KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Leonardas Garnionis

Daugiaklientinių bendradarbiavimo sistemų duomenų sinchronizacijos ir konfliktų sprendimo algoritmų taikymas XML dokumentams

Magistro darbas

Vadovas
dr. V. Pilkauskas
2006-05

Kaunas, 2006

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Leonardas Garnionis

Daugiaklientinių bendradarbiavimo sistemų duomenų sinchronizacijos ir konfliktų sprendimo algoritmų taikymas XML dokumentams

Magistro darbas

Kalbos konsultantė
I. Mickienė
2006-05

Vadovas
dr. V. Pilkauskas
2006-05

Recenzentas
doc. D. Rubliauskas
2006-05

Atliko
IFM-0/1 gr. stud.
Leonardas Garnionis
2006-05-15

Kaunas, 2006

Turinys

.

Application of Groupware System Data Synchronization and Conflict Resolution Algorithms for XML Documents

Summary

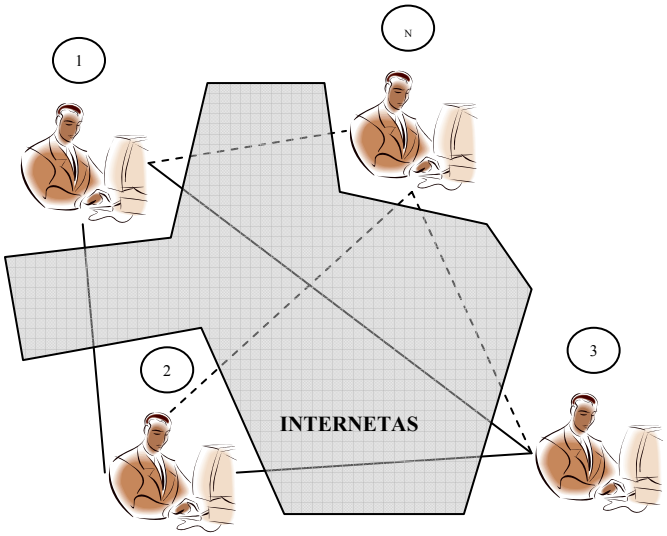
Real-time collaborative editing systems are groupware systems that allow members of a team to simultaneously edit shared documents from different sites. Shared objects involved in the team activity are subject to concurrent accesses and real-time constrains. Starting with the dOPT algorithm of Ellis and Gibbs various algorithms using operational transformation for maintaining consistency in collaborative systems have been proposed: adOPTed, GOT, GOTO and etc. All these algorithms are based on a linear representation of the document.

In this work we discuses about algorithms based on a tree representation of the document. We propose several algorithms based on dOPT algorithms for operation transformations for XML based documents. Experimental development helps to evaluate and demonstrate practical application for algorithms testing. Detailed commented experiment supplements formal description with practical strategies and specifics.

1. Įvadas

Magistro darbo tikslas - išanalizuoti grupinio bendradarbiavimo sistemų duomenų sinchronizacijos ir konfliktų sprendimo algoritmų veikimo principus. Pagal atliktą tyrimą sudaryti algoritmus XML dokumento grupiniam redagavimui.

Realaus laiko bendradarbiavimo sistema [3, 4, 5] skirta dokumentų redagavimui – tai grupinio darbo programinės įrangos visuma, kuri leidžia grupės nariams tuo pačiu metu redaguoti bendrus dokumentus iš skirtingų sistemos mazgų. Viena iš galimų, tokios sistemos, koncepcinė schema pateikta pirmame paveiksle.



1 pav. Sistemos koncepcinė schema

Kaip matyti, sistema yra paskirstyta tarp geografiškai nutolusių vartotojų. Vartotojų skaičius sistemoje teoriškai yra neribojamas. Visi vartotojai sistema naudojasi per internetinę prieigą. Bendri dokumentai yra replikuojami kiekvienam sistemos mazgui, todėl kiekvienas sistemos vartotojas turi pradinio dokumento kopiją, kurią jis gali redaguoti lokaliame sistemos mazge. Padalinti dokumentai, įtraukti į komandos veiklą, yra ryšių subjektai tarp konkuruojančių vartotojų ir realaus laiko. Realaus laiko aspektas lemia, kad kiekvienas vartotojas iš karto mato kiekvienos savo operacijos efektą. Tai pasiekiama iš karto vykdant lokaliai sugeneruotas operacijas lokaliai dokumento kopijai, ir kaip įmanoma greičiau atliekant kitų, nutolusių vartotojų operacijas. Įvykdžius lokalią operaciją ir norint užtikrinti greitą atsaką, ji tuoj pat turėtų būti išsiunčiama nutolusiems mazgams ir ten toliau vykdoma. Vienas svarbiausių realaus laiko bendradarbiavimo sistemos reikalavimų – aukštas konkurencijos lygis. Jis parodo, kad bet koks vartotojų kiekis, turi turėti galimybę tuo pačiu metu redaguoti bet kurią padalinto dokumento dalį. Žinodami apie tai, kiek įvykių sistemoje turi priežastingą priklausomybę nuo kitų įvykių, galime nustatyti konkurencingumo lygi skaičiavimuose. Visi

įvykiai kurie nėra priežastingai priklausomi nuo kitų įvykių, gali būti atliekami tuo pačiu metu.

Priežastingumas arba priežastingo pirmavimo ryšys tarp įvykių paskirstytose sistemose yra pagrindinis įrankis samprotaujant, analizuojant ir darant išvadas apie skaičiavimus. Žinios apie priežastingą priklausomybę tarp įvykių padeda spręsti įvairias problemas iškylančias projektuojant paskirstytas sistemas.

Paskirstytų skaičiavimų sistemos susideda iš aibės procesų, kurie bendradarbiauja ir varžosi tam, kad pasiektų bendrą tikslą. Šie procesai nesidalina bendra globalia atmintimi ir bendrauja išimtinai tik pranešimais, kurie siunčiami per bendravimo tinklą. Komunikacijos tinklo delsimo laikas yra baigtinis, bet nenusakomas. Proceso veiksmai yra skirstomi į tris tipus: vidiniai įvykiai, pranešimo siuntimo, pranešimo priėmimo. Vidiniai įvykiai įtakoja tik ta procesą, kuriame jie yra išgeneruojami. Vidiniai įvykiai yra rikiuojami į eilę pagal jų įvykimo laiką. Pranešimo siuntimo ir pranešimo priėmimo įvykiai turi reikšmės informacijos srautams tarp procesų. Išoriniai įvykiai sukuria priežastingą priklausomybę tarp siuntėjo ir gavėjo procesų.

Žmonių bendradarbiavimas reikalauja padalinto dokumento kopijų, logiškumo išlaikymo, kuris yra kitoks nei tradicinėse duomenų bazėse, ar paskirstytose kompiuterinėse sistemose. Pirma, žmogus – vartotojas turi galimybę spręsti kai kurias logiškumo išlaikymo problemas, išvengiant pagrindinių sistemos apribojimų. Antra, žmogus – vartotojas gali turėti specialių, sąveikos ir bendradarbiavimo ryšių nevaržomų, poreikių. Šie poreikiai gali turėti netradicinių logiškumo reikalavimų, kurie nėra numatyti pagrindinėje sistemoje.

Šiame darbe bus apžvelgti grupinio bendradarbiavimo sistemų veikimo principai, struktūrą, charakteristikos, iškylančias projektavimo problemas. Bus aptariama, kaip grupinio bendradarbiavimo sistemų pagrindinės problemos yra sprendžiamos operacijų transformavimo algoritmų pagalba (apie tai rašoma 2, 3 ir 4 skyriuose). Nuo linijinių dokumentų, tokių kaip tekstas, bus pereinama prie medinės struktūros dokumentų HTML[1] ir bendresnio atvejo – XML[2] (apie tai rašoma 5 ir 6 skyriuose).

2. Redagavimo operacijos

Kiekvienas sistemos vartotojas redaguodamas savo lokalią kopiją, generuoja operacijas ir jas vykdo. Kad padalinto dokumento kopijai būtų įvykdyta operacija, ji pirmiausia turi būti sugeneruota. Lokaliai sugeneruotos operacijos (lokalios operacijos) yra iš karto įvykdomos ir išsiunčiamos kitiems sistemos mazgams. Operacijos priimtos iš kitų sistemos mazgų, vadinamos *nutolusiomis*. Priimta nutolusi operacija turi būti įvykdyta lokaliai padalinto dokumento kopijai.

2.1. Ryšys tarp operacijų

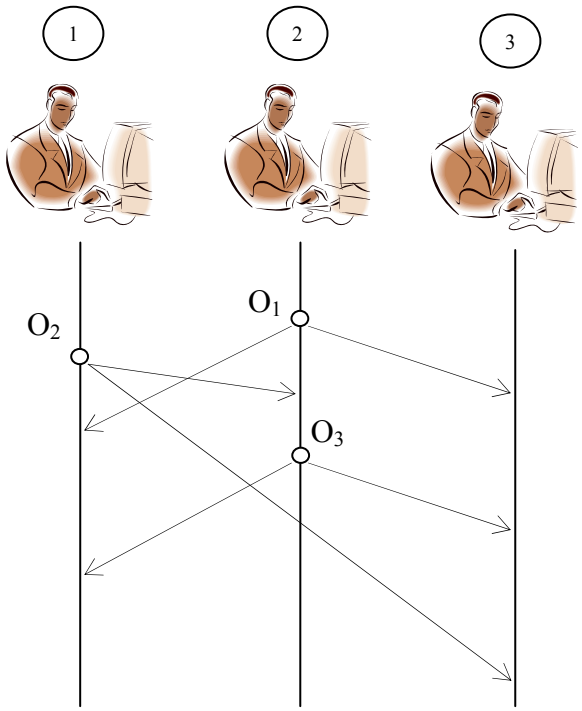
Priežastingo sąlygotumo (kauzalumo) tvarkos sąryšis. Ši operacija žymima simboliu „$\rightarrow$“. Operacijų kauzalumu nusakyti išskiriami trys atvejai. Sakykime, bendru atveju turime dvi operacijas O_a ir O_b sugeneruotas sistemos mazguose i ir j, tai $O_a \rightarrow O_b$ kai:

- $i = j$, t.y. operacijos atliktos tame pačiame sistemos mazge, ir O_a operacijos generacija buvo atlikta prieš O_b generaciją;
- $i \neq j$, t.y. operacijos atliktos skirtinguose sistemos mazguose, ir O_a operacijos įvykdymas buvo atliktas prieš O_b generaciją;
- egzistuoja tokia operacija O_x ,su kuria $O_a \rightarrow O_x$ ir $O_x \rightarrow O_b$.

Tokios operacijos yra viena nuo kitos priklausomos ir todėl reikia laikytis jų atlikimo tvarkos.

Konkuruojančių operacijų sąryšis. Operacija žymima simboliu „$||$“. Dvi operacijos - O_a ir O_b yra konkuruojančios, kai $O_a \rightarrow O_b$ rezultatas nėra lygus $O_b \rightarrow O_a$ rezultatui, t. y. $O_a \,||\, O_b \Rightarrow \neg((O_a \rightarrow O_b) \vee (O_b \rightarrow O_a))$. Tokios operacijos yra viena nuo kitos nepriklausomos ir gali būti vykdomos bet kokia tvarka skirtinguose sistemos mazguose.

Žemiau pateiktame pavyzdyje, nurodoma sistemoje sugeneruotų operacijų galima vykdymo tvarka. Reikėtų prisiminti, kad lokaliai sugeneruota operacija yra iš karto vykdoma.



2 pav. Operacijų generacija ir įvykdymas

Šioje situacijoje sistemą sudaro trys mazgai. Pirmame mazge sugeneruojama operacija O_2, antrame sugeneruojamos operacijos O_1 ir O_3. Šioms operacijoms galima parašyti šias išraiškas:

- $O_1 \| O_2$. Operacija O_1 ir operacija O_2 yra konkuruojančios, nes sąryšio $O_1 \to O_2$ rezultatas nėra lygus sąryšio $O_2 \to O_1$ rezultatui.
- $O_1 \to O_3$. Operacija O_1 priežastingai lemia operacijos O_3 vykdymo rezultatą.
- $O_2 \to O_3$. Kaip ir ankstesniame pavyzdyje, operacija O_2 priežastingai lemia operacijos O_3 vykdymo rezultatą.

2.2. Logiškumo (konsistencijos) išlaikymas

Logiškumo išlaikymas grupinio redagavimo sistemose – svarbiausias uždavinys. Sistema garantuoja logiškumą, jeigu ji palaiko šias savybes: priežastingo sąlygotumo išlaikymą, konvergenciją, vartotojo intencijos išlaikymą.

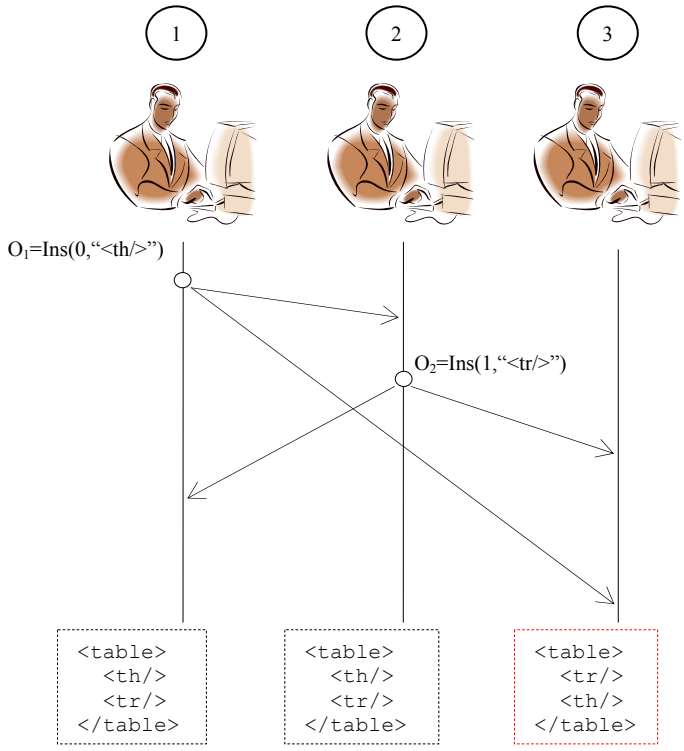
2.3. Priežastingas sąlygotumas

Jei $O_a \to O_b$, tai operacija O_a atliekama prieš operaciją O_b visuose sistemos mazguose. Žemiau pateiktame pavyzdyje nurodomas priežastingo sąlygotumo neišlaikymo atvejis ir parodyta kaip ši problema yra sprendžiama.

Sakykime, duotas pradinis dokumentas

```
<table>
</table>
```

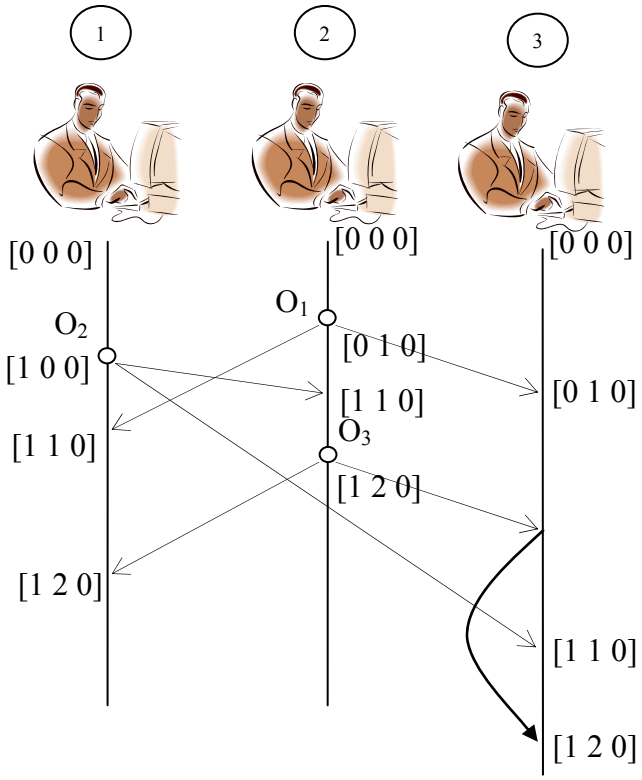
kurį redaguoja trys vartotojai (visos operacijos atliekamos mazgo <table> viduje).



3 pav. Priežastingo sąlygotumo neišlaikymas

Atlikus operacijas O_1 ir O_2 pirmajame ir antrajame sistemos mazguose, galima matyti, kad šiuose sistemos mazguose galutiniai dokumentai yra teisingi, o trečiame mazge – neteisingas. Toks rezultatas yra todėl, kad trečiajame sistemos mazge nėra išlaikytas operacijų O_1 ir O_2 kauzalumas. Šiuo atveju trečiajame mazge pirmiau buvo įvykdyta operacija O_2, o tik po to – O_1. Norint išlaikyti kauzalumą, pirmiausia reikėtų vykdyti operaciją O_1, o po to operaciją O_2.

4 paveiksle naudojant vektorinio laikrodžio mechanizmą pateiktas vienas priežąstingto sąlygotumo išlaikymo pavyzdžių. Apie loginių laikrodžių sistemų tipus bus kalbama „4. Loginis laikas“ skyriuje.



4 pav. Priežastingo sąlygotumo išlaikymas panaudojant vektorinį laikrodį

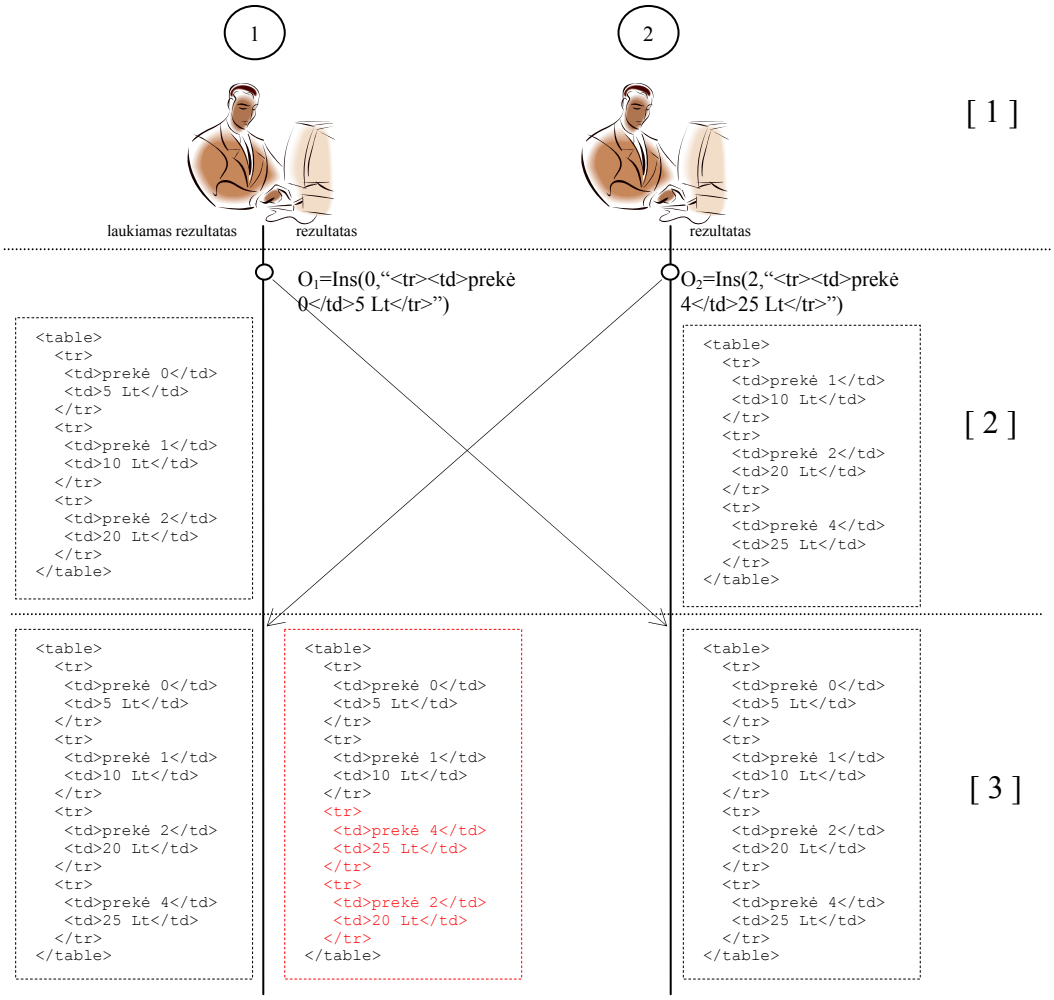
2.4. Konvergencija

Kopijų konvergencija [6] nusako, kad visuose sistemos mazguose atlikus tą patį operacijų rinkinį, dokumentų kopijos turi būti identiškos.

Sakykime, duotas pradinis dokumentas,

```
<table>
  <tr>
    <td>prekė 1</td>
    <td>10 Lt</td>
  </tr>
  <tr>
    <td>prekė 2</td>
    <td>20 Lt</td>
  </tr>
</table>
```

kurį redaguoja du vartotojai. Jie pildo lentelę prekių duomenimis, norėdami prekes išrikiuoti kainos didėjimo tvarka (visos operacijos atliekamos mazgo <table> viduje).



5 pav. Konvergencijos neišlaikymas

[1] Pradinio dokumento kopijos abiejuose sistemos mazguose yra vienodos.

[2] Norint užtikrinti aukštą konkurencijos ir atsako lygį, padalinto dokumento kopijos, po įvykdytų operacijų, gali būti skirtingos.

[3] Dokumentų kopijos turi konvertuoti po tų pačių operacijų sekos įvykdymo.

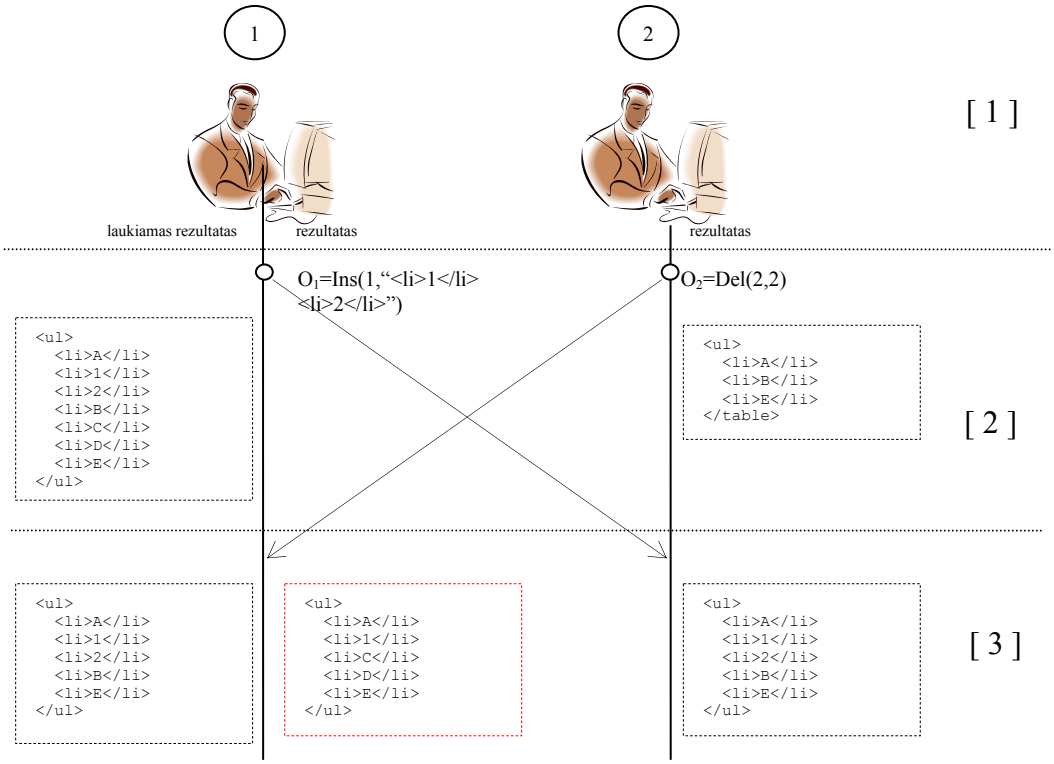
2.5. Vartotojo intencija

Rezultatas, atlikus operaciją O bet kokiam dokumentui sistemoje, turi būti toks pat kaip ir rezultatas, gautas tame sistemos mazge, kuriame buvo sugeneruota operacija O.

Sakykime, duotas pradinis dokumentas

```
<ul>
  <li>A</li>
  <li>B</li>
  <li>C</li>
  <li>D</li>
  <li>E</li>
</ul>
```

kurį redaguoja du vartotojai. (Visos operacijos atliekamos mazgo <ul> viduje).



6 pav. Vartotojo intencijos neišlaikymas

[1] Pradinio dokumento kopijos abiejuose sistemos mazguose yra vienodos.

[2] Operacijos O_1 intencija – įterpti du mazgus pirmoje pozicijoje. Operacijos O_2 intencija – ištrinti du mazgus pradedant antrąją pozicija. Operacijos O_2 intencija lokaliame sistemos mazge yra išsaugoma, nes operacija yra iš karto įvykdoma.

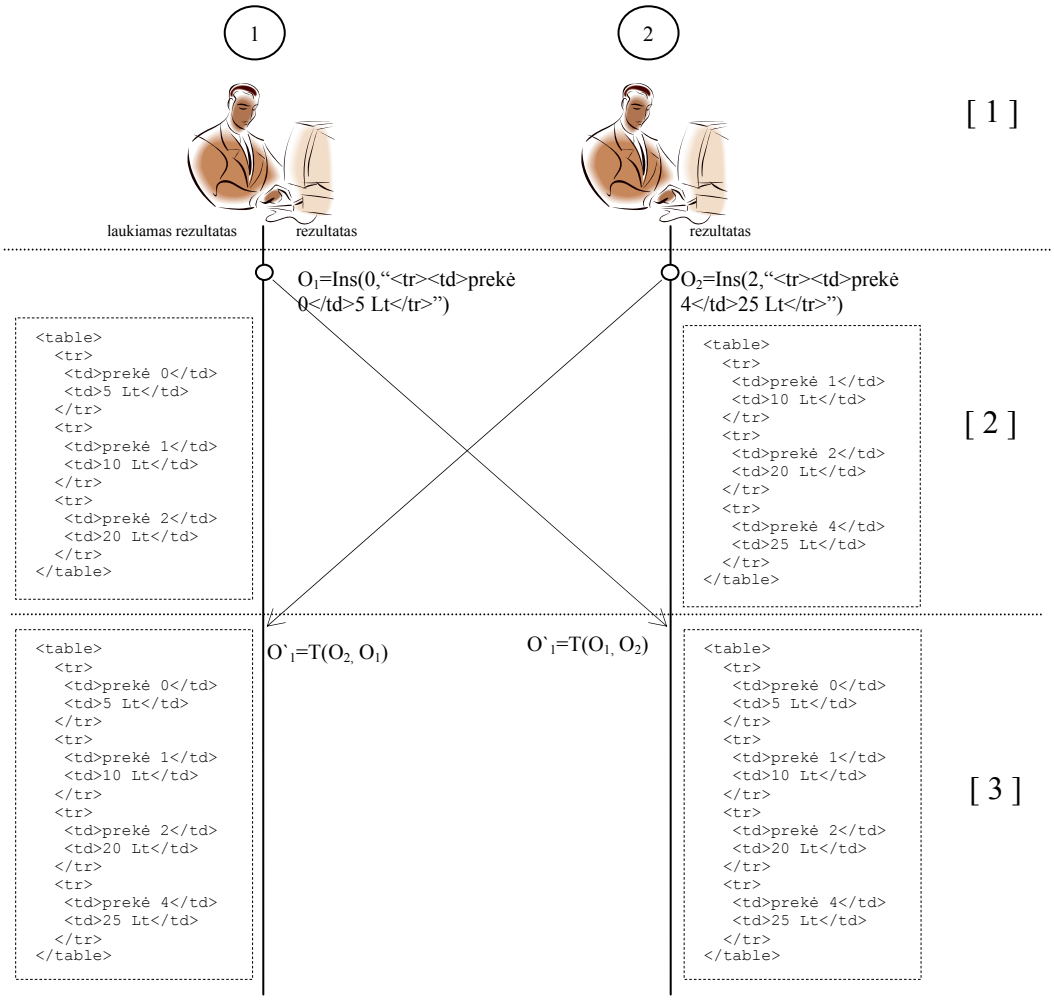
[3] Operacijos O_2 intencija yra pažeidžiama, kuomet ši operacija yra įvykdoma nutolusiame dokumente.

3. Operacijų transformavimas

Duotos dvi konkuruojančios operacijos O_1 ir O_2. Sukurkime operacijas transformuojančią taisyklę T. Jei $O_1' = T(O_1, O_2)$ ir $O_2' = T(O_2, O_1)$, tai $[O_1 \circ O_2'] \equiv [O_2 \circ O_1']$. Operacijų transformaciją iliustruoja pavyzdys. Sakykime, duotas pradinis dokumentas

```
<table>
  <tr>
    <td>prekė 1</td>
    <td>10 Lt</td>
  </tr>
  <tr>
    <td>prekė 2</td>
    <td>20 Lt</td>
  </tr>
</table>
```

kurį redaguoja du vartotojai. Jie pildo lentelę prekėmis, norėdami prekes išrikiuoti kainos didėjimo tvarka. (Visos operacijos atliekamos mazgo <table> viduje).



7 pav. Operacijų transformacija.

[1] Pradinio dokumento kopijos abiejuose sistemos mazguose yra vienodos.

[2] Vietinės operacijos dokumento kopijai yra vykdomos iš karto ir nėra transformuojamos.

[3] Nutolusios konkuruojančios transformacijos prieš vykdymą yra transformuojamos.

Operacijų transformacijos pagalba galima pasiekti dokumento kopijų konvergenciją ir išlaikyti vartotojų intenciją. Operacijų transformavimas neriboja vartotojų veiksmų, tokių būdu vartotojai gali bet kokį dokumento objektą redaguoti bet kuriuo laiku.

3.1. Bendra OT algoritmų struktūra

Visiems operacijų transformavimo algoritmams būdinga tokia struktūra:

- Valdymo algoritmas atskirtas nuo transformacijų funkcijų. Operacijų transformavimo funkcijos yra orientuojamos į sritį, t.y. priklauso nuo sistemos pobūdžio su kokiais dokumentų tipais ji operuoja. Jos pritaikomos operuoti eilutėmis, sąrašais, sekomis, medžiais ir t.t. Ir tai nedaro įtakos kontrolės algoritmams.
- Valdymo algoritmas yra visada bendras ir sudarytas iš užklausų eilės ir istorijos buferio.

3.2. dOPT algoritmas

dOPT (paskirstytas operacijų transformavimo) algoritmas pirmą kartą buvo pristatytas 1989 metais. dOPT algoritmas operuoja redaguojamojo dokumentuoto kopijomis visuose sistemos mazguose. Pradiniu momentu visos kopijos yra vienodos. Kiekvienas mazgas veikia to paties algoritmo principu. Šis algoritmas apdoroja pranešimus, kurie apibūdina šiuos įvykius:

- sugeneruota lokali redagavimo operacija (sistemos mazge S);
- sugeneruota nutolusi redagavimo operacija kitame sistemos mazge $s \neq S$.

Kiekvienai lokaliai redagavimo operacijai algoritmas modifikuoja lokalią dokumento kopiją ir perduoda pranešimą apie atliktą operaciją likusiems sistemos mazgams. Kiekviena nutolusi redagavimo operacija yra užlaikoma iki to momento, kol yra įvykdomos visos operacijos, atliktos iki jos. Tuomet užlaikyta redagavimo operacija yra transformuojama. Operacijos transformacija kompensuoja operacijų, atliktų po jos, pakeitimus. Galiausiai algoritmas paredaguoja lokalią dokumento kopiją, atvaizduodamas transformuotas operacijas.

dOPT algoritme redaguojamas objektas yra kintamo ilgio simbolių masyvas.

Yra trys redagavimo operacijos:

- insert[c, i] – įterpia simbolį c į poziciją i. Nuo šios pozicijos visi simboliai perkeliami viena pozicija į dešinę.

- delete[i] – ištrina simbolį pozicijoje *i*. Nuo šios pozicijos visi simboliai perkeliami viena pozicija į kairę.
- nop – dokumentas nėra modifikuojamas.

3.3. dOPT transformacijų funkcijos

Operacijų transformacijos reikalingos tada, kai dviejuose skirtinguose sistemos mazguose atitinkamai sugeneruojamos tokių rūšių operacijos:

- įterpti – įterpti;
- įterpti – ištrinti;
- ištrinti – įterpti;
- ištrinti – ištrinti.

Dabar aptariamas kiekvienas atvejis atskirai, kur *p1* ir *p2* – pozicijos kuriose operuoja vartotojai, *c1* ir *c2* – objektai, su kuriais operuoja vartotojai, *u1* ir *u2* – vartotojų identifikatoriai, *I* – sutampanti operacija, *Ins[p, c]* – įterpimo operacija, *Del[p]* trynimo operacija. Kiekviena operacijų transformavimo funkcija turi du parametrus. Pirmas parametras – tai nutolusi operacija, antras – lokali operacija. Nutolusi operacija pagal atliktos vietinės operacijos poziciją yra atitinkamai transformuojama.

Įterpti – įterpti atvejis. Šiuo atveju du sistemos vartojai terpia po simbolį. Operacijų transformacijos funkcija *TII* (transform insert – insert) tikrina, kurioje dokumento vietoje terpiami simboliai. Nutolusi operacija nebus transformuojama tada, kai pozicija yra arčiau dokumento pradžios, operacijų pozicijos sutampa ir nutolusio vartotojo identifikatorius mažesnis už lokalaus vartotojo identifikatorių.

TII operacijos tekstas:

```
TII( Ins[p1,c1], Ins[p2,c2] )
{
    if  p1 < p2 or ( p1 = p2 and u1 < u2 )
        return Ins[p1, c1];
    else
        return Ins[p1+1, c1];
}
```

Įterpti – ištrinti atvejis. Šiuo atveju vienas sistemos vartotojas terpia objektą į dokumentą, antrasis – trina. *TID* (transform insert – delete) operacijų transformavimo funkcija atlieka šiuos veiksmus: jei terpimo operacija vyksta prieš trynimo operaciją – tai operacija

nėra transformuojama, jei trynimas vyksta toje pačioje pozicijoje arba artimesnėje dokumento pradžiai, tuomet įterpimo operacija turi būti transformuojama.

TID operacijos tekstas:

```
TID( Ins[p1,c1], Del[p2] )
{
    if p1 < p2
        return Ins[p1, c1];
    else
        return Ins[p1-1,c1];
}
```

Ištrinti – įterpti atvejis. Šiuo atveju vienas sistemos vartotojas trina objektą iš dokumento, antrasis – įterpia. *TDI* (transform delete – insert) operacijų transformavimo funkcija atlieka šiuos veiksmus: jei nutolusi operacija yra atliekama arčiau dokumento pradžios nei lokali įterpimo operacija, tuomet ji nėra transformuojama, jei nutolusi operacija yra atliekama toliau terpimo operacijos, tai tuo atveju ją reikia transformuoti.

TDI operacijos tekstas:

```
TDI( Del[p1], Ins[p2,c2] )
{
    if p1 < p2
        return Del[p1];
    else
        return Del[p1+1];
}
```

Ištrinti – ištrinti atvejis. Šiuo atveju abu sistemos vartotojai trina dokumento objektus. *TDD* (transform delete – delete) operacijų transformavimo funkcija atlieka šiuos veiksmus: jei nutolusi operacija atliekama arčiau dokumento pradžios, operacija nėra transformuojama, jei lokali operacija atliekama arčiau dokumento pradžios, tuomet nutolusi operacija yra transformuoja, kai pozicijos sutampa – grąžinama vienetinė operacija.

TDD operacijos tekstas:

```
TDD( Del[p1], Del[p2] )
{
    if p1 < p2
        return Del[p1];
```

```
        else if p1 > p2
            return Del[p1-1];
        else
            return I;
    }
```

dOPT pagrindu yra sukurtas treeOPT [12] algoritmų visuma skirta medinės struktūros dokumentų grupiniam redagavimui. Šie algoritmai veikia dOPT principu, rekursyviai kiekvienam dokumento lygiui.

4. Loginis laikas

Paskirstytose sistemose visi veiksmai trunka tam tikrą laiko tarpą. Įvykių kauzalumo koncepcija yra fundamentalumas projektuojant ir analizuojant paskirstytas sistemas. Laiko tėkmė yra pagrindas kauzalumo kontrolės įrankis tarp įvykių. Paskirstytose sistemose nėra realizuotas fizinis laikas. Fizinį laiką paskirstytose sistemose galima realizuoti tik jį aproksimavus. Paskirstytose sistemose skaičiavimai atliekami etapais ir daro įtaką loginiam laikui, kuris vyksta periodais.

4.1. Loginio laikrodžio sistema

Loginių laikrodžių sistema susideda iš laiko domeno T ir loginio laikrodžio C. Aibės T elementai iš dalies yra išdėlioti „mažiau“ (<) sąryšiu. Sąryšis „mažiau“ dažniausiai vadinamas „atsitikęs prieš“, arba priežastingu pirmumu. Šis sąryšis panašus į „anksčiau nei“ sąryšį, apibūdinantį fizikinį laiką. Loginis laikas C yra funkcija, kuri vaizduoja įvykį paskirstytos sistemos elementui. Ši funkcija žymima $C(e)$ ir vadinama laiko žyme įvykiui e, laiko T srityje:

$$C : H \mapsto T$$

ir atitinka:

$$e_1 \to e_2 \Rightarrow C(e_1) < C(e_2)$$

Ši eilės tipo išraiška yra vadinama *laiko nuoseklumo sąlyga*. Kai T ir C tenkina šią sąlygą:

$$e_1 \to e_2 \Leftrightarrow C(e_1) < C(e_2)$$

tuomet galima sakyti, kad laikrodžių sistema yra griežtai nuosekli.

4.2. Realizacija

Kiekvienas procesas p_i operuoja su duomenų struktūromis. Šios duomenų struktūros turi tenkinti tokias sąlygas:

- lokalus loginis laikrodis, žymimas lc_i, kuris padeda procesui p_i matuoti savo progresą;
- globalus loginis laikrodis, žymimas gc_i, kuris vaizduoja proceso p_i lokalų laiką į globalų laiką. Tai padeda procesui priskirti nuoseklias laiko žymes savo lokaliems įvykiams. Dažniausiai lc_i yra gc_i dalis.

Protokolas, kuris užtikrina, kad procesų loginis laikas ir jo vaizdavimas į globalų laiką yra valdomas atitinkamai, turi atitikti šias dvi $R1$ ir $R2$ taisykles:

- $R1$ lemia, kaip procesas atnaujina lokalų loginį laikrodį (seka jo progresą) kai yra vykdomas įvykis, siuntimo, gavimo arba vidinės operacijos metu.
- $R2$ lemia, kaip procesas atnaujina savo globalų loginį laiką, naujinant vaizdavimą globaliu laiku ir globaliu progresu. Ši taisyklė nustato, kuri informacija apie loginį laiką priskiriama pranešimams ir, kaip priimantis procesas vartoja tą atnaujintą informaciją save atvaizduodamas globaliu laiku.

Loginio laiko sistemose kiekvienas procesas turi loginį laikrodį, kurio progresas yra valdomas abiejų taisyklių $R1$ ir $R2$. Kiekvienam įvykiui yra priskirta laiko žyma ir priežastingumo ryšiai tarp įvykių gali būti numatyti pagal jų laiko atžymas. Laiko atžymos, priskirtos kiekvienam įvykiui atitinka fundamentalią monotonišką savybę. Jei įvykis a yra priežastingas įvykiui b, tai laiko žymė įvykiui a yra mažesnė nei įvykiui b.

4.3. Skaliarinis laikas

1978 metais Lamportas [6] pristatė skaliarinį laiką, kuris vaizduoja absoliučiai išrikiuotus įvykius paskirstytose sistemose. Šiame metode laiko domenas yra sveikųjų neneigiamų skaičių aibė. Lokalus loginis proceso p_i laikas ir jo lokalus globalaus laiko atvaizdas yra suspausti į vieną sveikąjį skaičių C_i.

Taisyklės $R1$ ir $R2$ kurios atnaujina laikrodžius.

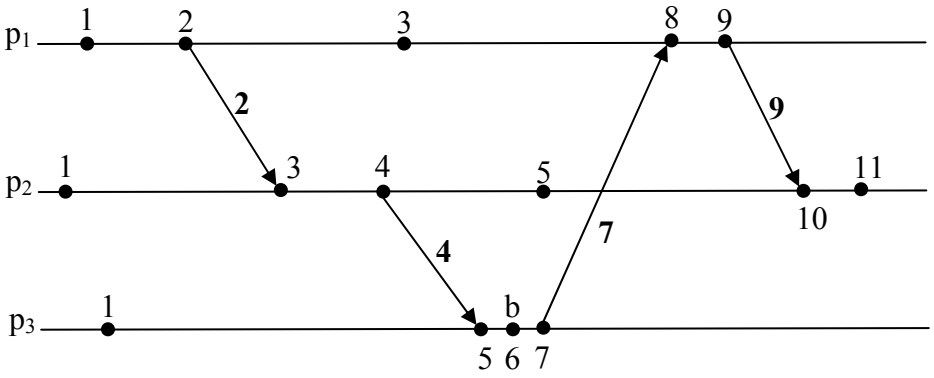
- $R1$: prieš įvykdant įvykį (siuntimo, priėmimo, arba vidinį) procesas p_i atlieka tokį veiksmą:

  $C_i = C_i + d$, kur $(d > 0)$

- $R2$: kiekvienas pranešimas neša laiko reikšmę iš savo siuntėjo išsiuntimo metu. Kai procesas p_i priima pranešimą su laiko žyme C_{msg}, jis atlieka tokius veiksmus:

  1. $C_i = \max(C_i, C_{msg})$,
  2. įvykdo $R1$,
  3. pristato pranešimą.

8 paveiksle pademonstruotas skaliarinio laiko progresas, kai d yra lygus 1.



8 pav. Skaliarinio laiko progresas, kai d = 1

4.5. Vektorinis laikas

Fidge [7], Mattern [8], Schmuck [9], atskirai vienas nuo kito, sukūrė vektorinio laiko mechanizmą. Vektorinio laiko sistemoje laiko domenas T vaizduojamas kaip n-matis masyvas, neneigiamų sveikųjų skaičių. Kiekvienas procesas p_i operuoja vektoriumi $vt_i[1..n]$, kur $vt_i[i]$ yra lokalus proceso p_i loginis laikrodis ir aprašo loginio laiko progresą procese p_i. $vt_i[j]$ aprašo paskutines proceso p_i žinias apie proceso p_j lokalų laiką. Jei $vt_i[j] = x$, procesas p_i žino, kad lokalus laikas procese p_j progresavo iki x reikšmės. Vektorius vt_i sudaro proceso p_i globalaus laiko vaizdą. Procesas p_i vartoja šį vektorių įvykių žymėjimui. Taisyklės $R1$ ir $R2$ kurios atnaujina laikrodžius.

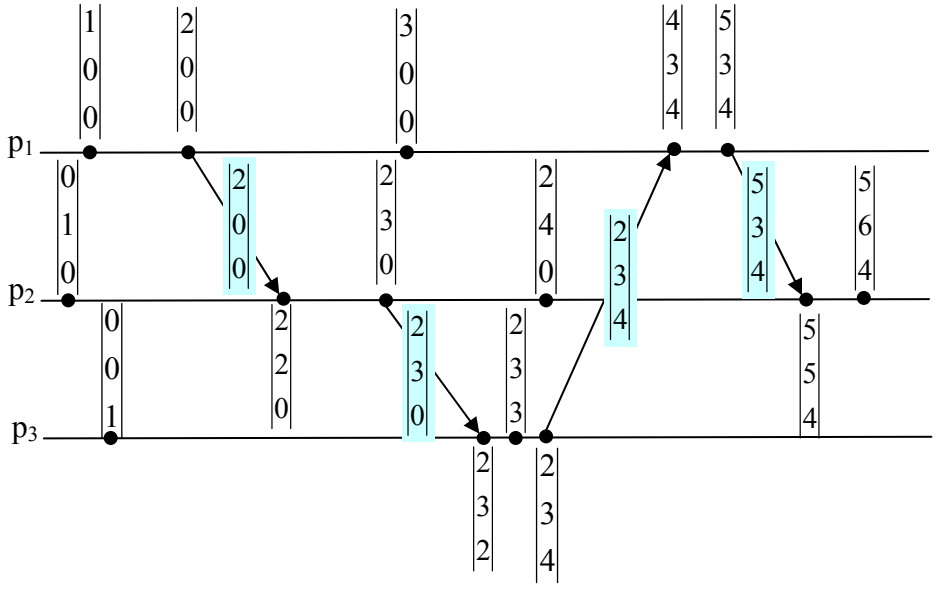
- $R1$: prieš vykdant įvykį (siuntimo, priėmimo, arba vidinį) procesas pi atlieka tokį veiksmą:

  $vt_i[i] = vt_i[i] + d$, kur $(d > 0)$

- $R2$: kiekvienas pranešimas m neša vektorinio laiko reikšmę iš savo siuntėjo išsiuntimo metu. Kai procesas p_i priima pranešimą su vektorinio laiko žyme (m, vt), jis atlieka tokius veiksmus:

  1. $1 \le k \le n$ 1 : $vt_i[k] = \max(vt_i[k], vt[k])$,
  2. įvykdo $R1$,
  3. pristato pranešimą m.

9 paveiksle pademonstruotas vektorinio laiko progresas, kai d yra lygus 1.



9 pav. Vektorinio laiko progresas, kai d = 1

Vektorinio laiko izomorfizmas

Izomorfizmas – dviejų aibių su vienodu operacijų skaičiumi elementų abipusiškai vienareikšmė atitiktis, išlaikanti sandaros savybes.

Vektorinių laikų vk ir vh palyginimas:

$$vh \le vk \Leftrightarrow \forall x : vh[x] \le vk[x]$$

$$vh < vk \Leftrightarrow (vh \le vk) \vee (\exists x : vh[x] < vk[x])$$

$$vh \,||\, vk \Leftrightarrow \neg(vh < vk) \vee \neg(vk < vh)$$

Jei įvykiai paskirstytoje sistemoje yra pažymėti laiko žymėmis naudojant vektorinio laiko sistemą, mes turime tokią priklausomybę:

Jei du įvykiai x ir y turi laiko žymes vh ir vk, tai:

$$x \to y \Leftrightarrow vh < vk$$

$$x \,||\, y \Leftrightarrow vh \,||\, vk\ .$$

Taip čia yra izomorfizmas tarp aibės dalinai išrikiuotų įvykių išgeneruotų paskirstytų skaičiavimų ir jų laiko žymių. Jei procesai išgeneravę įvykius yra žinomi, tuomet vektorinių laiko žymų palyginimas gali būti supaprastintas:

Jei įvykiai x ir y atitinkamai yra išgeneruoti procesuose p_i ir p_j, o jiems priskirtos laiko žymės (vh,i) ir (vh, j), tai turime:

$$x \to y \Leftrightarrow vh[i] < vk[i]$$

$$x \,||\, y \Leftrightarrow (vh[i] > vk[i]) \vee (vh[j] > vk[j]).$$

4.5. Matricinis laikas

Matricinio laiko sistemose laikas yra vaizduojamas kaip aibė nxn matricų, kurių elementus sudaro neneigiami sveikieji skaičiai. Procesas p_i aptarnauja $mt_i[1..n,1..n]$ matricą kur:

- $mt_i[i,i]$ nusako proceso p_i lokalų loginį laiką ir proceso pi atliekamų veiksmų progresą,
- $mt_i[k,i]$ nusako proceso p_i žinias apie proceso p_k žinias apie proceso p_l lokalų loginį laiką. mt_i matricą nusako proceso p_i lokalų, globalaus loginio laiko, vaizdą.

Eilutė $mt_i[i,]$ neturi jokios informacijos, bet vektorinis laikrodis vt_i nusako viską apie vektorinius laikrodžius. procesas p_i naudoja šias taisykles $R1$ ir $R2$ savo laikrodžio atnaujinimui:

- $R1$: prieš įvykdant įvykį, jis atnaujina lokalų loginį laiką:

  $mt_i[i,i] = mt_i[i,i] + d$, kur $(d > 0)$
- $R2$: kiekvienas pranešimas m neša matricinio laiko reikšmę iš savo siuntėjo išsiuntimo metu mt. Kuomet procesas p_i priima pranešimą su matricinio laiko žyme (m, mt) iš proceso p_j, jis atlieka sekančius veiksmus
  1. $1 \le k \le n$: $mt_i[i,k] = \max(mt_i[i,k], mt[j,k])$

     $1 \le k,l \le n$: $mt_i[i,k] = \max(mt_i[k,l], mt[k,l])$
  2. įvykdo $R1$.

  pristato pranešimą.

5. Medinė duomenų struktūra

Visi aukštesnio lygio teksto redaktoriai paremti medinės struktūros modeliu. Dokumentai, kuriais tokie redaktoriai operuoja, turi medžio struktūrą. Būtent HMTL [1], XML [2], XHTML [10], SGML [11] dokumentai yra medžio struktūros. Trimatės erdvės bendradarbiavimo sistemose, tokiose kaip žaidimai, stimuliatoriai, treniruokliai, nutolusios operacijos pavojingose aplinkose, scenų vaizdavimui naudojamos medinės struktūros duomenų modelis.

Seka ir medis yra bazinės duomenų struktūros nariai. Pavyzdžiui, objektai grafiniuose redaktoriuose ir kompiuterinio projektavimo įrankiuose, B – medžiai ir R – medžiai duomenų bazių sistemose.

5.2. Prielaidos

Suformuluojamos prielaidos grupinio bendradarbiavimo sistemai, kuri manipuliuoja su medinių struktūrų dokumentais. Paskirstytų procesų aibė manipuliuoja su tuo pačiu medžiu ar su medžių aibe. Duomenys kopijuojami, taip paslepiamas komunikacijų tinklo gaišaties laikas.

Sutartinę duomenų medžio struktūra:

- Kiekviena šaka gali turėti bet kokį skaičių vaikų.
- Kiekviena šaka gali turėti aibę savybių, kurios aprašomos šiuo dvejetu: $\{\langle raktas, reikšmė\rangle\}$.
- Neatsižvelgsime į medžių ir šakų semantiką.

5.3. XML, HTML ir XHTML

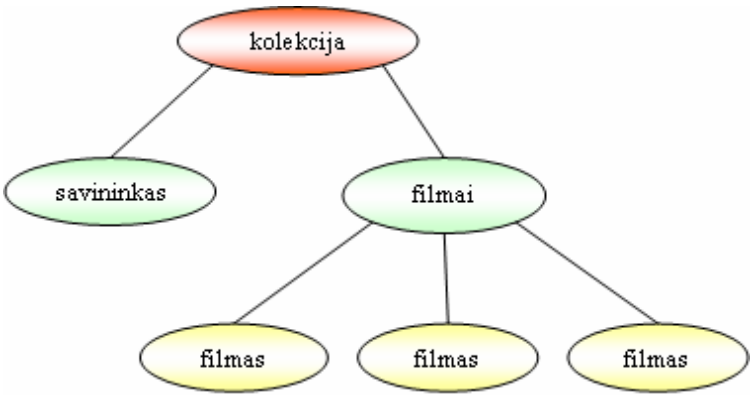
XML dokumentai yra medinės struktūros ir jais remiantis galima lengvai aprašyti struktūrizuotus duomenis. HTML dokumentus galime laikyti daliniu XML dokumento atveju. Norint HTML dokumentą parašyti kaip taisyklingą XML dokumentą, – reikia įvesti tam tikrus patikslinimus. HTML dokumentas leidžia kai kuriuos mazgus aprašinėti be uždarančios dalies. Pavyzdžiui <*img*>, <*br*> mazgus pagal XML taisykles reikia rašyti su užbaigimu: <*img* />, <*br* />. HTML dokumento rašymo kaip XML dokumento taisyklės yra apibrėžtos XHTML standarte.

Dabar reikėtų palyginti, kaip duomenys gali būti aprašomi XML dokumentu, ir taip pat pateikti kaip tie patys rezultatai gali būti atvaizduoti HTML pagalba. Pademonstruosime XML ir HTML dokumentų medinę struktūrą.

Sakykime turime video filmų kolekciją kuri priklauso konkrečiam savininkui. Tada šiuos duomenis galime atvaizduoti tokia XML dokumento struktūra:

```
<kolekcija>
  <savininkas>Antanas</savininkas>
  <filmai>
    <filmas garsas="ne" metai="1925">Operos vaiduoklis</filmas>
    <filmas garsas="taip" metai="1989">Operos vaiduoklis</filmas>
    <filmas garsas="taip" metai="1948">Virvė</filmas>
  </filmai>
</kolekcija>
```

Medinė, aukščiau pateikto, XML dokumento struktūra pateikta 10 paveiksle.

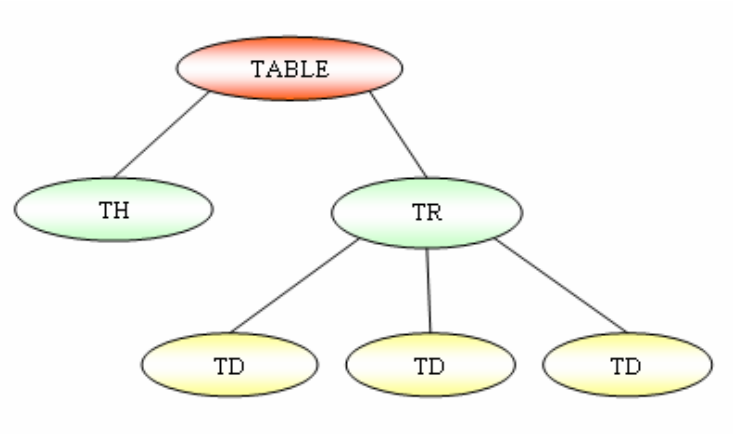


10 pav. Xml dokumentas.

Tuos pačius duomenis prašysime HTML dokumento forma:

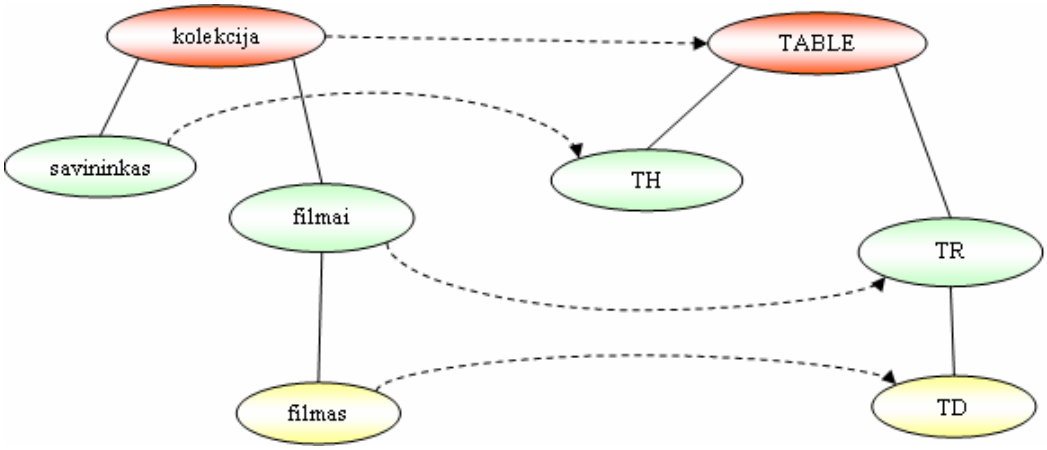
```
<TABLE>
     <TH colspan=3>Kolekcija. Savininkas Antanas </TH>
     <TR>
      <TD>Operos vaiduoklis</TD>
      <TD>Operos vaiduoklis</TD>
      <TD>Virvė</TD>
    </TR>
</TABLE>
```

Medinė, aukščiau pateikto, HTML dokumento struktūra pateikta 11 paveiksle.



11 pav. HTML dokumentas.

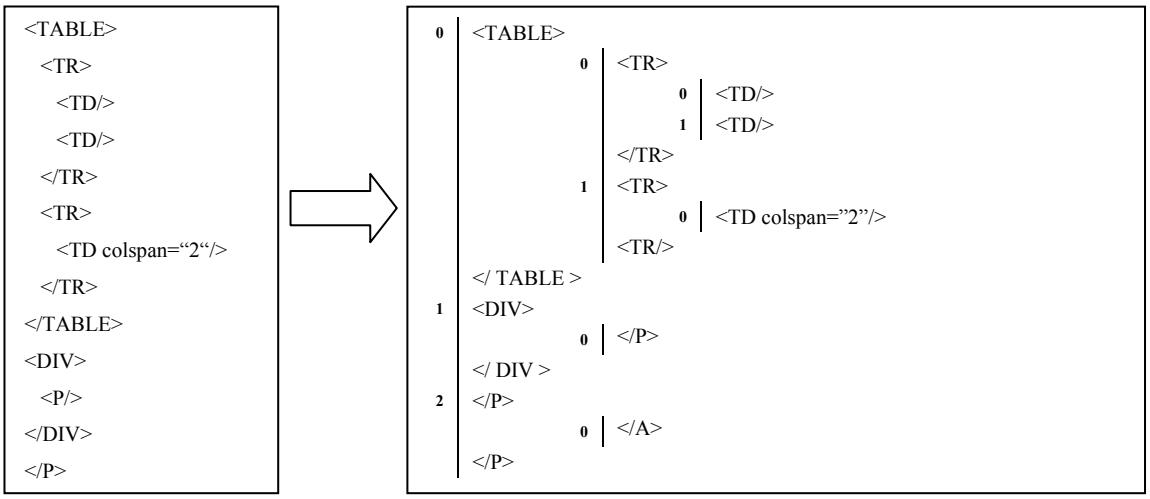
Žemiau pateiktame 12 paveiksle pavaizduotas XML dokumento atvaizdavimas į HTML dokumentą.



12 pav. XML dokumento atitikmuo HTML dokumentui.

5.4. Mazgų lokalizavimas

Mazgai ir mazgų atributai lokalizuojami remiantis jų vardais, pozicija nuo dokumento pradžios ir gyliu. Žemiau pateiktame pavyzdyje matyti HTML dokumento fragmentas ir jo mazgų žymėjimas. Remiantis šiuo pavyzdžiu, sudarytos mazgų ir mazgų atributų lokalizavimo taisyklės.



13 pav. Mazgų lokalizavimas.

Mazgo lokalizavimas. Mazgo lokalizavimui naudojamas jo vardas ir vektorinis kelias nuo dokumento pradžios:

<pavadinimas, vektorinis kelias>

pavyzdžiui:

<TABLE, [0]> – taip atvaizduotas pirmasis dokumento elementas <TABLE>.
<TD, [0,1,0]> – taip atvaizduojamas <TD> elementas, kuris yra elemento <TABLE> antrame <TR> elemente.

Atributo lokalizavimas. Atributo lokalizavimui naudojamas jį turinčio mazgo lokalizavimo struktūra ir vardu. Atributo vardą vadintinas raktu:

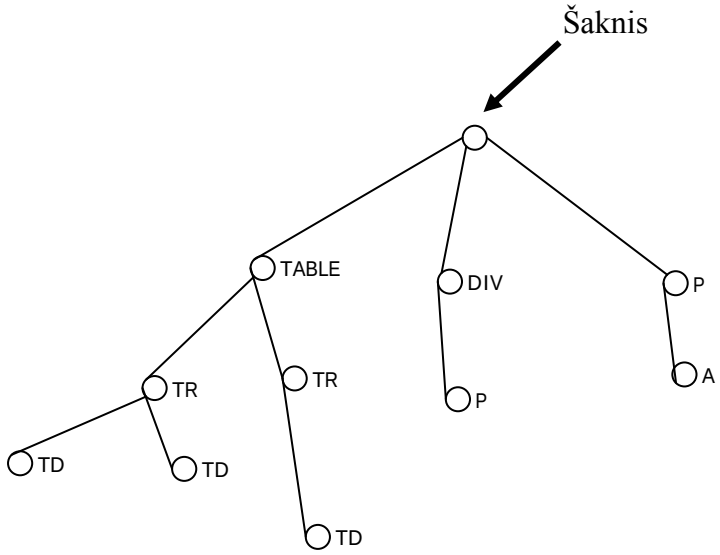
<mazgas, raktas>

Pavyzdžiui:

<<TD, [0,1,0]>, colspan> – taip atvaizduojamas <TD> elemento, kuris yra elemento <TABLE> antrame <TR> elemente, atributas. Šio atributo reikšmė yra 2.

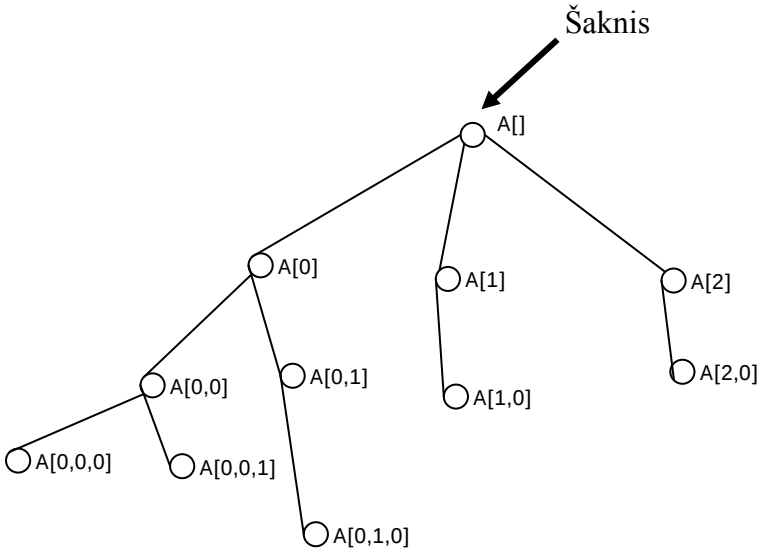
Pastaba. Palengvinant skaičiavimus ir visos dokumento struktūros supratimą, patogu patį dokumentą vaizduoti kaip medžio viršutinę šaką, kurią galima vaizduoti taip: <[]>. Mums nebūtina žinoti ir žymėtis medžio elementų vardus, pakanka žinoti tik jų padėtis dokumente, t.y. žinoti jų vektorinius lokalizacijų kelius. Priėmus šiuos supaprastinimus, aukščiau pateiktą dokumentą galima pavaizduoti tokiomis medinėmis struktūromis, kaip pavaizduota 14 ir 15

paveiksluose. „Šaknis“ – tai dokumentas, kuriame yra duomenys. Tokie žymėjimai naudojami tolimesniuose pavyzdžiuose, taip pat ir aiškinant operacijų transformacijų algoritmų veikimo principus.



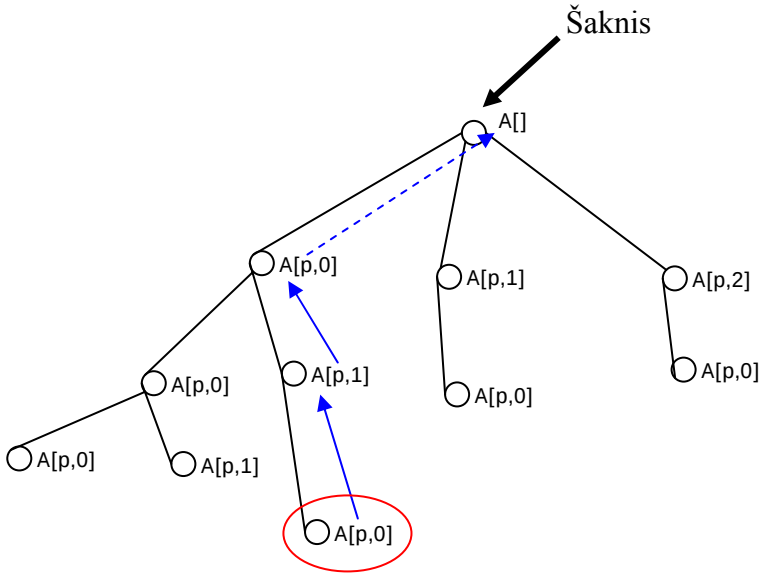
14 pav. Dalinis HTML dokumentas.

15 paveiksle „A“ yra dokumento vardas, jo žymė.



15 pav. Mazgų žymėjimas vektoriniais keliais.

Medžio mazgų vektorinius kelius galima supaprastintai žymėti rekursyviu būdu. Tokiu būdu dalis vektorinio kelio yra žymima *p*, kur *p* – šakos tėvinė šaka. Iš *p* galima gauti tėvinės šakos vektorinį kelią, todėl vaikinės šakos vektorinį kelią galime žymėti tokiu vektoriumi [*p*, *n*], kur *n* – vaikinės šakos pozicija tėvinėje šakoje. Pilnas pažymėtos šakos indeksas gaunamas kylant aukštyn medžiu iki šaknies. p keičiamas į tėvo indeksą. Medinės struktūros apėjimo pavyzdys pateiktas 16 paveiksle.



16 pav. Rekursinis mazgų žymėjimas.

16 paveiksle pavaizduoto pavyzdžio, raudonai pažymėto mazgo, vektorinio kelio formavimas:

1 žingsnis: A[p, 0]

2 žingsnis: A[p,1,0].

3 žingsnis: A[p,0,1,0].

4 žingsnis: p pakeičiamas tuščia aibe, todėl jis yra praleidžiamas. Gautas rezultatas yra A[0,1,0].

5.5. Operacijų tipai

Operacijos, kurios gali būti atliktos medinės struktūros dokumentui, skirstomos į struktūrines ir keitimo. Struktūrinės operacijos skiriamos dvi: *įterpimo*, *trynimo*. Keitimo operacija yra viena: *keitimo*. Struktūrinės operacijos keičia dokumento struktūrą įterpdamos arba ištrindamos dokumento mazgus.

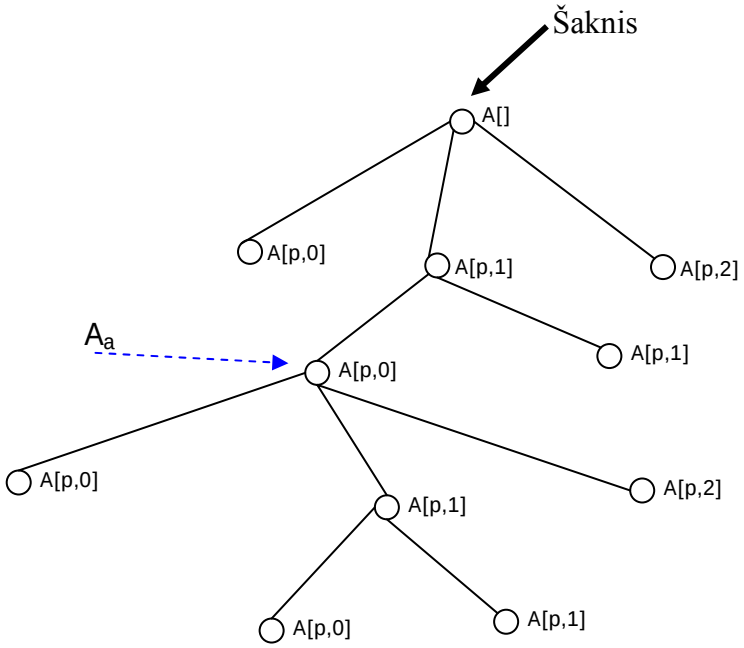
5.5.1. Struktūrinės operacijos

Operacija $insert(N, n, M)$ įterpia M šaką (medį) į šakos N poziciją n. Ši operacija taip pat įterpia naują šaką į poziciją n, jei M yra tuščias medis.

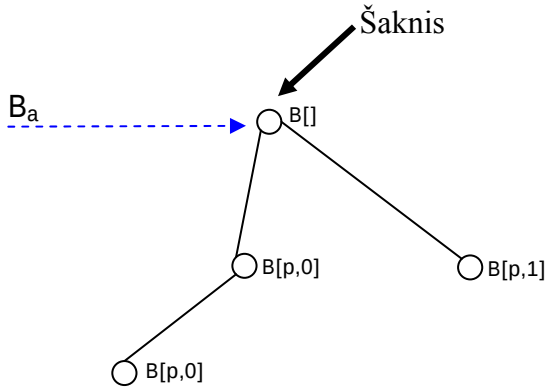
Operacijos parametrai:

- N – mazgas kuriame atliekama operacija;
- n – N mazgo vaiko pozicija, kurioje bus į terota šaka M;
- M – terpiama šaka (medis).

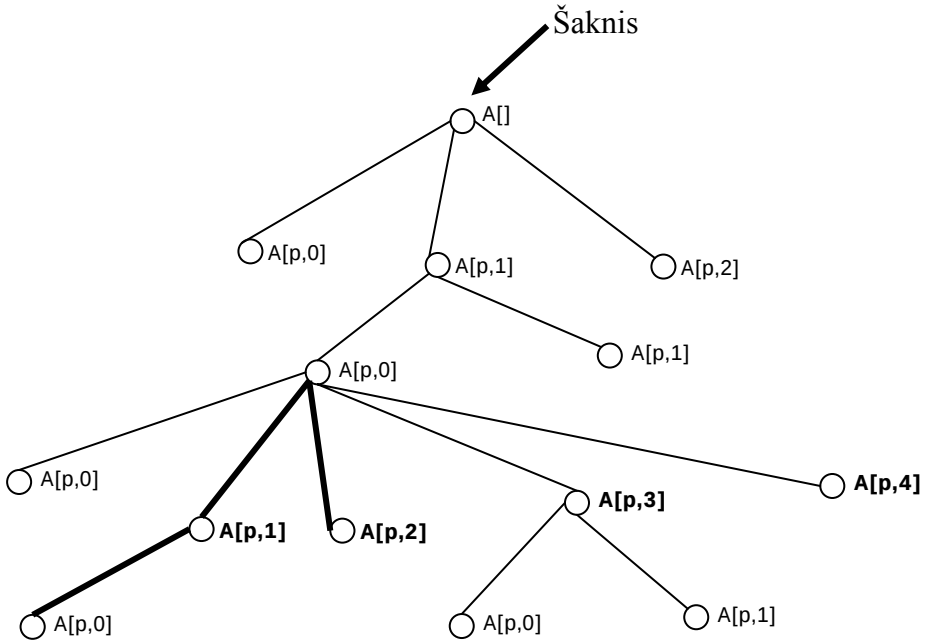
Žemiau pateikiamas pavyzdys, kaip į medžio A (17 paveikslas) šaką A_a yra įterpiamas medis B (18 paveikslas). Tokiam veiksmui atlikti panaudota terpimo operacija $insert(A_a, 1, B_a)$. Šios operacijos atklikimo rezultatas pavaizduotas 19 paveiksle. Mazgai kurie keičia savo pozicijas yra paryškinti. Naujai įterptos šakos taip pat yra paryškintos.



17 pav. A_a šaka



18 pav. B_a šaka



19 pav. insert(A_a,1, B_a) veikimo rezultatas

Operacija $delete(N,n,M)$ trina šakos N vaiką pozicijoje n. Tam, kad nebūtų prarasti duomenys, ištrinta šaka yra pervardinama į M.

Operacijos parametrai:

- N – šaka kurioje atliekama operacija;
- n – šakos M pozicija;
- M – trinama šaka.

5.5.2. Keitimo operacijos

Operacija $change(N,k,f)$ keičia šakos N atributo k reikšmę į naują reikšmę f.

Operacijos parametrai:

- N – mazgas kuriame atliekama operacija;
- k – atributas;
- f – nauja atributo reikšmė.

5.6. Operacijų transformavimo veikimo principas

Šioje dalyje pateikiamos algoritmų ir prototipinių funkcijos, kurias rekomenduojama naudoti grupinio bendradarbiavimo sistemų, skirtų medinių duomenų struktūrų dokumentams, realizavimui. Operacijų transformacijų algoritmai medinės struktūros dokumentui yra tokie

pat, kaip ir linijinių dokumentų operacijų transformavimo dOPT algoritmams. Taigi, mazgų redagavimo operacijų transformavimo algoritmai yra šie:

- įterpti – įterpti;
- įterpti – ištrinti;
- ištrinti – įterpti;
- ištrinti – ištrinti.

Skirtingai nuo linijinių dokumentų, medinės struktūros dokumentams, ypač pagrįstų XML pagrindu, būdingos mazgų atributų redagavimo operacijos. Jų transformacijų algoritmai yra šie:

- keisti – įterpti;
- keisti – ištrinti;
- keisti – keisti.

Skaičiavimų palengvinimui, reikėtų įvesti keletą prototipinių funkcijų, kurios būtų nepriklausomos nuo sistemos ir galėtų būti realizuotos pagal konkrečios sistemos specifikaciją. Tokios funkcijos būtų:

- $tree(N)$ – funkciją grąžina nuorodą į medį, kuriam priklauso šaka N, arba nuorodą į ją pačia, jei šaka yra medis;
- $site(N)$ – funkcija grąžina sistemos mazgo identifikacijos numerį, kuriame buvo modifikuota šaka N;
- $vector(N)$ – funkcija grąžina šakos N vektorinę poziciją (kelią) medyje;
- $navigate(P,V)$ - funkcija grąžina medžio P šaką, kuri randasi V vektorinėje pozicijoje.

Visos operacijų transformavimo funkcijos, pagal savo parametrus yra vienodos. Sakykime, O_a ir O_b yra sugeneruotos tai pačiai dokumento būsenai, kur:

- O_a – nutolusi operacija;
- O_b – lokali operacija.

Operacija O_b yra atlikta prieš operaciją O_a. Tuomet, kiekviena funkcija turės du parametrus, tai operacijos O_a ir O_b. Operacijų transformacijos funkcija, jei reikia, visada modifikuoja operaciją O_a. Taigi, bendru atveju operacijų transformavimo funkcija atrodytų šitaip:

$$\mathrm{IT}(O_a, O_b)$$

5.6.1. Adresų palyginimo algoritmas

Visuose operacijų su mazgais transformavimo algoritmuose reikia žinoti redaguojamų mazgų poziciją vienas kito atžvilgiu. Mazgų pozicijos vienas kito atžvilgiu nustatymui, pateiksime compare(N_a, N_b) funkciją. Jos parametrai – du mazgai N_a ir N_b. Funkcija grąžina mazgo N_b poziciją mazgo N_a atžvilgiu. Ši funkcija gali grąžinti šiuos rezultatus:

- PREFIX – mazgas N_b yra mazgo N_a prefiksas, t.y. N_a mazgas yra mazgo N_b šaka;
- SUFFIX – mazgas N_b yra mazgo N_a sufiksas, t.y. N_b mazgas yra mazgo N_a šaka;
- DIFFERENT – mazgai N_a ir N_b priklauso skirtingiems medžiams arba skirtingoms medžio šakoms, kurios nėra viena kitos prefiksas arba sufiksas;
- SAME – mazgai N_a ir N_b yra ekvivalentūs, t.y. rodo į ta pačia medžio poziciją.

Algoritmo $compare(N_a, N_b)$ tekstas

```
compare(N_a, N_b)
{
        P_a = tree(N_a)
        P_b = tree(N_b)
        V_a = vector(N_a)
        V_b = vector(N_b)
        if (P_a = P_b)
                for (i ← 0...(V_a − 1))
                        if (i = V_b)
                                return PREFIX
                        elif (navigate(P_a, i) ≠ navigate(N_b, i))
                                return DIFFERENT
                if (V_a = V_b)
                        return SAME
                else
                        return SUFFIX
        else
                return DIFFERENT
}
```

Funkcijos veikimo principas

Jei mazgas N_b ir mazgas N_a nepriklauso tam pačiam medžiui, tuomet funkcija grąžina DIFFERENT. Jei mazgas N_b ir mazgas N_a priklauso tam pačiam medžiui, tuomet iš mazgo N_a vektorinio kelio pradedant medžio, kuriam priklauso mazgas N_a, šaknimi formuojama aibė vektorių. Kiekvienas po to einantis vektorius yra ilgesnis vienu elementu už N_a pozicijos vektoriaus. Tarkime turime vektorių $A\langle[],0,1,10\rangle$, tuomet aibė formuojamų vektorių būtų tokia: $\{\langle[]\rangle,\langle[],0\rangle,\langle[],0,1\rangle,\langle[],0,1,1\rangle\}$. Kiekvienas toks vektorius yra lyginamas su N_b vektoriniu keliu. Jei kuris nors vektorius lygus N_b vektoriniam keliu, tuomet mazgas N_b yra mazgo N_a prefiksas, funkcija grąžina PREFIX. Jei kuris nors dalinis N_b vektorinis kelias nesutampa su tokio pat ilgio N_a daliniu vektoriniu keliu, tuomet funkcija grąžina DIFFERENT. Jei po dalinių vektorinių kelių palyginimo rezultatas nubūna aiškus, tuomet yra palyginami mazgo N_b ir mazgo N_a vektoriniai keliai. Jei keliai yra vienodi, tuomet funkcija grąžina SAME, kitu atveju funkcija grąžina SUFFIX.

5.7. XML operacijų su mazgais transformacijos

Kaip ir linijinių dokumentų atveju, operacijų transformacijos reikalingos, kai dviejuose skirtinguose sistemos mazguose atitinkamai sugeneruojamos tokios mazgų redagavimo operacijos:

- įterpti – įterpti;
- įterpti – ištrinti;
- ištrinti – įterpti;
- ištrinti – ištrinti;

Skirtingai nei linijinių dokumentų atveju, operacijoms negalima pritaikyti operacijų transformavimo. Ar pritaikyti operacijos transformaciją ar ne – priklauso nuo redaguojamų mazgų padėties vienas kito atžvilgiu. Operacijų transformacijas reikia taikyti tik tuo atveju, kaip vienas mazgas yra prefiksas arba kito mazgo sufiksas, arba redaguojamas tas pats mazgas. Kai mazgai nėra vienas kito sufiksas arba prefiksas operacijos transformavimas nereikalingas.

5.7.1. Įterpti – įterpti

Šiuo atveju du sistemos vartojai į dokumentą terpia naujus mazgus. Operacijų transformavimo algoritmas *TII* (transform insert – insert) tikrina, kurioje dokumento vietoje

terpiami nauji mazgai. Nutolusi operacija nebus transformuojama tuo atveju, kai pozicija yra arčiau dokumento pradžios, operacijų pozicijos sutampa ir nutolusio vartotojo identifikatorius mažesnis už lokalaus vartotojo identifikatorių.

TII (transform insert – insert) operacijų transformavimo algoritmo tekstas

TII($insert(N_a, n_a, M_a), insert(N_b, n_b, M_b)$)
{
$N_a' \leftarrow N_a$
$n_a' \leftarrow n_a$
$P_a = tree(N_a)$
$P_b = tree(N_b)$
$V_b = vector(N_b)$
$N_{ba} = navigate(P_a, V_b)$
if $(P_a = M_b)$
$N_a' \leftarrow (P_b, N_b + [n_b] + N_a)$
elif $(compare(N_a, N_b) = PREFFIX)$
if $((n_b < N_{ba}) \vee ((n_b = N_{ba}) \wedge (site(N_a) < site(N_b))))$
$N_a' \leftarrow V_b + 1$
elif $(compare(N_a, N_b) = SAME)$
if $((n_b < n_a) \vee ((n_b = n_a) \wedge (site(N_a) < site(N_b))))$
$n_a' \leftarrow n_a + 1$
return $insert(N_a', n_a', M_a)$
}

Algoritmo veikimo principas

Vienas sistemos vartotojas terpia naują medžio šaką M_a į šakos N_a vaiko poziciją n_a, kitas sistemos vartotojas terpia naują medžio šaką M_b į šakos N_b vaiko poziciją n_b. Jei O_a įterpiama aukščiau O_b - transformacija nevykdoma, Jei O_a įterpiama žemiau O_b - keičiama O_a pozicija.

Jei šaka N_a priklauso medžiui M_b, tuomet ji yra pertvarkoma. Tokių būdu šaka N_a tampa priklausoma tam pačiam medžiui, kaip ir šaka N_b ir yra įkomponuojama į šakos N_b vaiko poziciją n_b.

Kitu atveju palyginamos šakos N_a ir N_b. Jei:

- mazgas N_b yra mazgo N_a prefiksas, tuomet palyginame n_b su paskutine vektoriaus i elemento šakoje N_a reikšme. Jeigu n_b yra mažesnis už tą reikšmę arba lygus jai ir sistemos mazgo numeris, kuriame redaguojama šaka N_a yra mažesnis už sistemos mazgo numerį, kuriame redaguojama šaka N_b, tuomet vektoriaus reikšmė V_b yra padidinama vienetu ir vektorius V_b yra atitinkamai pakeičiamas šakos N_a vektoriaus dalimi;
- mazgai N_a ir N_b yra ekvivalentūs, tuomet palyginame n_b ir n_a. Jeigu n_b reikšmė yra mažesnė už n_a reikšmę arba jos yra lygios ir sistemos mazgo numeris, kuriame redaguojama šaka N_a yra mažesnis už sistemos mazgo numerį, kuriame redaguojama šaka N_b, tuomet pozicija n_a yra padidinama vienetu.

Atlikus (arba ne) šakos N_a ir pozicijos n_a, į kurią turi būti įterpta nauja šaka M_a, modifikacijas į ją yra įterpiama nauja šaka M_a.

5.7.2. Ištrinti – įterpti

Šiuo atveju vienas sistemos vartotojas trina objektą iš dokumento, antrasis – terpia. *TDI* (transform delete – insert) operacijų transformavimo algoritmas atlieka šiuos veiksmus: jei nutolusi operacija yra atliekama arčiau dokumento pradžios nei lokali įterpimo operacija, tuomet ji nėra transformuojama, jei nutolusi operacija yra atliekama toliau terpimo operacijos, tai tuo atveju ją reikia transformuoti.

TDI (transform delete – insert) operacijų transformavimo algoritmo tekstas

$\mathsf{TDI}(delete(N_a, n_a, M_a), insert(N_b, n_b, M_b))$
{
$N_a' \leftarrow N_a$
$n_a' \leftarrow n_a$
$P_a = tree(N_a)$

$V_b = vector(N_b)$

$N_{ba} = navigate(P_a, V_b)$

if $(compare(N_a, N_b) = PREFFIX)$

if $((n_b < N_{ba}) \vee ((n_b = N_{ba}) \wedge (site(N_a) < site(N_b))))$

$N_a^{'} \leftarrow V_b + 1$

elif $(compare(N_a, N_b) = SAME)$

if $((n_b < n_a) \vee ((n_b = n_a) \wedge (site(N_a) < site(N_b))))$

$n_a^{'} \leftarrow n_a + 1$

return $delete(N_a^{'}, n_a^{'}, M_a)$

}

Algoritmo veikimo principas

Vienas sistemos vartotojas trina medžio N_a šaką M_a pozicijoje n_a, kitas sistemos vartotojas terpia naują medžio šaką M_b į šakos N_b vaiko poziciją n_b.

Palyginamos šakos N_a ir N_b. Jei:

- mazgas N_b yra mazgo N_a prefiksas, tuomet palyginame n_b su paskutine vektoriaus i elemento šakoje N_a reikšme. Jeigu n_b yra mažesnis už tą reikšmę arba lygus jai ir sistemos mazgo numeris, kuriame redaguojama šaka N_a yra mažesnis už sistemos mazgo numerį, kuriame redaguojama šaka N_b, tuomet vektoriaus reikšmė V_b yra padidinama vienetu ir vektorius V_b yra atitinkamai pakeičiamas šakos N_a vektoriaus dalimi;
- mazgai N_a ir N_b yra ekvivalentūs, tuomet palyginame n_b ir n_a. Jeigu n_b reikšmė yra mažesnė už n_a reikšmę arba jos yra lygios ir sistemos mazgo numeris, kuriame redaguojama šaka N_a yra mažesnis už sistemos mazgo numerį, kuriame redaguojama šaka N_b, tuomet pozicija n_a yra padidinama vienetu.

Atlikus (arba ne) šakos N_a ir pozicijos n_a, iš kurios turi būti ištrinta šaka M_a, modifikacijas iš jos yra ištrinama šaka M_a.

5.7.3. Įterpti – ištrinti

Šiuo atveju vienas sistemos vartotojas terpia naują šaką į dokumentą, antrasis – trina. *TID* (transform insert – delete) operacijų transformavimo algoritmas atlieka šiuos veiksmus: jei terpimo operacija vyksta prieš trynimo operaciją – tai operacija nėra transformuojama, jei trynimas vyksta toje pačioje pozicijoje arba artimesnėje dokumento pradžiai, tuomet įterpimo operacija turi būti transformuojama.

TID (transform insert – delete) operacijų transformavimo algoritmo tekstas

TID($insert(N_a, n_a, M_a), delete(N_b, n_b, M_b)$)
{

$N_a^{'} \leftarrow N_a$

$n_a^{'} \leftarrow n_a$

$P_a = tree(N_a)$

$V_b = vector(N_b)$

$N_{ba} = navigate(P_a, V_b)$

if $(compare(N_a, N_b) = SAME)$

if $(n_b < n_a)$

$n_a^{'} \leftarrow n_a^{'} - 1$

elif $(compare(N_a, N_b) = PREFFIX)$

if $(n_b < N_{ba})$

$N_a^{'} \leftarrow V_b - 1$

elif $((n_b = N_{ba}) \wedge (site(N_a) = site(N_b)))$

$N_a^{'} \leftarrow (M_b, navigate(P_a, V_b + 1))$

return $insert(N_a^{'}, n_a^{'}, M_a)$

}

Algoritmo veikimo principas

Vienas sistemos vartotojas terpia naują medžio šaką M_a į šakos N_a vaiko poziciją n_a, kitas sistemos vartotojas trina medžio N_b šaką M_b pozicijoje n_b.

Palyginamos šakos N_a ir N_b. Jei:

- mazgai N_a ir N_b yra ekvivalentūs, tuomet palyginame n_b ir n_a. Jeigu n_b reikšmė yra mažesnė už n_a reikšmę, tuomet pozicija n_a yra mažinama vienetu;
- mazgas N_b yra mazgo N_a prefiksas, tuomet palyginame n_b su paskutine vektoriaus i elemento šakoje N_a reikšme. Jei:
    - n_b yra mažesnis už ją, tuomet vektoriaus reikšmė V_b yra mažinama vienetu ir vektorius V_b yra atitinkamai pakeičiamas šakos N_a vektoriaus dalimi.
    - n_b reikšmė lygi paskutinei vektoriaus i elemento, šakoje N_a, reikšme ir sistemos mazgo numeris, kuriame redaguojama šaka N_a yra lygus sistemos mazgo numerį, kuriame redaguojama šaka N_b, tuomet šaka N_a tampa M_b medžio šaka, V_b vektorius paskutinis elementas yra didinamas vienetu.

Atlikus (arba ne) šakos N_a ir pozicijos n_a, į kurią turi būti įterpta nauja šaka M_a, modifikacijas į ją yra įterpiama nauja šaka M_a.

5.7.4. Ištrinti – ištrinti

Šiuo atveju abu sistemos vartotojai trina dokumento objektus. *TDD* (transform delete – delete) operacijų transformavimo algoritmas atlieka šiuos veiksmus, jei nutolusi operacija atliekama arčiau dokumento pradžios, operacija nėra transformuojama, jei lokali operacija atliekama arčiau dokumento pradžios, tuomet nutolusi operacija yra transformuoja, kai pozicijos sutampa – grąžinama vienetinė operacija.

TDD (transform delete – delete) operacijų transformavimo algoritmo tekstas

TDD($delete(N_a, n_a, M_a)$, $delete(N_b, n_b, M_b)$)
{
$N_a^{'} \leftarrow N_a$
$n_a^{'} \leftarrow n_a$
$P_a = tree(N_a)$
$V_b = vector(N_b)$
$N_{ba} = navigate(P_a, V_b)$
if $(\text{comapre}(N_a, N_b) = SAME)$

if $(n_b < n_a)$

$n_a^{'} \leftarrow n_a - 1$

elif $((n_b = n_a) \wedge (site(N_a) = site(N_b)))$

return *IDENTITY*

elif $(\text{compare}(N_a, N_b) = PREFFIX)$

if $(n_b < N_{ba})$

$N_a^{'} \leftarrow V_b - 1$

elif $(n_b = N_a[i])$

$N_a^{'} \leftarrow (M_b, navigate(P_a, V_b + 1))$

return $delete(N_a^{'}, n_a^{'}, M_a)$

}

Algoritmo veikimo principas

Vienas sistemos vartotojas trina medžio N_a šaką M_a pozicijoje n_a, kitas sistemos vartotojas trina medžio N_b šaką M_b pozicijoje n_b.

Palyginamos šakos N_a ir N_b. Jei:

- mazgai N_a ir N_b yra ekvivalentūs, tuomet lyginame n_b ir n_a reikšmes. Jei:
  - n_b reikšmė yra mažesnė už n_a reikšmę, tuomet pozicija n_a yra mažinama vienetu;
  - n_b reikšmė lygi paskutinei vektoriaus i elemento, šakoje N_a, reikšme ir sistemos mazgo numeris kuriame redaguojama šaka N_a yra lygus sistemos mazgo numerį, kuriame redaguojama šaka N_b, tuomet grąžinama vienetinė operacija, kuri nieko neatlieka;
- mazgas N_b yra mazgo N_a prefiksas, tuomet lyginame poziciją n_b ir paskutinę vektoriaus i elemento, šakoje N_a, reikšmes. Jei
  - n_b reikšmė yra mažesnė už paskutinę vektoriaus i elemento, šakoje N_a, reikšme, tuomet vektoriaus reikšmė V_b yra mažinama vienetu ir vektorius V_b yra atitinkamai pakeičiamas šakos N_a vektoriaus dalimi.

- n_b reikšmė lygi paskutinei vektoriaus i elemento, šakoje N_a, reikšme, tuomet šaka N_a tampa M_b medžio šaka, V_b vektorius paskutinis elementas yra didinamas vienetu.

Atlikus (arba ne) šakos N_a ir pozicijos n_a, iš kurios turi būti ištrinta šaka M_a, modifikacijas iš jos yra ištrinama šaka M_a.

5.8. XML operacijų su atributais transformacijos

Kaip ir linijinių dokumentų atveju, operacijų transformacijos reikalingos, kai dviejuose skirtinguose sistemos mazguose atitinkamai sugeneruojamos tokios mazgų atributų redagavimo operacijos:

- keisti – įteprti;
- keisti – ištrinti;
- keisti – keisti.

5.8.1. Keisti – įterpti

Šiuo atveju, vienas sistemos vartotojas keičia šakos atributo reikšmę, kitas sistemos vartotojas terpia naują šaką į dokumentą.

TCI (transform change – insert) operacijų transformavimo algoritmo tekstas

TCI($change(N_a, k, f)$, $insert(N_b, n, M)$)
{
$N_a^{'} \leftarrow N_a$
$P_a = parent(N_a)$
$P_b = parent(N_b)$
$V_b = vector(N_b)$
$N_{ba} = navigate(P_a, V_b)$
if $(P_a = M)$
$N_a^{'} \leftarrow (P_b, N_b[:] + [n] + N_a[:])$
elif $((compare(N_a, N_b) = PREFFIX) \wedge (n \le V_b))$
$N_a^{'} \leftarrow V_b + 1$
return $change(N_a^{'}, k, f)$
}

Algoritmo veikimo principas

Šiuo atveju, vienas sistemos vartotojas keičia šakos N_a atributo k reikšmę į naują reikšmę f, kitas sistemos vartotojas terpia naują medžio šaką M į šakos N_b vaiko poziciją n.

Jei šaka N_a priklauso medžiui M, tuomet ji yra pertvarkoma. Tokių būdu šaka N_a tampa priklausoma tam pačiam medžiui, kaip ir šaka N_b ir yra įkomponuojama į šakos N_b vaiko poziciją n.

Jei mazgas N_b yra mazgo N_a prefiksas ir poziciją n yra mažesnė už paskutinę vektoriaus i elemento, šakoje N_a, reikšme, tuomet vektoriaus reikšmė V_b yra didinama vienetu ir vektorius V_b yra atitinkamai pakeičiamas šakos N_a vektoriaus dalimi.

Atlikus (arba ne) šakos N_a modifikacijas, jos atributui k priskiriama nauja reikšmė f.

5.8.2. Keisti - ištrinti

Šiuo atveju, vienas sistemos vartotojas keičia šakos atributo reikšmę, kitas sistemos vartotojas trina šaką iš dokumento.

TCD (transform change – delete) operacijų transformavimo algoritmo tekstas

TCD($change(N_a, k, f), delete(N_b, n, M)$)
{
$N_a' \leftarrow N_a$
$P_a = tree(N_a)$
$V_b = vector(N_b)$
$N_{ba} = navigate(P_a, V_b)$
if $(compare(N_a, N_b) = PREFFIX)$
if $(n < N_{ba})$
$N_a' \leftarrow V_b - 1$
elif $(n_b = N_{ba})$
$N_a' \leftarrow (M, navigate(P_a, V_b + 1))$
return $change(N_a', k, f)$

}

Algoritmo veikimo principas

Jei mazgas N_b yra mazgo N_a prefiksas, tuomet yra lyginama pozicija n ir paskutinė vektoriaus i elemento, šakoje N_a, reikšmė. Jei:

- n_b reikšmė yra mažesnė už paskutinę vektoriaus i elemento, šakoje N_a, reikšme, tuomet vektoriaus reikšmė V_b yra mažinama vienetu ir vektorius V_b yra atitinkamai pakeičiamas šakos N_a vektoriaus dalimi;
- n_b reikšmė lygi paskutinei vektoriaus i elemento, šakoje N_a, reikšme, tuomet šaka N_a tampa M medžio šaka, V_b vektorius paskutinis elementas yra didinamas vienetu.

Atlikus (arba ne) šakos N_a modifikacijas, jos atributui k priskiriama nauja reikšmė f.

5.8.3. Keisti – keisti

Šiuo atveju, abu sistemos vartotojas keičia šakos atributo reikšmę.

TCC (transform change – change) operacijų transformavimo algoritmo tekstas

$$O_a = change(N_a, k_a, f_a)$$
$$O_b = change(N_b, k_b, f_b)$$

```
TCC(O_a, O_b)
{
    if ((comapre(N_a, N_b) = SAME) ∧ (k_a = k_b))
        if (O_a < O_b)
            return IDENTITY
        else
            return O_a
    return O_a
}
```

Algoritmo veikimo principas

Jei mazgai N_b ir N_a yra ne vienas ir tas pats mazgas, grąžinama operacija O_a. Jei mazgai N_b ir N_a yra vienas ir tas pats mazgas, bet atributai yra skirtingi, tuomet grąžinama

operacija O_a. Jei mazgai N_b ir N_a yra vienas ir tas pats mazgas ir redaguojamas tas pats mazgo atributas, tuomet tikriname operacijų pirmumą globalioje operacijų sekoje. Jei O_a yra įvykusi anksčiau operacijos O_b, tuomet grąžinama vienetinė operacija, kitu atveju grąžinama operacija O_a.

6. Praktinis panaudojimas

Šioje darbo dalyje praktiškai patikrinsime mūsų pasiūlytų operacijų transformavimo algoritmų, XML dokumentams, teisingumą. Tam tikslui sukursime eksperimentinio XML dokumento duomenų struktūrą, struktūrinių operacijų, kurias galima atlikti su dokumentu, duomenų struktūras, programinę įrangą, imituojančia paskirstytą sistema.

6.1. Eksperimento aprašymas

Eksperimento tikslas – suprojektuoti eksperimentinę programinę įranga ir patikrinti pasiūlytos metodikos teisingumą. Eksperimento atlikimo tvarka:

- Sumodeliuoti XML dokumento duomenų struktūrą, kuria sudaro XML dokumento mazgo ir mazgo atributų duomenų struktūros.
- Apsibrėžti struktūrinių operacijų tipus. Šiomis operacijomis bus atliekami dokumento struktūriniai pakeitimai.
- Sumodeliuoti struktūrinės operacijos duomenų struktūrą.
- Sukurti programinę įrangą imituojančią paskirstytą sistemą. Ši programinė įranga turi suteikti galimybę imituoti redaguojamo dokumento paskirstymą sistemos mazgams, sistemos mazgus (vartotojus), jų atliekamus redagavimo veiksmus paskirstytam dokumentui.
- Eksperimentiškai patikrinsime visus mazgų redagavimo operacijų transformavimo algoritmus: įterpti – įterpti, įterpti – ištrinti, ištrinti – įterpti, ištrinti – ištrinti.

Eksperimento atlikimui pasirinkome šias priemones:

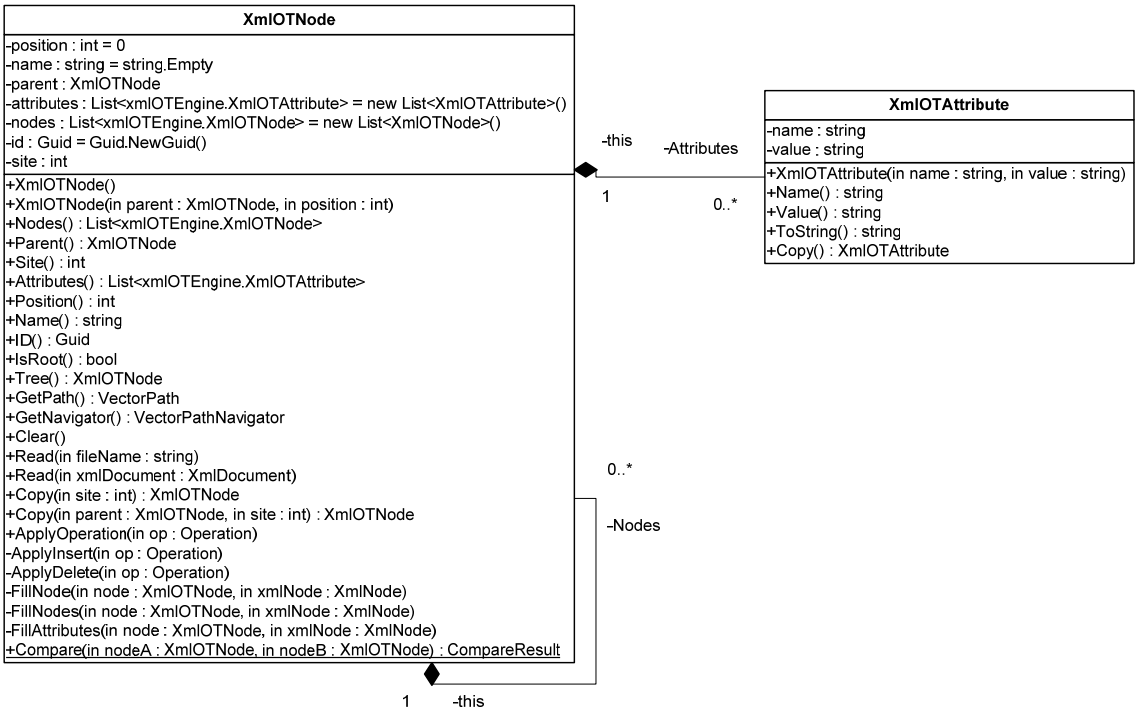
- Microsoft® .NET Framework v2.0.
- Microsoft® Visio® for Enterprise Architects (11.4301.6568).
- Microsoft® Visual Studio® 2005.
- Microsoft® Visual C# v2.0.
- Doc-O-Matic 5.1.

6.2. Eksperimento eiga

6.2.1. Programinės įrangos projektavimas

Pirmiausia, sukursime duomenų struktūrą aprašančia medinį dokumentą. Šiame eksperimente, tai bus XML dokumento eksperimentinė duomenų struktūra aprašanti XML dokumento mazgą. Ši duomenų struktūra turi agregatinį ryšį pati į save ir ryšį su duomenų

struktūra – aprašančia XML dokumento mazgo atributą. 20 paveiksle pavaizduota eksperimentinio dokumento struktūra, panaudojant UML diagramas.



20 pav. Dokumento UML diagrama

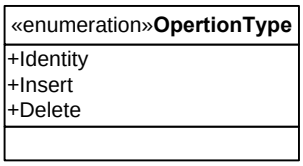
Pagrindinių savybių aprašymas:

	Pavadinimas	Aprašymas
	Attributes	Mazgo atributai.
	ID	Mazgo identifikacinis numeris.
	IsRoot	Grąžina ar ši šaka yra dokumento šaknis.
	Name	Mazgo pavadinimas.
	Nodes	Grąžina dokumento mazgo vaikinius mazgus.
	Parent	Tėvinis mazgas.
	Position	Mazgo pozicija tėvinėje šakoje.
	Site	Mazgo numeris.
	Tree	Grąžina dokumento šaknį.

Pagrindinių metodų aprašymas:

	Pavadinimas	Aprašymas
	ApplyOperation	Įvykdo medžiui struktūrinę operaciją.
	Clear	Išvalo mazgo struktūrą.
S	Compare	Palygina du medžio mazgus.
	Copy	Kopijuoja dokumento mazgą.
	GetNavigator	Sukuria mazgui navigatorių.
	GetPath	Grąžina mazgo vektorinį kelią.
	Read	Nuskaito Xml dokumentą ir iš jo suformuoja XmlOTNode struktūrą.

OperationType – apibrėžiai struktūrinių operacijų tipus. Pagal teoriją, jos yra dvi, tai: terpimo ir trynimo, bet tuščia operacija, kuri neįtakoja dokumento struktūrai (paprasčiausiai ji nėra vykdoma). 21 paveiksle pavaizduotas operacijų tipų vardinis tipas, panaudojant UML diagramas.

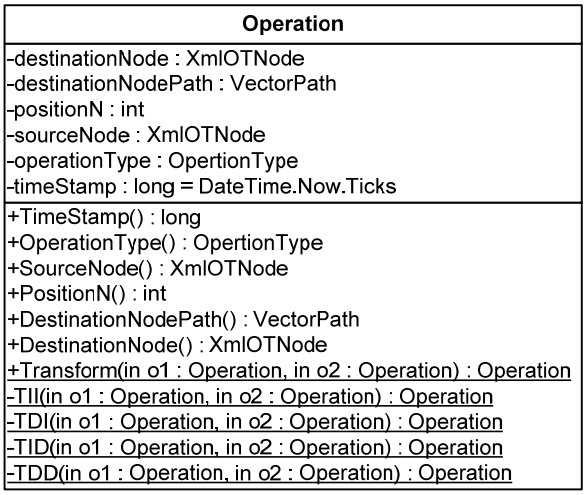


21 pav. Operacijų tipų UML diagrama

OperationType nariai:

Pavadinimas	Aprašymas
`Identity`	Tuščia operacija. Dokumento struktūra nekeičiama.
`Insert`	Struktūrinė terpimo operacija.
`Delete`	Struktūrinė trynimo operacija.

Operation - klasė aprašanti struktūrinę operaciją. Jos pagalba yra formuojami duomenys terpimo, trynimo operacijoms. Jei tai duomenys terpimo operacijai, tai pateikiamas dokumento mazgas kuriame atliekama operacija, pozicija, kurioje bus įterpta nauja šaka ir terpiama šaka. Jei tai duomenys trynimo operacijai, tai pateikiamas dokumento mazgas kuriame atliekama operacija, pozicija, iš kurios bus ištrintas dokumento mazgas ir grąžinamas ištrintas mazgas. Jei tai tuščia operacija – duomenys nepateikiami.



22 pav. Operacijos UML diagrama

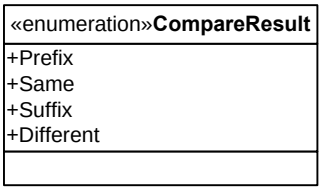
Pagrindinių savybių aprašymas:

	Pavadinimas	Aprašymas
	DestinationNode	Mazgas kuriame atliekama operacija.
	DestinationNodePath	Mazgo, kuriame atliekama operacija, vektorinis kelias.
	OperationType	Struktūrinės operacijos tipas.
	PositionN	Operacijos atlikimo pozicija mazge.
	SourceNode	Dokumento mazgas kuris įterpiamas į dokumentą arba ištrinamas iš dokumento.

Pagrindinių metodų aprašymas:

	Pavadinimas	Aprašymas
	Transform	Operacijų transformacijos funkcija.

CompareResult – struktūra aprašo dviejų medžio mazgų palyginimo rezultatą. Lyginant du dokumento mazgus *A* ir *B* yra grąžinamas *B* mazgo rezultatas *A* mazgo atžvilgiu. Struktūros UML diagrama pateikta 23 paveiksle.

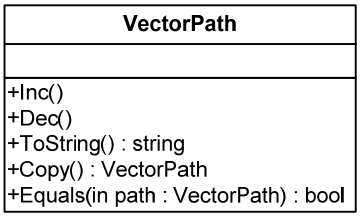


23 pav. ComapreResult UML diagrama

CompareResult struktūros nariai:

Narys	Aprašymas
`Prefix`	Lyginant du mazgus A ir B, mazgas B yra mazgo A prefiksas.
`Same`	Lyginant du mazgus A ir B, mazgai yra tie patys.
`Suffix`	Lyginant du mazgus A ir B, mazgas B yra mazgo A sufiksas.
`Different`	Lyginant du mazgus A ir B, mazgai yra nepriklausomi vienas nuo kito.

VectorPath – aprašo dokumento mazgo vektorinį kelią nuo dokumento pražios. *VectorPath* UML diagrama pateikta 24 paveiksle.

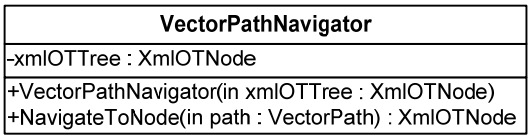


24 pav. VectorPath UML diagrama

Pagrindinių metodų aprašymas:

	Pavadinimas	Aprašymas
	Copy	Sukuria vektorinio kelio kopiją.
	Dec	Didina paskutinio vektoriaus elemento reikšmę 1.
	Equals	Palygina ar su vektoriniai keliai yra lygus.
	Inc	Didina paskutinio vektoriaus elemento reikšmę 1.
	ToString	Atvaizduoja vektorinį kelia šiuo formatu: „[],i,i+1,...“

VectorPathNavigator – aprašo medžio mazgų navigatorių. Tai pagalbinė klasė, kuri padeda nuo duoto dokumento mazgo naviguoti jo vaikinius mazgus pagal *VectorPath* klasę. *VectorPathNavigator* UML diagrama pateikta 25 paveiksle.



25 pav. VectorPath UML diagrama

Pagrindinių metodų aprašymas:

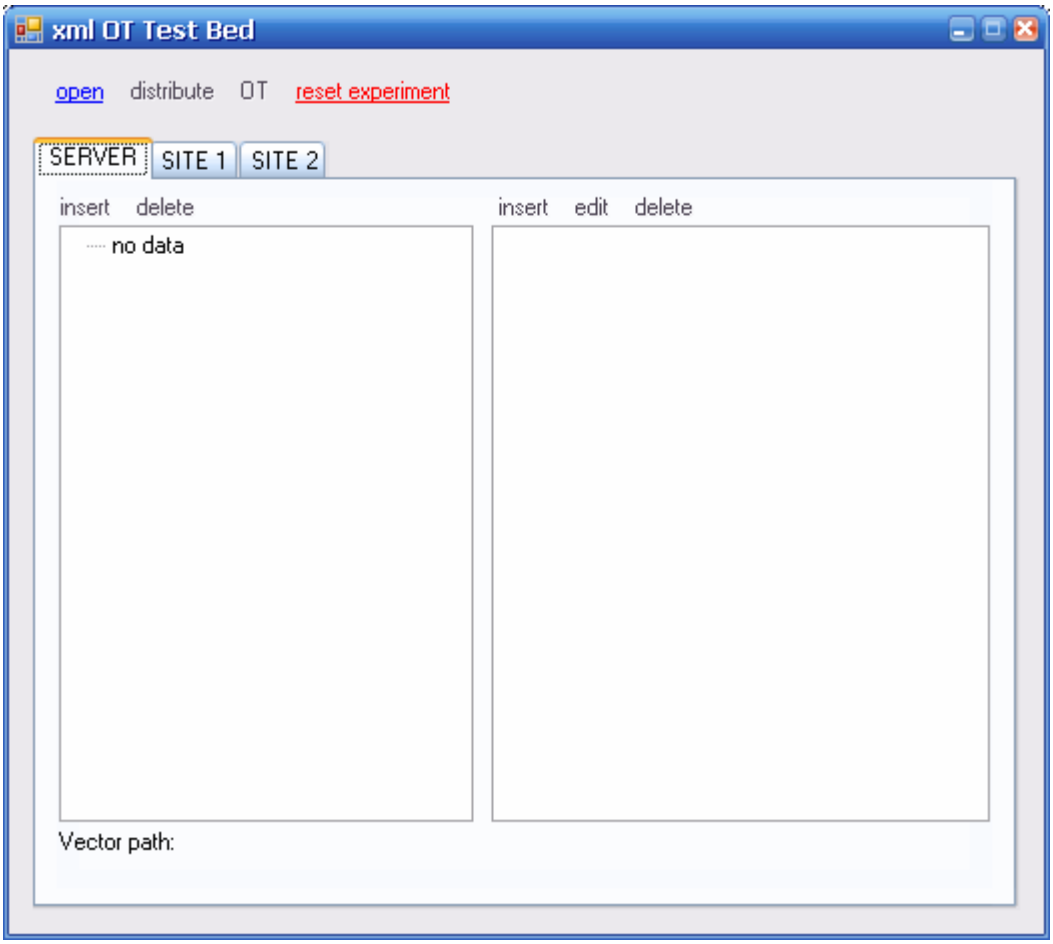
	Pavadinimas	Aprašymas
	NavigateToNode	Randa medyje mazgą pagal vektorinį kelią.
	VectorPathNavigator	Sukuria naują navigatorių konkrečiam medžiui.

6.2.2. Vartotojo sąsaja

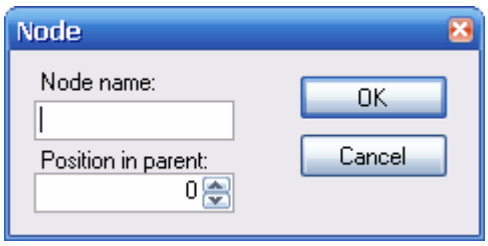
Eksperimentinės programinės įrangos vartotojo sąsaja turi tenkinti šiuos reikalavimus:

- turi leisti užkrauti bet kokius eksperimentinius duomenis;
- patogiai atvaizduoti redaguojamą dokumentą;
- suteikti galimybę redaguoti dokumentą;
- turi leisti, bet kuriuo metu nutraukti atliekamą eksperimentą;
- imituoti tris paskirstytos sistemos mazgus: serverį ir du klientinius mazgus.

Vartotojo sąsaja pateikta 26 paveiksle, naujo mazgo kūrimo forma pateikta 27 paveiksle.



26 pav. Vartotojo sąsaja

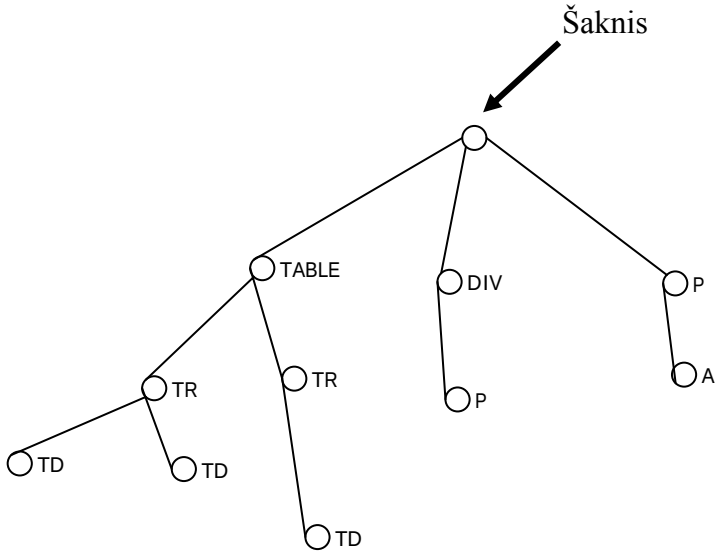


27 pav. Naujo mazgo kūrimo forma

6.2.3. Metodų testavimas

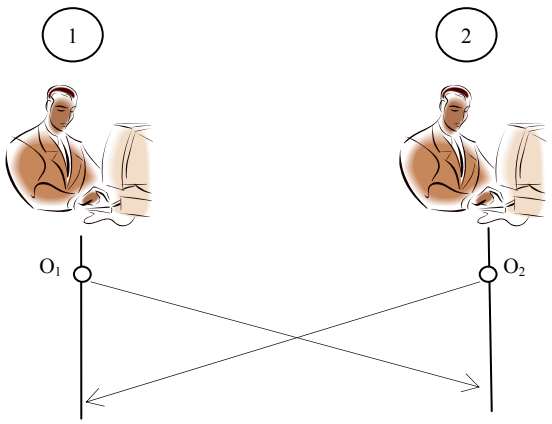
Pasiruošimas eksperimentams

Eksperimentui paruošime duomenis, kurių struktūra pavaizduota 28 paveiksle. Šis duomenų failas bus XML dokumentas – experiment.xml, jo turinys pateiktas 9.1. priede.



28 pav. Eksperimentiniai duomenys

Pirmojo sistemos mazgo veiksmus žymėsime raudona spalva, antrojo – mėlyna. Kiekvienas mazgas galės atlikti po vieną struktūrinę operaciją. Abi operacijos bus konkuruojančios. Koncepcinė operacijų vykdymo schema pateikta 29 paveiksle. Pirma įvykdomos lokalios operacijos, po to – nutolusios. Nutolusi operacija prieš vykdymą yra transformuojama.



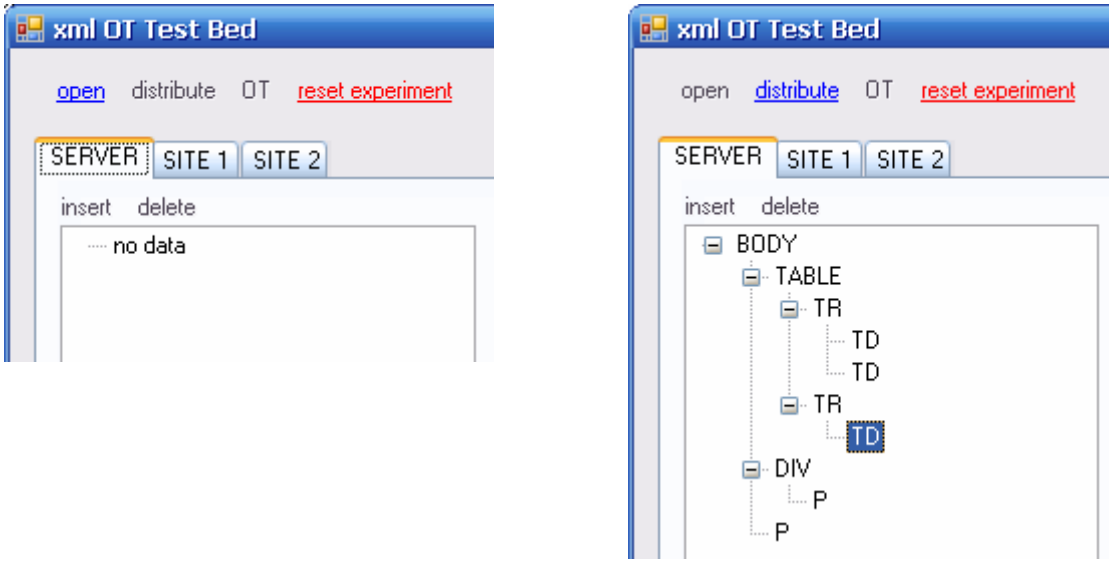
29 pav. Operacijų vykdymo schema

Eksperimentas nr. 1

Duomenys

Šioje eksperimento dalyje atliksime du žingsnius:

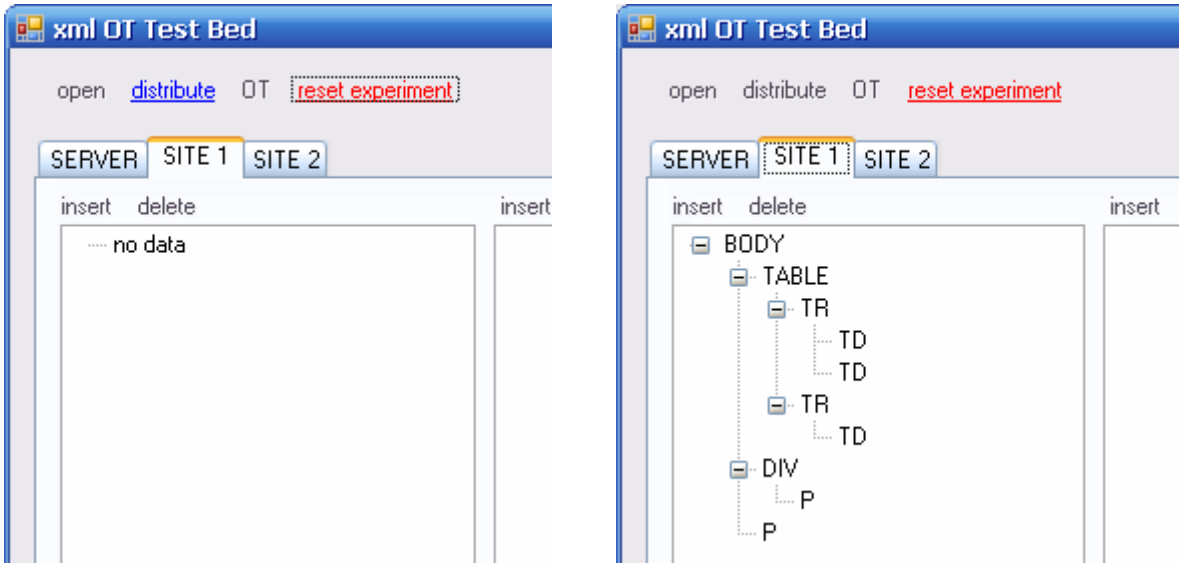
1. OPEN – į serverinę sistemos dalį užkrausime pradinių duomenų failą. Eksperimentinės programos būsenos pavaizduotos 30 paveiksle, (a) – prieš užkrovimą, (b) – po užkrovimo.



(a) (b)

30 pav. Duomenų užkrovimas į serverio dalį

2. DISTRIBUTE – paskirstysime serveryje saugomą dokumentą sistemos klientinėms dalims. Eksperimentinės programos būsenos pavaizduotos 31 paveiksle, (a) – prieš paskirstymą, (b) – po paskirstymo.

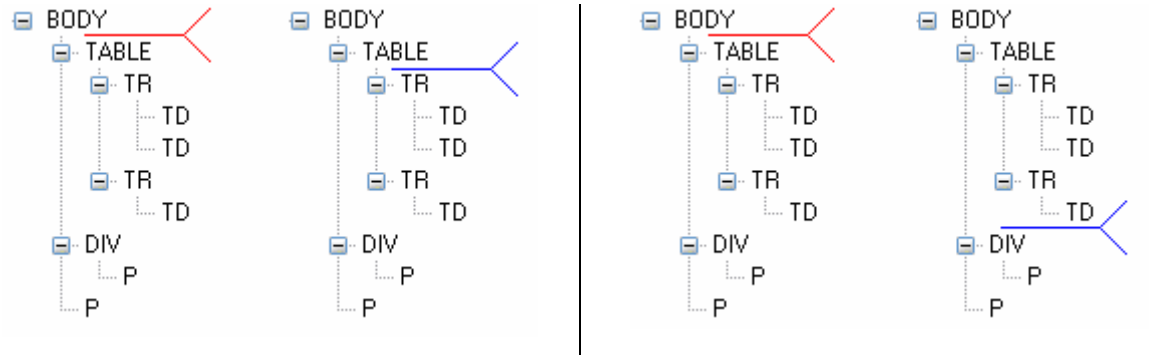


(a) (b)

31 pav. Duomenų paskirstymas

Eksperimento eiga

Šiame eksperimente patikrinsime operacijų *įterpti – įterpti* transformavimo funkcijos veikimą. 32 paveiksle pateikti du terpimo atvejai. (a) atvejis – pademonstruotas variantas, kai terpimas vyksta skirtinguose dokumento mazguose, bet jie yra susieti. (b) atvejis – terpimas vyksta tame pačiame mazge. (Dokumentas kairėje priklauso pirmam sistemos mazgui, o dešinėje – antram sistemos mazgui).

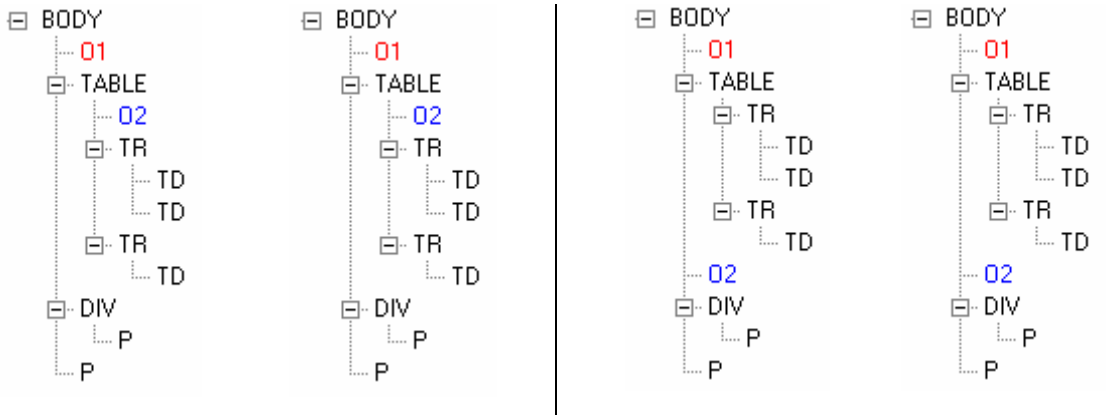


(a) (b)

32 pav. Įterpti – įterpti

Eksperimento rezultatai

Atlikus operacijų transformacijas sistemos klientiniuose mazguose, dokumentų kopijos buvo vienodos. Rezultatai pateikti 33 paveiksle.



(a) (b)

33 pav. OT įterpti – įterpti rezultatai

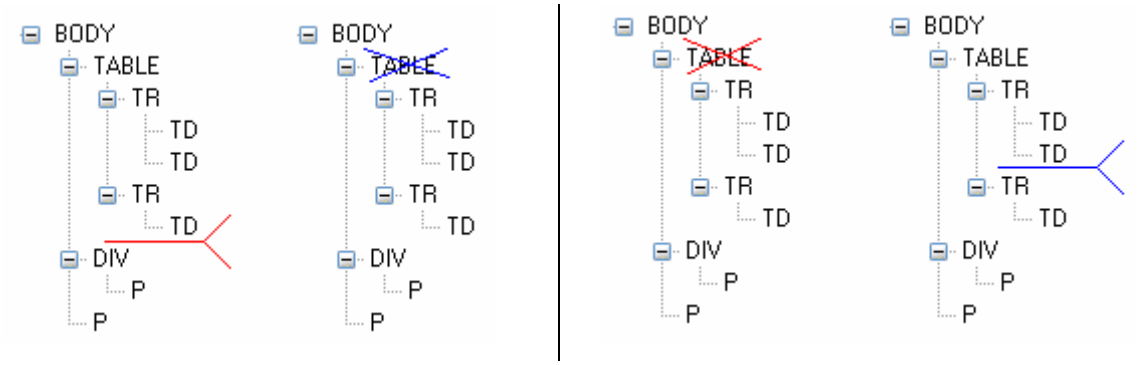
Eksperimentas nr. 2

Duomenys

Žiūrėti eksperimentą nr. 1.

Eksperimento eiga

Šiame eksperimente patikrinsime operacijų *įterpti – ištrinti, ištrinti – įterpti* transformavimo funkcijų veikimą. Viename mazge atliekant trynimo, kitame terpimo operacijas, suveikia operacijų transformavimo funkcijos grąžinančios atitinkamai terpimo ir trynimo operacijas.

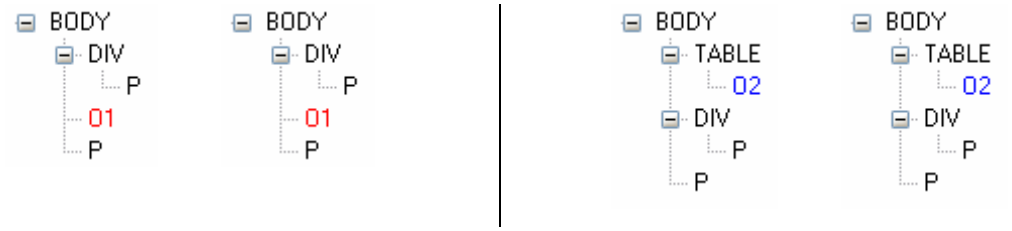


(a) (b)

34 pav. Įterpti – ištrinti, ištrinti – įterpti

Eksperimento rezultatai

Atlikus operacijų transformacijas sistemos klientiniuose mazguose, dokumentų kopijos buvo vienodos. Rezultatai pateikti 35 paveiksle.



(a) (b)

35 pav. OT įterpti – ištrinti, ištrinti – įterptirezultatai

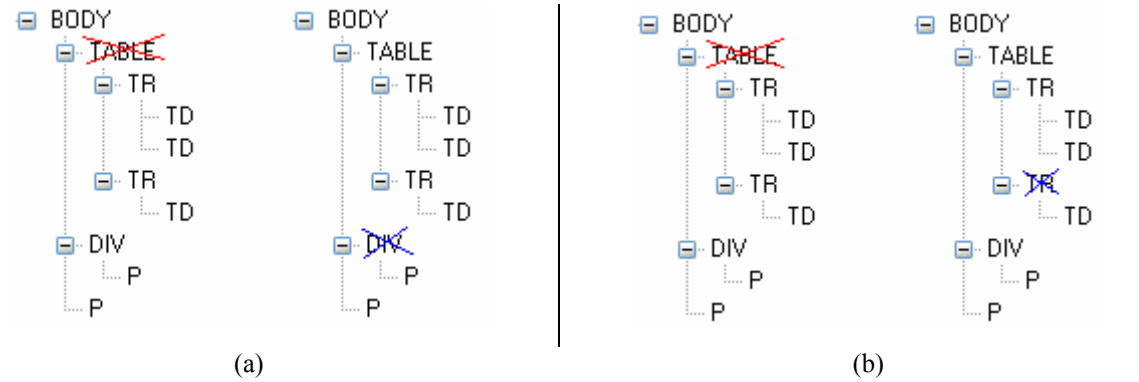
Eksperimentas nr. 3

Duomenys

Žiūrėti eksperimentą nr. 1.

Eksperimento eiga

Šiame eksperimente patikrinsime operacijų *ištrinti – ištrinti* transformavimo funkcijos veikimą. (a) atvejis – terpimas vyksta tame pačiame mazge. (b) atvejis – pademonstruotas variantas, kai trynimas vyksta skirtinguose dokumento mazguose, bet jie yra susieti.

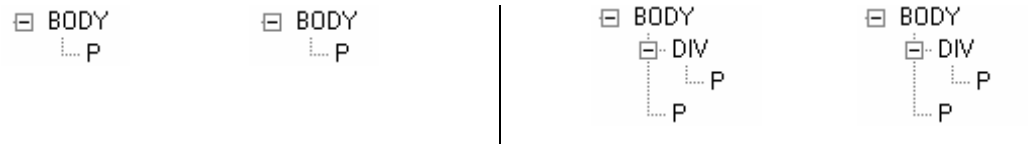


(a) (b)

36 pav. Ištrinti – ištrinti

Eksperimento rezultatai

Atlikus operacijų transformacijas sistemos klientiniuose mazguose, dokumentų kopijos buvo vienodos. Rezultatai pateikti 37 paveiksle.



(a) (b)

37 pav. OT Ištrinti – ištrinti rezultatai

6.2.4. Eksperimento įvertinimas

Sukūrus eksperimentinio XML dokumento duomenų struktūrą, struktūrinių operacijų, kurias galima atlikti su dokumentu, duomenų struktūras, programinę įrangą, imituojančia paskirstytą sistema, praktiškai realizavome operacijų transformavimo algoritmus ir patikrinome jų teisingumą. Iš atliktų tyrimų galime teigti, kad mūsų pasiūlyti operacijų transformavimo algoritmai yra teisingi.

7. Išvados

Išanalizavus grupinio bendradarbiavimo sistemų duomenų sinchronizacijos ir konfliktų sprendimo algoritmus, buvo nustatyta, kad dauguma jų yra patobulinti dOPT algoritmo variantai. Šių algoritmu šeima yra skirta tekstiniams dokumentams redaguoti. Medinės struktūros dokumentams pasiūlytas tik vienas algoritmas treeOPT, kuris vartoja dOPT algoritmus iteraciniu būdu.

Darbe pasiūlyta eilė operacijų transformavimo algoritmų XML dokumentų redagavimo operacijoms. Tai nėra iteracinio pobūdžio algoritmai. Jie paremti redaguojamų šakų viena kitos pozicijų įvertinimu.

Skirtingai nei tekstiniams dokumentams, XML dokumentai turi nestruktūrinius elementu – mazgų atributus. Todėl darbe pateikta mazgų atributų redagavimo operacijų transformavimo algoritmai.

Eksperimentinėje darbo dalyje suprojektuota eksperimentinio XML dokumento duomenų struktūrą, struktūrinių operacijų, kurias galima atlikti su dokumentu, duomenų struktūras, programinę įrangą, imituojančia paskirstytą sistema. Šios programinės įrangos pagalba praktiškai patikrinta operacijų transformacijų algoritmų teisingumas.

8. Literatūra

1. HyperText Markup Language (HTML) Home page. http://www.w3.org/MarkUp/
2. T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language Recommendation http://www.w3.org/TR/2000/REC-xml-20001006, World Wide Web Consortium, Oct. 2000.
3. Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality-preservation and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, 1998.
4. Maher Suleiman, Michèle Cart, and Jean Ferrié. Concurrent operations in a distributed and mobile collaborative environment. *International Conference on Data Engineering* , Orlando, Florida, USA, 1998.
5. Nicolas Vidot, Michèle Cart, Jean Ferrié and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. *Computer Supported Cooperative Work,* Philadelphia, Pennsylvania, USA, 2000.
6. L. Lamport. Time, clocks and the ordering of events in a distributed systems. *Communication of the ACM*, 21(7), 1978.
7. Fidge, L. J. *Timestamp in message passing systems that preserves partial ordering*. Proc. 11th Australian Comp. Conf., (Feb. 1988). Pp. 56-66.
8. Mattern, F. *Virtaul time and global state of distributed systems*. Proc. "Parallel and distributed algorithms" Conf., (Cosnard, Quinton, Raynal, Robert Eds), North –Ilolland, (1988), pp. 215-226.
9. Schmuck, F. The use of efficient broadcast in asynchronous distributed systems. Ph. D. Thesis, Cornell University, TR88-928, (1988), 124 pages.
10. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). A Reformulation of HTML 4 in XML 1.0. W3C Recommendation 26 January 2000, revised 1 August 2002 http://www.w3.org/TR/xhtml1/
11. Overview of SGML Resources. Dan Connolly, Editor of the HTML 2.0 specification. Created Nov 1995 last updated $Date: 2004/03/26 21:54:24 $ by $Author: connolly $ http://www.w3.org/MarkUp/SGML/
12. Claudia-LAvinia Ignat, Moira C. Norrie. Customizable Collaborative Editor Relaying on treeOPT Algoritm. ETH Zurich, Switzerland.

9. Priedai

9.1. Objekto atributų žymenys

Žymuo	Aprašymas
	Objekto savybė.
	Objekto metodas.
	Statinis objekto atributas.

9.2. Eksperimentinis duomenų failas experiment.xml

```
<BODY>
<TABLE>
  <TR>
    <TD/>
    <TD/>
  </TR>
  <TR>
    <TD colspan="2"/>
  </TR>
</TABLE>
<DIV>
  <P/>
</DIV>
<P/>
</BODY>
```