurti funkcijas. Tad reiks sukurti šį sprendimą taip, kad būtų įmanoma įvykdyti su XQuery funkcijomis. XQuery funkcijos aprašas:

```
declare function prefix:function_name($parameter AS datatype)
AS returnDatatype
{
 ...funkcijos kodas...
}
```

6.1.2. Realizacija „*insert*“ funkcijos

Tad šiuo atveju galime priimti sprendimą, kaip realizuosime *insert* operatorių. Kad galėtumėme realizuoti tokią dviejų iš eilės einančių funkcijų struktūrą kaip ***insert .. in ..***, įvedame *transf_insert* funkciją, kurios parametrai būtų operatoriai, kelias, ir užklausos.

Realizuotas sprendimas formalaus modelio panaudojant XQuery funkciją (Visos papildomos funkcijos yra skyriuje „Priedai“, „1. *Insert* operatoriaus funkcijos kodas“ dalyje):

```
declare function local:transf_insert( $dokumentas as xs:string,
                                      $aplinkybes as xs:string,
                                      $naujasmedis as node()*,
                                      $kelias as xs:string )
      as item()*
 {
      let $source:=doc($dokumentas)
      return local:copy( $source/node(),
```

```
                        $aplinkybes,
                        $naujasmedis,
                        local:dynamic-path($source,$kelias ) )
};
```

Funkcijos parametrai:

- `$dokumentas` atitinka dokumentą ar medį, kuriam vykdysime įterpimą
- `$naujasmedis` atitinka medį, kurį norime įterpti.
- `$aplinkybes` atitinka kaip bus įterpiamas `$naujasmedis` t.y. viena iš šių reikšmių: `'into', 'following', 'preceding'`.

    - *into*.

    Pavyzdys:

    Funkcijos iškvietimas „catalog.xml“ dokumentui (Dokumento struktūra pateikta skyriuje „Priedai“: „5. „Catalog.xml“ XML dokumento kodas.“), kuriame į visus elementus, kurie turi *upc* atributą, įterpiame naują elementą <*element*>

    ```
    local:transf_insert("catalog.xml",'into',<elemen><g
    r="ssss"></g></elemen>,"catalog/cd[@upc]")
    ```

    Apibrėžimas – ši aplinkybė reiškia, kad užklausos pagalba galima įterpti atributą, elementą ar kt. savybę į medžio, ar jo šakos tam tikrą vietą.

    - *following*

    Pavyzdys:

    Funkcijos iškvietimas „catalog.xml“ dokumentui (Dokumento kodas pateiktas skyriuje „Priedai“: „5. „Catalog.xml“ XML dokumento kodas.“), kuriame po visų elementų, kurie turi *upc* atributą, įterpiame naują elementą <*element*> :

```
local:transf_insert("catalog.xml",'following',<elemen><g
r="ssss"></g></elemen>,"catalog/cd[@upc]")
```

Apibrėžimas – ši aplinkybė reiškia, kad užklausos pagalba galima įterpti elementą po tam tikros medžio šakos.

- *preceding*

Pavyzdys:

Funkcijos iškvietimas „catalog.xml“ dokumentui (Dokumento kodas pateiktas skyriuje „Priedai“: „5. „Catalog.xml“ XML dokumento kodas.“), kuriame prieš visus elementus, kurie turi *upc* atributą, įterpiame naują elementą <*element*> :

```
local:transf_insert("catalog.xml",'preceding',<elemen><g
r="ssss"></g></elemen>,"catalog/cd[@upc]")
```

Apibrėžimas – ši aplinkybė reiškia, kad užklausos pagalba galima įterpti elementą prieš tam tikrą medžio šaką.

- `$kelias` reiškia kelią iki šakos, kurioje vykdysime įterpimą. Kelias yra XPath reiškinys.

6.2. *Delete* operatoriaus kūrimas

6.2.1. Formalus „*delete*“ aprašymas

Apibrėžiamas operatoriaus veiksmas: Šis operatorius padaro kopiją medžio, kuriame tam tikros šakos yra jau ištrintos.

Aprašome *delete* operatorių ir kokius operatorius jis pats turės (aprašas iš [BMM03] tyrimo medžiagos):

```
delete p in q
```

Siūlomas sprendimas, realizuojant XQuery funkcija:

```
declare function prefix:transf_delete(kelias AS datatype, u1
AS datatype)
AS returnDatatype
{
 ...funkcijos kodas...
}
```

6.2.2. Realizacija „*delete*“ funkcijos

Realizuotas sprendimas formalaus modelio panaudojant XQuery funkciją (Visos papildomos funkcijos yra skyriuje „Priedai“, „2. *Delete* operatoriaus funkcijos kodas“ dalyje):

```
declare function local: transf_delete( $dokumentas as xs:string,
                                       $kelias as xs:string )
      as item()*
 {
      let $source:=doc($dokumentas)
      return local:copy( $source/node(),
                         local:dynamic-path($source,$kelias ) )
 };
```

Funkcijos parametrai:

- `$kelias` reiškia kelią nuo šakninio elemento medyje iki elemento, kurį mes norime ištrinti. XPath reiškinys.
- `$dokumentas` yra dokumentas ar medis, kuriame vykdysime trynimą.

  Pavyzdys:

  ```
  local: transf_delete ("catalog.xml","catalog/cd/@upc=74646938720")
  ```

  Užklausa grąžina kopiją medžio, kuriame visi <*cd*> elementai su atributu *upc*, kuris yra lygus `74646938720`, yra ištrinami.

6.3. *Replace* operatoriaus kūrimas

6.3.1. Formalus „*replace*“ aprašymas

Apibrėžiamas operatoriaus veiksmas: Šis operatorius padaro kopiją medžio, kuriame tam tikros šakos yra pakeičiamos. Keitimo vieta yra aprašoma kelio nuoroda, taip pat kaip ir operatoriuje *insert*.

Aprašome *replace-with* operatorių ir kokius operatorius jis pats turės (aprašas iš [BMM03] tyrimo medžiagos):

replace p **as** $v **with** q′ **in** q

Siūlomas sprendimas, realizuojant XQuery funkcija:

```
declare function prefix:transf_delete(kelias AS datatype, u1
AS datatype, u2 AS datatype)
AS returnDatatype
{
 ...funkcijos kodas...
}
```

6.3.2. Realizacija „*replace*“ funkcijos

Realizuotas sprendimas formalaus modelio panaudojant XQuery funkciją (Visos papildomos funkcijos yra skyriuje „Priedai“, „3. *Replace* operatoriaus funkcijos kodas“ dalyje):

```
declare function local:transf_replace( $dokumentas as xs:string,
                                       $naujasdokumentas as node()*,
                                       $kelias as xs:string )
      as item()*
 {
      let $source:=doc($dokumentas)
      return local:copy( $source/node(),
                         local:dynamic-path($source,$kelias ),
                         $naujasdokumentas,
                         )
```

```
};
```

Funkcijos parametrai:

- `$kelias` reiškia kelią nuo šakninio elemento medyje iki elemento, kurį mes norime pakeisti. XPath reiškinys.
- `$dokumentas` yra dokumentas ar medis, kuriame vykdysime keitimą.
- `$naujasdokumentas` yra dokumentas ar medis, kurį pakeisime vietoj šakos nurodytos kelio reiškiniu.

Pavyzdys:

```
local: transf_replace
("catalog.xml","catalog/cd/@upc=74646938720","<new><changed></changed><
/new>")
```

Užklausa grąžina kopiją medžio, kuriame visi <*cd*> elementai su atributu *upc*, kuris yra lygus `74646938720`, yra pakeičiami į <*new*> elementą.

6.4. *Move* operatoriaus kūrimas

6.4.1. Formalus „*move*“ aprašymas

Apibrėžiamas operatoriaus veiksmas: Šis operatorius padaro kopiją medžio, kuriame medis ar tam tikros šakos yra perkeliamos į naują vietą (naują kelią).

Aprašome *move* operatorių ir kokius operatorius jis pats turės (aprašas iš [BMM03] tyrimo medžiagos):

```
move p (into | preceding | following) p' in q
```

Siūlomas sprendimas, realizuojant XQuery funkcija:

```
declare function prefix:transf_move($u1 AS datatype,
aplinkybes AS datatype, kelias AS datatype, u2 AS datatype)
AS returnDatatype
```

```
{
 ...funkcijos kodas...
}
```

6.4.2. Realizacija „*move*“ funkcijos

Realizuotas sprendimas formalaus modelio panaudojant XQuery funkciją (Visos papildomos funkcijos yra skyriuje „Priedai“, „4. *Move* operatoriaus funkcijos kodas“ dalyje):

```
declare function local: transf_move ( $dokumentas as xs:string,
                                      $aplinkybes as xs:string,
                                      $naujaskelias as xs:string,
                                      $kelias as xs:string )
      as item()*
 {
      let $source:=doc($dokumentas)
      return local:copy( $source/node(),
                         $aplinkybes,
                         local:dynamic-path($source,$ naujaskelias)),
                         local:dynamic-path($source,$kelias ) )
 };
```

Funkcijos parametrai:

- `$dokumentas` atitinka dokumentą ar medį, kuriame vykdysime perkelimą
- `$naujaskelias` atitinka kelią, kur perkelsime pasirinktą medį. XPath reiškinys.
- `$aplinkybes` atitinka kaip bus perkeliamas pasirinktas medis t.y. viena iš šių reikšmių: `'into', 'following', 'preceding'`.
- `$kelias` reiškia medį, kurį perkelsime. XPath reiškinys.

Pavyzdys:

```
local: transf_move
("catalog.xml","catalog/cd/artist/,"precending","catalog/cd/@upc=746469
38720")
```

Užklausa grąžina kopiją medžio, kuriame prieš visus <*artist*> elementus perkeliamas medis, kurio kelio reiškinys yra `catalog/cd/@upc=74646938720`.

7. XSLT ir XQuery transformacijų palyginimas

Buvo rasta gana nemažai informacijos šia tema. Visų pirmą išskirčiau diskusiją apie XQuery ir XSLT [Mc08]. Taip pat buvo naudingi ir kiti palyginimo šaltiniai apie XQuery ir XSLT [Ei05] [Tv08] [Ma06]. Nors jų prigimtinis tikslas ir skiriasi viena yra užklausų kalba, o kita yra transformavimo kalba, tačiau jie yra suderinami vienas su kitu ir tai leidžia palyginti XQuery ir XSLT.

Taigi galima išskirti keletą kriterijų ir bruožų pagal ką lyginsime XSLT ir XQuery.

- Trynimas – trynimas pasinaudojant funkcijomis.
- Suradimas – suradimas struktūros panaudojant funkcijomis.
- Perkėlimas – perkelti struktūra iš vienos vietos į kitą pasinaudojant funkcijomis.
- Įterpimas – įterpti struktūrą į tam tikrą kitos struktūros vietą pasinaudojant funkcijomis
- Struktūra yra naudojama ta pati, tad galima bus apskaičiuoti prieš tai paminėtų veiksmų atlikimo laiką tarp skirtingų kalbų
- Struktūrą keisti į sudėtingesnę t.y. didinti grafo lygį, pridėti į užklausas reguliaraus kelio reiškinius[2]
- Naudojami palyginimuose reguliarūs reiškiniai turi atitikti XPath rekomendaciją [BBCFKRS10].
- Palyginimas vienos transformacijos (pvz:. įterpimo), vienai duomenų struktūrai turi būti panaudotas su reguliaru reiškiniu iš : elemento ir elemento su požymiu (atributu, tekstu).

Visuose palyginimuose pasinaudosime sukurtomis XQuery funkcijomis :

- transf_insert
- transf_delete
- transf_move
- transf_replace

, ir XSLT sukurtom analogiškom transformacijomis tai pačiai duomenų struktūrai.

[2] Pasinaudota Altova XML Generator pasinaudojant aprašyta DTD struktūra

XQuery užklausų kalbai vykdyti yra reikalingas procesorius. Šiuo atveju procesorius yra programinė įranga, kuri leidžia ieškoti vartotojo užklaustos informacijos XML dokumente. Šiuose palyginimuose bus naudojamas „Saxon-B 9.x“ procesorius. Jis palaiko visas palyginimuose reikalingas technologijas t.y:

- XSLT 2.0
- XPath 2.0
- XQuery 1.0

7.1. Funkcionalumo tyrimas

Šio tyrimo tikslas parodyti, kad naujos sukurtos XQuery funkcijos ir XSLT transformacijos veikia vienodai t.y. gauna vienodus rezultatus, ir tokius rezultatus, kurių mes laukėme. Šiame palyginime pasinaudosime XML dokumentu *catalog.xml* (Dokumento struktūra pateikta skyriuje „Priedai“: „5. „Catalog.xml“ XML dokumento kodas.“) ir atliksime modifikavimo veiksmus, ir tada parodysime, kad gauti užklausų rezultatai vienodi.

7.1.1. Įterpimo transformacija

7.1.1.1. Elemento įterpimo transformacija

Transformacijos struktūra:

Struktūra : „catalog.xml“
Kelio reiškinys : `//cd`
Įterpiama nauja struktūra :

```
<elemen>
      <g r="ssss">
            test
      </g>
</elemen>
```

Paaiškinimas : Į struktūrą *catalog.xml* įterpiama aprašyta nauja struktūra, o įterpiama į visus elementus *artist*.

XQuery funkcija:

```
local:transf_insert("catalog.xml",'into',<elemen><g
r="ssss"></g></elemen>,//cd)
```

XSLT transformacija:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>
     <xsl:template match="//cd">
          <xsl:copy>
               <xsl:copy-of select="@*"/>
               <xsl:apply-templates/>
               <elemen><g r="ssss"></g></elemen>
          </xsl:copy>
     </xsl:template>
     <xsl:template match="@*|node() | text()">
          <xsl:copy>
                <xsl:copy-of select="@*"/>
                <xsl:apply-templates/>
                </xsl:copy>
     </xsl:template>
</xsl:transform>
```

7.1.1.2. Transformacijos rezultatai

Kadangi įterpiama buvo į visus *<cd>* elementus, tad reikia patvirtinti ar buvo įterpta abejose kalbose elementas *<elemen>*. Struktūra, kuri turėjo būti gauta (parodyta, kaip turi atrodyti pirmas *<cd>* elementas, o kituose *<cd>* elementuose analogiškai paskutinis elementas turi būti *<element>...</element>*) :

```
<catalog>
     <cd upc="602498678299">
          <artist>U2</artist>
          <title>How to Dismantle an Atomic Bomb</title>
          <price>13.98</price>
          <label>Interscope Records</label>
          <date>2004-11-23</date>
          <elemen>
               <g r="ssss">test</g>
          </elemen>
     </cd>
</catalog>
```

Atlikęs šias transformacijas gavau tokius pačius rezultatus tarp kalbų. Tad galima teigti, kad XQuery *transf_insert* transformacija su elementu be požymio, bei XSLT sukurta transformacija yra vienodos, o jų rezultatai buvo gauti tokie, kokių ir buvo tikimasi.

7.1.1.3. Elemento su požymiu įterpimo transformacija

Transformacijos struktūra:

Struktūra : „catalog.xml“

Kelio reiškinys : `//cd[@upc=75679244222]`

Įterpiama nauja struktūra :

```
<elemen>
      <g r="ssss">
            test
      </g>
</elemen>
```

Paaiškinimas : Į struktūrą *catalog.xml* įterpiama aprašyta nauja struktūra, o įterpiamą į visus elementus cd, kurių atributas *upc* yra lygus `75679244222`.

XQuery funkcija:

```
local:transf_insert("catalog.xml",'into',<elemen><g
r="ssss"></g></elemen>,//cd[@upc=75679244222])
```

XSLT transformacija:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>
     <xsl:template match="cd[@upc='75679244222']">
          <xsl:copy>
               <xsl:copy-of select="@*"/>
               <xsl:apply-templates/>
               <elemen><g r="ssss"></g></elemen>
          </xsl:copy>
     </xsl:template>
     <xsl:template match="@*|node() | text()">
          <xsl:copy>
                <xsl:copy-of select="@*"/>
                <xsl:apply-templates/>
                </xsl:copy>
     </xsl:template>
</xsl:transform>
```

7.1.1.4. Transformacijos rezultatai

Kadangi įterpiama buvo į *cd* elementą su atributu *upc*, kuris lygus 75679244222, tad reikia patvirtinti ar buvo įterpta abejose kalbose elementas <*elemen*>. Struktūra, kuri turėjo būti gauta:

```
<cd upc="75679244222">
     <artist>Led Zeppelin</artist>
     <title>Physical Graffiti</title>
     <price>22.99</price>
     <label>Atlantic</label>
     <date>1994-08-16</date>
     <elemen>
          <g r="ssss">test</g>
     </elemen>
</cd>
```

Atlikęs šias transformacijas gavau tokius pačius rezultatus abejose kalbose. Tad galima teigti, kad XQuery *transf_insert* su požymiu, šiuo atveju su atributu, bei XSLT sukurta transformacija yra vienodos, o jų rezultatai buvo gauti tokie, kokių ir buvo tikimasi.

7.1.2. Ištrynimo transformacija

7.1.2.1. Elemento ištrynimo transformacija

Transformacijos struktūra:

Struktūra : „catalog.xml“
Kelio reiškinys : `//artist`
Paaiškinimas : Iš struktūros *catalog.xml* ištrinami visi <*artist*> elementai.

XQuery funkcija:

```
local:transf_delete("catalog.xml",//artist)
```

XSLT transformacija:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>
     <xsl:template match="@*|node()">
          <xsl:copy>
               <xsl:apply-templates select="@*|node()"/>
          </xsl:copy>
     </xsl:template>
     <xsl:template match="artist"/>
</xsl:transform>
```

7.1.2.2. Transformacijos rezultatai

Kadangi ištrinami visi <*artist*> elementai, tad reikia patvirtinti ar buvo ištrinti jie abejose kalbose. Struktūra, kuri turėjo būti gauta (parodyta, kaip turi atrodyti pirmas <*cd*> elementas, o kituose <*cd*> elementuose analogiškai neturi būti <*artist*> elemento):

```
<catalog>
     <cd upc="602498678299">
          <title>How to Dismantle an Atomic Bomb</title>
          <price>13.98</price>
          <label>Interscope Records</label>
          <date>2004-11-23</date>
```

```
    </cd>
</catalog>
```

Atlikęs šias transformacijas gavau tokius pačius rezultatus tarp kalbų. Tad galima teigti, kad XQuery *transf_delete* transformacija su elementu be požymio, bei XSLT sukurta transformacija yra vienodos, o jų rezultatai buvo gauti tokie, kokių ir buvo tikimasi.

7.1.2.3. Elemento su požymiu ištrynimo transformacija

Transformacijos struktūra:

Struktūra : „catalog.xml“
Kelio reiškinys : `//artist[.='Led Zeppelin']`
Paaiškinimas : Iš struktūros *catalog.xml* ištrinami visi <*artist*> elementai su tekstu „Led Zeppelin“.

XQuery funkcija:

```
local:transf_delete("catalog.xml",//artist[.='Led Zeppelin'])
```

XSLT transformacija:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>
    <xsl:template match="@*|node()">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()"/>
        </xsl:copy>
    </xsl:template>
    <xsl:template match="//artist[.='Led Zeppelin']"/>
```

```
</xsl:transform>
```

7.1.2.4. Transformacijos rezultatai

Kadangi ištrinami visi <*artist*> elementai, kurių reikšmė yra lygi „Led Zeppelin“, reikia patvirtinti ar buvo visi ištrinti abejose kalbose. Struktūra, kuri turėjo būti gauta (parodyta, kaip turi atrodyti pirmas <*cd*> elementas, o kituose <*cd*> elementuose analogiškai neturi būti <*artist*> elemento su tekstine reikšme „Led Zeppelin“):

```
<catalog>
     <cd upc="75679244222">
          <title>Physical Graffiti</title>
          <price>22.99</price>
          <label>Atlantic</label>
          <date>1994-08-16</date>
     </cd>
</catalog>
```

Atlikęs šias transformacijas gavau tokius pačius rezultatus tarp kalbų. Tad galima teigti, kad XQuery *transf_delete* transformacija su elementu kartu su požymiu, bei XSLT sukurta transformacija yra vienodos, o jų rezultatai buvo gauti tokie, kokių ir buvo tikimasi.

7.1.3. Perkėlimo transformacija

7.1.3.1. Elemento perkėlimo transformacija

Transformacijos struktūra:

Struktūra : „catalog.xml“
Kelio reiškinys (iš kur perkeliamas) : `//artist`
Kelio reiškinys (į kur perkeliamas) : `//cd`

Paaiškinimas : Struktūroje *catalog.xml* visi <*artist*> elementai yra perkeliami į visus <*cd*> elementus.

XQuery funkcija:

```
local:transf_move("catalog.xml",'into',//artist,//cd)
```

XSLT transformacija:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>
     <xsl:template match="//cd"/>
     <xsl:template match="//artist">
          <xsl:copy-of select="//cd"/>
     </xsl:template>
     <xsl:template match="@*|node() | text()">
          <xsl:copy>
               <xsl:copy-of select="@*"/>
               <xsl:apply-templates/>
          </xsl:copy>
     </xsl:template>
</xsl:transform>
```

7.1.3.2. Transformacijos rezultatai

Kadangi perkeliami visi <*artist*> elementai, tad reikia patvirtinti ar buvo perkelti jie abejose kalbose. Struktūra, kuri turėjo būti gauta : Tai visuose <*cd*> elementuose prie jų turimų elementų pridedami visi <*artist*> elementai iš viso dokumento.

Atlikęs šias transformacijas gavau tokius pačius rezultatus tarp kalbų. Tad galima teigti, kad XQuery *transf_move* transformacija su elementu be požymio, bei XSLT sukurta transformacija yra vienodos, o jų rezultatai buvo gauti tokie, kokių ir buvo tikimasi.

7.1.3.3. Elemento su požymiu perkėlimo transformacija

Transformacijos struktūra:

Struktūra : „catalog.xml“
Kelio reiškinys (iš kur perkeliamas) : `//artist[.='U2']`
Kelio reiškinys (į kur perkeliamas) : `//cd[@upc=75679244222]`
Paaiškinimas : Struktūroje *catalog.xml* visi <*artist*> elementai su tekstine reikšme „U2“ yra perkeliami į visus <*cd*> elementus su atributu <*upc*>, kuris lygus `75679244222`.

XQuery funkcija:

```
local:transf_move("catalog.xml",'into',//artist[.='U2'],//cd[@upc=7
5679244222])
```

XSLT transformacija:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>
     <xsl:template match="//cd[@upc=75679244222]"/>
     <xsl:template match="//artist[.='U2']">
          <xsl:copy-of select="//cd[@upc=75679244222]"/>
     </xsl:template>
     <xsl:template match="@*|node() | text()">
          <xsl:copy>
               <xsl:copy-of select="@*"/>
```

```
            <xsl:apply-templates/>
        </xsl:copy>
    </xsl:template>
</xsl:transform>
```

7.1.3.4. Transformacijos rezultatai

Kadangi perkeliami visi <*artist*> elementai su požymiais, tad reikia patvirtinti ar buvo perkelti jie abejose kalbose. Struktūra, kuri turėjo būti gauta : Tai visuose <*cd*> elementuose, kurių atributas *upc*, prie jų turimų elementų pridedami visi <*artist*> elementai, su tekstine reikšme lygia „U2“, iš viso dokumento.

Atlikęs šias transformacijas gavau tokius pačius rezultatus tarp kalbų. Tad galima teigti, kad XQuery *transf_move* transformacija elemente su požymiu, bei XSLT sukurta transformacija yra vienodos, o jų rezultatai buvo gauti tokie, kokių ir buvo tikimasi.

7.1.4. Pakeitimo transformacija

7.1.4.1. Elemento pakeitimo transformacija

Transformacijos struktūra:

Struktūra : „catalog.xml“
Kelio reiškinys: `//artist`
Nauja struktūra : `<elemen><g r="ssss">dfgdfgd</g></elemen>`
Paaiškinimas : Struktūroje *catalog.xml* visi <*artist*> elementai yra pakeičiami į elementą <*elemen*>

XQuery funkcija:

```
local:transf_replace("catalog.xml",<elemen><g
r="ssss">dfgdfgd</g></elemen>,//artist)
```

XSLT transformacija:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>
     <xsl:template match="//artist">
          <elemen><g r="ssss">dfgdfgd</g></elemen>,//
     </xsl:template>
     <xsl:template match="@*|node() | text()">
         <xsl:copy>
             <xsl:copy-of select="@*"/>
             <xsl:apply-templates/>
         </xsl:copy>
     </xsl:template>
</xsl:transform>
```

7.1.4.2. Transformacijos rezultatai

Kadangi pakeičiami visi *<artist>* elementai, tad reikia patvirtinti ar buvo perkelti jie abejose kalbose. Struktūra, kuri turėjo būti gauta : Vietoj visų *<artist>* elementų turi būti *<elemen><g r="ssss">dfgdfgd</g></elemen>* elementai.

Šios transformacijos davė analogiškus rezultatus. Tad galima teigti, kad XQuery *transf_replace* transformacija elementui, bei XSLT sukurta transformacija yra vienodos, o jų rezultatai buvo gauti tokie, kokių ir buvo tikimasi.

7.1.4.3. Elemento su požymiu pakeitimo transformacija

Transformacijos struktūra:

Struktūra : „catalog.xml“

Kelio reiškinys: `//artist[.='U2']`

Nauja struktūra : `<elemen><g r="ssss">dfgdfgd</g></elemen>`

Paaiškinimas : Struktūroje *catalog.xml* visi <*artist*> elementai, su tekstine reikšme „U2“ yra pakeičiami į elementą <*elemen*>

XQuery funkcija:

```
local:transf_replace("catalog.xml",<elemen><g
r="ssss">dfgdfgd</g></elemen>,//artist[.="U2"])
```

XSLT transformacija:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>
     <xsl:template match="//artist[.='U2']">
          <elemen><g r="ssss">dfgdfgd</g></elemen>,//
     </xsl:template>

     <xsl:template match="@*|node() | text()">
         <xsl:copy>
             <xsl:copy-of select="@*"/>
             <xsl:apply-templates/>
         </xsl:copy>
     </xsl:template>
</xsl:transform>
```

7.1.4.4. Transformacijos rezultatai

Kadangi pakeičiami visi <*artist*> elementai su požymiu, tad reikia patvirtinti ar buvo perkelti jie abejose kalbose. Struktūra, kuri turėjo būti gauta : Vietoj visų <*artist*> elementų, su tekstine reikšme „U2“ turi būti <*elemen*><*g r="ssss"*>*dfgdfgd*</*g*></*elemen*> elementai.

Šios transformacijos davė analogiškus rezultatus. Tad galima teigti, kad XQuery *transf_replace* transformacija elementui su požymiu, bei XSLT sukurta transformacija yra vienodos, o jų rezultatai buvo gauti tokie, kokių ir buvo tikimasi.

7.1.5. Funkcionalumo tyrimo rezultatai

Buvo išbandytos visos sukurtos XQuery transformavimo funkcijos. Taip pat palyginti gauti rezultatai tarp XQuery ir XSLT transformacijų. Transformacijos buvo atliekamos su elementu, bei su elementu ir požymiu. Kiekvienos transformacijos rezultatai sutapo tarp XQuery funkcijų bei XSLT transformacijų. Kadangi XSLT naudojamas duomenų modelis yra toks pats kaip ir XPath 2.0, bei XQuery 1.0 duomenų modelis (XDM) [BFMMNW10]. Tai iš to, kad visose transformacijose buvo gauti vienodi rezultatai, galime teigti, kad naujos XQuery transformavimo funkcijos yra gerai apibrėžtos.

7.2. Transformacijų efektyvumo palyginimas

Kadangi buvo nustatyta, kad naujos transformavimo funkcijos yra gerai apibrėžtos, tai galima lyginti jų efektyvumą. Tam reikės susikurti XML dokumentus. Transformacijom vykdyti pasinaudosime DTD, tam kad visuose bandymuose turėtume vienodą XML struktūrą. Naudojamas DTD - „normalus.dtd“. Tam, kad galėtumėme lyginti turime susikurti skirtingus XML dokumentus[3].

7.2.1. Pirmas palyginimas

Visų pirmą, pasinaudodami aprašytu DTD aprašu, sugeneruosime dokumentą su tam tikrom charakteristikomis:

- Failas: Test1.xml (62 kb)
- Šakninis elementas: `<catalog>`
- Struktūra:

  `<product>` (20 elementų su 5 atributais ir jų reikšmėmis)

[3] Naudojamas įrankis – Altova XMLSpy – Generate XML funkcija

<specifications> (20 elementų su 2 atributais ir jų reikšmėmis)

<options> (vienas elementas su 3 atribsutais su reikšmėmis)

<price> (20 elementų su 4 atributais ir jų reikšmėmis)

- Elementų kiekis : 821 elementas.

Užklausos naudojamos vienodos, atitinkamose transformacijose. Tyrimo objektai – užklausa, dokumento dydis, užklausos atlikimo laikas. Taip pat yra nustatoma ar gauta užklausa yra validi[4].

Funkcija	Atlikimo laikas	XML dokumento dydis	Validus
XQuery : transf_insert	11,5 sec	62 kb	Taip
XSLT : įterpimo transformacija	1 sec	62 kb	Taip
XQuery : transf_delete	7 sec	62 kb	Taip
XSLT : ištrynimo transformacija	0,5 sec	62 kb	Taip
XQuery : transf_move	13 sec	62 kb	Taip
XSLT : perkėlimo transformacija	2 sec	62 kb	Taip
XQuery : transf_replace	8 sec	62 kb	Taip
XSLT : pakeitimo transformacija	0,6 sec	62 kb	Taip

1 lentelė. Transformacijų palyginimo lentelė Test1.xml dokumentui

7.2.2. Antras palyginimas

Šiame palyginime pasinaudosime XML dokumentu gautu po įterpimo transformacijos pirmame palyginime. Charakteristikos:

- Failas: Test2.xml (70 kb)
- Šakninis elementas: <catalog>
- Struktūra:

[4] Validavimo įrankis – Altova Validate priemonė

<product> (20 elementų su 5 atributais ir jų reikšmėmis)

<specifications> (20 elementų su 2 atributais ir jų reikšmėmis)

<elemen> (1 elementas)

<g> (1 elementas su vienu atributu ir jo reikšme)

<options> (vienas elementas su 3 atributais su reikšmėmis)

<price> (20 elementų su reikšmėmis ir su 4 atributais ir jų reikšmėmis)

- Elementų kiekis : 860 elementų.

Užklausos naudojamos vienodos, atitinkamose transformacijose. Tyrimo objektai – užklausa, dokumento dydis, užklausos atlikimo laikas.

Funkcija	Atlikimo laikas	XML dokumento dydis	Validus
XQuery : transf_insert	10,6 sec	70 kb	Taip
XSLT : įterpimo transformacija	0,8 sec	70 kb	Taip
XQuery : transf_delete	11,6 sec	70 kb	Taip
XSLT : ištrynimo transformacija	0,5 sec	70 kb	Taip
XQuery : transf_move	13 sec	70 kb	Taip
XSLT : perkėlimo transformacija	1,6 sec	70 kb	Taip
XQuery : transf_replace	9 sec	70 kb	Taip
XSLT : pakeitimo transformacija	1,4 sec	70 kb	Taip

2 lentelė. Transformacijų palyginimo lentelė Test2.xml dokumentui

7.2.3. Trečias palyginimas

Šiame palyginime pasinaudosime XML dokumentu gautu po įterpimo transformacijos pirmame palyginime. Charakteristikos:

- Failas: Test3.xml (232 kb)

- Šakninis elementas: <catalog>
- Struktūra:

<product> (20 elementų su 5 atributais ir jų reikšmėmis)

<specifications> (20 elementų su 2 atributais ir jų reikšmėmis)

<elemen> (1 elementas)

<g> (1 elementas su vienu atributu ir jo reikšme)

<test> (1 elementas)

<specifications> (5 elementų su 2 atributais ir jų reikšmėmis)

<options> (vienas elementas su 3 atributais su reikšmėmis)

<price> (20 elementų su reikšmėmis ir su 4 atributais ir jų reikšmėmis)

- Elementų kiekis : 3620 elementų.

Užklausos naudojamos vienodos, atitinkamose transformacijose. Tyrimo objektai – užklausa, dokumento dydis, užklausos atlikimo laikas.

Funkcija	Atlikimo laikas	XML dokumento dydis	Validus
XQuery : transf_insert	12 sec	232 kb	Taip
XSLT : įterpimo transformacija	2,2 sec	232 kb	Taip
XQuery : transf_delete	10 sec	232 kb	Taip
XSLT : ištrynimo transformacija	1,5 sec	232 kb	Taip
XQuery : transf_move	14 sec	232 kb	Taip
XSLT : perkėlimo transformacija	2,7 sec	232 kb	Taip
XQuery : transf_replace	10 sec	232 kb	Taip
XSLT : pakeitimo transformacija	2,4 sec	232 kb	Taip

3 lentelė. Transformacijų palyginimo lentelė Test3.xml dokumentui

7.2.4. Ketvirtas palyginimas

Šiame palyginime pasinaudosime XML dokumentu gautu po įterpimo transformacijos pirmame palyginime. Charakteristikos:

- Failas: Test4.xml (1,42 MB)
- Šakninis elementas: `<catalog>`
- Struktūra:

  `<product>` (99 elementų su 5 atributais ir jų reikšmėmis)

  `<specifications>` (99 elementų su 2 atributais ir jų reikšmėmis)

  `<options>` (vienas elementas su 3 atributais su reikšmėmis)

  `<price>` (99 elementų su reikšmėmis ir su 4 atributais ir jų reikšmėmis)
- Elementų kiekis : 19701 elementų.

Užklausos naudojamos vienodos, atitinkamose transformacijose. Tyrimo objektai – užklausa, dokumento dydis, užklausos atlikimo laikas.

Funkcija	Atlikimo laikas	XML dokumento dydis	Validus
XQuery : transf_insert	13,5 sec	1,42 MB	Taip
XSLT : įterpimo transformacija	3 sec	1,42 MB	Taip
XQuery : transf_delete	12 sec	1,42 MB	Taip
XSLT : ištrynimo transformacija	3 sec	1,42 MB	Taip
XQuery : transf_move	13,5 sec	1,42 MB	Taip
XSLT : perkėlimo transformacija	3,7 sec	1,42 MB	Taip
XQuery : transf_replace	8 sec	1,42 MB	Taip
XSLT : pakeitimo transformacija	3 sec	1,42 MB	Taip

4 lentelė. Transformacijų palyginimo lentelė Test4.xml dokumentui

7.2.5. Penktas palyginimas

Šiame palyginime pasinaudosime XML dokumentu gautu po įterpimo transformacijos pirmame palyginime. Charakteristikos:

- Failas: Test5.xml (4,21 MB)
- Šakninis elementas: `<catalog>`
- Struktūra:

`<product>` (99 elementų su 5 atributais ir jų reikšmėmis)

`<specifications>` (99 elementų su 2 atributais ir jų reikšmėmis)

`<test>` (1 elementas)

`<specifications>` (5 elementų su 2 atributais ir jų reikšmėmis)

`<options>` (vienas elementas su 3 atributais su reikšmėmis)

`<price>` (99 elementų su reikšmėmis ir su 4 atributais ir jų reikšmėmis)

- Elementų kiekis : 68706 elementų.

Užklausos naudojamos vienodos, atitinkamose transformacijose. Tyrimo objektai – užklausa, dokumento dydis, užklausos atlikimo laikas.

Funkcija	Atlikimo laikas	XML dokumento dydis	Validus
XQuery : transf_insert	16 sec	4,21 MB	Taip
XSLT : įterpimo transformacija	6 sec	4,21 MB	Taip
XQuery : transf_delete	13 sec	4,21 MB	Taip
XSLT : ištrynimo transformacija	7 sec	4,21 MB	Taip
XQuery : transf_move	18 sec	4,21 MB	Taip
XSLT : perkėlimo transformacija	9 sec	4,21 MB	Taip
XQuery : transf_replace	13 sec	4,21 MB	Taip
XSLT : pakeitimo transformacija	9 sec	4,21 MB	Taip

5 lentelė. Transformacijų palyginimo lentelė Test5.xml dokumentui

7.2.6. Šeštas palyginimas

Šiame palyginime pasinaudosime XML dokumentu gautu po įterpimo transformacijos pirmame palyginime. Charakteristikos:

- Failas: Test6.xml (6,23 MB)
- Šakninis elementas: `<catalog>`
- Elementų kiekis : ~101000 elementų.

Užklausos naudojamos vienodos, atitinkamose transformacijose. Tyrimo objektai – užklausa, dokumento dydis, užklausos atlikimo laikas.

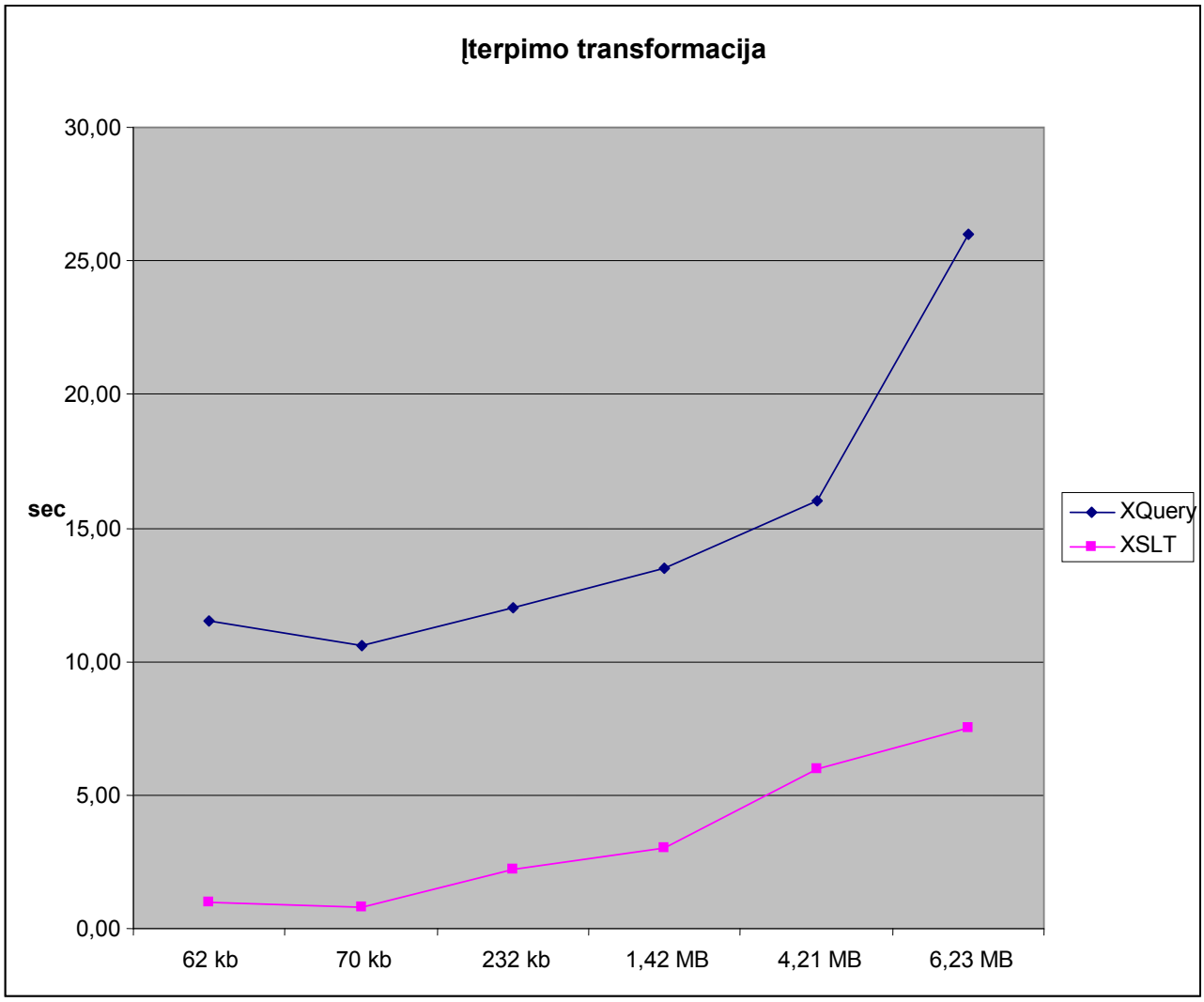
Funkcija	Atlikimo laikas	XML dokumento dydis	Validus
XQuery : transf_insert	26 sec	6,23 MB	Taip
XSLT : įterpimo transformacija	7,5 sec	6,23 MB	Taip
XQuery : transf_delete	13 sec	6,23 MB	Taip
XSLT : ištrynimo transformacija	13 sec	6,23 MB	Taip
XQuery : transf_move	22 sec	6,23 MB	Taip
XSLT : perkėlimo transformacija	10 sec	6,23 MB	Taip
XQuery : transf_replace	15 sec	6,23 MB	Taip
XSLT : pakeitimo transformacija	13 sec	6,23 MB	Taip

6 lentelė. Transformacijų palyginimo lentelė Test6.xml dokumentui

7.3. Efektyvumo tyrimo rezultatai

Gauti transformacijų rezultatai iš visų 6 lentelių atvaizduoti 4 grafikuose, kurie atitinka keturias transformacijas. X ašyje bus XML failo dydis, o y ašyje atlikimo laikas sekundėmis.

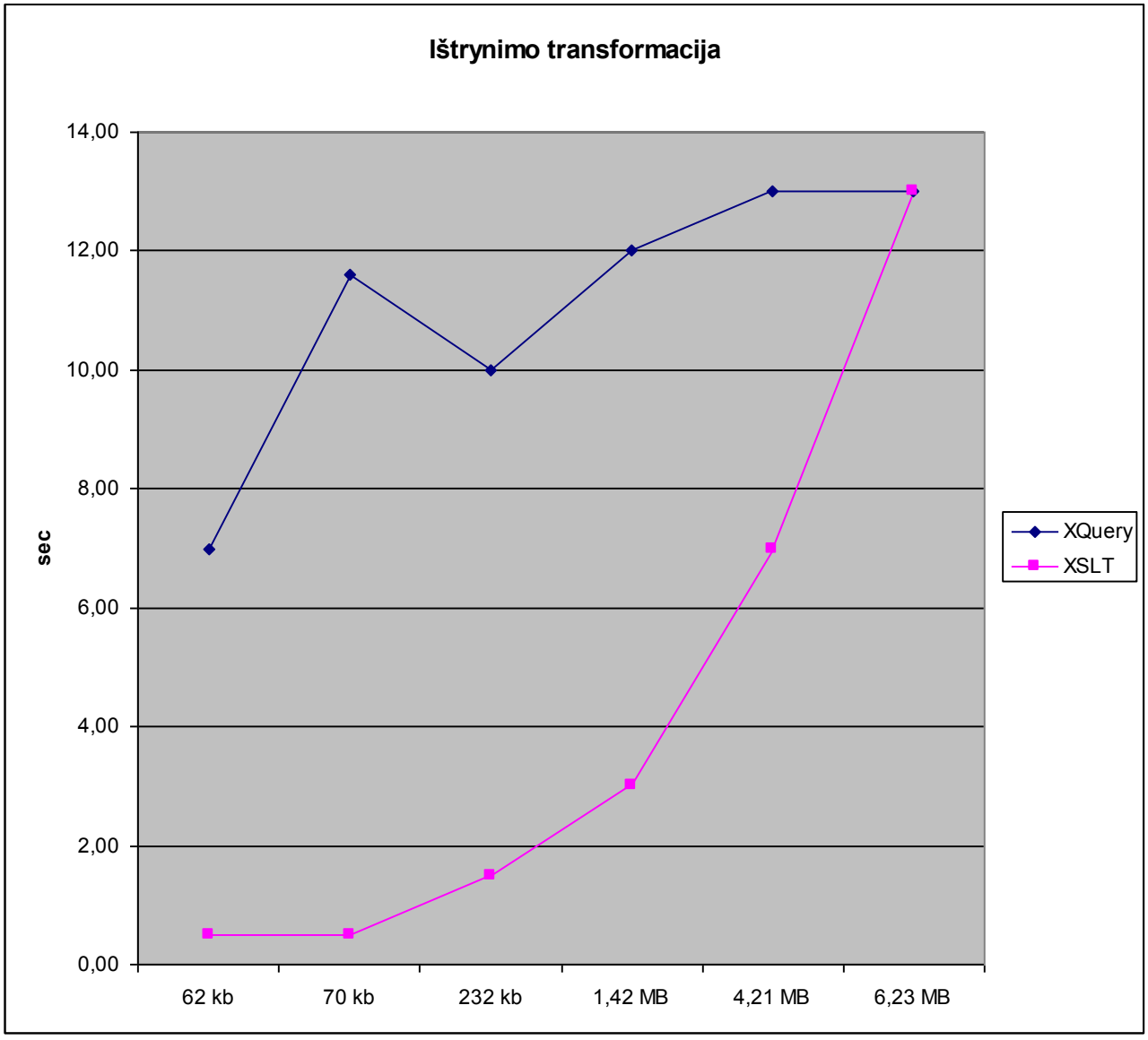
Įterpimo transformacija



1 Grafikas. Įterpimo transformacijos XQuery ir XSLT

Kaip matome XQuery sukurtomis funkcijomis nėra labai efektyvu atlikti įterpimo transformacijas. Jas žymiai efektyviau atlikti XSLT transformacijose.

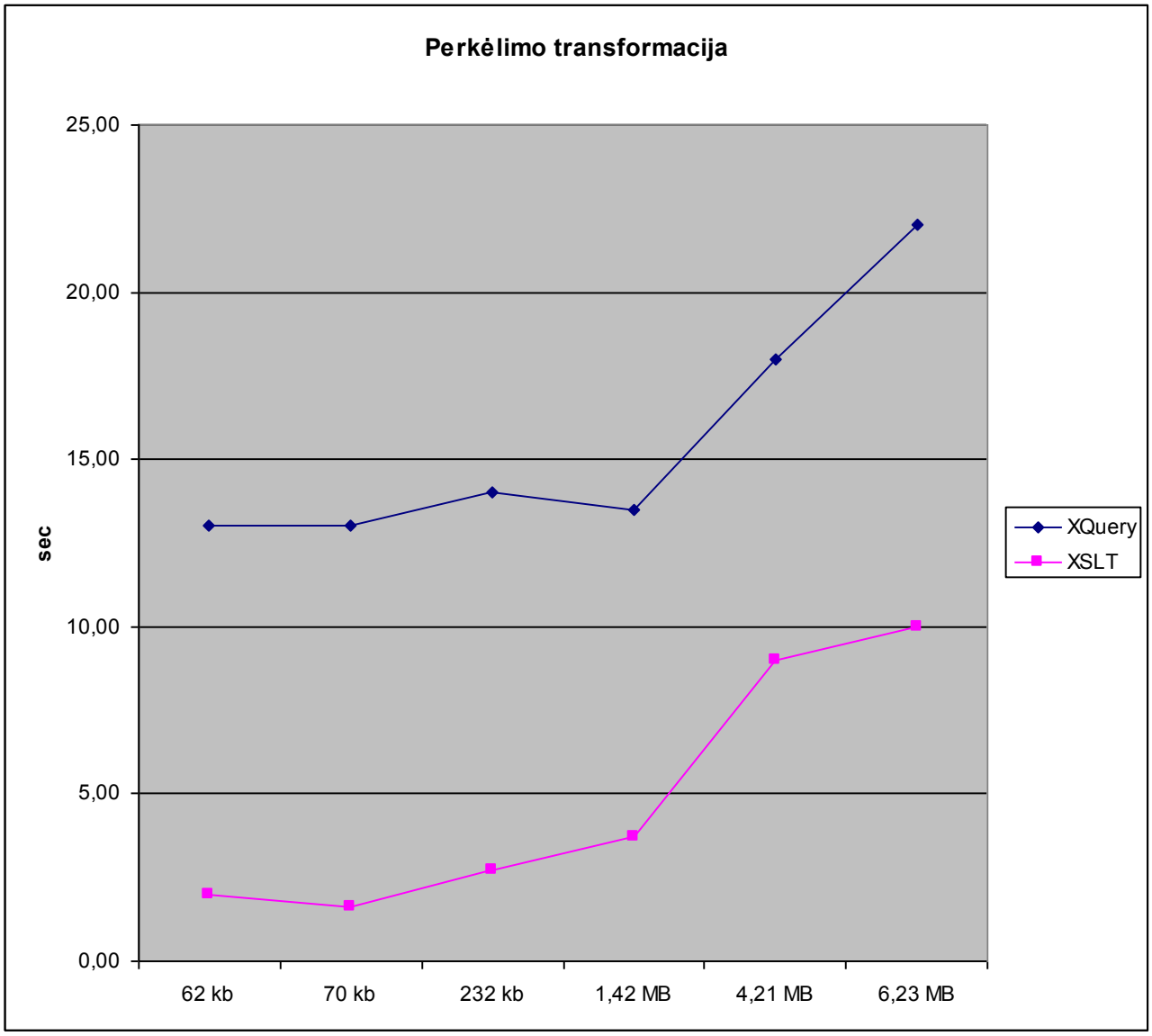
Ištrynimo transformacija



1 Grafikas. Įterpimo transformacijos XQuery ir XSLT

Kaip matome XQuery sukurtomis funkcijomis nėra labai efektyvu atlikti ištrynimo transformacijas mažo dydžio failuose. Jas žymiai efektyviau atlikti XSLT transformacijose, tačiau kai XML dokumento dydis didėja, XQuery pasiekia tą patį efektyvumą.

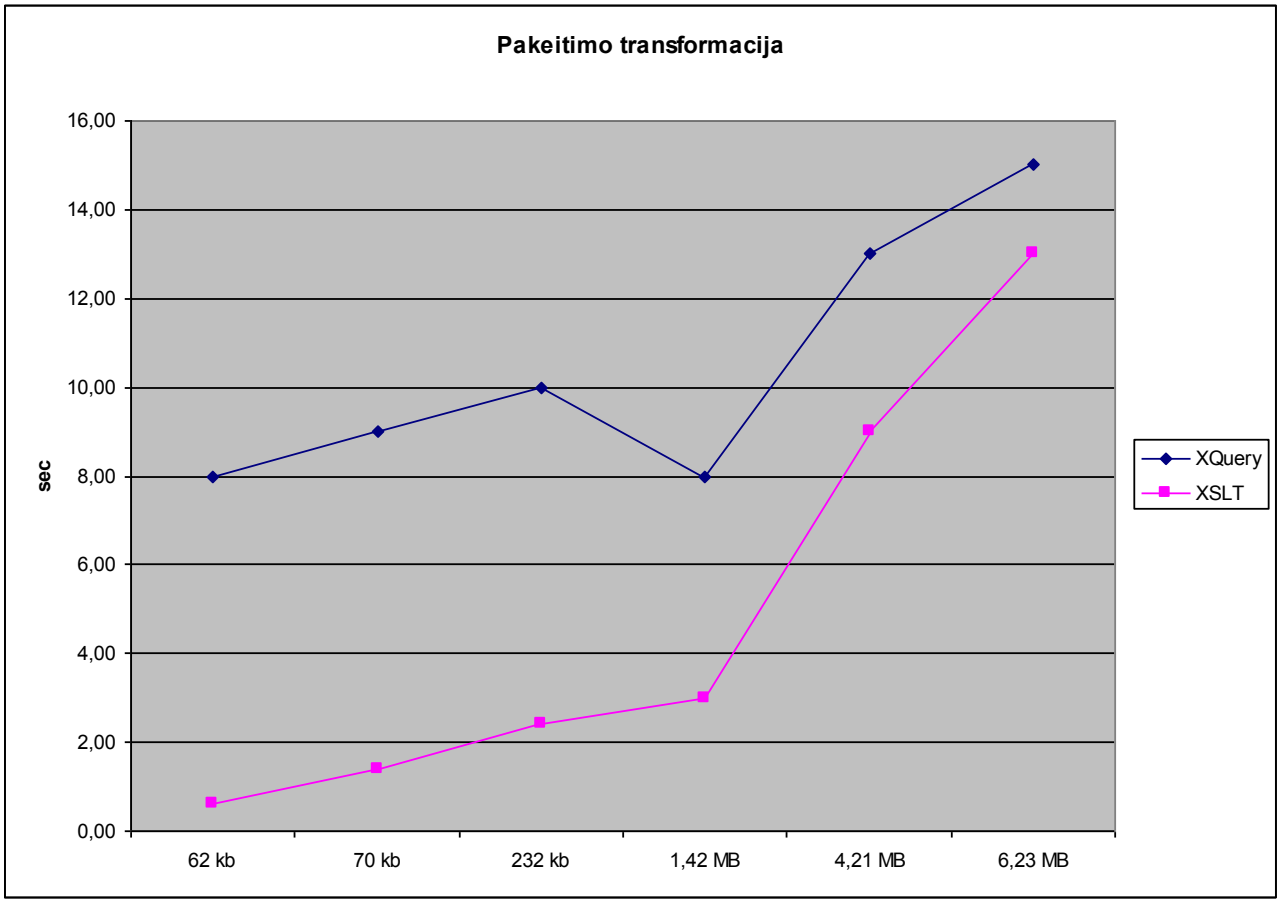
Perkėlimo transformacija



1 Grafikas. Įterpimo transformacijos XQuery ir XSLT

Kaip matome XQuery sukurtomis funkcijomis nėra efektyvu atlikti perkėlimo transformacijas. Jas žymiai efektyviau atlikti XSLT transformacijose.

Pakeitimo transformacija



1 Grafikas. Įterpimo transformacijos XQuery ir XSLT

Kaip matome XQuery sukurtomis funkcijomis nėra labai efektyvu atlikti perkėlimo transformacijas mažo dydžio failuose. Jas žymiai efektyviau atlikti XSLT transformacijose, tačiau kai XML dokumento dydis didėja, XQuery pasiekia panašų efektyvumą.

8. Išvados

Šiame darbe mūsų tikslas buvo praplėsti XQuery užklausų kalbą transformavimo funkcijomis ir tuo pačiu palyginti jas su XSLT transformacijomis. Buvo siekiama:

- Supaprastinti medžio transformavimo reiškinius
- Sukurti transformavimo funkcijas, kurios padėtų įvykdyti norimas transformacijas
- Parodyti, kad XQuery naujos funkcijos leidžia lengviau išreikšti transformacijas

Taigi buvo sukurtos keturios transformavimo funkcijos:

- `transf_insert` – Jos pagalba mes įterpiame medį ar šaką į norimą medžio vietą.
- `transf_delete` – Jos pagalba mes ištriname medį ar šaką iš norimos medžio vietos.
- `transf_move` – Jos pagalba mes perkeliame medį ar šaką iš norimos medžio vietos į kitą.
- `transf_replace` – Jos pagalba mes pakeičiame medį ar šaką į norimą naują medį.

Funkcionalumo tyrimo skyriuje, buvo aprašytos transformacijos XSLT ir XQuery kalbose. Įvykdžius kiekvieną transformacijos užklausą buvo gauti analogiški rezultatai dvejose kalbose. Taigi – XQuery operatoriai buvo apibrėžti teisingai.

Kaip matome „7.1.4.1.Elemento pakeitimo transformacija“ skyriuje transformacija XQuery kalboje išreikšta viena funkcija, tuo tarpu XSLT 12 eilučių. Ir beveik kiekviename tyrimo etape matome, kad XSLT atlikti tą pačią transformaciją reikalauja daugiau kodo eilučių. Sistemose, kuriose vykdoma daug transformacijų, tokiose, kaip „mapping“ tipo sistemose ar didelių duomenų bazių sistemose, transformacijos vykdomos dažnai. Tad kuo transformacija lengviau išreiškiama tuo lengviau ją suprasti, modifikuoti ar sukurti. Tad šiais pavyzdžiais parodome, kaip galima lengviau išreikšti transformacijas.

Pasinaudodami skyriuje 7.2. „Transformacijų palyginimas“ gauta informacija galima teigti, kad naujos funkcijos nėra tokios efektyvios kaip XSLT transformacijos, nedidelio dydžio dokumentu transformavime. Kai kurios funkcijos pranašumą įgyja didelio dydžio dokumentuose (pvz:.

Palyginimo lentelėje ~ 8 MB dokumentas). Jų atlikimo laikas tada susivienodina su XSLT transformacijom.

Tolesniuose darbuose vystant transformavimo operatorius – funkcijas reiktų bandyti surasti efektyvesnius algoritmus toms XQuery transformavimo funkcijoms, kurių transformavimo funkcijų efektyvumas nepasiekia XSLT ir nustatyti ar išvis jį įmanoma pasiekti. Taip pat reiktų XQuery sukurtas transformacijas perkelti iš funkcijų lygio į operatorių, bei „Prologo“ lygį. Tikslas, kad XQuery kalbą sudarytų transformavimo operatoriai, kaip pvz:. SQL kalboje „SELECT FROM“ sąlygos, taip ir XQuery kalboje „INSERT INTO“. Tai leistų dar lengviau išreikšti transformacijas, nei dabar jos yra išreikštos. Taip pat būtų atsisakoma tam tikrų dabar naudojamų funkcijų, kurios operatorių lygmenyje yra nereikalingos ir tai leistų sutaupyti laiko resursų užklausos vykdymo metu.

9. Literatūros sąrašas

[BMM03] E. Bruno, J. Le Maitre, E. Murisasco. Extending XQuery with Transformation Operators, 2003.

[Le01] E. Lenz. XQuery: Reinventing the Wheel?, 2001.

[Mc08] D. McCreary. Teaching XSLT vs. Teaching XQuery. Discussion on forum later. http://www.oreillynet.com/xml/blog/2008/06/teaching_xslt_vs_teaching_xque.html, 2008.

[Ei05] J. D. Eisenberg. Comparing XSLT and XQuery, http://www.xml.com/pub/a/2005/03/09/xquery-v-xslt.html March 09, 2005.

[Tv08] J. Tverskov. XQuery and XSLT compared, http://www.xmlplease.com/xquery-xslt December 22, 2008.

[Ma06] B. Marchal. Working XML: Comparing XSLT 2.0 and XQuery, http://www.ibm.com/developerworks/xml/library/x-wxxm34/index.html ,04 Apr 2006.

[Mai86] D.Maier. A logic for objects. In Workshop on "Foundations on Deductive Database and Logic Programming", 1986.

[Cl99a] J. Clark. XSL Transformations (XSLT specifcation), http://www.w3.org/TR/WD-xslt , 1999

[El01] J. Elkada. A Graph-oriented Query Language for Semi-Structured Data: Theoretical and Practical Analysis, 2001

[MS07] A. Møller, M. I. Schwartzbach. XML Graphs in Program Analysis, 2007.

[Cl99b] J. Clark. Xml Path Language (XPATH), http://www.w3.org/TR/xpath , 1999.

[BBCFKRS10] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, J. Siméon. The XPath 2.0 Standard, W3C XSL/XML Query Working Groups, http://www.w3.org/TR/xpath20/ , 2010.

[BFMMNW10] A. Berglund, M. Fernández, A. Malhotra, J. Marsh, M. Nagy, N. Walsh. XQuery 1.0 and XPath 2.0 Data Model (XDM), http://www.w3.org/TR/xpath-datamodel/ , 2010.

[CSD00] Computer Sciences Department University of Wisconsin-Madison. The Niagara Internet Query System , http://www.w3.org/TR/xpath , 2000.

[CFMR05] D. Chamberlin, P. Fankhauser, M. Marchiori, J. Robie XML Query (XQuery) Requirements W3C Working Draft, 3 June 2005. http://www.w3.org/2006/06/XML/query.html

[NDM00] J. Naughton, D. DeWitt, D. Maier, et al., "The Niagara Internet Query System." http://www.cs.wisc.edu/niagara/Publications.html , 2000.

[XML98] XML (eXtensible Markup Language) http://www.w3.org/TR/1998/REC-xml-19980210, 10 February, 1998.

[GMW99] R. Goldman, J. McHugh, J. Widom Stanford University. From Semistructured Data to XML:Migrating the Lore Data Model and Query Language, 1999.

[AQMWW97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. L. Wiener. The Lorel Query Language for Semistructured Data, 1997.

[B97] P. Buneman. Semistructured Data, 1997.

[JMZ00] D. Jindal, S. Muthukrishnan, O. Zaki. Query Containment for XML-QL. University of Wisconsin-Madison. April 28, 2000.

10. Priedas

1. *Insert* operatoriaus funkcijos kodas.

```
{-------------------------------------------------------------------------------------------------------------------}
declare function local:index-of-node ($nodes as node()* ,$nodeToFind as node() )
as xs:integer* {

      for $seq in (1 to fn:count($nodes))
            return $seq[$nodes[$seq] is $nodeToFind]

};
{-------------------------------------------------------------------------------------------------------------------}
declare function local:substring-before-if-contains ( $arg as xs:string? ,
$delim as xs:string )  as xs:string? {

   if (contains($arg,$delim))
      then substring-before($arg,$delim)
      else $arg

} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:substring-after-if-contains ( $arg as xs:string?,$delim
as xs:string )  as xs:string? {

   if (contains($arg,$delim))
      hen substring-after($arg,$delim)
      else $arg

} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:name-test($testname as xs:string? ,$names as xs:string* )
as xs:boolean {

   $testname = $names
   or
   $names = '*'
   or
   local:substring-after-if-contains($testname,':') = (for $name in $names
   return substring-after($name,'*:'))
   or
   substring-before($testname,':') =(for $name in $names[contains(.,':*')]
   return substring-before($name,':*'))

} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:dynamic-path ( $parent as node() ,$path as xs:string )
as item()* {

  let $nextStep := local:substring-before-if-contains($path,'/')
  let $finalSteps  := substring-after($path,'=')
  let $restOfSteps := substring-after($path,'/')

  for $child in
    ($parent/*[local:name-test(name(),$nextStep)],
```

```
     $parent/@*[local:name-test(name(),
      local:substring-before-if-contains(substring-after($nextStep,'@'),'='))])
  return
      if ( string-length(local:substring-before-if-contains(substring-
      after($nextStep,'@'),'='))!=0) then
            if (local:substring-before-if-contains(substring-
            after($nextStep,'@'),'=')=local-name($child)) then
                  if ( string-length($finalSteps)!=0)  then
                        if ($finalSteps = data($child))
                              then $child
                              else local:dynamic-path($child, $restOfSteps)
                  else $child
            else local:dynamic-path($child, $restOfSteps)
      else  if ($restOfSteps)
         then local:dynamic-path($child, $restOfSteps)
         else $child
} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:is-node-in-sequence-deep-equal ( $node as node()? ,$seq
as node()* )  as xs:boolean {
      some $nodeInSeq in $seq satisfies deep-equal($nodeInSeq,$node)
} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:copy($element as element(),$aplinkybes as
xs:string,$newnodes as node()*,$gautkelias1 as node()*) as element() {
      if ($element=$gautkelias1)
      then
            for $child in $element
              return if (local:is-node-in-sequence-deep-
equal($child,$gautkelias1)=true() or local:is-node-in-sequence-deep-
equal($child/@*,$gautkelias1)=true())
                  then if ($aplinkybes='into') then ( element {node-
name($child)} {$child/@*,insert-before($child/node(),local:index-of-
node($child/node(),$child/node()[last()])+1,$newnodes)})
                        else if ($aplinkybes='following') then ( insert-
before($child,local:index-of-node($child,$child)+1,$newnodes))
                              else ( insert-before($child,local:index-of-
node($child,$child),$newnodes))
                              else  if ($child instance of element())
                                    then
local:copy($child,$aplinkybes,$newnodes,$gautkelias1)
                                    else $child else element {node-
name($element)}
             {$element/@*,
                   for $child in $element/node()
                         return if (local:is-node-in-sequence-deep-
equal($child,$gautkelias1)=true() or local:is-node-in-sequence-deep-
equal($child/@*,$gautkelias1)=true())
                              then if ($aplinkybes='into')
                                    then ( element {node-name($child)}
{$child/@*,insert-before($child/node(),local:index-of-
node($child/node(),$child/node()[last()])+1,$newnodes)})
                                          else if ($aplinkybes='following')
                                                then ( insert-
before($child,local:index-of-node($child,$child)+1,$newnodes))
                                                else ( insert-
before($child,local:index-of-node($child,$child),$newnodes)) else  if ($child
instance of element())
      then local:copy($child,$aplinkybes,$newnodes,$gautkelias1)
```

```
      else $child
              }
   };
{-------------------------------------------------------------------------------------------------------------------}
declare function local:transf_insert($dokumentas as xs:string,$aplinkybes as
xs:string, $newnodes as node()*,$gautkelias as xs:string) as item()*{

      let $source:=doc($dokumentas)
      return local:copy($source/node(),$aplinkybes,$newnodes,local:dynamic-
      path($source,$gautkelias))

};
{-------------------------------------------------------------------------------------------------------------------}
```

2. *Delete* operatoriaus funkcijos kodas.

```
{-------------------------------------------------------------------------------------------------------------------}
declare function local:index-of-node ($nodes as node()* ,$nodeToFind as node() )
as xs:integer* {

   for $seq in (1 to fn:count($nodes))
      return $seq[$nodes[$seq] is $nodeToFind]

 };
{-------------------------------------------------------------------------------------------------------------------}

declare function local:substring-before-if-contains ( $arg as xs:string? ,
$delim as xs:string )  as xs:string? {

   if (contains($arg,$delim))
      then substring-before($arg,$delim)
      else $arg

} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:substring-after-if-contains ( $arg as xs:string?,$delim
as xs:string )  as xs:string? {

   if (contains($arg,$delim))
      hen substring-after($arg,$delim)
      else $arg

} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:name-test($testname as xs:string? ,$names as xs:string* )
as xs:boolean {

      $testname = $names or $names = '*' or local:substring-after-if-
      contains($testname,':') =
      (for $name in $names return substring-after($name,'*:'))

      or

      substring-before($testname,':') =(for $name in $names[contains(.,':*')]
      return substring-before($name,':*'))
} ;
{-------------------------------------------------------------------------------------------------------------------}
```

```
declare function local:dynamic-path ( $parent as node() ,$path as xs:string )
as item()* {
  let $nextStep := local:substring-before-if-contains($path,'/')
  let $finalSteps  := substring-after($path,'=')
  let $restOfSteps := substring-after($path,'/')
  for $child in
    ($parent/*[local:name-test(name(),$nextStep)],
     $parent/@*[local:name-test(name(),
                                  local:substring-before-if-
      contains(substring-after($nextStep,'@'),'='))])
  return
      if ( string-length(local:substring-before-if-contains(substring-
      after($nextStep,'@'),'='))!=0) then
            if (local:substring-before-if-contains(substring-
            after($nextStep,'@'),'=')=local-name($child)) then
                  if ( string-length($finalSteps)!=0)  then
                        if ($finalSteps = data($child))
                              then $child
                              else local:dynamic-path($child, $restOfSteps)
                  else $child
            else local:dynamic-path($child, $restOfSteps)
  else  if ($restOfSteps)
         then local:dynamic-path($child, $restOfSteps)
         else $child
    } ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:is-node-in-sequence-deep-equal ( $node as node()? ,$seq
as node()* )  as xs:boolean {
      some $nodeInSeq in $seq satisfies deep-equal($nodeInSeq,$node)
} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:transf_delete($dokumentas as xs:string,$kelias as
xs:string) as item()*{
      let $source:=doc($dokumentas)
      return local:copy($source/node(),local:dynamic-path($source,$kelias))
};
```

3. *Replace* operatoriaus funkcijos kodas.

```
{-------------------------------------------------------------------------------------------------------------------}
declare function local:index-of-node ($nodes as node()* ,$nodeToFind as node() )
as xs:integer* {

   for $seq in (1 to fn:count($nodes))
      return $seq[$nodes[$seq] is $nodeToFind]

 };
{-------------------------------------------------------------------------------------------------------------------}
declare function local:is-node-in-sequence-deep-equal ( $node as node()? ,$seq
as node()* )  as xs:boolean {
      some $nodeInSeq in $seq satisfies deep-equal($nodeInSeq,$node)
} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:copy($element as element(),$newnodes as
node()*,$gautkelias1 as node()*) as element() {
      element {node-name($element)}
```

```
      {$element/@*,for $child in $element/node() return if (local:is-node-in-
sequence-deep-equal($child,$gautkelias1)=true() or local:is-node-in-sequence-
deep-equal($child/@*,$gautkelias1)=true())
            then for $newchild in $newnodes/node() return $newchild
             else  if ($child instance of element()) then
local:copy($child,$newnodes,$gautkelias1)
                                                  else $child
      }
};
{-------------------------------------------------------------------------------------------------------------------}
declare function local:transf_replace dokumentas as node()*, $newnodes as
node()*,$gautkelias as node()*) as item()*{
      let $source:=doc($dokumentas)
      return local:copy($source/node(),$newnodes,$gautkelias)
};
{-------------------------------------------------------------------------------------------------------------------}
```

4. *Move* operatoriaus funkcijos kodas.

```
{-------------------------------------------------------------------------------------------------------------------}
declare function local:index-of-node ($nodes as node()* ,$nodeToFind as node() )
as xs:integer* {

   for $seq in (1 to fn:count($nodes))
      return $seq[$nodes[$seq] is $nodeToFind]

 };
{-------------------------------------------------------------------------------------------------------------------}
declare function local:is-node-in-sequence-deep-equal ( $node as node()? ,$seq
as node()* )  as xs:boolean {
      some $nodeInSeq in $seq satisfies deep-equal($nodeInSeq,$node)
} ;
{-------------------------------------------------------------------------------------------------------------------}
declare function local:copy($element as element(),$aplinkybes as xs:string,
$newnodes as node()*,$gautkelias1 as node()*) as element() {
            element {node-name($element)}
              {$element/@*,
                    for $child in $element/node()
                          return if (local:is-node-in-sequence-deep-
equal($child,$newnodes)=true())
                                          then if (local:is-node-in-sequence-
deep-equal($element,$gautkelias1)=true())  then ( element {node-
name($child)}{$child/node()}) else ()
                                         else if (local:is-node-in-sequence-
deep-equal($child,$gautkelias1)=true() or local:is-node-in-sequence-deep-
equal($child/@*,$gautkelias1)=true())
                                                then if (local:is-node-in-
sequence-deep-equal($newnodes,$child/node())=true()) then
local:copy($child,$aplinkybes,$newnodes,$gautkelias1)
                                        else              if
($aplinkybes='into')
                                                                  then ( element
{node-name($child)} {$child/@*,insert-before($child/node(),local:index-of-
node($child/node(),$child/node()[last()])+1,$newnodes)})
                                                                  else if
($aplinkybes='following')
```

```
      then ( insert-before($child,local:index-of-
node($child,$child)+1,$newnodes))

      else ( insert-before($child,local:index-of-node($child,$child),$newnodes))
                                                  else  if ($child instance of
element())
                                                                        then
local:copy($child,$aplinkybes,$newnodes,$gautkelias1)
                                                                        else $child
             }
};
{-------------------------------------------------------------------------------------------------------------}
declare function local:transf_move($dokumentas as node()*, $aplinkybes as
xs:string, $iskur as node()*,$kelias as node()*) as item()*{
 let $source:=$dokumentas
       return local:copy($source/node(),$aplinkybes,$iskur,$kelias)
 };
{-------------------------------------------------------------------------------------------------------------}
```

5. „Catalog.xml“ XML dokumento kodas.

```
{-------------------------------------------------------------------------------------------------------------}

<catalog>
  <cd upc="602498678299">
     <artist>U2</artist>
     <title>How to Dismantle an Atomic Bomb</title>
     <price>13.98</price>
     <label>Interscope Records</label>
     <date>2004-11-23</date>
  </cd>
  <cd upc="75679244222">
     <artist>Led Zeppelin</artist>
     <title>Physical Graffiti</title>
     <price>22.99</price>
     <label>Atlantic</label>
     <date>1994-08-16</date>
  </cd>
  <cd upc="75678367229">
     <artist>Rush</artist>
     <title>Rush in Rio</title>
     <price>13.98</price>
     <label>Atlantic</label>
     <date>2003-10-21</date>
  </cd>
  <cd upc="74646938720">
     <artist>Billy Joel</artist>
     <title>Songs in the Attic</title>
     <price>10.99</price>
     <label>Sony</label>
     <date>1998-10-20</date>
  </cd>
  <cd upc="75678263927">
     <artist>Led Zeppelin</artist>
     <title>Houses of the Holy</title>
     <price>10.98</price>
     <label>Atlantic</label>
```

```
      <date>1994-07-19</date>
   </cd>
   <cd upc="8811160227">
      <artist>Jimi Hendrix</artist>
      <title>Are You Experienced?</title>
      <price>12.99</price>
      <label>Experience Hendrix</label>
      <date>1997-04-22</date>
   </cd>
</catalog>
```

{----------}

6. „Normalus.dtd“ DTD dokumento kodas.

{----------}

```
<!ENTITY AUTHOR "John Doe">
<!ENTITY COMPANY "JD Power Tools, Inc.">
<!ENTITY EMAIL "jd@jd-tools.com">

<!ELEMENT CATALOG (PRODUCT+)>

<!ELEMENT PRODUCT
(SPECIFICATIONS+,OPTIONS?,PRICE+,NOTES?)>
<!ATTLIST PRODUCT
NAME CDATA #IMPLIED
CATEGORY (HandTool|Table|Shop-Professional) "HandTool"
PARTNUM CDATA #IMPLIED
PLANT (Pittsburgh|Milwaukee|Chicago) "Chicago"
INVENTORY (InStock|Backordered|Discontinued) "InStock">

<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ATTLIST SPECIFICATIONS
WEIGHT CDATA #IMPLIED
POWER CDATA #IMPLIED>

<!ELEMENT OPTIONS (#PCDATA)>
<!ATTLIST OPTIONS
FINISH (Metal|Polished|Matte) "Matte"
ADAPTER (Included|Optional|NotApplicable) "Included"
CASE (HardShell|Soft|NotApplicable) "HardShell">

<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST PRICE
MSRP CDATA #IMPLIED
WHOLESALE CDATA #IMPLIED
STREET CDATA #IMPLIED
SHIPPING CDATA #IMPLIED>
```