KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

Simonas Mozūra

APSIMOKYMO ALGORITMŲ TAIKYMAS AUTOMATIZUOTAM DOKUMENTO SPECIFIKACIJOS ATKŪRIMUI

Baigiamasis magistro darbas

Vadovas
doc. dr. D. Makackas

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

APSIMOKYMO ALGORITMŲ TAIKYMAS AUTOMATIZUOTAM DOKUMENTO SPECIFIKACIJOS ATKŪRIMUI

Baigiamasis magistro darbas
Informatikos studijų programa (kodas 621I10003)

Vadovas
doc. dr. D. Makackas
2015-05-25

Recenzentas
doc. dr. T. Blažauskas
2015-05-25

Darbą atliko
Simonas Mozūra
2015-05-25

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

(Fakultetas)

Simonas Mozūra

(Studento vardas, pavardė)

Informatikos studijų programa, 621I10003

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Pavadinimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 15 m. Gegužės 25 d.

Kaunas

Patvirtinu, kad mano, **Simono Mozūros**, baigiamasis projektas tema „Apsimokymo algoritmų taikymas automatizuotam dokumento specifikacijos atkūrimui“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

SANTRAUKA

Praktikoje naudojami dokumentų formatai vis dažniau yra pateikiami XML byla arba jų rinkiniu. XML suteikia gerą dokumento duomenų skaitomumą, todėl šis formatas yra plačiai naudojamas informacijos keitimuisi tiek tarp vartojo ir sistemos, tiek tarp sistemų. Šiuo atveju tikimasi, kad tiek siunčiami, tiek gaunami dokumentai bus vienodos bei nekintančios specifikacijos. Problema iškyla, kai norint vykdyti duomenų gavimą arba mainus, duomenis teikiančioji pusė nesuteikia struktūros specifikacijos, pagal kurią duomenys bus siunčiami, ir niekaip neįsipareigoja pastarosios nekeisti.

Kita pasireiškianti problema – duomenų saugojimui naudojami dokumentai, kurių struktūra yra grįsta XML (MS Office, Open Office paketų dokumentai, Magic Draw projektų failai). Šiose failuose naudojamų XML bylų specifikacijos yra arba neskelbiamos viešai, nepilnos arba yra didelės apimties.

Norint sukurti programinius komponentus darbui su minėtais dokumentais, užtrunkama daug laiko. Šio darbo tikslas yra pasiūlyti metodus, kurie padėtų automatizuoti šių bibliotekų sudarymą ir regeneravimą. Šiame darbe tikslo sprendimui pasiūlyta keletas metodų, kurie, naudodamiesi sukaupta informacija apie gaunamų XML bylų struktūrą, gebėtų sugeneruoti programinį kodą nuoseklinimui. Kartu buvo sprendžiama, kaip NET aplinkoje, nestabdant programinės įrangos, būtų galima pakeisti dokumentų apdorojimo biblioteką.

Siūlomų metodų analizė parodė, kad visi nagrinėti metodai gali būti pritaikyti priklausomai nuo keliamo tikslo ir žinių apie gaunamus dokumentus. XSD evoliucijos metodą tikslinga taikyti, kai reikia palaikyti visas galimas dokumentų versijas, tačiau labai išauga turima dokumento specifikacija. Svoriais paremto metodo taikymas užtikrina specifikacijos išgavimą, kuri nedaug skirsis nuo tikros dokumento specifikacijos. Tai galioja, jei dokumento specifikacija kinta ne iš esmės. Maišant šiuos metodus, atsisakoma pilno versijų palaikymo, tačiau sugeneruota dokumento specifikacija geba „išsivalyti“ nuo nebenaudojamų elementų. Taikant genetinio (evoliucinio) algoritmo principus, pavyksta pagerinti gaunamų dokumentų skaitomumą, tačiau išauga skirtumų skaičius tarp sugeneruotos specifikacijos ir originalios dokumento specifikacijos.

SUMMARY

In practice document formats are increasingly used one XML file or a set of XML files. XML provides a good document data readability. This format is widely used in the exchange of information between used application and between systems. In this case, it is expected that both sent and received documents will have uniform and unchanging specifications. The problem comes when in the exchange of data, originating side does not give specifications for the structure on which the data will be sent and has no obligation to maintain the latter.

Another occurring problem – data reading data in the documents that have XML based structure (MS Office, Open Office suite documents, Magic Draw project files). Specifications for these files are not published publicly, are incomplete or are voluminous.

The creation of software components that work with the documents in question, takes a lot of time. The aim of this thesis is to suggest ways to help automate creation of these libraries and their regeneration. In this paper, several approaches are offered on how, using information gathered about incoming XML file structure, to be able to generate code for serialization and how to load this document processing library into .NET environment without interrupting running software.

Analysis showed that all consideration of methods can be applied depending on the goals and the knowledge of incoming documents. XSD evolution based approach is appropriate when you need to maintain all possible versions of the documents. However it increases the size of specification. Weight-based approach provides specifications that are not much different from the real document specifications. This method could be applied if the document specification does not change the substance. Blending these methods disables full version support; however generated document specification is able to abandon obsolete items. When using some ideas of genetic (evolutionary) algorithm, method manages to improve the readability of incoming documents in the cost of differences between generated specification and the original document specification.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

1 ĮVADAS

Naudojami dokumentų formatai vis dažniau yra pateikiami XML byla arba jų rinkiniu. XML suteikia gerą dokumento duomenų skaitomumą, todėl šis formatas yra plačiai naudojamas informacijos keitimuisi tiek tarp vartojo ir sistemos, tiek tarp sistemų. Šiuo atveju tikimasi, kad tiek siunčiami, tiek gaunami dokumentai bus vienodos bei nekintančios specifikacijos. Problema iškyla, kai norint vykdyti duomenų gavimą arba mainus, duomenis teikiančioji pusė nesuteikia struktūros specifikacijos, pagal kurią duomenys bus siunčiami, ir niekaip neįsipareigoja pastarosios nekeisti.

Įprastiems vartotojams taip pat yra pateikiami įvairių dokumentų formatai, vis dažniau sukuriami naudojantis XML arba jų rinkiniu. XML suteikia skaitomumą failo dydžio kaina, kurią šiek tiek apmažina galimai naudojamos glaudinimo bibliotekos. Pavyzdžiui, beveik kasdien sutinkami „Word“ šeimos dokumentai: teksto redaktoriaus (*.docx), skaičiuoklės (*.xlxs), prezentacijų (*.pptx) dokumentai. Taip pat ir atvirojo formato dokumentų šeima, kai kurie specifinės panaudos failai, pvz Magic draw programos generuotas projekto failas.

Viena priežastis, kodėl pereita prie tokio formato, yra XML specifikacijos vienareikšmiškumas, nepriklausomai nuo naudojamos platformos: operacinės sistemos, procesoriaus tipo, darbinės atminties kiekio. Perėjus prie tokio dokumentų formato, palengvinamas taikomųjų programų kūrėjams darbas, integruojant sprendimus į sistemas, nepriklausomai, ar tai būtų „Windows“, „Linux“ ar kuri nors mobilioji operacinė sistema.

Tačiau atsiranda kitos problemos. Jeigu bylą apibrėžiantis formatas yra labai platus, programų kūrėjai priversti naudotis formato specifikacijomis, kurios analogiškai yra ne ką mažiau plačios. Pvz.: pilna Open XML (Word šeimos) formato specifikacija yra daugiau nei 6000 puslapių dydžio. Taip pat esant dideliam užsibrėžtų aprėpti elementų kiekiui, programavimo darbai gali užtrukti pernelyg didelį laiko tarpą. Darbui su Open xml dokumentu bibliotekos dydis yra virš 50000 programinio kodo eilučių. Ir net tokia didelė biblioteka neleidžia jos naudoti mobiliuosiuose įrenginiuose.

Praktikoje susiduriama su dokumentais, kurie aprašomi ne viena XML byla, bet jų aibe (kaip Word, Open Office). Tokių dokumentų analizė rankiniu būdu tampa neįmanoma dėl failų dydžio. Praktikoje susiduriama su nepakankama dokumentacija naudojamiems dokumento formatams. Magic Draw projekto failo struktūros specifikacija nėra prieinama viešai.

Šio darbo pagrindinis tikslas yra pasiūlyti sprendimą, kurio pagalba būtų automatizuotas XML pagrindu struktūrizuotų dokumentų skaitymo bibliotekos ir jų modifikacijų kūrimas, nežinant jų struktūros specifikacijos.

Šiam tikslui pasiekti keliami **uždaviniai**:

- Atlikti analizę metodų, kurie naudojami XML dokumentų aibės bendros specifikacijos radimui;
- Pasiūlyti metodus, leidžiančius atlikti XML dokumentų specifikacijos automatizuotą analizę;
- Pasiūlyti metodiką, kaip, pasinaudojus XML bylos struktūra, sugeneruoti nuoseklinimo (serialization) klases šio failo elementų skaitymui;
- Pasiūlyti metodus, kaip atnaujinti jau sugeneruotą programinį komponentą.

Darbo pagrindinė idėja - turint vienodo formato failų imtį, galima, pasinaudojus XML skaitomumu, bandyti atstatyti, kas yra apibrėžta specifikacijoje: elementų vardus, sąryšius tarp elementų, laukų tipus. Iš visos šios žinių bazės sugeneruoti programinį kodą, kuris gali būti panaudotas komponento kūrimui.

Susiduriama su problema, kad imtis gali būti nepakankama. Todėl dar vienas iš uždavinių šiame darbe - sugeneruotos bibliotekos automatinis atsinaujinimas. Tai yra, jeigu sutikti pavyzdžiai už

pradinės aibės ribų turi šiek tiek kitokią informaciją, numatyti metodus komponento pasitobulinimui su minimaliu žmogaus įsikišimu.

Darbe išsikeltiems uždaviniams spręsti buvo sukurta programinė įranga:

- Analizės modulis vienoda specifikacija duomenis saugantiems XML formato failams skaityti.
- Generatorius, kuriantis komponentą pastarųjų failų skaitymui.
- Metodai, sutikus failus, kurių specifikacija nebuvo padengta pradiniu momentu, komponento atsinaujinimui.

Magistro tezes sudaro 8 skyriai. Antrajame skyriuje aptariama XML formatai bei matematiniai modeliai, kurie gali būti taikomi šių dokumentų formatų analizei, pasiūlant taikymo principus, apžvelgiami jau sukurti sprendimai. Trečiame skyriuje pateiktas sistemos projektas, o jo realizacijos etapai aprašyti skyriuose nuo ketvirto iki šešto. Septintame skyriuje pateikiami eksperimentinių metodų tyrimo rezultatai. Aštuntame - darbo rezultatai ir išvados.

2 APSIMOKANČIŲ ALGORITMŲ TAIKYMAS XML DOKUMENTŲ ANALIZĖJE

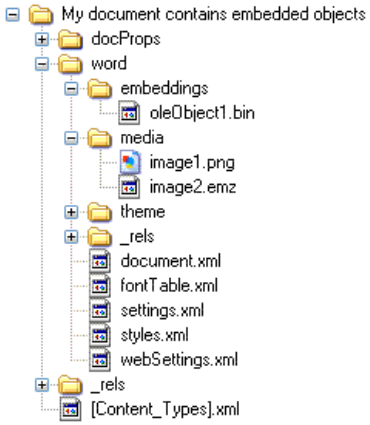
Šiame skyriuje yra pateikiama XML aprašyti dokumentų formatai bei egzistuojantys sprendimai šių dokumentų analizei ir modifikavimui. Taip pat apžvelgiamos metodikos bei algoritmai, kurie gali būti vienokiu ar kitokiu būdu taikyti automatizuojant XML struktūros dokumentų analizės komponento sudarymui. Šio uždavinio sprendimui apžvelgiami dokumentų formatai Open XML ir Open Document. Taip pat apžvelgiami esami programiniai sprendimai minėtų dokumentų analizei bei analizuojama, kaip būtų galima taikyti neuroninius bei genetinius algoritmus dokumente esančių struktūrų analizei.

2.1 Dokumentų formatų apžvalga

Šiuo metu dominuoja dvi teksto redaktorių grupės, t. y., Microsoft Word ir Open Office šeimų teksto redaktoriai. Nuo 2007 metų šie redaktoriai sudaro dokumentus XML formatu. Žemiau pateikiama šių formatų struktūros apžvalga.

2.1.1 Microsoft Office atvirojo XML formato dokumentas

2005 metais Microsoft kompanija perėjo prie Office Open XML (arba OOXML) atvirojo standarto specifikacijos dokumentų, pateikčių bei skaičiuoklių kūrimo. Atviras formatas naudingas organizacijoms, kurios ketina šį formatą realizuoti į programinę įrangą, gebančią skaityti šiuos dokumentus. Taip pat komercinėms ir valstybinėms įstaigoms, kurios šį formatą naudos bei apmokymus vykdantiems asmenims. Pagrindinė kompanija, atlikusi standartizavimo darbus – Ecma International [1]. Nuo Microsoft „Office“ 2007 metų versijos OOXML formatas tapo numatytu visoms po jos sekusioms programinės įrangos versijoms. 2010 versija palaikė ISO 29500 Transitional formato skaitymo ir rašymą galimybes, nuo 2013 yra palaikomas ISO 29500 Strict formatas.



2.1 pav. OOXML Hierarchinė struktūra

OOXML dokumentas yra .zip archyvo failas, turintis hierarchinę aplankų struktūrą (2.1 pav.). Sudedamosios dalys (*.docx failas):

- Document.xml – pagrindinis failas, kuriame yra surašyta visa dokumento informacija. Šiame faile galima rasti visą tekstą, kuris vaizduojamas teksto redaktoriuje.
- Styles.xml – aprašomi stiliai lentelėms, paragrafams, nuorodoms ir t.t.
- Settings.xml – informacija teksto redagavimo įrankiui, apie kai kuriuos nustatymus.
- [Content_Types].xml – MIME tipų informacija apie dokumentą ir jo dalių turinį.

- Media aplankas – saugoma vaizdinė, garsinė arba kita medžiaga naudojama dokumente.
- _rels aplankas – turi savyje document.rels.xml failą, kuriame aprašomi failų, esančių media aplanke, naudojimas.

```
<w:body>
  <w:p w:rsidR="0046449F" w:rsidRDefault="00447965" w:rsidP="0
    <w:pPr>
      <w:pStyle w:val="Title"/>
    </w:pPr>
    <w:r>
      <w:t>
        Kalendorinis darbo planas
      </w:t>
    </w:r>
  </w:p>
  <w:tbl>
    <w:tblPr>
      <w:tblStyle w:val="TableGrid"/>
      <w:tblW w:w="0" w:type="auto"/>
      <w:tblLook w:val="04A0" w:firstRow="1" w:lastRow="0" w:f
    </w:tblPr>
    <w:tblGrid>
      <w:gridCol w:w="4927"/>
      <w:gridCol w:w="4927"/>
    </w:tblGrid>
    <w:tr w:rsidR="00447965" w:rsidTr="00447965">
```

2.2 pav. Document.xml turinio pavyzdys

Document.xml (2.2 pav.) failo daugiausia sudaro paragrafai (žymimas <w:p>) bei lentelės (žymimas <w:t>) kartu su savo vaikais.

2.1.2 Pernešamo dokumento formatas (PDF)

PDF formatas buvo pristatytas dar 1993 Adobe Acrobat programos pristatymo metu. Nuo to laiko PDF formatas buvo de facto globalus standartas, naudojamas saugiam ir patikimam informacijos apsikeitimui [2]. 2007 metų sausio 29 dieną Adobe išleido 1.7 formato versiją paremtą ISO 32000 standartu.

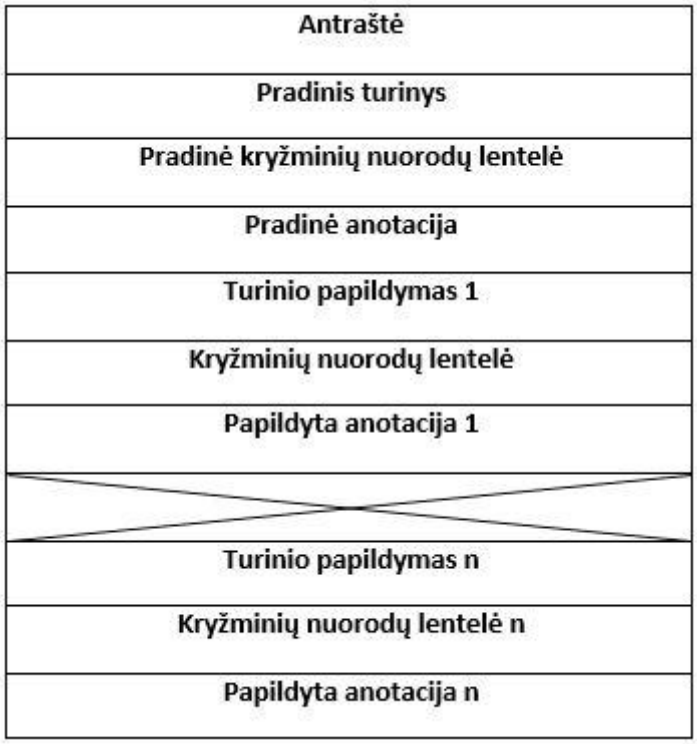
PDF formatas leidžia demonstruoti dokumentus, kurie turi tekstą, paveiksliukus, multimedijos elementus, nuorodas į WEB ir t.t. Taip pat dokumentas gali būti apsaugotas slaptažodžiu, leisti Javascript įskiepus.

2.3 pav. PDF dokumento struktūra [2]

Pagrindinis skirtumas nuo *docx* ar *odt* dokumentų yra tai, kad visa informacija yra laikoma viename faile, o ne zip archyve su sava struktūra. Failą galima atsidaryti su paprastu teksto įrankiu. PDF failo struktūrą (2.3 pav.) galima išskirstyti :

- Antraštė – pirmoji failo eilutė, nurodo PDF dokumento versijos numerį.
- Turinys – saugomi objektai. Tai gali būti tekstas, duomenų srautai, kiti elementai. Šioje dalyje aprašoma informacija, kuri yra rodoma vartotojui.

- Xref lentelė. Kryžminių nuorodų lentelėje saugomos nuorodos į visus objektus, esančius dokumente.
- Anotacija. Nurodo kaip skaityti dokumentą jį atidariusiai programinei įrangai. Pagrindinis šios dalies tikslas yra programinę įrangą nukreipti į nuorodų lentelę. PDF dokumentas yra skaitomas iš apačios į viršų.



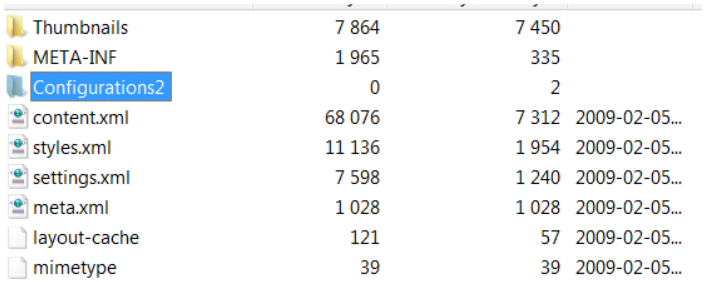
2.4 pav. Naujausia PDF struktūra

Norint palaikyti naujos informacijos pridėjimą PDF struktūra (2.4 pav.), buvo modifikuota taip, kad nauja informacija yra pridedama į dokumento pabaigą. Tokiu būdu ji yra išsaugoma greičiau ir efektyviau.

Pagrindiniai iššūkiai, skaitant ir rašant į PDF dokumentą, iškyla dėl sudėtingos teksto saugojimo struktūros. Tekstas yra koduojamas ASCII formatu, taip pat vienas teksto gabalas negali užimti daugiau nei 65535 baitų. Viršijant šį skaičių, reikia kurti naują objektą.

2.1.3 Atvirojo dokumento formatas

Atvirojo standarto tekstinių dokumentų formatas (OD) buvo sukurtas OASIS įmonės kaip alternatyva tuo metu naudotiems uždariems, patentais saugomiems formatams. Šį formatą buvo galima naudoti be jokių apribojimų. Paskutinė versija (1.2) buvo išleista 2011 metų rugsėjo mėnesį [3]. Formatą sudaro atskiros dalys dokumentams, skaičiuoklėms bei pateiktims.



2.5 pav. ODT struktūra

Open Document Text (ODT) dokumentas yra zip failas, turintis panašią struktūrą (2.5 pav.) kaip ir OOXML failas. Pagrindinės sudedamosios dalys:

- Content.xml. Svarbiausias failas, laikantis visą vartotojui svarbią informaciją, išskyrus kitu objektus (paveiksliukus ir pan).
- Styles.xml. Laiko stilių informaciją. OD labai plačiai naudoja stilius. Stiliai taip pat gali būti aprašyti ir Content.xml faile.
- Meta.xml. Metaduomenis laikantis failas. Tai gali būti autorius, paskutinio pakeitimo data ir t.t.
- Settings.xml. Saugomi nustatymai kaip pritraukimas, kursoriaus pozicija.
- Configurations aplankas. Saugomi dokumente naudojami išoriniai objektai kaip paveiksliukai.

```
<office:body>
  <office:text text:use-soft-page-breaks="true">
    <text:p text:style-name="P1">
      Kalendorinis darbo planas
    </text:p>
    <table:table table:style-name="Table2">
      <table:table-columns>
        <table:table-column table:style-name="TableColumn3"/>
        <table:table-column table:style-name="TableColumn4"/>
      </table:table-columns>
      <table:table-row table:style-name="TableRow5">
        <table:table-cell table:style-name="TableCell6">
          <text:p text:style-name="P7">
            Kauno technologijos universitetas
          </text:p>
```

2.6 pav. Content.xml turinys

Lyginant 2.1.1 skyriuje aprašytą document.xml struktūrą ir content.xml (2.1 pav.) galima matyti, kad pastarojo vardai yra kur kas labiau išplėsti, taip pat praktiškai kiekvienam elementui yra priskiriamas atskiras stilius.

2.2 Metodai ir algoritmai

Žemiau esančiuose skyriuose apžvelgiama algoritmai, kurie gali būti panaudoti apsimokymui dokumento analizės stadijoje.

2.2.1 Neuroniniai tinklai

Dirbtinis neuroninis tinklas – informacijos apdorojimo paradigma, kuri buvo įkvėpta nervinės sistemos (neuronų) veikimo principų.

Dirbtinis neuronas – biologinio neurono abstrakcija, pagrindinis dirbtinių neuroninių tinklų komponentas. Dirbtinis neuronas turi keletą įėjimų (x0, x1, x2, …, xn) ir vieną išėjimą (y). Išėjimo reikšmė gaunama pagal (1) formulę

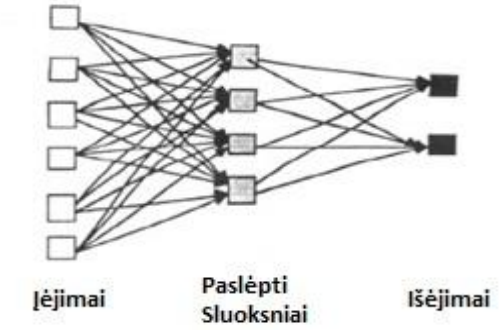
$$y = f(\sum_{j=0}^{n} w_j x_j) \quad (1)$$

Čia w – vadinami įėjimų svoriais, o funkcija f – aktyvavimo funkcija.

2.2.1.1 Neuroninių tinklų tipai, jų mokymas

Neuroniniai tinklai gali būti:

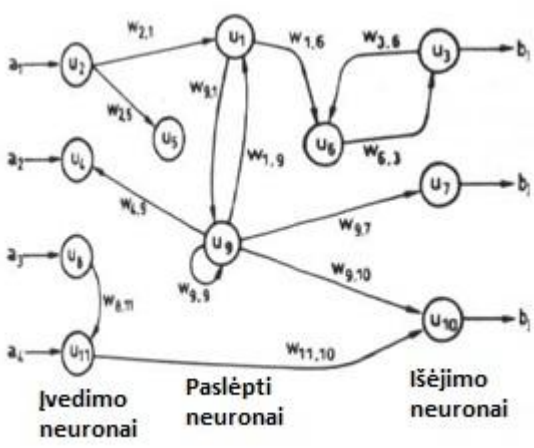
- Tiesioginio sklidimo (2.7 pav. Tiesioginio sklidimo tinklas)



2.7 pav. Tiesioginio sklidimo tinklas [4]

Tiesioginio sklidimo tinkle išeities reikšmės perduodamos tik tolimesniems tinklo elementams. Juose nėra atgalinio ryšio bei ciklų. Galiausiai gaunama išeities reikšmės.

- Tinklai su atgaliniu ryšiu (2.8 pav.)



2.8 pav. Tinklas su atgaliniu ryšiu [4]

Tinkle su atgaliniu ryšiu tolimesnių sluoksnių, to paties sluoksnio neuronai gali siųsti signalus vieni kitiems ar patys sau bei sudaryti ciklus. Jie yra gerokai sudėtingesni, tačiau lengviau kintantys.

Neuroninių tinklų apsimokymą galima skaidyti į:

- Prižiūrimas mokymasis. Žinomas „teisingas“ atsakymas. Tinklui pateikiamas įvedimo išvedimo rinkiniai, tinklas koreguojamas taip, kad ateityje pateiktų teisingą atsakymą.
- Neprižiūrimas mokymasis. Tinklas nežino, ar gautas atsakymas yra teisingas ir klasifikuoja duomenis pagal savo vidines žinias.

Neuroniniams tinklams apmokyti taip pat gali būti panaudoti genetiniai, evoliuciniai algoritmai.

2.2.2 Evoliuciniai ir genetiniai algoritmai

Evoliuciniai algoritmai stengiasi spręsti sudėtingus, atkartodami Darvino evoliucijos, procesus. Tai galima apibūdinti, kaip krūvą padarų, vadinamų individais, ieškančių sprendimo erdvėje ir taip evoliucionuojančių. [5]

Genetiniai algoritmai yra tam tikra evoliucinių algoritmų klasė. Ji naudoja gamtoje egzistuojančius gyvybės evoliucinius mechanizmus: paveldėjimą, mutaciją, natūraliąją atranką ir rekombinaciją.

Tiek evoliuciniai, tiek genetiniai algoritmai prasideda nuo pradinės sprendimų (individų) populiacijos sudarymo. Sprendimas yra išreiškiamas chromosomomis (individais). Tada iš vienos populiacijos yra imami individai ir sukuriama nauja populiacija. Tikima, kad naujoji karta bus

geresnė už senąją. Imant individus naujai populiacijai sudaryti, yra atsižvelgiama į tai, kad būtų kuo tikslesnis sprendimas, tada individas gali būti pasirinktas. Toks procesas yra kartojamas iki užbrėžto tikslo.

Algoritmo eskizas:

1. Sugeneruoti n chromosomų, kurios reprezentuoja problemų sprendimus, ir sudarys pradinę populiaciją.
2. Apskaičiuoti kiekvienos chromosomos (sprendimo) x tinkamumą f(x).
3. Kuriama nauja populiacija:
   3.1. Atrinkti ir suporuoti individus į n/2 porų.
   3.2. Sukryžminti x ir y chromosomas, kad gautume naujus individus.
   3.3. Mutuoti chromosomas.
   3.4. Patalpinti naujas chromosomas i populiaciją.
4. Kartoti algoritmą nuo 2 punkto tol, kol yra patenkintos tam sprendinį tenkinančios sąlygos.
5. Grąžinti sprendimą. [5]

2.2.2.1 Chromosomos kodavimas

Yra gana daug chromosomų kodavimo būdų, dažniausiai jie yra susiję su sprendžiama problema. Dažniausiai yra naudojamas dvejetainis kodavimas, simbolių eilutė, slankaus taško skaičiai, kiti būdai (pvz. masyvas) [6].

1 0 1 1 0 0 1 0 1 1 1 0 1

2.9 pav. Dvejetainis kodavimas

Dvejetainio kodavimo atveju (**Error! Not a valid bookmark self-reference.**) chromosomą sudaro dviejų bitų, 0 ir 1, eilutė. Toks metodas dėl savo paprastumo buvo naudojamas pačiuose pirmuose genetinių algoritmų bandymuose ir vis dar lieka gana populiarus. Šia eilute galima užkoduoti praktiškai bet ką, tačiau šis metodas nėra natūralus daugumos uždavinių sprendimui.

9 7 13 12 2 3 4 8 9

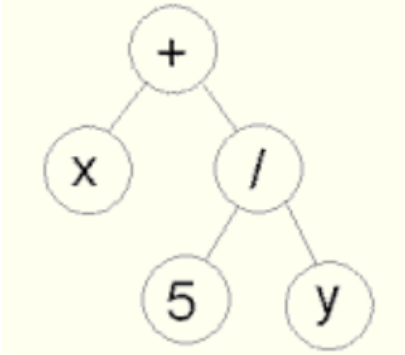
2.10 pav. Perstatymų kodavimas

Kitas metodas yra perstatymų kodavimas. Jis gali būti naudojamas rikiavimo problemose, tokiose kaip keliaujančio pirklio. Tokiame kodavime chromosoma yra numerių eilutė (2.10 pav.), nusakanti eilę miestų, kuriuose jau apsilankė pirklys. .

(atgal), (atgal), (dešinėn), (priekin), (kairėn)

2.11 pav. Reikšminis kodavimas

Problemose, kuriose yra naudojamos sudėtingesnės reikšmės, kaip pvz. realūs skaičiai, bitų kodavimas būtų per sudėtingas ir nepraktiškas. Todėl yra naudojamas tiesioginis kodavimas, ir kiekvienas genas atspindi tikrąją reikšmę. Čia chromosomos eilutė gali būti sudaryta iš bet kokių reikšmių, nuo realių skaičių iki objektų. Reikšminis kodavimas yra tinkamas specifinėms problemoms spręsti, tačiau dažniausiai reikalauja kurti specifinius kryžminimo ir mutacijos operatorius[]. Taip pat toks kodavimas tinkamas apmokyti neuroniniams tinklams.

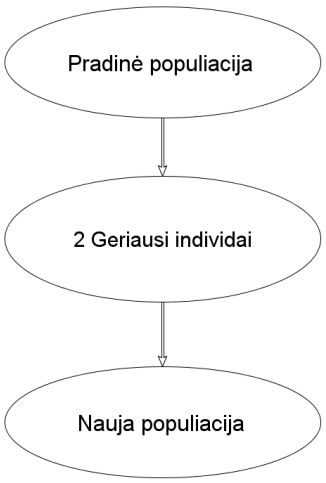


2.12 pav. Kodavimas medžiu [7]

Medžio kodavimas yra daugiausia naudojamas vystyti programas arba išraiškas. Medžio kodavime (2.12 pav.) chromosoma yra kažkokių objektų medis. Tai gali būti funkcijos, programinio kodo komandos programavimo kalbose [7]. Šitoks būdas yra labai geras programų skaidyme į reikšmines dalis.

2.2.2.2 Selekcija

Selekcija – procesas, kai iš populiacijos yra parenkami 2 tėvai kryžminimui.



2.13 pav. Selekcijos schema

Prieš kryžminimo etapą chromosomos gali būti įvertinamos panaudojus tinkamumo funkciją. Geriausiai įvertinti individai yra paimami kryžminti. Kuo didesnis parinkimo slenkstis, tuo geresni individai yra parenkami.

Atrankos spaudimas – laipsnis, nuo kurio geresni individai turi didesnį palankumą.

Dauginimosi imtis – individai, kurie yra kryžminami.

Galimi selekcijos metodai:

- *Ruletės metodas.*

Algoritmo žingsniai:

1. Susumuoti visas reikšmes pagal tinkamumą. Tegu sumos rezultatas T.
2. Kartoti:
    a. Pasirinkti atsitiktinį skaičių *k* tarp 0 ir T
    b. Eiti per individus sumuojant reikšmes. Ta reikšmė, kurios pridėjimas sumą padaro didesne nei *k* yra parenkama.
3. Metodas kartojamas kol pasiekiamas norimas individų skaičius

Ruletės metodas yra nesudėtingas realizuoti, bet sukuria labai daug triukšmo. Šis algoritmas palankus chromosomoms su didesne tinkamumo funkcijos reikšme. Tačiau, pavyzdžiui, jeigu vienos chromosomos reikšmė yra labai didelė, ji yra beveik garantuotai parenkama, tokiu būdu gali būti per greitai pasiektas konvergavimas. Taip pat gali būti atmetami geriausi individai sprendimui. [8]

- *Atsitiktinis parinkimas***.** Šiuo atveju parenkami 2 individai iš populiacijos ir sukryžminami.
- *Pasirinkimas pagal rangą*

Algoritmo žingsniai:

1. Surikiuoti individus pagal tinkamumo funkcijos reikšmę
2. Kiekvienai chromosomai priskirti parenkamumo tikimybę r.
3. Imti 2 individus.
4. Generuoti atsitiktinį skaičių R.
5. Jei r < R imti pirmą individą, jei r >=R imti antrą.
6. Tai kartoti ir antram tėvui.

Ši technika neleidžia per greito konvergavimo, taip pat sprendžia imties plėtimo problemas, kaip stagnacija ir išankstinis konvergavimas. Pasirinkimas pagal rangą yra labiau praplečiamas metodas lyginant su kitais.[8]

- *Turnyru paremta selekcija*

Turnyro selekcijos metu atrankos spaudimas yra palaikomas rengiant turnyrą tarp s varžovų (čia s – turnyro dydis). Laimėtojas, individas su didžiausiu tinkamumu tarp s dalyvių, yra įdedamas į dauginimosi imtį. Bendras dauginimosi imties tinkamumo vidurkis yra didesnis nei vidutinis populiacijos tinkamumas. Tokiu būdu yra užtikrinamas tolimesnis tinkamumo augimas [9].

2.2.2.3 Kryžminimas

Kryžminimas ir mutacija yra pagrindiniai genetinio algoritmo operatoriai, lemiantys problemų sprendimų vystymąsi. Jų tipų pasirinkimas ir realizavimas priklauso nuo problemos bei naudojamų chromosomų kodavimo.

Kryžminimas – metodas, kuris iš 2 tėvų sudaro naują individą (vaiką) populiacijai.

Bendras kryžminimo algoritmas:

1. Parinkti 2 tėvus
2. Perkirsti chromosomą ties atsitiktine vieta
3. Kopijuoti informaciją iš abiejų tėvų.

Paprasčiausias būdas vykdyti kryžminimą – pasirinkti atsitiktinę vietą iš vieno ir kito tėvo, nukopijuoti informaciją iš pirmojo pradžios ir antrojo galo, tai sukuriant naują vaiką.

Galimos kryžminimo operacijos:

- Per vieną tašką;
- Per 2 taškus;
- Per N taškų;

- Normalinis kryžminimas. Naudojama atsitiktinai generuojama kaukė. Abu tėvai turi būti vienodo ilgio.
- Trijų tėvų kryžminimas. Imama geno reikšmė iš 2 tėvų. Jeigu ji sutampa, yra kopijuojama, jeigu ne, kopijuojama iš 3 – čio tėvo.
- Rikiuotas kryžminimas.

2.2.2.4 Mutacija

Po kryžminimo operacijos seka mutacija. Mutacija neleidžia algoritmui sustoti lokaliame minimume. Mutacija leidžia atgauti prarastą genetinę informaciją, taip pat gali būti panaudota sumaišyti esamą. Mutacijos operatorius tradiciškai laikomas paprastos paieškos operatoriumi.[10]

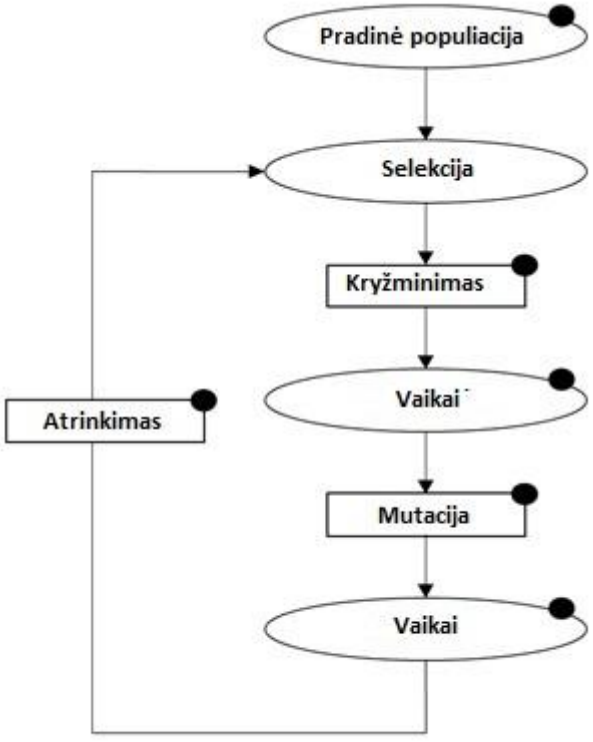
Mutacijos technikos:

- Apvertimas. Reikšmė pakeičiama priešinga.
- Apkeitimas. 2 genai sukeičiami vietomis.
- Apgręžimas. Chromosoma yra apgręžiama į kitą pusę.

2.2.3 Idėjiniai (memetic) algoritmai

R. Dawkins knygoje „The Serfish Gene“ nurodė, kad „meme“ yra imitacijos vienetas kultūriniame perdavime, kuris kai kuriais aspektais yra panašus su genais [11].

Jeigu genetiniai algoritmai bando atkartoti biologinę evoliuciją, tai idėjiniai – kultūrinę [11]. Idėjiniai algoritmai yra kombinacija tarp globalios paieškos ir lokalios, euristinės paieškos, kurią atlieka kiekvienas iš individų.



2.14 pav. Idėjinio algoritmo šablonas [13]

Idėjini algoritmai savo struktūra (2.14 pav.) yra beveik identiški evoliuciniams/genetiniams algoritmams. Turimi visi tie patys žingsniai: populiacijos individų kryžminimasis, mutacijos palikuonių gavimas. Pagrindinis skirtumas, jog kiekviename iš žingsnių gali būti sprendžiama lokali problema. Pavyzdžiui, pradinė populiacija gali būti gauti specifinės problemos euristinius sprendimų,

kryžminimosi operacija gali būti apribota kažkokios srities ar atvaizdavimo specifikos tam, kad suteiktų geresnius paieškos metodus evoliuciniai algoritmo daliai [13].

2.2.4 Genetiniai algoritmai dirbant su XML dokumentais

Genetiniai algoritmai panaudoti XML schemų ir XML paremtų duomenų atvaizdžiams į reliacines duomenų bazes. Dauguma egzistuojančių metodų atvaizdžiams kurti pvz. modeliais paremti, nenaudoja užklausų istorijos geresniems rezultatams pasiekti. Eksperimentai parodė, kad genetinio algoritmo parengti atvaizdžiai buvo efektyvesni negu įprasti metodai. [14]

Taip pat yra pavyzdžiai, kai genetiniai algoritmai yra naudojami ieškoti reikšminės informacijos iš duomenų pateiktų XML formatu [15] ir pagreitinti informacijos paiešką indeksuojant XML laukus [16].

Genetiniai algoritmai taip pat gali būti panaudojami norint pasiekti didesnį programos testavimo padengimą. Naudojant XML dokumentus ir evoliucionuojant jų struktūrą, buvo pasiektas kur kas didesnis programinio kodo padengimas testais automatiniu būdu, lyginant su įprastais metodais. [17]

2.2.5 Kaip panaudoti metodus?

Darbo pagrindinis tikslas – neturint jokių žinių apie dokumento struktūros specifikaciją, ją atkurti tik iš pateiktų dokumentų.

Imant neuroninį tinklą, galima pastebėti, kad pradines viršūnes galime laikyti aukščiausio lygmens XML failo elementais, o jų vaikus – žemesniojo lygmens. Galiausiai reikia išvesti skaitymo ir rašymo funkcijas. Šiam darbui reikės naudoti tinklą su atgaliniu ryšiu, nes elementai gali saugoti vieni kitus.

Pagrindinė neuroninio tinklo paskirtis – surasti ryšius tarp elementų, taip pat minimizuoti funkcijų kiekį, reikalingą skaityti vienodus atributus.

Genetiniai algoritmai gali būti pritaikyti tiek neuroniniams tinklams apmokyti, tiek pastarųjų gautą rezultatą apdoroti dar kartą. Galimas variantas bandyti naudoti didelį skaičių tinklų pastarųjų rezultatą apjungti, mutuoti ir t.t.

Idėjų algoritmų principai gali būti taikomi komponento ir sistemos gyvavimo ciklo palaikymui bei regeneravimo procesams atlikti. Tokiu atveju vienos evoliucijos pakopos rezultatas bus komponentas, o, atsiradus naujiems duomenims, procesas būtų kartojamas jau žinant, kas gali XML schemos evoliucijoje sistemose.

Norint „suspėti“ su besikeičiančiamomis XML specifikacijomis, yra atliekami tyrimai, kaip šie pasikeitimai įtakoja sistemas ir kaip būtų galima sumažinti šių pasikeitimų žalą [18]. Panašiai yra peržvelgiama evoliucija, kai schemos pateikiamos dokumento tipo aprašais (DTD). Nurodomi operatoriai, kurie palengvintų šio tipo specifikacijų tobulinimą, tačiau nežiūrima į sudaromas problemas seniems dokumentams [19]. Dažniausiai schemos evoliucija yra nagrinėjama duomenų bazių lygyje [20].

2.3 Metaprogramavimas

Metaprogramavime sistema, meta-programa, manipuliuoja programomis-objektais. Meta-programa gali kurti programas, apjungti šių programų fragmentus į didesnes programas-objektus, stebėti struktūrinius šių programų pasikeitimus [21].

Meta-programos gali būti dviejų tipų:

- Generuojančios programos;

- Analizuojančios programos.

2.3.1 Generuojančios meta-programos

Programa generatorius dažniausiai naudojama susieti aibę panašių problemų su aibe panašių sprendimų. Kiekvienai problemai bandoma ieškoti atitinkamo sprendinio. Tai yra vykdoma konstruojant kitą programą (programą-objektą). Dažniausiai generuota programa yra labiau specializuota spręsti nurodytą problemą ir veikia efektyviau negu bendras sprendimas.

Programų generatoriai gali būti:

- Statiniai. Sugeneruotas programinis kodas rašomas į diską ir kompiliuojamas įprastiniais kompiliatoriais.
- Generatoriai, veikiantys vykdymo metu. Programa parašo (sukonstruoja) kitą programą, tada iškart ją (jas) įvykdo.[18]

2.4 Save regeneruojančios sistemos

2004 metais buvo sukurta save regeneruojančios sistemos. (SRS) programa finansuota JAV gynybos pažangių tyrimų projektų agentūros. Pagrindinis tikslas – sukurti karybai sistemą, kuri gebėtų teikti savo kritinės svarbos paslaugas, nepaisant kibernetinės atakos ar galimų klaidų sistemos viduje. SRS turėjo užtikrinti, kad sistema mokysis iš savo klaidų.

Šios programos 4 pagrindinės zonos:

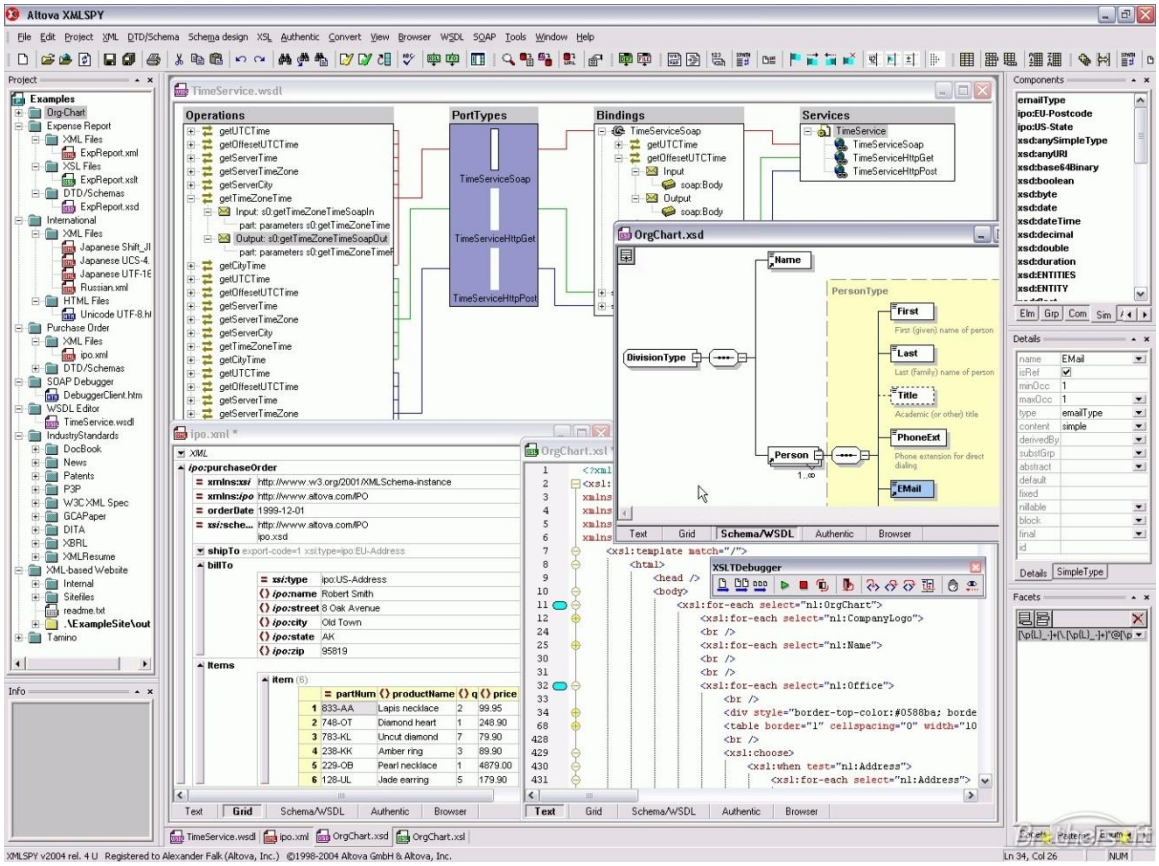
- Biologijos įkvėptas įvairumas (Biologically-Inspired Diversity). Automatiškai pagaminti 100 skirtingų, tačiau funkciškai ekvivalenčių elementų taip, kad ne daugiau 33 versijos turėtų tuos pačius defektus.
- Kognityvinis imunitetas ir regeneracija. Tiksliai nustatyti 10% sistemos problemų priežasčių ir efektyviai išspręsti bent pusę jų.
- Grūdėtas (granular), plečiamas pasikartojimas. Trigubas sumažinimas vėlavimo gaunant pastovius duomenis, toleruoti iki 5 trikių pagrindiniame serveryje, tuo pat metu išlaikant duomenų gavimą ir apdirbimą iki 10000 vartotojų.
- Tvarkymasis su užpuolimais. Atmušti arba sulėtinti 10% puolimų iš vidaus.

Šio projekto rezultatas PMOP ir AWDRAT sistemos, kurių tikslai atmušti atakas ir surasti kompromisus sistemoms. Abi pasikliauja stebėjimu, diagnostika ir saviadaptacija [12].

2.5 Esami sprendimai

2.5.1 XMLSPY

XMLSPY yra XML yra teksto redaktorius leidžiantis kurti, analizuoti, tvarkyti XML bylas, generuoti XSL, XSLT, OOXML ir kitus failų tipus. Taip pat leidžiantis atvaizduoti XML struktūrą: ryšius, elementų tipus ir t.t (2.15 pav.).



2.15 pav. XMLSPY programos pavyzdys

XMLSPY yra mokama programa. Licencijų kaina gali būti nuo 400$ iki 800$. Nemokamai suteikiama 1 mėnesio licencija. Pagrindiniai privalumai:

- Gali dirbti su keletu XML failų, gali dirbti su OOXML (Office šeimos dokumentais);
- Gali generuoti programinį kodą;
- Iš dalies gali patobulinti programinį kodą (reikia per naują kelti failus arba pridėti naujus, tada generuoti programinį kodą iš naujo).

Trūkumai:

- Yra tik kaip atskira programa, menkos sistemų integracijos galimybės;
- Rankinis darbas.

2.5.2 Xsd2Code

Xsd2Code yra laisvai prieinama ir laisvai platinama biblioteka, leidžianti programinio kodo generavimą iš XSD struktūros bylų. Ši biblioteka platinam pagal MIT licenciją, todėl gali būti integruojama, modifikuojama ir naudojama sistemose be jokių apribojimų.

Privalumai:

- Programinio kodo generavimas iš XSD failo;
- Gali generuoti klases įvairiais būdais, priklausomai nuo to, kokioje programinėje įrangoje bus sugeneruotas programinis kodas, naudojamas (įprastos nuoseklinimo klasės, WEB paslaugų kontraktai, MVVM klasės);
- Nemokama, lengvai integruojama ir naudojama;
- Gali generuoti standartinius metodus.

Trūkumai:

- Naudojasi tik XSD failais;

- Negali atnaujinti esamo programinio kodo.

2.5.3 Visual Studio šablonų kalba T4

T4 šablonų kalba yra plačiai naudojama Microsoft Visual Studio aplinkoje bei bibliotekose, kurios generuoja pagalbinį programinį (pvz. Entity Framework). Šablonai įprastai naudojami su Visual Studio pagalba, tačiau gali būti panaudoti ir neesant šios programos.

Privalumai:

- Leidžia generuoti programinį kodą, tačiau generatoriaus naudojamus šablonus reikia pasirašyti pačiam;
- Gali generuoti bet kurios kalbos programinį kodą;
- Didelės praplėtimo, prisitaikymo galimybės.

Trūkumai:

- Žemo lygio sprendimas.

2.5.4 XML Schema Definition Tool (Xsd.exe)

Kartu su Visual Studio yra platinamas XML Schema Definition Tool, leidžiantis generuoti XSD dokumentus iš XML bylų, taip pat leidžia generuoti programinį kodą iš XSD failo. Tai yra konsolinė programa, kuri yra platinama su Visual Studio.

Privalumai:

- Programinio kodo generavimas iš XSD dokumento;
- Gali generuoti XSD dokumentą iš XML bylos.

Trūkumai:

- Platinimo principas (tik su Visual Studio);
- Jokių praplėtimo galimybių.

2.5.5 Apibendrinimas

2.1 lentelė. Esamų sprendimų analizės rezultatai

Įrankis	**XML skaitymo galimybės**	**XML analizavimo galimybės**	**Generavimas**	**Atnaujinimas**
Xml spy	Yra	Yra	Yra	Yra, dalinai
Xsd2Code	Yra xsd	Nera	Yra	Nėra
T4	Žemo lygio sprendimas, visos funkcijos kuriamos programuotojo			
Xsd.exe	Reikia versti į xsd	Ne	Taip, iš xsd	Ne

Nors komercinis sprendimas XMLSPY turi beveik visa reikalaujamą funkcionalumą, susiduriama su problema, jog dokumentas, iš kurio generuojamas programinis kodas, turi būti idealus, t.y., turintis visus elementus, atributus ir ryšius. Kitu atveju, susitikus naują dokumentą, reikėtų žmogaus įsiterpimo, kuris tą naują dokumentą keltų į projektą, iš kurio buvo generuotas programinis kodas bei atnaujinti tą sugeneruotą programinį komponentą.

Kiti įrankiai pradeda reikalauti specifinių dalykų: kitokio failo tipo (XSD), pačiam vartotojui pasirašyti šablonus ir jų logiką (T4)

2.6 Dinamiškai prijungiami komponentai

Dinamiškas bibliotekos užkrovimas yra procesas, kada programa savo vykdymo metu gali prisijungti biblioteką į savo adresų erdvę, peržiūrėti ir, esant reikalui, naudotis prijungta biblioteka. Ši funkcija yra plačiai naudojama „Linux“ aplinkoje.

Didžioji dauguma programavimo kalbų leidžia prijungti išorinius programinius komponentus dinamiškai pagrindinės programos vykdymo metu. [22]

.net

.Net sistemoje išoriniai komponentai yra įvardijami kaip DLL (Dynamic Language Library). Šioje sistemoje DLL yra prijungiami naudojant Assembly.Load metodą iš System.Reflection bibliotekos. Šitaip prijungta biblioteka negali būti panaikinta vykdymo metu arba į jos vietą užkrauta kita biblioteka tokiu pat vardu.[23] Vienintelis būdas pasiekti tokį rezultatą yra išmesti visus iki vieno užkrautus komponentus.[24]

Visi šie metodai tinka programavimo kalboms: Visual Basic, Visual C# ir Visual C++.

Java

Java programavimo kalboje yra 2 būdai prijungti išorines bibliotekas:

- Naudojant System.Load ir System.LoadLibrary metodus. Šiuo atveju gali būti užkrauti arba .so failai arba sistemoje (priklauso nuo OS) jau egzistuojantys failai;
- Jar failų dinaminis įkėlimas. Naudojami metodai iš java.net.URLClassLoader [25].

Vienintelis būdas pakeisti užkrautą komponentą yra, jeigu visi naudojami objektai iš jo yra surinkti šiukšlių rinktuvės.

Python

Kadangi Python nėra kompiliuojama, o interpretuojama programavimo kalba, išorinių komponentų pridėjimas yra kur kas paprastesnis.

- Programiniame kode įrašyti *import* sakiniai. Lyginant su C#, C++ arba Java kalbomis, kur naudojamos bibliotekos turi būti aprašytos failo viršuje, Python leidžia užkrauti modulį bet kurioje programinio kodo vietoje. Tai turi minusų, dėl klaidų tikimybės (nėra užkraunamo modulio, klasės ir pan) galimas modulių kešavimas;
- Naudojant importlib.import_module. Leidžia importuoti ir pkg failus [26];
- Imp modulis. Leidžia importuoti .py klases, taip pat ir kompiliuotus python failus .pyc.

Python leidžia per naują užkrauti įkrautus modulius naudojant reload funkciją.

3 DOKUMENTŲ SPECIFIKACIJOS ATKŪRIMAS

3.1 Reikalavimai sistemai

Funkciniai reikalavimai:

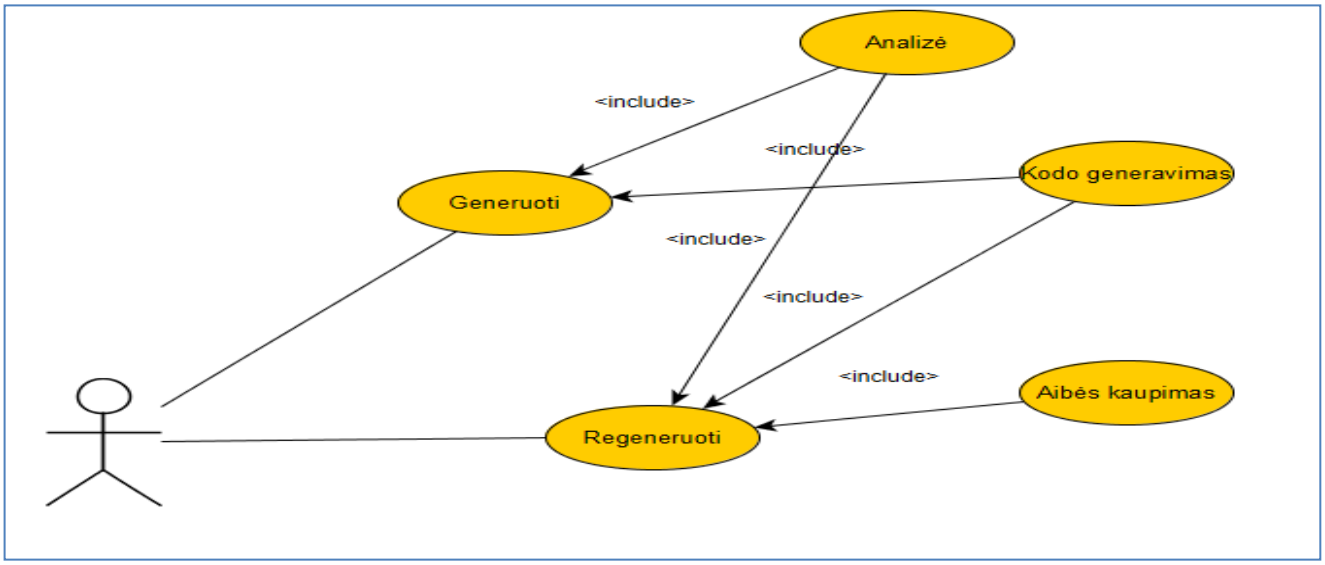
1. Sugeneruoti testinę dokumentų imtį pagal nustatytus parametrus.
2. Komponentą nusakančių kokybės parametrų skaičiavimas.
3. Komponentas privalo save regeneruoti, jei netenkina kokybės reikalavimų.
4. Komponentas turi gebėti modifikuoti dokumentą, kurti naują dokumentą.

Nefunkciniai reikalavimai:

1. Leidžiami dokumentų formatai yra OOXML ir OD dokumentų šeimos.
2. Sistemos prototipas realizuotas .net arba Python programavimo kalba.
3. Prototipas veikia Windows operacinės sistemos platformoje.

3.2 Sistemos panaudojimo atvejai

Numatomą sistemos dalį, atsakančią už specifikacijos atkūrimą, veikimą, galima išskaidyti į generavimą ir sistemos regeneraciją (3.1 pav.).

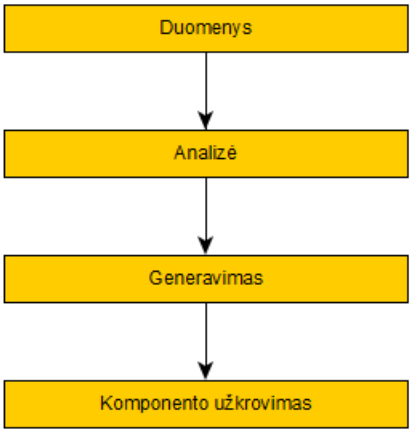


3.1 pav. Sistemos panaudos atvejų diagrama

Tiek generavimo, tiek regeneravimo atveju yra vykdoma dokumento analizė bei kodo generavimas. Regeneravimui taip pat gali būti panaudota sukaupta aibė dokumentų, kurie vėliau bus panaudoti analizei.

3.3 Pagrindiniai specifikacijos atkūrimo etapai

Numatomą funkcionalumą galima skaldyti į 4 vienas po kito einančius etapus (3.2 pav.): duomenis, analizę, programinio kodo generavimą, sukurto komponento užkrovimą į sistemą. Kiekviename lygyje yra susiduriama su iššūkiais, problemomis, reikalaujančiomis vienokio ar kitokio sprendimo. Pasirenkami sprendimai gali smarkiai įtakoti vėliau esančių lygių atliekamą funkcionalumą.

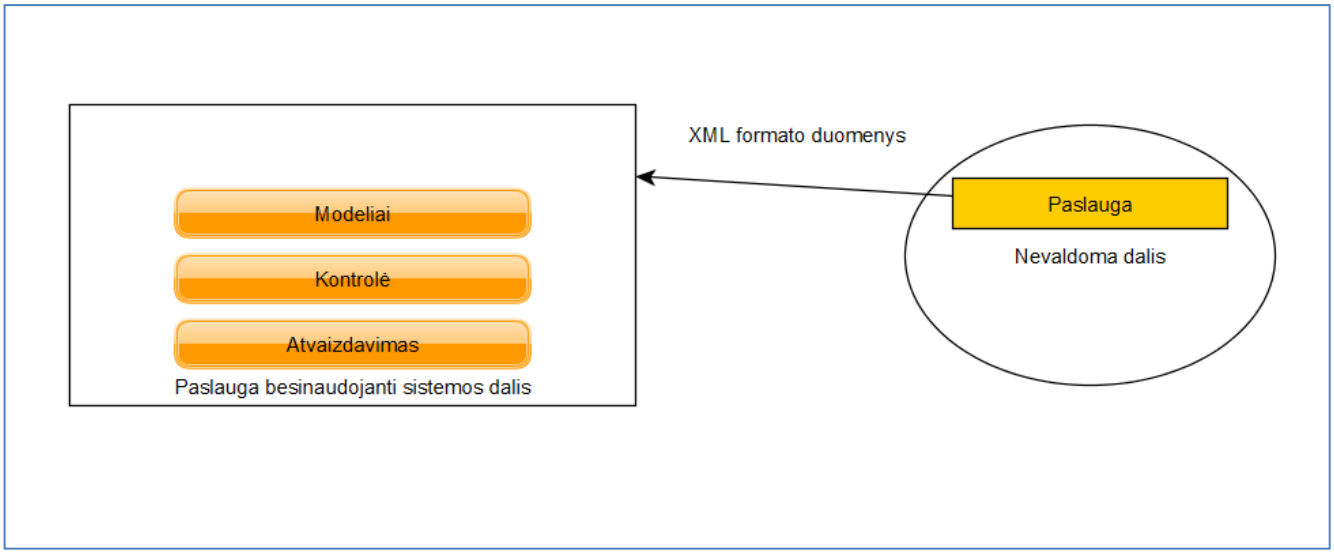


3.2 pav. Pagrindinės metodo dalys

Atlikus visus šiuos veiksmus, laikoma, kad po to sistema sprendžia savo uždavinį su turimu komponentu, t.y., komponentas suteikia tik bazinį funkcionalumą komponentui nuoseklinti. Norimas pasiekti funkcionalumas, uždaviniai, problemos bei realizuoti metodai aprašomi tolimesniuose šio darbo skyriuose. 4 skyriuje aptariami atliekami veiksmai su duomenimis, gavus naują dokumentą, 5 skyrius nurodo kelių dokumentų analizės metodus, 6 skyrius detalizuoja kodo generavimo ir programinio komponento įkrovimo veiksmus.

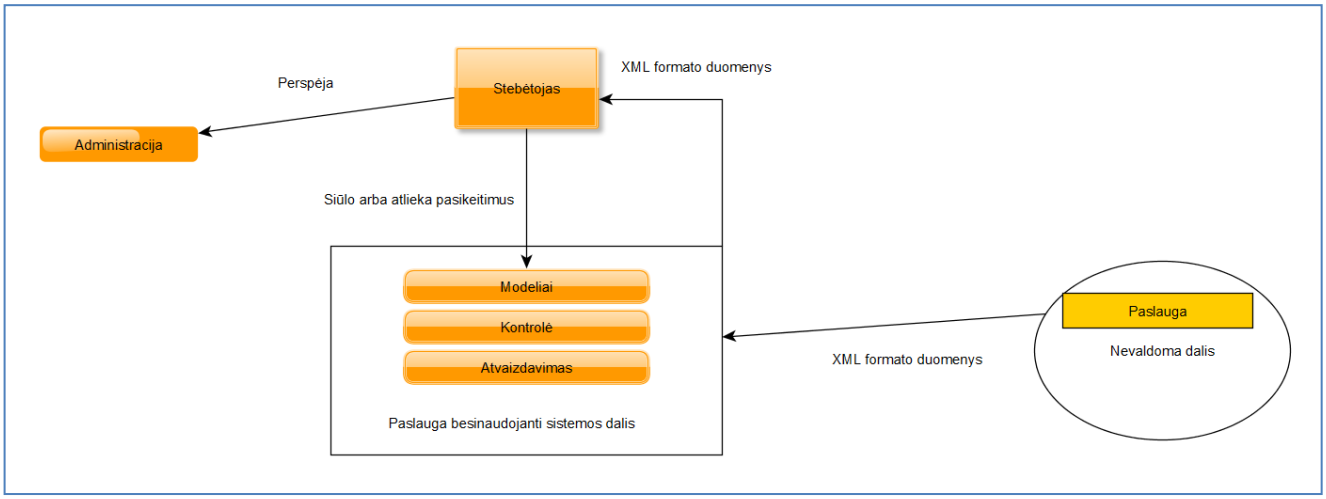
3.4 Prototipo projektas

Prototipui pasirinka sistema, susidedanti iš 2 dalių: trečiosios šalies valdomos paslaugos dalies ir sistemos dalies, kurią galime valdyti (3.3 pav.). Įprastai, sistema gauna XML formato duomenis, juos nuskaito (modelio dalis), apdoroja (kontrolės dalis) ir atvaizduoja. Tokia sistema yra nejautri pasikeitimams iš nevaldomos dalies, o didesni pasikeitimai iššaukia klaidas programiniame kode.



3.3 pav. Įprasta paslauga besinaudojanti sistema

Siūlomo prototipo atveju (3.4 pav.), įvedamas stebėtojo komponentas. Kiekvieną kartą, gavus duomenis, jie yra tikrinami. Jeigu aptinkami pasikeitimai, stebėtojas gali arba siūlyti arba atlikti pakeitimus.



3.4 pav. Sistemos prototipo architektūra

Siūlant yra tik sugeneruojamos atnaujintos klasės, tačiau komponentas gali būti neįkraunamas į sistemą. Jeigu pakeitimai yra atliekami, senas programos komponentas yra išmetamas, programinis kodas įkompiliuojamas į naują ir, sukūrus naują domeną, komponentas yra įkraunamas į jį.

4 DOKUMENTO SPECIFIKACIJOS ATKŪRIMO DUOMENŲ LYGIS

Duomenų lygyje turima 1 arba daugiau XML formato bylų. Šiame lygyje yra sprendžiama, ar sistema yra pajėgi gauti duomenis iš minėtosios bylos (bylų). Šiame skyriuje aprašoma būdai, kokiais galima bandyti nustatyti, ar komponentas pajėgus užkrauti bylą, pateikiami bandytų dokumentų pavyzdžiai, taip pat gauti rezultatai.

4.1 Duomenų analizė

Pradiniu laiko momentu reikia atsižvelgti į tai, ar komponentas jau yra sudiegtas ar ne, gali susidaryti 2 situacijos:

1. komponentas yra įdiegtas anksčiau. Šiuo atveju yra pereinama prie patikrinimo;
2. komponento nėra, sistema pereina visus 3.2 pav. žingsnius.

Patikrinimo metu yra žiūrima, ar turimas komponentas gali sėkmingai nuoseklinti dokumentą, t.y., visi elementai, ryšiai ir atributai yra sudedami į atitinkamus objektus. Šiuo atveju, jei nors vienas elementas, atributo reikšmė negali būti užkrauta, laikoma, kad sistema nėra pajėgi apdirbti dokumento. Taip pat klaidomis laikoma, jei gautame dokumente elementas turi daug vaikų, o turima klasė gali atvaizduoti tik vieną.

XML failų sistemai patikrinimą galima atlikti keliais būdais:

1. Pabandymas tiesiog užkrauti dokumentą. Nuoseklinimo metodai leidžia užkrauti dokumentą, jeigu nėra kritinių klaidų, t.y., vardai atitinka, tenkinamos visos reikalaujamų laukų (elementų, atributų) reikšmės. Šis metodas yra galimas naudoti, kai iš viso dokumento naudojama tik labai maža laukų dalis. Taip naudojant šį metodą, sunku gauti duomenų, kuriuos būtų galima vėliau panaudoti.
2. XSD schemos patikra. Šiuo atveju jau turi būti sukurta schema, kuri yra naudojama patikroje. Sudaryta schema privalo atitikti nuoseklinimo klases, kitu atveju teisingas patikros rezultatas nebūtinai garantuos teisingą duomenų užkrovimą. Taip pat naudojantis schema galima gauti ir naudingos informacijos apie skirtumus nuo žinomos schemos ir paduoto dokumento. Galiausiai, turint XSD schemos dokumentą, galima jį panaudoti programinių klasių generavimui su bibliotekomis.
3. Pilna XML dokumento bei esamų nuoseklinimo klasių analizė. Metodika remiasi galimybe nuskaičius XML bylą bei panaudojus *System.Reflection* esančius metodus, suvesti tiek XML dokumentą, tiek turimas klases į vienodą formatą ir atlikti palyginimus tarp jų. Taip galima išgauti daug informacijos apie skirtumus, tačiau kyla abejonių dėl metodo sudėtingumo ir greitaveikos.

Duomenų surinkimo fazėje taip pat reikia spręsti, kaip grupuoti duomenis, turinčius skirtingas vardų erdves.

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

4.1 pav. Pavyzdys su skirtingomis vardų erdvėmis

Pavyzdyje matomas dokumentas (4.1 pav.), kuris turi 2 *table* elementus skirtingose vardų erdvėse. Kaip grupuoti tokius elementus? Kaip vieną ar kelis atskirus?

Šiame darbe yra naudojama XSD schemos dokumentas kaip XML dokumentų specifikacija. Taip pasirinkta dėl to, kad schema vienareikšmiškai nusako dokumento struktūrą yra lengvai skaitomas ir yra rekomenduojamas formatas *World Wide Web Consortium* (W3C) [27]. Toks formatas leidžia naudotis jau sukurtais sprendimais programinio kodo generavimui.

4.2 Duomenų pavyzdžiai

Šiame skyriuje naudojamų XSD dokumentų pavyzdžiai pateikiami 4.1 lentelėje.

4.1 lentelė. Naudojamų XSD dokumentų pavyzdžiai

```
<?xml version="1.0"?>
<Product Weight="10">
<Name>
    Kamuolys "Spalding"
</Name>
<Price>
    10.2
</Price>
</Product>
```

```
<?xml version="1.0" ?>
<Cart>
<Product Weight="10">
    <Name>
      Kamuolys "Spalding"
    </Name>
    <Price>
      10.2
    </Price>
</Product>
</Cart>
```

```
<?xml version="1.0" ?>
<Cart>
<Product Weight="10">
    <Name>
      Kamuolys "Spalding"
    </Name>
    <Specification>
      <Height>10</Height>

<Color>Oranžinis</Color>
    </Specification>
    <Price>
      10.2
    </Price>
  </Product>
  <Product Weight="10">
    <Name>
      Raketė
    </Name>
    <Specification>
      <Height>10</Height>
    </Specification>
    <Price>
      17
    </Price>
  </Product>
</Cart>
```

Duomenys yra paimami pirkinių krepšelio principu. Pirmiausia duodama paprasto produkto struktūra, bandoma žiūrėti, ar yra teisingai atpažįstami laukai, jų vardai bei tipai. Toliau struktūra yra daroma sudėtingesnė bei platesnė. Pridedama sąrašo struktūra, vidiniai elementai, įvairesni tipai.

4.3 Reikalavimai transformavimo iš XML į XSD schemą

Transformuojant XML į XSD schemą sprendimui keliami tokie reikalavimai:

1. Teisingas vardų atpažinimas. Sprendimas privalo atpažinti visus elementų, elementų atributų vardus, esančius pateiktame dokumente.
2. Teisingas tipų atpažinimas. Pradiniame etape yra reikalaujama, kad būtų bent dalinai atpažinti tipai, esantys to dokumento elementų savybėse (property).
3. Teisingas ryšių atpažinimas. Jeigu elementas turi vieną vaiką, nurodoma, kad jis turi tik vieną vaiką.

4.4 .net suteikiamų bibliotekų veikimo rezultatai

Šiame skyriuje pateikiamas pradinės analizės etapas, kuriame XSD schemos generavimas atliktas su .NET teikiamais įrankiais. Panaudota schemos generavimas iš *DataSet*, *xsd.exe* įrankis, taip pat schemos generavimas naudojantis *XDocument*.

4.4.1 Schemos generavimas naudojantis DataSet

Schemos generavimą naudojantis *DataSet* klase galima atlikti tokiu būdu:

```
public static void XmlToXsd(string xmlPath, string xsdPath)
        {
            var doc = new XmlDocument();
            doc.Load(xmlPath);
            var ds = new DataSet();
            ds.ReadXml(new XmlNodeReader(doc));
            ds.WriteXmlSchema(xsdPath);
        }
```

Į metodą paduodama nuoroda failinėje sistemoje į XML dokumentą bei nuoroda, kur saugoti gautą XSD schemą.

Imant pirmą dokumentą iš pateiktų 4.1 lentelėje, gaunamas rezultatas:

```
<?xml version="1.0" standalone="yes"?>
<xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="Product">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="0" msdata:Ordinal="0" />
        <xs:element name="Price" type="xs:string" minOccurs="0" msdata:Ordinal="1" />
      </xs:sequence>
      <xs:attribute name="Weight" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="Product" />
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Kaip matoma buvo sugeneruota šiek tiek per daug elementų, tačiau pastarieji yra tiesiog sisteminiai elementai, kurie netrukdo dirbti su tokia schema. Kur kas didesnė problema yra, kad *Product* elemento visi atributai bei vaiko elementai yra nurodomi kaip simbolių eilutės, nors akivaizdu, kad *Price* yra slankaus kablelio skaičiaus elementas, o *Weight* yra sveikas skaičius.

Viena tokio generavimo priežasčių yra ta, kad DataSet visas reikšmes saugo kaip objektus ir nepriskiria jiems tipų, užkraunant duomenis iš XML failo. Tai galima išspręsti suteikus DataSet esančioms lentelėms tipus.

Generuojant XSD schemą antrajam dokumentui gaunama:

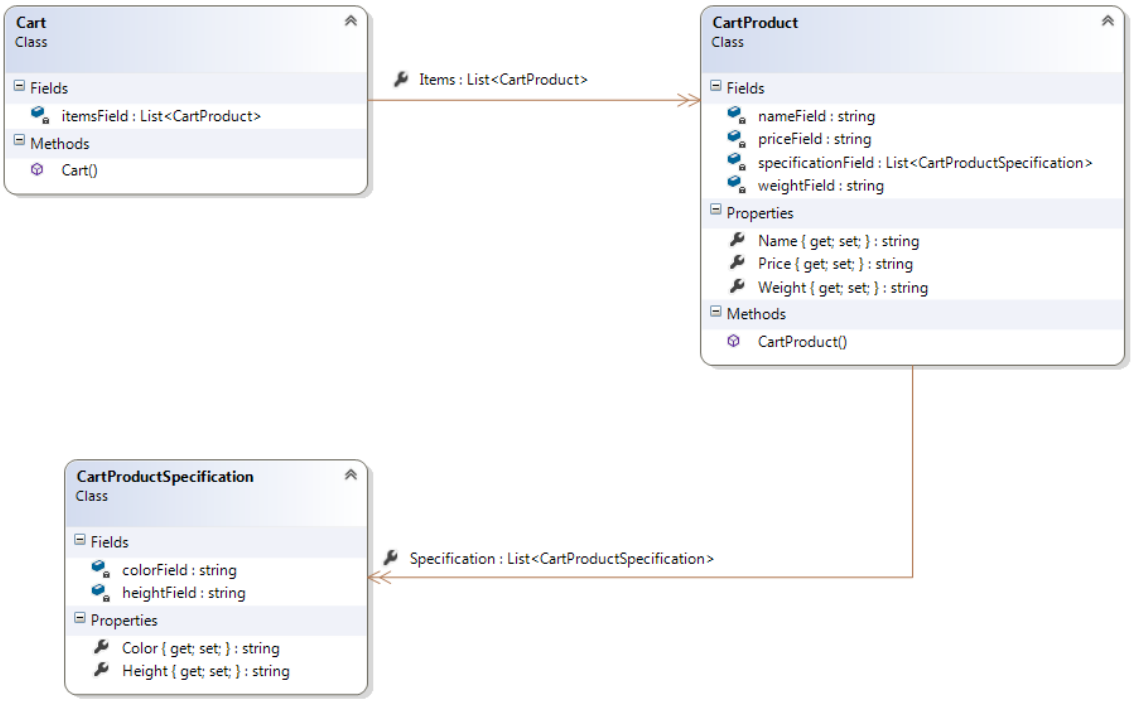
```
   <?xml version="1.0" standalone="yes"?>
   <xs:schema id="Cart" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
     <xs:element name="Cart" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
       <xs:complexType>
         <xs:choice minOccurs="0" maxOccurs="unbounded">
           <xs:element name="Product">
             <xs:complexType>
               <xs:sequence>
                 <xs:element name="Name" type="xs:string" minOccurs="0" msdata:Ordinal="0"
/>
                 <xs:element name="Price" type="xs:string" minOccurs="0"
msdata:Ordinal="1" />
               </xs:sequence>
               <xs:attribute name="Weight" type="xs:string" />
             </xs:complexType>
           </xs:element>
         </xs:choice>
       </xs:complexType>
     </xs:element>
   </xs:schema>
```

Vėlgi turime tą pačią problemą su tipų neatpažinimu. Kita problema atsiranda, jog *Product* nurodomas kaip sąrašas (dėl <xs:choice minOccurs="0" maxOccurs="unbounded">). Tai reiškia, kad programiniame kode generuotame iš šitokios schemos *Product* bus sąrašas *Cart* klasėje, nors iš dokumento akivaizdu, jog elementas buvo tik vienas ir turėtų būti vienas.

Atlikus transformaciją su 3 dokumentu gaunama tokia schema:

```
   <xs:schema id="Cart" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
     <xs:element name="Cart" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
       <xs:complexType>
         <xs:choice minOccurs="0" maxOccurs="unbounded">
           <xs:element name="Product">
             <xs:complexType>
               <xs:sequence>
                 <xs:element name="Name" type="xs:string" minOccurs="0" msdata:Ordinal="0"
/>
                 <xs:element name="Price" type="xs:string" minOccurs="0"
msdata:Ordinal="2" />
                 <xs:element name="Specification" minOccurs="0" maxOccurs="unbounded">
                   <xs:complexType>
                     <xs:sequence>
                       <xs:element name="Height" type="xs:string" minOccurs="0" />
                       <xs:element name="Color" type="xs:string" minOccurs="0" />
                     </xs:sequence>
                   </xs:complexType>
                 </xs:element>
               </xs:sequence>
               <xs:attribute name="Weight" type="xs:string" />
             </xs:complexType>
           </xs:element>
         </xs:choice>
       </xs:complexType>
     </xs:element>
   </xs:schema>
```

Šį kartą, nors ir yra teisingai nurodyta, kad *Product* yra ne vienas ir turi būti atvaizduotas sąraše, tačiau *Specification* yra tik vienas, o yra nurodoma, jog yra jų daug. Šiai schemai sugeneruotas programinis kodas pasinaudojus Xsd2Code biblioteka vaizduojamas klasių diagrama 4.2 pav. Gautos 3 klasės: „Cart“, „CartProduct“ ir „CartProductSpecification“.



4.2 pav. Sugeneruotas programinis kodas naudojant *DataSet*

Matoma, kad *Cart* turi savyje *CartProduct* sąrašą, nes buvo keli *Product* elementai. Tačiau visuose *Product* elementuose buvo tik po vieną *Specification*, o su generuotame kode nurodoma, jog gali būti keli. Visi kiti laukai, atributai nurodomi kaip simbolių eilutės, nors dokumentuose kai kurie laukai tikrai gali būti sveiko arba slankaus kablelio skaičiaus. Sugeneruotas programinis kodas pateikiamas 9.1 priede.

Bandant naudotis kitais įrankiais, gauti visiškai identiški rezultatai. Iš to galima spręsti, kad ir kiti įrankiai naudoja arba tą pačią DataSet klasę, arba vidinį šios klasės funkcionalumą. Taip pat matoma, kad visi laukai yra nurodomi kaip simbolių eilutės. Šią problemą galima spręsti priverstinai *DataSet* įdedant tipus.

4.4.2 Schemos generavimas panaudojant DataSet su tipais

DataSet elementams suteikti tipus galima šiuo būdu:

1. Pereiti per visas visų lentelių eilutes.
2. Sudėti teisingus rastus duomenų tipus.

XSD failo kūrimas naudojant tipizuotą DataSet:

```
// xmlPath – kelias iki xml dokumento
// xsdPath – kelias iki xsd dokumento
public static void CreateWithTyped(string xmlPath, string xsdPath)
        {
            var doc = new XmlDocument();
            doc.Load(xmlPath);
            var ds = new DataSet();
            ds.ReadXml(new XmlNodeReader(doc));
            var typedDataSet = ds.Clone();
            typedDataSet.EnforceConstraints = false;
            for (var j = 0; j < typedDataSet.Tables.Count; j++)
            {
                var dt = ds.Tables[j];
                var newDt = typedDataSet.Tables[j];
                var types = Enumerable.Range(0, dt.Columns.Count).Select(x =>
typeof(int)).ToArray();
                foreach (DataRow dr in dt.Rows)
                {
```

```
                        for (var i = 0; i < dt.Columns.Count; i++ )
                        {
                            var data = dr[i].ToString();
                            int outInt = 0;
                            if (int.TryParse(data, out outInt))
                            {
                                continue;
                            }
                            double outDouble;
                            if (double.TryParse(data, out outDouble))
                            {
                                var oldType = types[i];
                                if (oldType != typeof(string))
                                {
                                    types[i] = typeof(double);
                                }
                                continue;
                            }
   //Susidedam nustatytą tipą
                            types[i] = typeof(string);
                        }
                    }
                    for (var i = 0; i < newDt.Columns.Count; i++)
                    {
                        if(dt.Columns[i].ColumnName.EndsWith("id",
StringComparison.OrdinalIgnoreCase))
                            continue;
   //Prisiskiriam nustatytą tipą
                        newDt.Columns[i].DataType = types[i];
                    }
   //Importuojam duomenis
                    foreach (DataRow oldDr in dt.Rows)
                    {
                        newDt.ImportRow(oldDr);
                    }
                }
                typedDataSet.EnforceConstraints = true;
                typedDataSet.WriteXmlSchema(xsdPath);
            }
```

Galima išskirti keletą vietų:

```
var typedDataSet = ds.Clone();
```

Šiuo veiksmu yra sudaromas esamo *DataSet* struktūros kopija, t. y., nukopijuojama visi lentelių, laukų aprašai iš senos *DataSet* į naująją, tačiau duomenis nėra perkeliami.

```
typedDataSet.EnforceConstraints = false;
```

Šiuo atveju yra leidžiama netikrinti, ar dedamų įrašų ryšiai yra teisingi. Kadangi duomenys dedami nebūtinai teisinga tvarka, ir orientuojamės labiau į tipų klaidų aptikimą, o ne dėl teisingų ryšių struktūros išlaikymo, taip užtikrinama, kad nebus klaidų dėl galbūt ne ta tvarka perkeliamų duomenų.

Atliekami bandymai su tais pačiais duomenų pavyzdžiais iš 4.1 lentelės:

Pirmo dokumento atveju:

```
<?xml version="1.0" standalone="yes"?>
<xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="Product">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="0" msdata:Ordinal="0" />
        <xs:element name="Price" type="xs:duoble" minOccurs="0" msdata:Ordinal="1" />
      </xs:sequence>
      <xs:attribute name="Weight" type="xs:int" />
    </xs:complexType>
  </xs:element>
  <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="Product" />
```

```
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Matoma, kad *Price* ir *Weight* jau turi tinkamus tipus. Vėlgi išlaikomas tas pats *NewDataSet* elementas. Įdomu, kad jis yra tik tuo atveju, kai tėra viena lentelė.

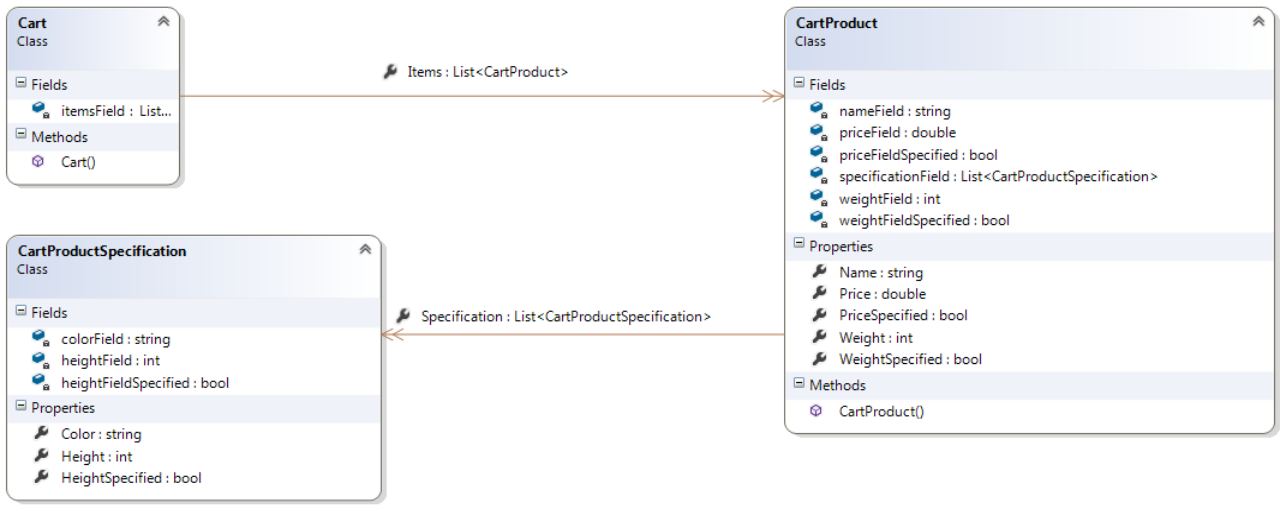
Bandymas su 3 dokumentu:

```
  <?xml version="1.0" standalone="yes"?>
  <xs:schema id="Cart" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="Cart" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="Product">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Name" type="xs:string" minOccurs="0" msdata:Ordinal="0"
/>
                <xs:element name="Price" type="xs:double" minOccurs="0"
msdata:Ordinal="2" />
                <xs:element name="Specification" minOccurs="0" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="Height" type="xs:int" minOccurs="0" />
                      <xs:element name="Color" type="xs:string" minOccurs="0" />
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="Weight" type="xs:int" />
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Šiuo atveju turime teisingai nurodytus tipus *Price*, *Weight* ir kitus laukus, kaip simbolių eilutes. Teisingai nurodyta, kad *Cart* turi daugiau nei vieną *Product*, tačiau kiekvienas *Product* turėjo tik po vieną *Specification*. Iš to galima spręsti, kad XSD kūrimas pasinaudojus DataSet visais atvejais elemento vaiką nurodo kaip sąrašą, net esant jam tik vienam.

Programinis kodas

Pagal paskutiniąją sugeneruotą XSD schemą pasinaudojus Xsd2Code biblioteka generuotas programinis kodas (klasių pavyzdžiai 9.2 priede).



4.3 pav. Programinis kodas naudojant *DataSet* su tipais

Kaip 4.3 pav. manome atsirado daugiau laukų, kurių nebuvo naudojantis DataSet nenurodžius tipų. Įdomu tai, kad visiems laukams, kurie nėra simbolių eilutės sugeneruota laukai formatu *{lauko vardas}Specified*. Taip turbūt bandoma užtikrinti, kad jeigu laukas yra tuščias taip būtų nurodyta šis faktas. Aišku, tai kelia klausimą, kodėl nebuvo galima tiesiog naudotis *nullable* laukais tiek *double*, tiek *int* atveju.

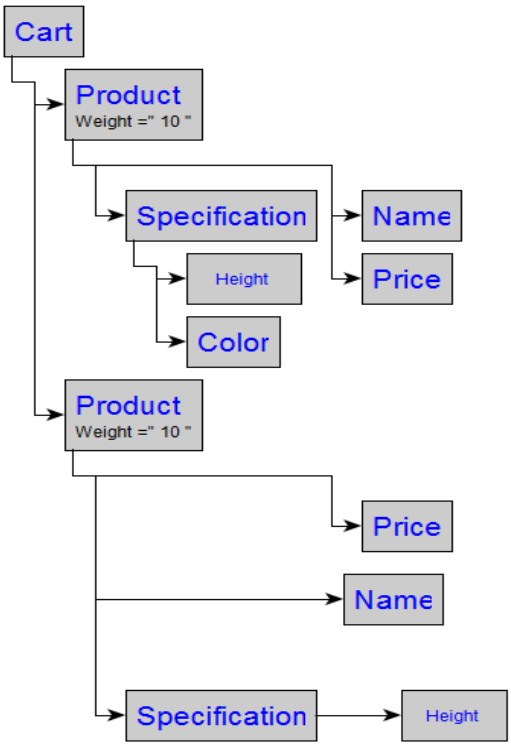
4.4.3 Išvados

Naudojantis DataSet bei kitas suteikiamais sprendimais (*xsd.exe*, *XDocument*), gaunamas rezultatas su visais laukais, kaip simbolių eilutėmis, nepaisant to, kad iš konteksto galima spręsti, jog pastarieji tokie nėra. Šią problemą galima spręsti naudojant *DataSet* su programiniu būdu įrašytais teisingais tipais į duomenų lentelės stulpelius. Tačiau tada iškyla kita problema, bet koks tėvo – vaiko ryšys yra nurodomas kaip vienas su daug, t.y. XSD faile visada nurodoma, kad elementas turi daug kitų (kompleksinių) elementų, net jei XML faile jų yra tik vienas.

Dėl šių priežasčių sukurtas atskiras funkcionalumas XML failo struktūrai transformuoti į XSD.

4.5 Sukurtas funkcionalumas

XML failą galima įsivaizduoti kaip medžio struktūrą, matomą 4.4 pav.



4.4 pav. Pavyzdinė XML dokumento struktūra

Paprasčiausias būdas sužinoti teisingus laukų tipus, ryšius tarp elementų yra apeiti visą medį.

Sukurto metodo žingsniai:

1. Susikurti struktūrą, saugoti.
2. Apeiti vieną elementą:
    a. gauti visus vaikus, visus atributus;
    b. atnaujinti atributų sąrašą;
    c. einant per vaikus, jeigu vaikas buvo sutiktas, nurodyti, kad bus sąrašo struktūra, jei ne – pridėti prie vaikų;

d. priskirti vaikui tipą;
e. jeigu vaikas kompleksinis – dėti į saugojimo struktūrą, kartoti algoritmą šiam vaikui.

Pagrindinius tipus galima aprašyti išvardijimu:

```
public enum OutputTypes : int
    {
        @int = 0,
        @double = 1,
        @string = 2,
        @class = 3,
        @listint = 4,
        @listdouble = 5,
        @liststring = 6,
        @listclass = 7
    }
```

Juos galima lyginti pagal svarbumą. Mažiausiai svarbus yra sveikasis tipas, labiausiai – klasių sąrašas. Vadinasi, jeigu bus sutikti keli sveikieji skaičiai tame lauke ir tik viena simbolių eilutė – laukui bus priskirtas simbolių eilutės tipas.

Klasė duomenims apie elementą saugoti:

```
public class NodeInfo
    {
        public Dictionary<string, OutputTypes> Childs { get; set; }
        public Dictionary<string, OutputTypes> Attributes { get; set; }
        public string Name { get; set; }
```

Čia saugomi žodynai su vaikų vardais ir priskirtais tipais bei atitinkamai atributai.

Metodai:

```
public void UpdateData(XmlNode node, Collector collector)
        {
            var appendList = new List<XmlNode>();
            var set = new HashSet<string>();
            foreach (XmlNode child in node.ChildNodes)
            {
                var outputType = OutputTypes.@class;
                if (child.HasChildNodes && child.FirstChild.Name != "#text")
                {
                    appendList.Add(child);
                }
                else
                {
                    outputType = GetType(child.InnerText);
                }
                OutputTypes data;
                if (!Childs.TryGetValue(child.Name, out data))
                {
                    Childs.Add(child.Name, outputType);
                }
                else
                {
                    if (data != outputType || set.Contains(child.Name))
                    {
                        Childs[child.Name] = MergeTypes(data, outputType);
                    }
                }
                set.Add(child.Name);
            }
            AppendAttributes(node);
            foreach (var item in appendList)
                collector.AppendData(item);
        }
```

Šiuo metodu sukuriamos arba atnaujinamos žinios apie elementą.

Pirmiausia imami jo vaikai, tada tikrinama, ar kiekvienas vaikas turi savų vaikų, ir ar pirmasis vaikas nėra tiesiog tekstas. Taip atrenkami kompleksiniai tipai, t.y., tie, kurie pavirs į klasės. Kitu atveju yra bandoma nuspręsti, koks tipas bus sukurtas: *int*, *double*, *string*.

Po to žiūrima, ar yra šis vaikas žinomas, jeigu ne – pridedame prie vaikų sąrašo, jeigu taip, tokiu atveju reikia patikrinti:

1. Ar žinomas vaiko tipas sutampa priskirtuoju dabar. Taip atrenkami pasikeitimai tarp tipų.
2. Gal šiame elemente jau vaikas sutiktas buvo, tokiu atveju galimas sąrašo atvejis ir tai reikia atitinkami fiksuoti.

Galiausiai visi kompleksiniai vaikai yra pridedami prie kaupiančiosios struktūros.

Collector

Šis objektas veikia kaip struktūra, kuri kaupia visus duomenis apie apeitus elementus.

```
public class Collector
    {
        public Dictionary<string, NodeInfo> NodeInfos { get; set; }
```

Jis savyje saugo žodyną, sudarytą iš elemento vardo kaip rakto ir duomenų apie elementą, kaip reikšmės.

Yra 2 metodai:

```
public void Traverse(XmlNode node)
        {
            var info = new NodeInfo();
            NodeInfos.Add(node.FirstChild.Name, info);
            info.UpdateData(node.FirstChild, this);
        }
public void AppendData(XmlNode node)
        {
            NodeInfo info = null;
            if (!NodeInfos.TryGetValue(node.Name, out info))
            {
                info = new NodeInfo();
                NodeInfos.Add(node.Name, info);
            }
            info.UpdateData(node, this);
        }
```

Traverse metodas veikia kaip įeities taškas, jo pagrindinė užduotis pradėti visą algoritmą. Jis prideda pirmąjį elementą ir tada eina per visus pastarojo vaikus.

AppendData metode yra pažiūrima, ar jau yra žinių apie tokį elementą, jeigu nėra – jis pridedamas. Bet kokiu atveju yra atnaujinami elemento duomenys.

Rezultatai:

Pirmam dokumentui:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Product">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="Name" type="xs:string" />
        <xs:element minOccurs="0" name="Price" type="xs:double" />
      </xs:sequence>
      <xs:attribute name="Weight" type="xs:int" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Iš karto matosi, jog nebėra „*NewDataSet*“ schemos elemento. Tipai nurodomi teisingai (kontekste), įdomu, kad ant Weight atributo uždėtas `use="required"`, tai reiškia, kad atributas bus privalomas.

Antras:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Cart">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" ref="Product" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Product">
```

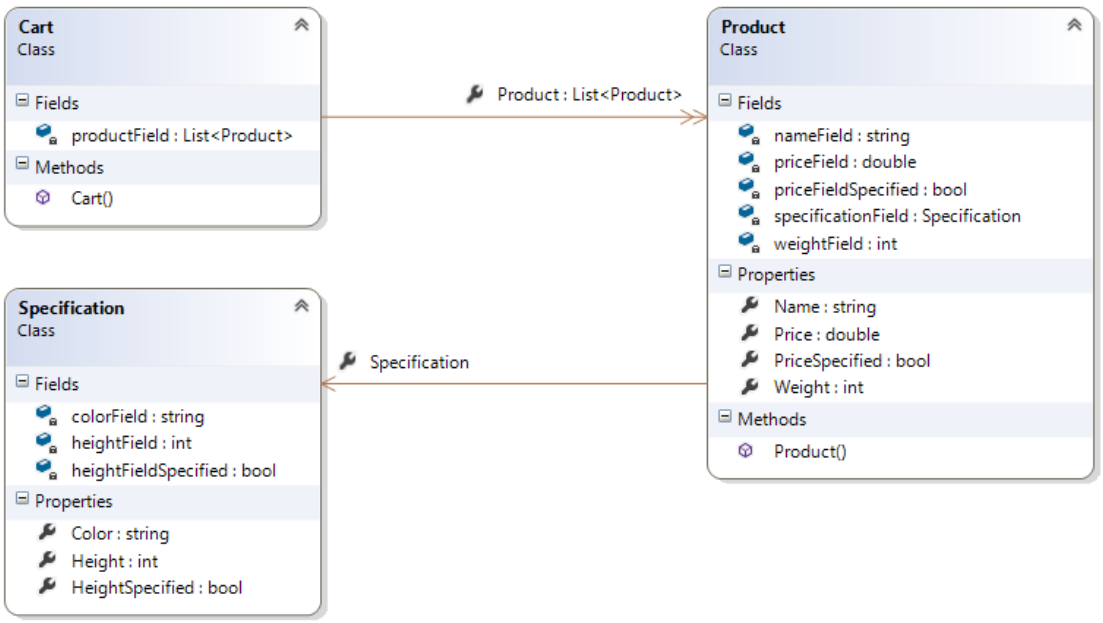
```
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="Name" type="xs:string" />
        <xs:element minOccurs="0" name="Price" type="xs:double" />
      </xs:sequence>
      <xs:attribute name="Weight" type="xs:int" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Matomas, kad Cart jau nebeturi neriboto skaičiaus produktų.

Trečias dokumentas:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Cart">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="Product" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Product">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="Name" type="xs:string" />
        <xs:element minOccurs="0" ref="Specification" />
        <xs:element minOccurs="0" name="Price" type="xs:double" />
      </xs:sequence>
      <xs:attribute name="Weight" type="xs:int" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="Specification">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="Height" type="xs:int" />
        <xs:element minOccurs="0" name="Color" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Ryšiai nurodyti teisingai: *Cart* turi daug *Product*, pastarieji turi tik po vieną *Specification*. Generuoto programinio kodo klasės nurodomos 4.5 pav.



4.5 pav. Generuotas programinis kodas naudojantis sukurtu funkcionalumu

Tipai nurodyti teisingai. *Xsd2Code* vis vien generuoja *PriceSpecified* lauką *Price*, tačiau nebegeneravo *Weight*, kas reiškia, jog *required* nurodymas pasakė bibliotekai, kad tas laukas bus visais atvejais.

4.6 XMLSPY

XSD schemos generavimui buvo pabandyta panaudoti „XMLSPY“ produktą, išbandytą su 3 dokumentu. Gautas rezultatas:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- W3C Schema generated by XMLSpy v2015 rel. 3 (http://www.altova.com) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Cart">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="Product" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
<xs:element name="Name" type="xs:string"/>
<xs:element name="Color">
	<xs:simpleType>
		<xs:restriction base="xs:string">
			<xs:enumeration value="Oranžinis"/>
		</xs:restriction>
	</xs:simpleType>
</xs:element>
<xs:element name="Price">
	<xs:simpleType>
		<xs:restriction base="xs:decimal">
			<xs:enumeration value="10.2"/>
			<xs:enumeration value="17"/>
		</xs:restriction>
	</xs:simpleType>
</xs:element>
<xs:element name="Height">
	<xs:simpleType>
		<xs:restriction base="xs:byte">
			<xs:enumeration value="10"/>
		</xs:restriction>
	</xs:simpleType>
</xs:element>
<xs:element name="Product">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="Name"/>
			<xs:element ref="Specification"/>
			<xs:element ref="Price"/>
		</xs:sequence>
		<xs:attribute name="Weight" use="required">
			<xs:simpleType>
				<xs:restriction base="xs:byte">
					<xs:enumeration value="10"/>
				</xs:restriction>
			</xs:simpleType>
		</xs:attribute>
	</xs:complexType>
</xs:element>
<xs:element name="Specification">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="Height"/>
			<xs:element ref="Color" minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
</xsschema>
```

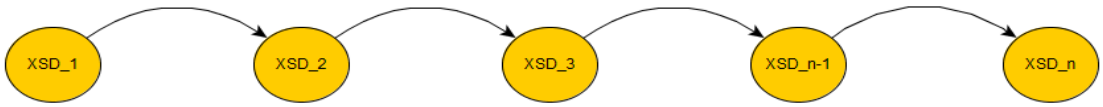
Dokumente galima matyti, kad XMLSPY generuoja truputį kitokios struktūros XSD schemą. Kiekvienam laukui, atributui yra kuriami atskiri tipai, ar tai būtų paprastas tipas (*xs:simpleType*) ar sudėtinis, su nuorodomis į kitus elementus. Taip pat elementams yra priskirti išvadinimai bei didesnė tipų aibė - vietoj jau naudotų *int, double, string* atsirado ir *decimal* bei *byte*.

5 DOKUMENTO ANALIZĖS LYGIS

Analizės lygyje yra atliekamas gauto dokumento bei esamos žinių basės pavertimas į objektus, kurie vėliau bus naudojami generuojant programinį kodą. Šiame etape labai svarbu yra pasirinkta strategija, kaip saugoma turima žinių bazė, ypač kalbant apie regeneraciją. Skyriuje pateikiami metodikos kaip galima saugoti, atnaujinti turimą informaciją.

5.1 Bendras metodų eskizas

Kadangi 4.1 skyriuje buvo nutarta naudotis XSD formato byla kaip dokumento specifikacija, taigi visą dokumento specifikacijos, keitimą galima įsivaizduoti kaip seriją kuriamų XSD dokumentų (5.1 pav.)

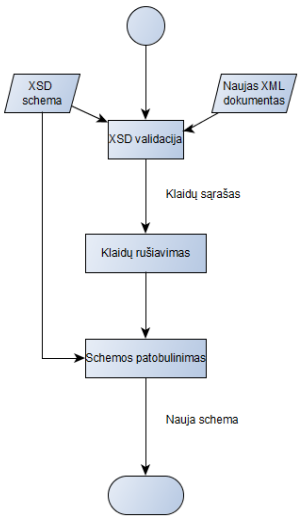


5.1 pav. XSD schemos kitimas per laiką

XSD dokumentai leidžia naudotis, jie yra universalūs aprašai, leidžiantys jais naudotis ne tik kuriamos sistemos dalies aplinkoje, bet ir kitose aplinkose esant reikalui. Su XSD dokumentu galima lengvai aptikti pasikeitimus XML byloje, o pasinaudojus šia informacija pakeisti turimą specifikaciją. Iš čia gaunama evoliucionuojančios XSD schemos metodas.

5.2 XSD evoliucija

XSD evoliucija galima apibūdinti XML dokumentą aprašančios schemos tobulinimą per laiką. Šis veiksmas gali būti vykdomas tiesiog atlikus schemos dokumento patikrą schemai ir panaudojus gautu klaidų sąrašą atitinkamai keisti schemą. Įprastai schema yra tikrinamas XML dokumentas, ar jis atitinka žinomą formatą, šiuo atveju schema yra tikrinama paties dokumento. Taigi, ši metodą dar galima vadinti atvirkštine dokumento patikra.



5.2 pav. XSD evoliucijos bendra veikimo schema

Į metodą (5.2 pav.) paduodama jau turima XSD schema ir naujas atėjęs XML dokumentas. Praleidžiama schemos patikra ir gaunamas klaidų sąrašas.

.Net aplinkoje schemos patikra vykdoma šiuo metodu:

```
//schema – XSD schema simbolių eilute
//xml – XML dokumentas simbolių eilute
public static List<SchemaValidationData> Validate(string schema, string xml)
        {
            var validationErrors = new List<SchemaValidationData>();
            XmlSchemaSet schemas = new XmlSchemaSet();
            schemas.Add("", XmlReader.Create(new StringReader(schema)));
            var doc = XDocument.Parse(xml);
            doc.Validate(schemas, (o, e) =>
            {
                var element = o as XObject;
                var vi = new SchemaValidationData()
                {
                    Object = element,
                    Message = e.Message
                };
                validationErrors.Add(vi);
            });
            return validationErrors;
        }
```

XmlSchemaSet yra sisteminė .net klasė, leidžianti atlikti dokumento patikrą esant reikalui ir pagal kelias schemas ir skirtingose vardų erdvėse. Patikra vykdoma sutiktų įvykių principu.

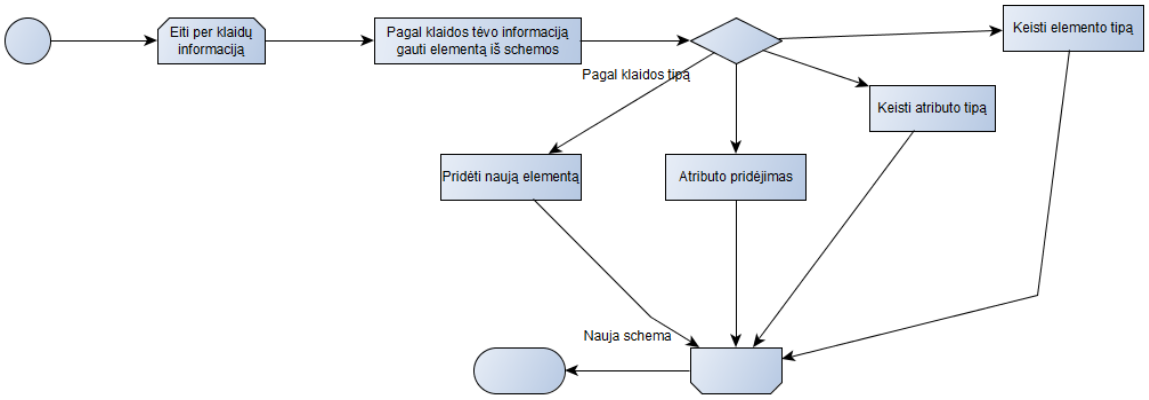
Klaidų informacijai saugoti naudojamos klasės, pateiktos prieduose 9.3 skyriuje. Svarbu paminėti, kad per *XObject* galima gauti elemento ar atributo tėvą, pagal kurį vėliau bus atlikta paieška schemoje.

Galimas klaidų sąrašas su paaiškinimais pateiktas 5.1 lentelėje. Reikia atkreipti dėmesį į pirmą klaidą. Ši klaida bus išmesta ir tuo atveju jeigu elementas yra nurodytas kaip vaikas, tačiau nurodyta, kad jis yra vienas. Taip pat ši klaida metama nepaisant to ar neatrastas elementas savyje turi vaikų ar ne.

5.1 lentelė. Patikros klaidų sąrašas

Klaida	**Paaiškinimas**
The element 'Cart' has invalid child element 'Product'	Nurodo, kad Cart elemente yra neatpažintas vaikas, kurį reikės pridėti.
Price attribute is not declared	Atributas nerastas
The 'Price' element is invalid - The value '...	Neatitinka elemento tipas
The 'Price' atribute is invalid - The value '...	Neatitinka atributo tipas

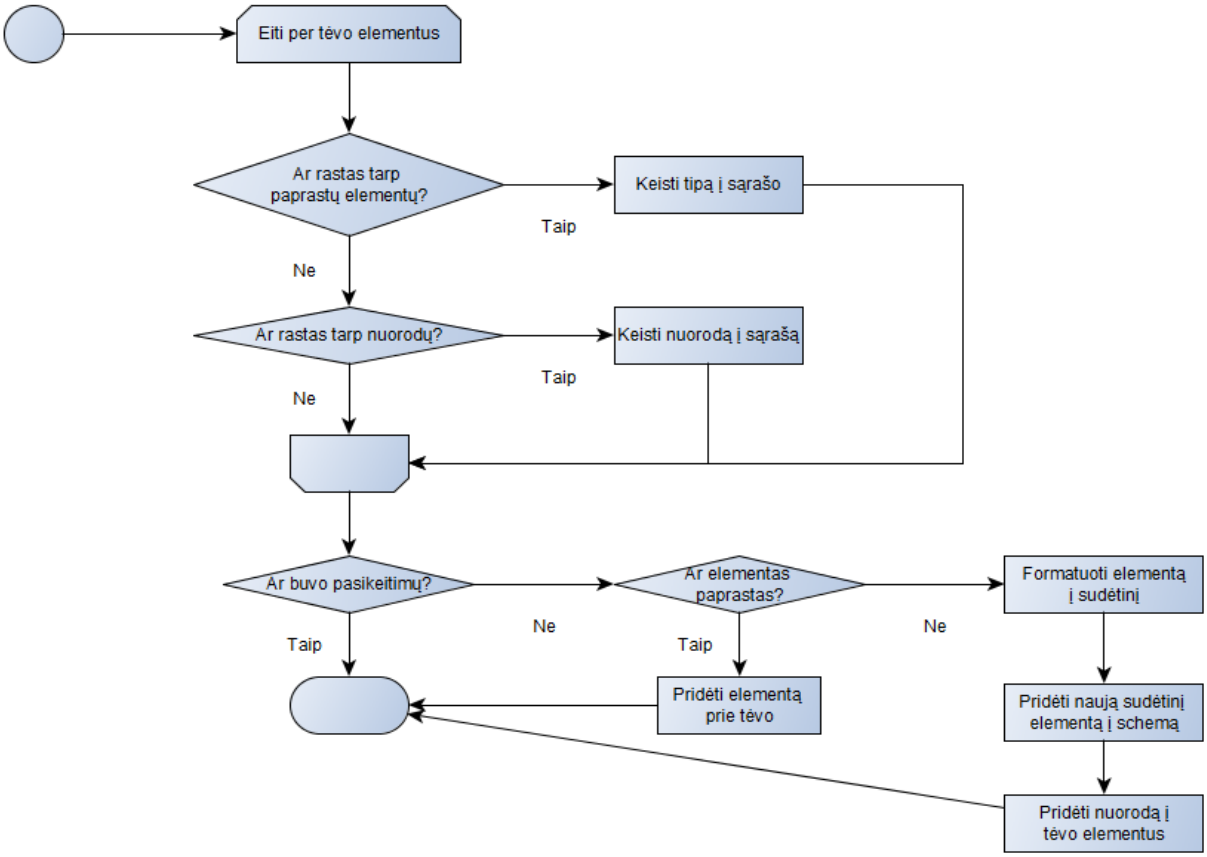
5.2.1 Schemos atnaujinimas



5.3 pav. XSD schemos atnaujinimo diagrama

XSD schema atnaujinama (5.3 pav.) einant per surinktą klaidų sąrašą, paėmus elemento, kuris sukėlė klaidą tėvą, iš schemos imamas elementas. Po to pagal klaidos tipą (5.1 lentelė) galima atlikti veiksmus su schema.

Atributo pridėjimas, paprastų tipų keitimas nėra sudėtingi veiksmai. Naujo elemento pridėjimo žinutė gali savyje slėpti kelis įmanomus veiksmų variantus, naujo elemento pridėjimas tampa gerokai sudėtingesniu veiksmu detalizuojamas 5.4 pav.



5.4 pav. Elemento pridėjimo veiksmas schemoje

Šiuo atveju logika tampa sudėtinga. Kadangi atsiradus tiek naujam elementui, tiek padidėjus tų pačių elementų skaičiui, klaidos žinutė yra ta pati, galima išskirti keturi atvejus:

1. Pridėtas naujas paprastas elementas, t.y. toks, kuris savyje turi tik tekstą, jam nereikia nuorodos.
2. Pridėtas naujas sudėtingas elementas, turintis savyje vaikų. Šiuo atveju pridedam netik nuorodą, tačiau ir elemento deklaraciją schemoje.
3. Atsirado keli paprasti elementai. Ieškoma pagal vardą.
4. Atsirado keli sudėtingi elementai. Ieškoma pagal nuorodos vardą.

Tėve esantys elementai gali būti tiek paprasti, tiek sudėniai (turintys savyje vaikų). Paprasti elementai schemoje žymimi nurodant jiems vardą, tuo tarpu sudėniai turi nuorodą į schemoje esantį elementą. Dėl to paieška tampa sudėtinga, nes gali atitikti ir elemento vardo ir nuorodos reikšmės. Esant atitikimams, rastas ryšys nurodomas kaip sąrašo tipas.

Neradus atitikimų reikia pridėti klaidą iššaukusį elementą. Čia vėlgi yra išskiriami 2 variantai:

1. kai elementas yra paprastas – pridedam tik prie tėvo;
2. Kai elementas sudėtinis – jį verčiam į schemos elementą ir pridedam kaip nuorodą.

5.2.2 Apibendrinimas

Schemos evoliucijos metodu kiekvienas neatitikimas iššauks schemos atsinaujinimą, šiuo būdu yra užtikrinama, kad dokumento specifikacija per daug nesiskirs nuo tos, kuri naudojama gaunamiems XML dokumentams sukurti. Taip pat šis metodas turi užtikrinti greitą reakciją kardinalaus specifikacijos pasikeitimo metu.

Deja, jeigu pačiame dokumente yra klaidų, jos bus fiksuojamos kaip priimtini pasikeitimai, t.y., bet koks netikslumas, susijęs su duomenų tipais, sąrašais, elementų vardais bus laikomas schemos neatitikimu ir atitinkamai ištaisytas. Triukšmingų dokumentų atveju tai gali reikšti labai greitą tolimą nuo realios specifikacijos.

Dėl šių priežasčių taip pat yra siūlomas kitas – svorinis metodas.

5.3 Svoriais paremtas schemos generavimo metodas

XML dokumentą galima aprašyti ryšių aibe tarp:

- Elementų;
- Atributų;
- Tipų.

Kiekvienas elementas turi savyje kitus elementus - tai būtų paprasti elementai, turintys savyje tik tekstą ar nuorodas į kitus elementus, taip pat atributus. Šiems sąryšiams galima priskirti kaupiamą svorį. Vėliau nurodžius slenkstį, nuo kurio elementai yra laikomi tinkamais schemai, ją formuoti.

Norint suteikti vėliau ateinantiems elementams didesnę svarbą, suteikiami svoriai palaipsniui yra didinami.

Algoritmą galima skaldyti į 3 dalis:

1. Inicializciją;
2. Dokumento pridėjimą;
3. Schemos atidavimą.

Inicializacija

Pradžioje yra paduodamas didinimo parametras ir slenkstis:

- Didinimo parametras – dokumentų kiekis iki kol bus padidintas suteiktas svoris.
- Slenkstis – skaičius nuo kurio ryšys yra pripažįstamas tinkamu schemai.

5.3.1 Dokumento pridėjimas

Šiuo žingsniu laikoma, kad dokumentas visada turi vienodą šakninį elementą.

Pirmiausia žiūrima ar turėtų dokumentų skaičius nėra dalus be liekanos iš didinimo parametro. Jeigu taip, suteikiamą svorį didinti per 1.

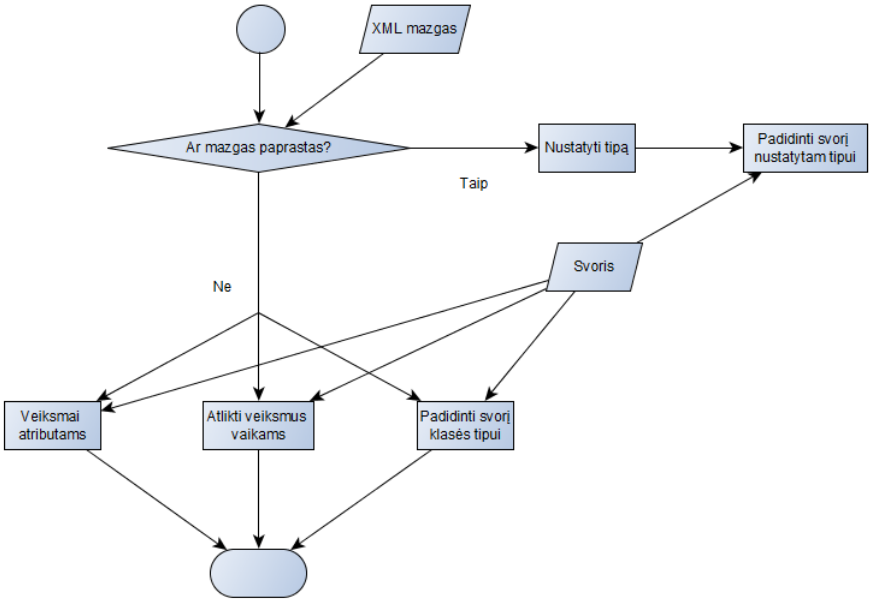
Pradėti nuo šakninio elemento ir eiti per jo vaikus (elementus ir atributus). Jeigu tarp žinomų vaikų nėra to elemento, jį pridėti. Jeigu elementas nėra paprastas (turintis savyje tik tekstą), atitinkamai vykdyti šią funkciją rekursyviai jam. Jeigu sutinkamas sąrašas, suteikiamas svoris kiekvienam elementui yra mažinamas tiek kartų, kiek yra sąraše elementų.

Kiekvienas XML mazgas saugo svorius:

- Tipui (sveikas skaičius, slankaus kablelio skaičius, simbolių eilutė, sudėtinis tipas);
- Pasikartojimui (būtinas, nebūtinas elementas, sąrašas ar vienas elementas).

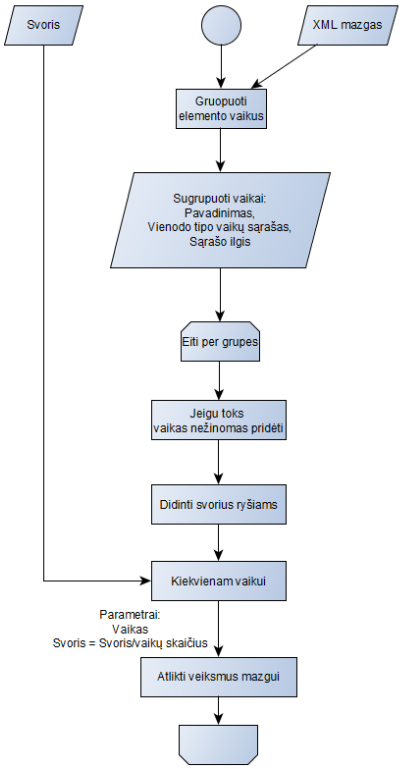
Taip pat yra saugomi duomenis apie nuorodas į kitus mazgus ir atributus.

Atributai saugo svorius tipui ir priklausomumui (būtinas atributas, nebūtinas).



5.5 pav. Mazgui atliekami veiksmai

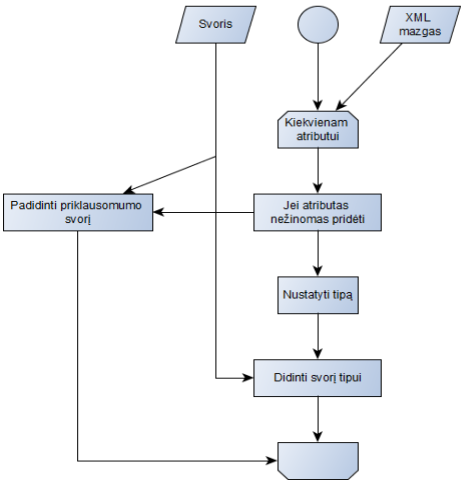
Kiekvienam mazgui yra atliekama veiksmų aibė. Tikrinama, ar jis nėra paprastas (t.y. turintis tik tekstą), jeigu taip – didinamas svoris nustatytam tipui iš vidinio teksto. Kitu atveju atliekami veiksmai vaikams bei atributams detalizuoti atitinkamai 5.6 pav. ir 5.7 pav..



5.6 pav. Veiksmai mazgo vaikams

Veiksmuose vaikams XML mazgo vaikai yra sugrupuojami pagal jų vardą. Gaunamas sąrašas grupių, kuriose yra nurodomas vaiko vardas, kiek vaikų su tuo vardu yra ir jų mazgai. Visi nežinomi vaikai pridedami. Vaiko tipui (vardui) priklausomai nuo jų skaičiaus yra padidinamas svoris sąrašo turėjimui.

Tada kiekvienam mazgui iš grupės yra rekursyviai atliekami veiksmai kaip mazgui (5.6 pav.). Tačiau, jeigu vaikų yra daug, perduodamas svoris yra mažinamas per tiek kartų, kiek yra vaikų sąraše. Taip daroma, nes atnaujinant vaiko mazgo duomenis svoriai bus padidinami tiek kartų, kiek mazgų yra sąraše, o tai reikštų per didelį skaičių.

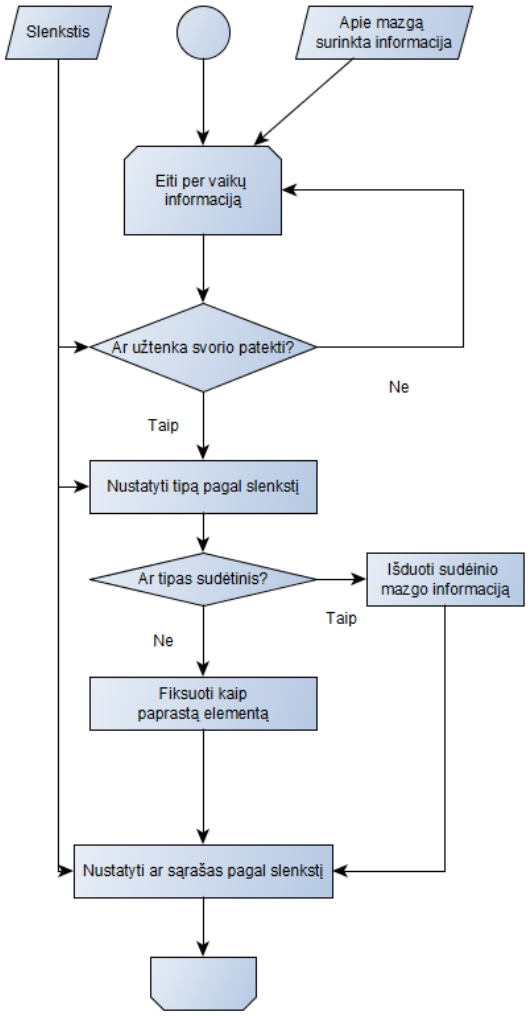


5.7 pav. Veiksmai mazgo atributams

Kiekvienam atributui yra tikrinama, ar jis yra žinomas, jeigu ne – pridedamas. Iš atributo reikšmės nustatomas tipas ir tam tipui atribute didinamas svoris. Atributai laikomi unikalūs kiekvienam mazgui, todėl, jeigu vardai ir kartojasi keliuose skirtinguose mazguose, tipai vieni kitų neįtakoja niekaip.

5.3.2 Schemos išdavimas

Šiame etape yra sukuriama nauja XSD schema. Metodo schema pavaizduota 5.8 pav.



5.8 pav. XSD schemos išdavimas

Rekursyviai einama per sukauptą mazgų informaciją. Tikrinama, ar yra viršijamas slenkstis priklausomybės svoryje. Taip pat tikrinamas svoris tipams, elementas yra sąrašas ar vienetinis elementas.

5.3.3 Apibendrinimas

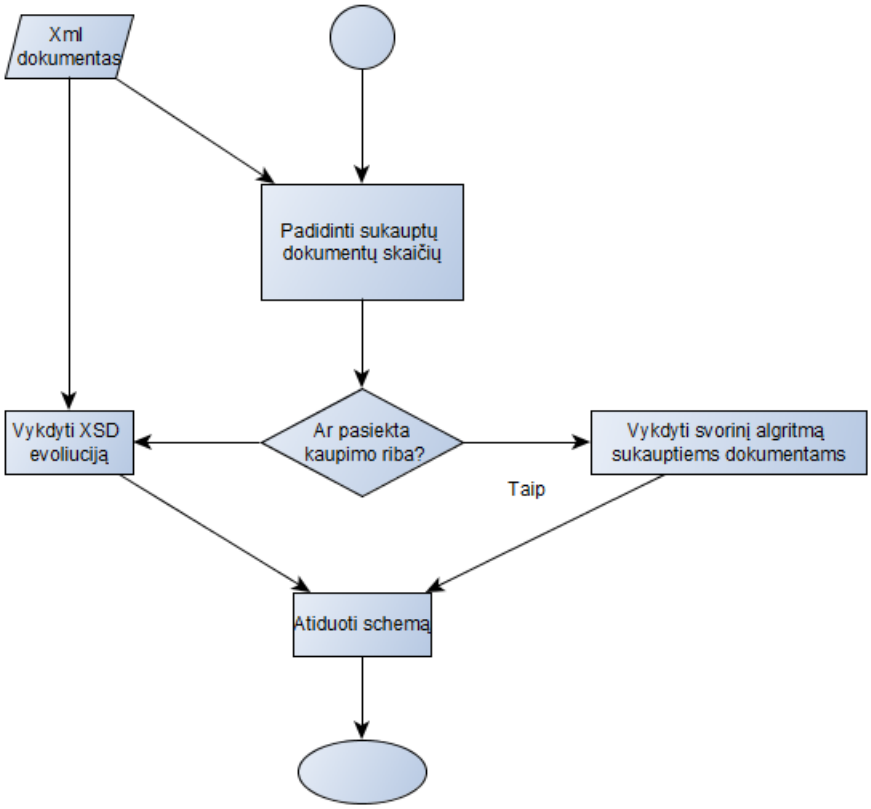
Svoriniu metodu tikimasi pasiekti dokumento specifikacijos atsparumą klaidoms. Klaidingi elementai bus per mažo svorio ir neperlips slenksčio. Tačiau atitinkamai, jeigu specifikacijoje atsiras pasikeitimų, jie taip pat neturės pakankamo svorio ir bus kurį laiką ignoruoti.

Norint turėti ir XSD evoliucijos tikslumą ir svorinio algoritmo atsparumą, galima naudoti maišytą algoritmą.

5.4 Maišytas metodas

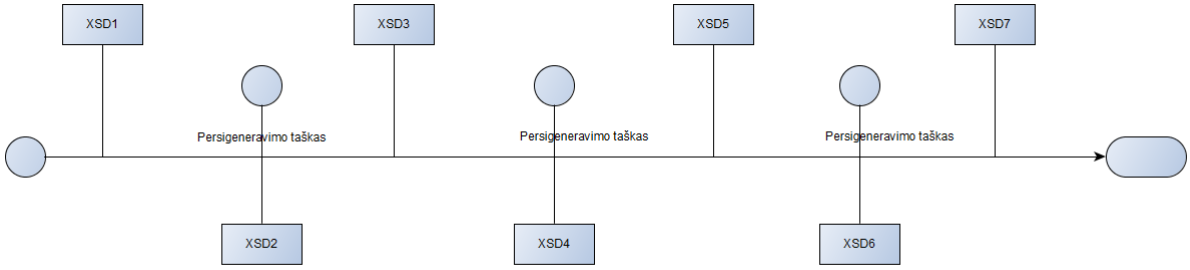
Tam tikrą laiką schemos evoliucija nėra prastas metodas, tačiau nėra atsparus šiukšlėms (kiekvienas netikslumas iššaukia schemos pasikeitimus). Todėl pabandyta sujungti svorinį ir evoliucijos metodą. Tam tikrą dokumentų skaičių yra vykdomas XSD evoliucijos metodas. Po to yra

įvykdomas svorinis metodas sukauptiems n dokumentų, kur n – sveikas teigiamas skaičius. Metodo schema pateikiama 5.9 pav.



5.9 pav. Maišto metodo bendra schema

Sukuriama riboto ilgio eilė, į kurią statomi dokumentai, kai eilė užsipildo sukauptiems dokumentams atliekamas svorinis metodas. Taip siekiama išvalyti schemą nuo šiukšlių, kurios galimai joje atsirado vykdant evoliucijos algoritmą.



5.10 pav. Maišytas metodas laike

Metodas laike vaizduojamas 5.10 pav. Pradžioje yra vykdomas XSD evoliucijos metodas iki pasiekiamas persigeneravimo taškas. Tada sukauptai dokumentų aibei yra įvykdomas svorinis metodas, ir sukurta nauja schema tampa ta, kuri bus tobulinama toliau iki sekančio persigeneravimo taško.

5.5 Genetinis metodas

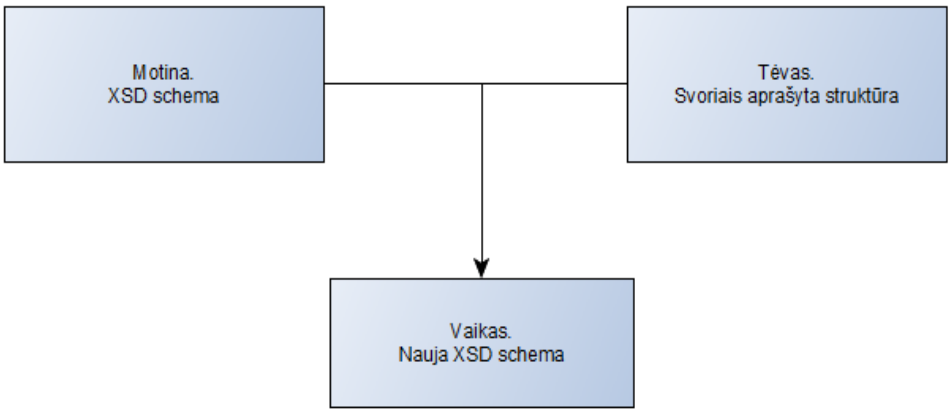
Maišyto algoritmo atveju 5.10 pav. pasiekus kaupimo ribą praktiškai visai prieš tai žinota informacija yra išmetama ir schema yra konstruojama iš naujo. Tai yra informacijos nuostoliai, kurių galima bandyti išvengti.

Maišytą metodą galima tobulinti įmaišius genetinių metodų idėjas.

Tiek po XSD evoliucijos tiek po maišyto metodo atlikimo informacijos formatą galima laikyti tokiu pačiu, tik kitaip užkoduotu. Vienoje pusėje yra griežtos struktūros dokumentas, kitoje svorių aibe aprašyta struktūra.

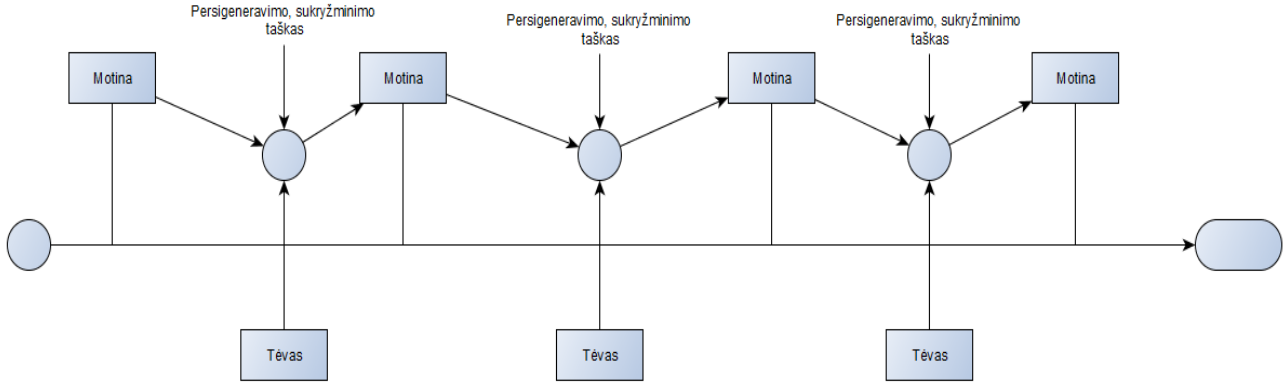
Selekcija

Kadangi jau turim 2 individus selekcija jau yra įvykdyta. Pavadinkime XSD schemos struktūrą „motina“, svoriais aprašytą struktūrą „tėvu“. Juos abu sukryžminus gaunamas vaikas, kuris tampa nauja motina 5.11 pav.**Error! Reference source not found.**



5.11 pav. Genetinio algoritmo individai

Bendrai algoritmo veikimas pavaizduotas 5.12 pav.

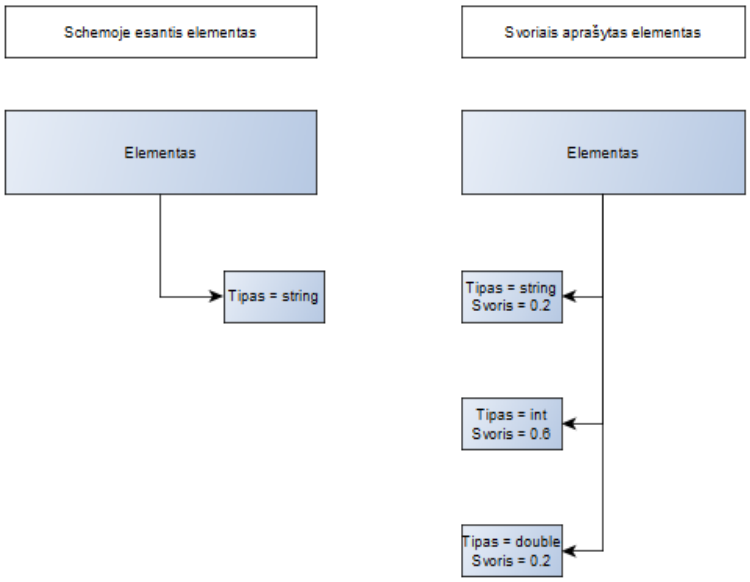


5.12 pav. Genetinio algoritmo veikimas laike

„Motina“ iki persigeneravimo (sukryžminimo) taško elgiasi kaip ir maišyto algoritmo atveju. Pasiekus nurodytą tašką, yra sugeneruojam „tėvo“ struktūra, įvykdomas kryžminimas, o po mutacijos yra sukuriama nauja „motina“.

Kryžminimas

Kryžminimas vyksta iš motinos paimant turimos informacijos (elementai, atributai, tipai) aibę ir tą aibę panaudojant padidinti atitinkamų elementų svoriams tėvo struktūroje. Pavyzdys pateikiamas tipų svoriams 5.13 pav.

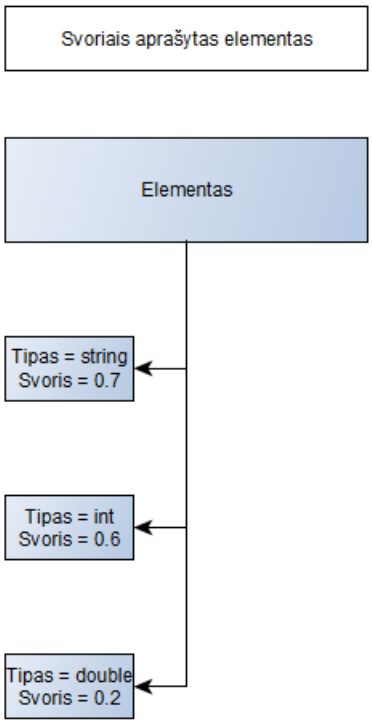


5.13 pav. Kryžminimo veiksmo pavyzdys

Pavyzdyje matoma, kad schemoje elemento tipas nurodomas kaip simbolių eilutė, tuo tarpu svoriais aprašytoje struktūroje galimi 3 tipai:

- Simbolių eilutė su svoriu 0.2
- Sveikasis skaičius su svoriu 0.6
- Slankaus kablelio skaičius su 0.2

Kryžminant simbolių eilutės svoris yra padidinamas (tarkime, per 0.5).

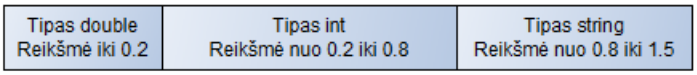


5.14 pav. Kryžminimo rezultatas

Po kryžminimo 5.14 pav. gaunamas simbolių eilutės svoris jau yra 0,7.

Mutacija

Mutacija šiuo atveju keičia naujos schemos išdavimo principą. Prieš tai buvo žiūrima, ar svoris viršija ribą. Šio metodo atveju svoriai yra išdėstomi atkarpoje 5.15 pav.



5.15 pav. Svorių išsidėstymas

Tuomet yra imamas atsitiktinis dydis nuo 0 iki maksimalios atkarpos reikšmės. Gautas rezultatas nurodo priskirtą tipą. Šis metodas analogiškai gali būti panaudotas elementų egzistavimui bei ryšiams gauti.

6 PROGRAMINIO KODO GENERAVIMAS IR KOMPONENTO UŽKROVIMAS

Programinio kodo generavimo metu sukuriamas komponentas vėliau įkraunamas į sistemą. Šiame skyriuje apžvelgiama, ką galima panaudoti programiniam kodui generuoti ir kokios problemos atsiranda norint atnaujinti komponentą.

6.1 Programinio kodo generavimas

Programinio kodo generavimo metu ir analizės metu sukaupti ir sugrupuoti duomenys panaudojami sugeneruoti dabar aktualias nuoseklinimo klases.

Programinio kodo generavimas .net aplinkoje galimas panaudojus T4 šablonus arba *CodeDom* elementus. Taip pat yra bibliotekos ir įrankiai generuoti programinį kodą iš XSD schemų. Generuojant iš XSD schemų galima naudotis *xml.exe* įrankiu. Tačiau šis įrankis yra galimas tik jeigu yra įdiegta *Visual Studio*, taigi tai labai riboja programos pernešimo galimybes. Viena iš bibliotekų generuoti programinį kodą iš XSD schemos yra *Xsd2Code.* Ši biblioteka neturi jokių specifinių reikalavimų ir gali būti pernešama į kitas aplinkas.

6.1.1 Programinio kodo generavimas naudojantis Xsd2Code biblioteka

4 skyriuje visas programinis kodas iš naudojantis XSD failais buvo generuojamas naudojant *Xsd2Code* biblioteką. Ji su savo užduotimis susitvarkė pakankamai gerai. Tiesa, generuojant ši biblioteka taip pat sugeneruoja didžiulį kiekį atributų, kurių nauda yra abejotina. Taip pat biblioteka savotiškai kuria vardus klasėms. Pvz.: *Cart* elemente esantis *Product* bus įvardintas kaip *CartProduct*, o dar žemiau esantis *Specification* kaip *CartProductSpecification*.

Problemos su šia biblioteka:

- Nėra galimybės nurodyti, kad klasių vardai būtų generuojami *PascalCase* t.y. pirma klasės raidė yra didžioji. Vadinasi, jeigu XML (kaip ir XSD) elemento vardas iš mažosios – klasė, savybė (property) bus iš mažosios. Įprastai C# programavimo kalboje kodo rašymo standartai prašo, kad klasė, savybė, metodas prasidėtų didžiąja raide.
- Didžiulis atributų kiekis, todėl negalima nurodyti, kokie atributai reikalingi, kokie ne. Daug sugeneruotų programinių atributų yra specifinėms reikmėms ir būtų naudinga galimybė jų atsisakyti, paliekant tik nuoseklinimui reikalingus.
- Negalima plėsti, kad generuotų papildomas funkcijas klasėse. Yra galimybė sugeneruoti kelias standartines kaip konstruktorių, klonavimo, nuoseklinimo skaitymą, rašymo metodus, tačiau nėra galimybės gauti specifinį funkcionalumą.
- Generuojami laukai *{vardas}Specified*, kurie nurodo kai kurių laukų egzistavimą, vietoj to, kad būtų generuojami laukai, leidžiantys null reikšmes (nullable).

6.1.2 T4 šablonų kalba

T4 šablonų kalboje galima sugeneruoti nuoseklinimo klasę, specifinius metodus ir t.t. Dar vienas T4 privalumas yra tai, kad galima generuoti kodą nepriklausomai nuo programavimo kalbos. Šablono rezultatas yra simbolių eilutė, kuri gali būti tiesiog surašyta į bet kokios galūnės tekstinį failą ir atitinkamai panaudota nebūtinai .net paremtoje sistemoje.

Programinio kodo generavimo metodas skaldomas į 2 dalis:
1. Failas su galūne .tt, kuriame aprašomas šablonas;
2. Dalinė (partial) klasė, kurioje galima aprašyti logiką;

Toks skaldymas leidžia šablone palikti labai nedaug funkcionalumo, o visą logiką perkelti į jau sistemoje esantį kodą. Taipogi naudojantis T4 kalba galima generuoti atskirus failus kiekvienai klasei. *Xsd2Code* atveju yra generuojamas vienas failas, kuriame yra visas programinis kodas.

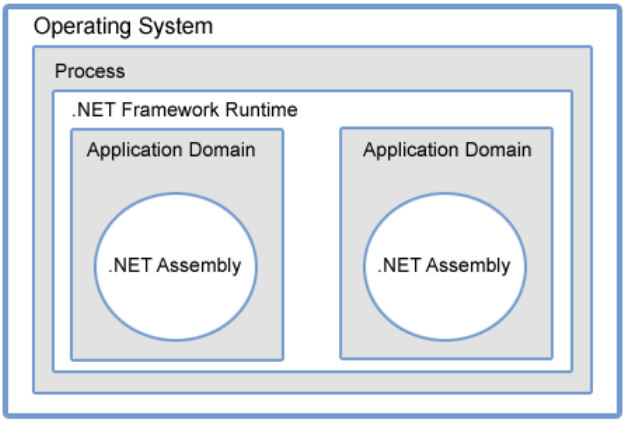
Šablono ir logikos failo pavyzdžiai prieduose 9.4 skyriuje.

6.2 Komponento užkrovimas

6.2.1 Dinamiškai prijungiami komponentai

Dinamiškai prijungiamo komponento (dll) prijungimas naudojant .net ir c# programavimo kalbą vyksta per *System.Assembly* bibliotekas. Naudojamas *Assembly.LoadFrom* metodas nurodant komponento vietą. Tačiau šioje stadijoje iškyla problemos, kaip pakeisti jau naudojamą komponentą.

Programos esančios .net aplinkoje veikia atskirose programų domenuose (AppDomain). Viena programa taip pat gali turėti kelis domenus (6.1 pav.**Error! Reference source not found.**).



6.1 pav. Programa su domenais

Domenai gali būti:

- Pagrindinis. Tai yra domenas, kuris yra sukuriamas tik paleidus programą. Jo nepatartina išmesti.
- Šalutiniai domenai. Jie yra kuriami per *AppDomain.Create* metodą, nurodant domeno vardą. Vėliau galima į sukurtą domeną užkrauti komponentus. Šį domeną galima išmesti, tačiau tik tada, jei jo duomenys nėra naudojami pagrindiniame.

Problemos iškyla, jeigu pagrindiniame domene reikia pasinaudoti sukurto šalutinio domeno komponentų funkcionalumu. Sukūrus šalutinį domeną ir įkrovus komponentą į pastarąjį, kviečiant metodus iš kito komponento, pastarasis vis vien yra pririšamas prie pagrindinio domeno, taip uždraudžiant komponento atlaisvinimą. Šią problemą nurodoma bandyti spręsti pasinaudojus *MarshalByRef* objektus, tačiau net ir tuo atveju, jeigu klasės panaudojamos pagrindiniame domene, komponentas nėra paleidžiamas, nors domenas yra.

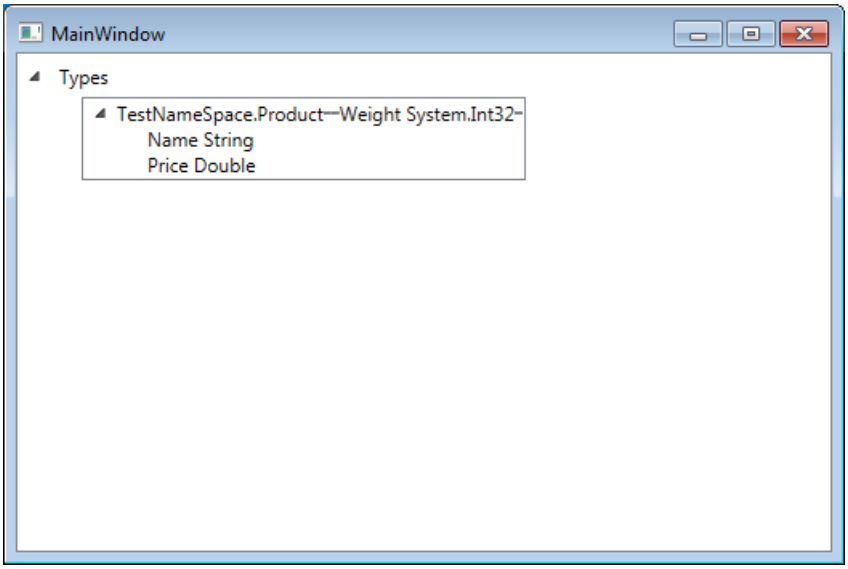
6.2.2 Testinė programa

Komponento atvaizdavimui naudojama WPF biblioteka.

Tarkime jau turime sugeneruotą komponentą. Norint jį užkrauti į programą ir po to jį atleisti, reikia susikurti atskirą *AppDomain* iš pastarajame užkrautų *assmebly* paskutiniajame bus sugeneruotos klasės. Tada galime pasiimti visus tipus ir tiesiog atvaizduoti. Pateikiamas programinis kodas, sukuriantis domeną ir užkraunantis komponentą į jį. Atlikus visus veiksmus, domenas šalinamas iš veikiančios programinės įrangos.

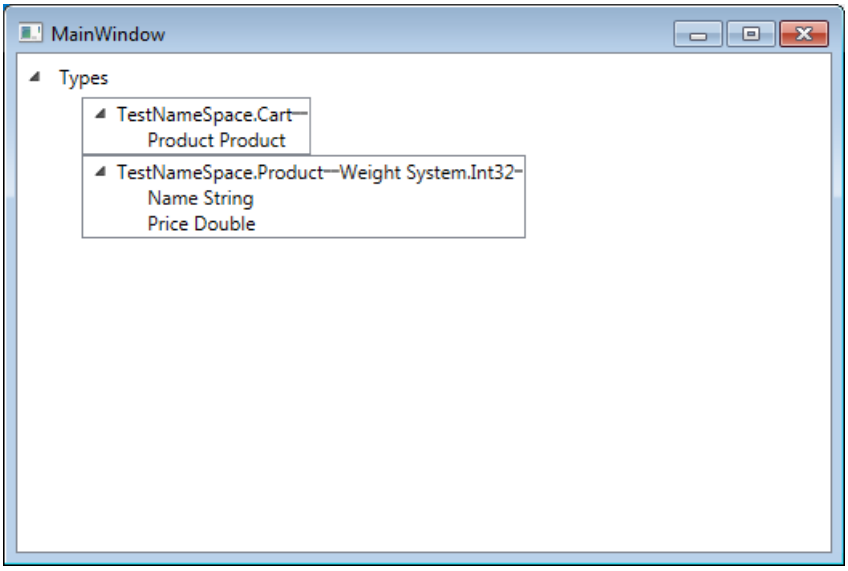
```
var domain = AppDomain.CreateDomain("XmlDomain");
domain.Load(AssemblyName.GetAssemblyName(dll));
var assml = domain.GetAssemblies().Last();
var types = assml.GetTypes();
//atvaizdavimo atnaujinimas
UpdateTree(types);
AppDomain.Unload(domain);
```

Rezultatai:



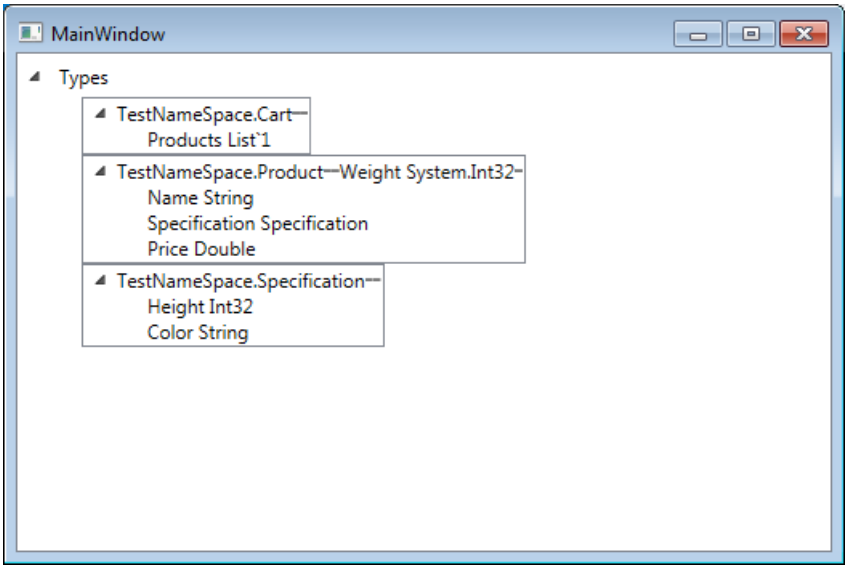
6.2 pav. Pradiniai duomenys

6.2 pav. matome užkrauto domeno tipų pavyzdį. Kaip ir duomenyse tėra viena klasė su atributu, bei Name ir Price vaikais.



6.3 pav. Naujas atėjęs dokumentas

Tęsiant darbą 6.3 pav., atėjo naujos specifikacijos dokumentas. yra užkrauta jau patobulinta bibliotekos versija. Matome, kad čia yra jau *Cart* savyje turinti *Product*.



6.4 pav. Galutinis specifikacijos pasikeitimas

Galiausiai 6.4 pav. Cart jau nurodoma, kad turi sąrašą Product, o pastarasis turi tik 1 specifikaciją.

6.3 Apibendrinimas

Programinio kodo generavimui galima naudoti standartinę *Xsd2Code* biblioteką arba susikurti savą T4 šabloną ir juo naudotis. Abu šie sprendimai leis kuri programinį kodą, kuris po to gali būti panaudotas sutransliuoti į komponentą.

Komponentas į sistemą gali būti įkraunamas susikūrus naują domeną ir kviečiant *Assembly.Load* metodą. Tačiau dll yra užkraunamas ir į naują domeną ir į pagrindinį. Tai reiškia, kad net ir pašalinus sukurtą naują domeną, dll lieka sistemoje. Vadinasi, jeigu tas dll bus keičiamas, trinamas programa išmes klaidą. Tačiau, jeigu bus sukurtas naujas komponentas ir naujas domenas su tuo pačiu vardu – programa išlaikys teisingą veikimą, tik sukurtų komponentų dll kaupsis.

7 EKSPERIMENTINIS SISTEMOS TYRIMAS

Šiame skyriuje pateikiami rezultatai testavus 5 skyriuje pateiktu metodus, jiems vieną po kito pateikus dokumentų imtį. Imčiai generuoti naudotas eksponentinis dėsnis, kur dokumentų intensyvumas yra 50 dokumentų per laiko vienetą. Imtyje gali būti procentas klaidingų dokumentų, taip pat dokumento specifikacija tam tikrais laiko intervalais gali kisti.

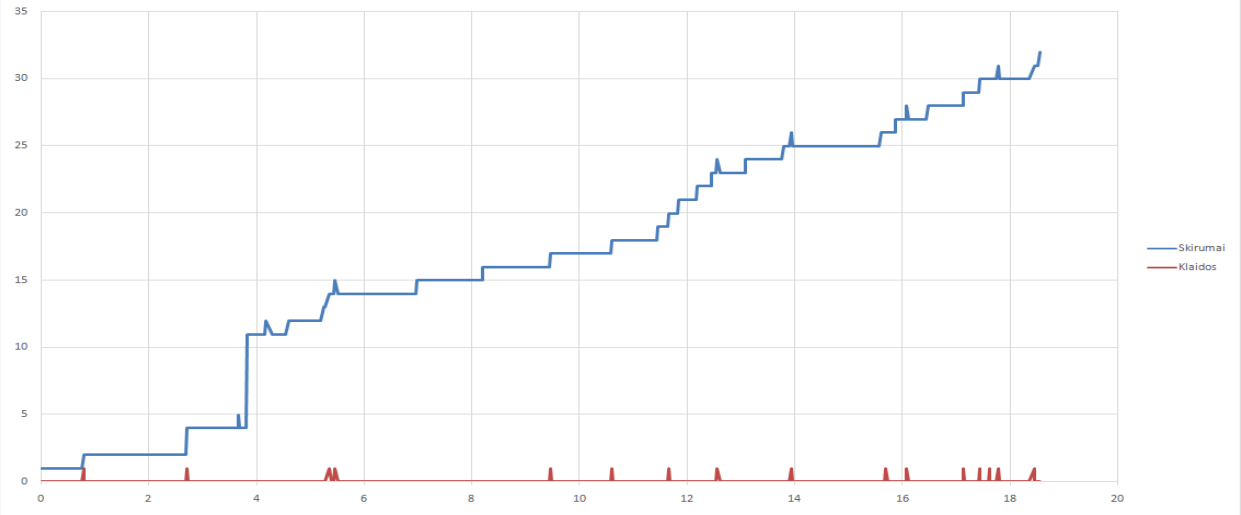
Dokumentų generavimui naudota *.Net xml power tools* bibliotekos dalis [28] galinti generuoti XML bylą iš pateiktos XSD schemos. Skirtumų nuo tikrosios schemos skaičiavimui naudota tos pačios bibliotekos dalis, leidžianti generuoti skirtumų tarp dviejų XML bylų dokumentą [29].

Tikrinimo principai: tikrinama, kiek sugeneruota schema turi skirtumų nuo tikrosios, taip pat, kiek sugeneruota schema, atlikus XML dokumento patikrą nurodo klaidų. Atlikti eksperimentai tiek su viena, tiek su keliomis skirtingomis schemomis.

7.1 Vienos specifikacijos dokumentai

Šiam tyrime naudojama viena dokumento specifikacija (XSD). Pagal ją generuojamos XML bylos yra pateikiamos algoritmams. Dokumentas gali turėti 5% klaidos tikimybę bei 10 kartų į specifikaciją yra įtraukiami elementai visam laikui.

7.1.1 XSD evoliucija



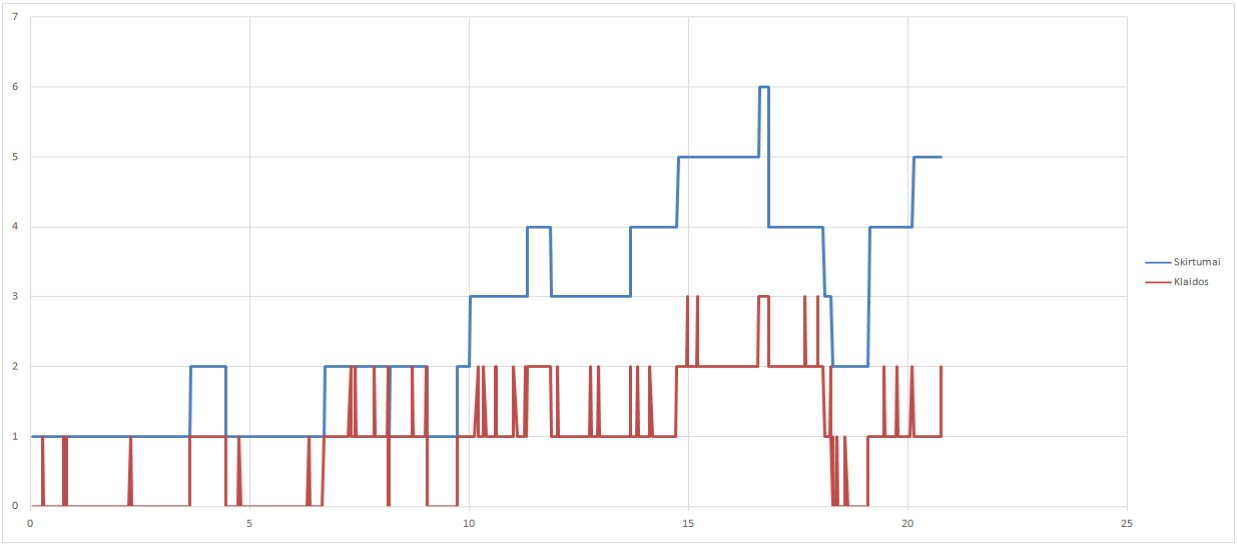
7.1 pav. XSD evoliucija

Evoliucijos atveju (7.1 pav.) matomas nuolatinis pokyčių kaupimas. T. y., gavus dokumentą su netiksliais duomenimis, jie yra fiksuojami ir informacija apie juos yra saugoma. Kiekvieno schemos persigeneravimo metu galima tikėtis, kad ji turės vidutiniškai 16,4 skirtumų lyginant su tikrąja schema. Tačiau šio metodų schemą panaudojus patikrinti dokumentą, galima yra tik 0,02 klaidos tikimybė. Dažniausiai net ir esant klaidoms kitas atėjęs dokumentas iššauks pasikeitimus schemoje, kurie tą klaidą tiesiog ištaisys.

7.1.2 Svorinis metodas

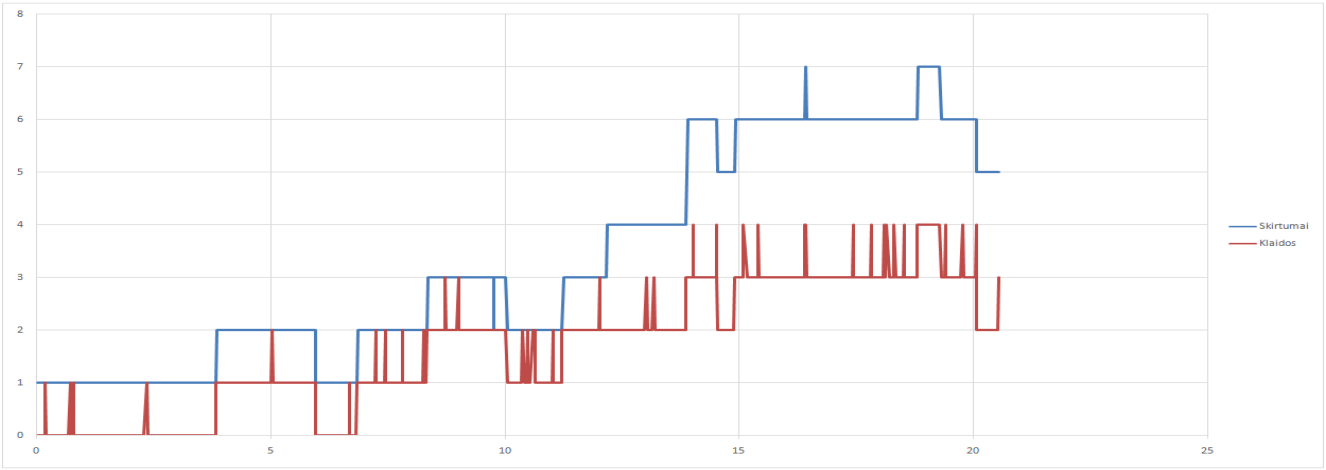
Šis metodas buvo išbandytas keičiant elementų priėmimo slenkstį.

Esant slenksčiui 0,3 grafikas vaizduojamas 7.2 pav.



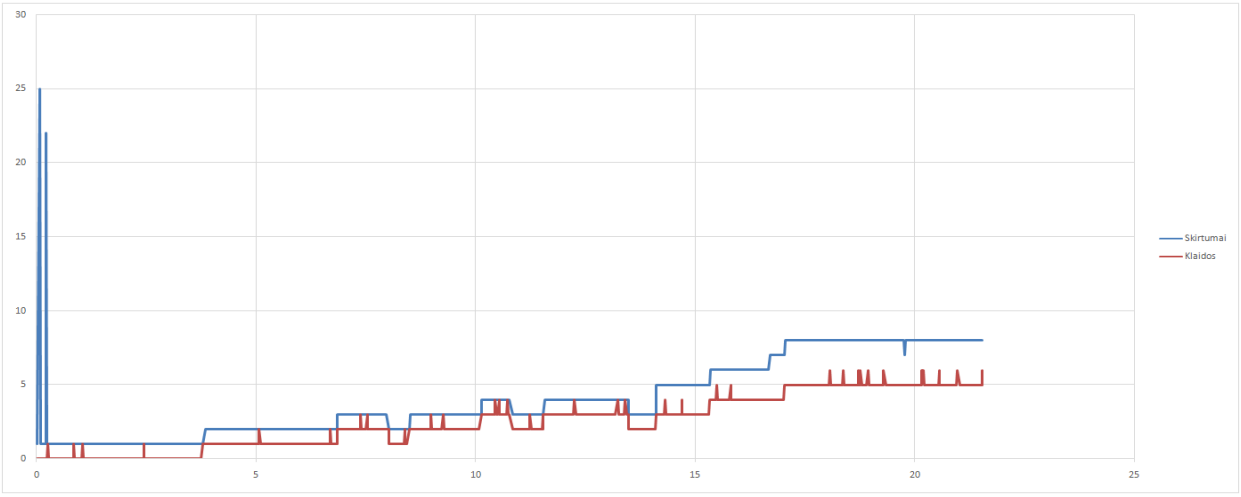
7.2 pav. Svorinis metodas. Naudojamas slenkstis – 0.3.

Iš grafiko galima matyti, kad skirtumai nuo originalios dokumento schemos nėra labai dideli, tik 2,5 skirtumo laike. Klaidų tikimybė taip pat nėra didelė: 0,8. Tačiau Galima matyti, kad tam tikrais atvejais metodas trumpais „smaigais“ bando susitvarkyti su elementais, kurie yra retesni.



7.3 pav. Svorinis metodas. Naudojamas slenkstis – 0,5

Esant slenksčiui 0,5 stebimas panašus veikimas (7.3 pav.). Bet šiuo atveju skirtumų skaičius siekia 3,3 ir klaidų skaičius yra vidutiniškai 1,6.



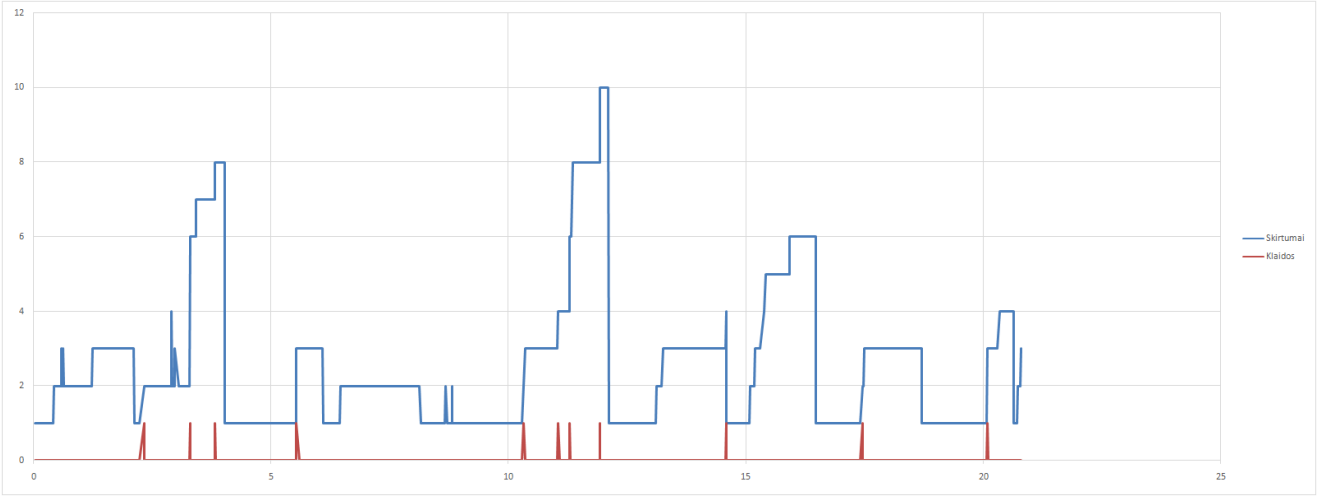
7.4 pav. Svorinis metodas. Naudojamas slenkstis – 0.7

Parinkus 0,7 slenkstį, gaunamas 4,07 vidutinis skirtumų skaičius ir 2,48 klaidos. Matoma (7.4 pav.), kaip pradžioje metodas kelis kartus sugeneravo schemą smarkai besiskyrusią nuo tikrosios. Taip dažniausiai būna dėl tipų skirtumo tarp skirtingų schemų, bet neįtakoja klaidų skaičiaus dokumentui.

7.1.3 Maišytas metodas

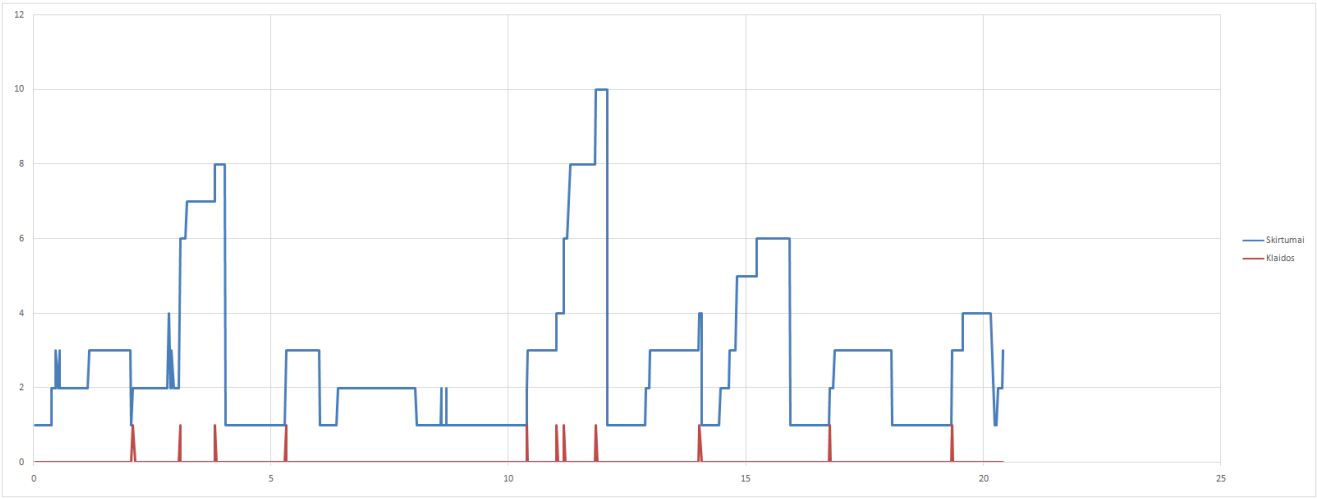
Maišyto algoritmo atveju taip pat buvo naudojami 3 svoriai atitinkami svorinio algoritmo atveju.

Grafike 7.5 pav. galima matyti, kada metodas persiskaičiuoja naudodamasis svoriniu algoritmu. Esant slenksčiui 0,3 ir persiskaičiuojant kas 100 dokumentų, vidutiniškai per laiko vienetą yra 2.6 skirtumo ir 0,01 klaidos.

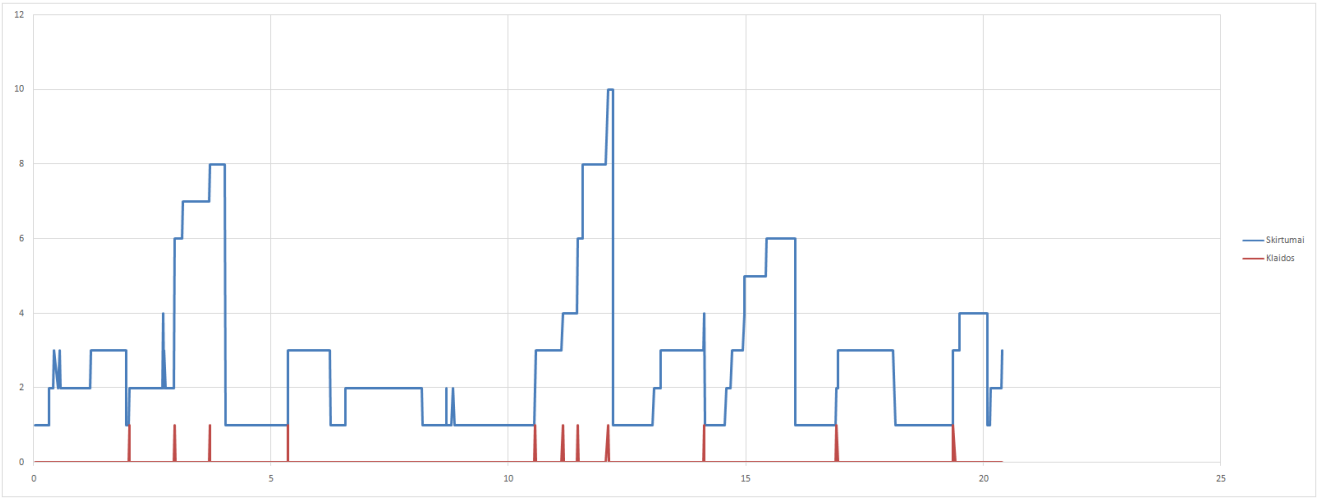


7.5 pav. Maišytas metodas. Slenkstis 0.3, persigeneravimas kas 100 dokumentų

Pasirinkus 0.5 slenkstį gaunama vidutiniškai 5,5 skirtumo ir 0,027 klaidos (7.6 pav.).



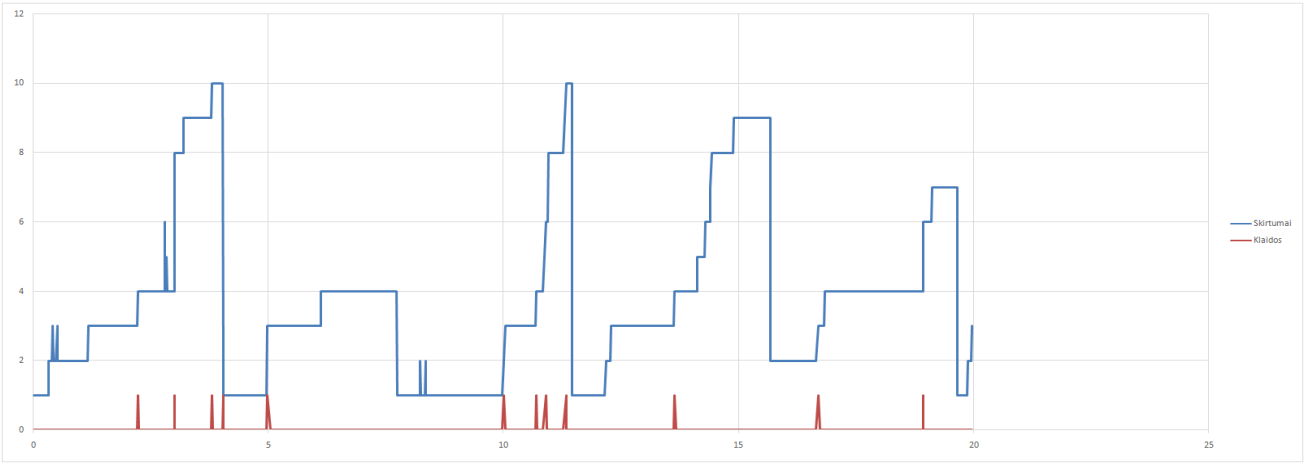
7.6 pav. Maišytas metodas. Slenkstis 0.5, persigeneravimas kas 100 dokumentų



7.7 pav. Maišytas metodas. Slenkstis 0.7, persigeneravimas kas 100 dokumentų

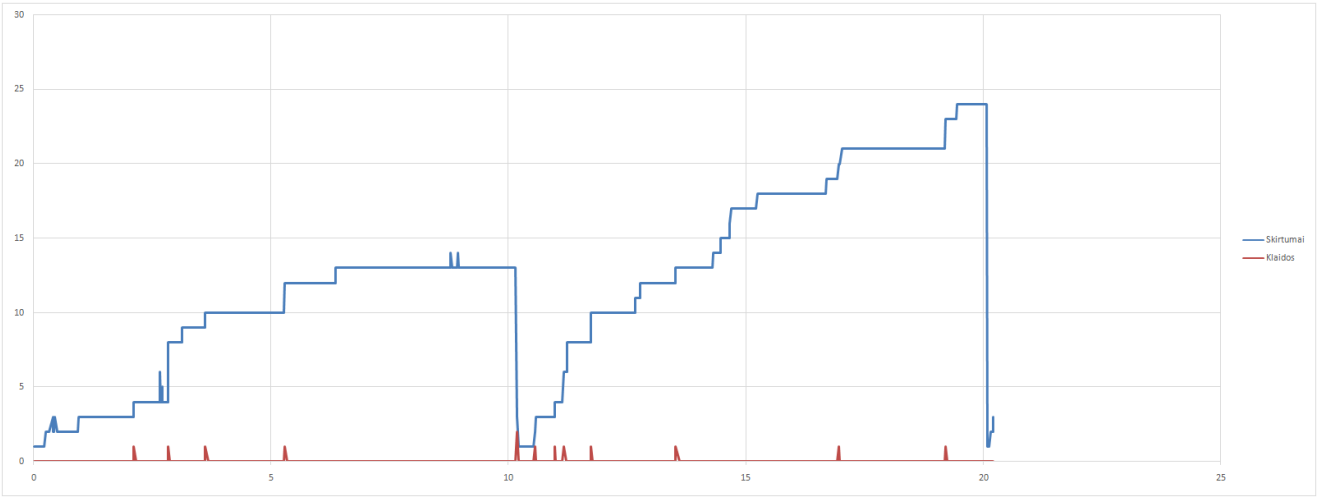
Esant 0,7 slenksčiui, vidutiniškai gaunama 0,03 klaidos ir 5,4 skirtumo per laiką.

Kadangi persiskaičiavimas vyksta tuo pačiu metu, skirtumų tarp metodų praktiškai nėra. Todėl paimtas slenkstis 0,5 ir keista dokumentų kaupimo trukmė.



7.8 pav. Maišytas metodas. Kaupiama 200 dokumentų, slenkstis – 0.5

Kaupiant 200 dokumentų iki persiskaičiamvimo, gaunama vidutiniškai 3,7 skirtumo ir 0,015 klaidos per laiko vienetą.

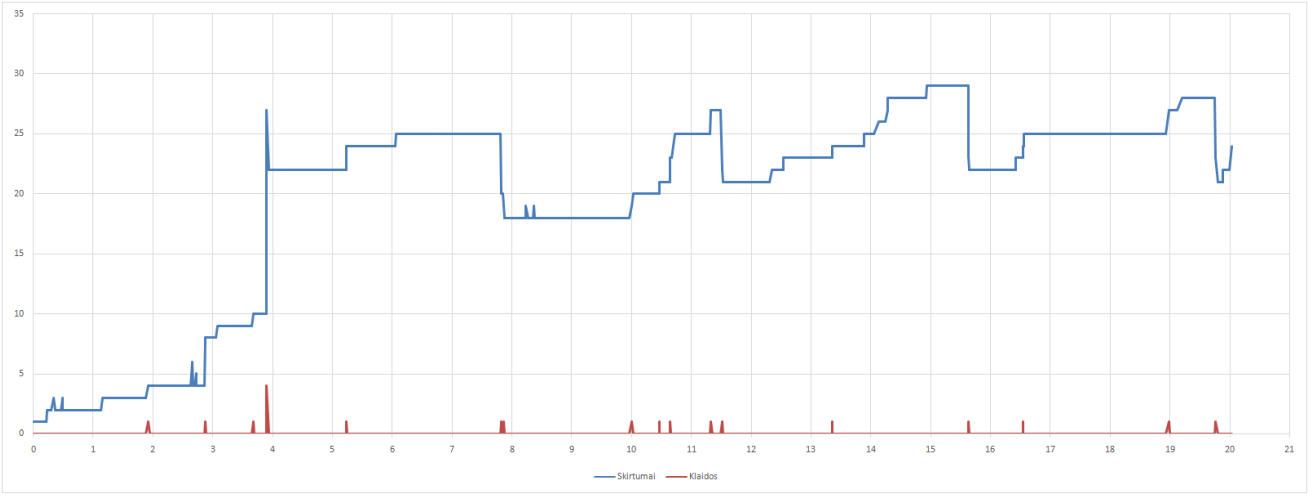


7.9 pav. Maišytas metodas. Kaupiama 500 dokumentų, slenkstis – 0.5

Kaupiant 500 dokumentų gauta 12,1 vidutinio skirtumo ir 0,012 klaidos. Galima matyti (7.9 pav.) kiek daug skirtumų dėl surinktų nespecifikuotų elementų XSD evoliucija surenka.

7.1.4 Genetinis metodas

Genetinis metodas buvo atliktas naudojant persigeneravimą kas 200 dokumentų.



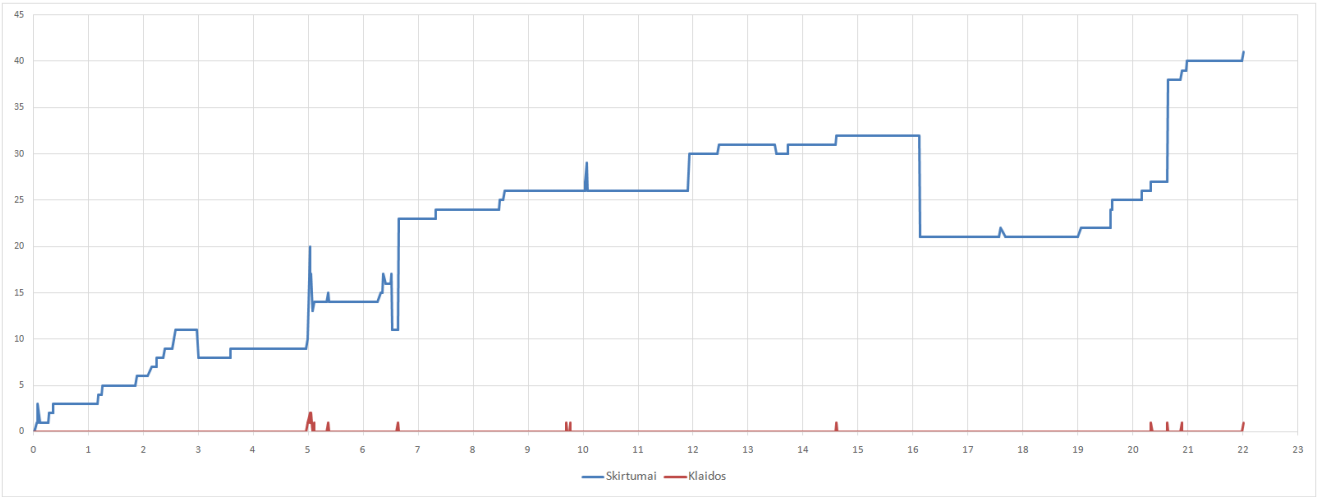
7.10 pav. Genetinio metodo rezultatai. Persigeneravimas kas 200 dokumentų.

Iš genetinio metodo rezultatų 7.10 pav. matoma, kad persigeneruojant skirtumų skaičius yra paliekamas kur kas didesnis: 19,5 skirtumo per laiko vienetą, klaidų skaičius gaunamas vidutiniškai 0,019 klaidos per laiko vienetą. Metodas nenoriai išsivalo, yra linkęs kaupti skirtumus panašiai kaip XSD evoliucija.

7.2 Kelių specifikacijų dokumentai

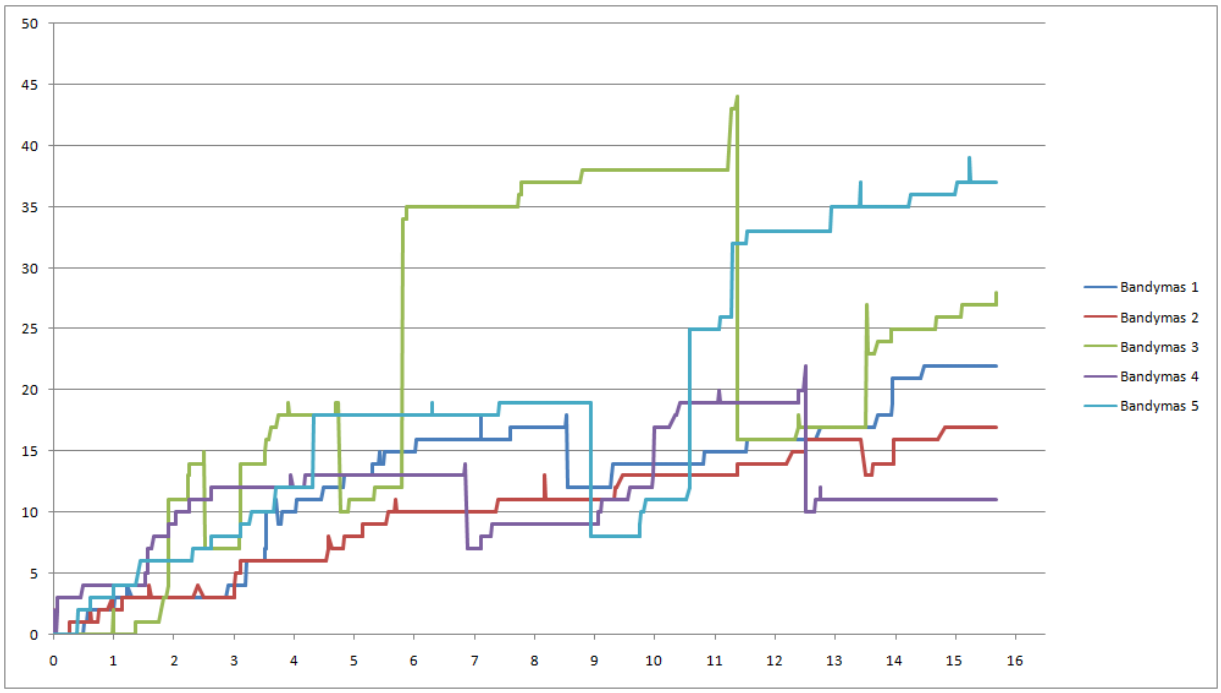
Atliekami eksperimentai, kai dokumento specifikacija kelis kartus kardinaliai pasikeičia. Paliekamas tik šakninis elementas. Stebima, kaip algoritmai susidoroja su kardinaliais pasikeitimais.

7.2.1 XSD evoliucija



7.11 pav. XSD evoliucija, daug dokumentų

Xsd evoliuciją atlikus su besikeičiančiomis schemomis gaunama vidutiniškai 21,3 skirtumo ir 0,015 klaidos per laiko vienetą. Grafike (7.11 pav.) stebimas ta pati skirtumų kaupimo tendencija.



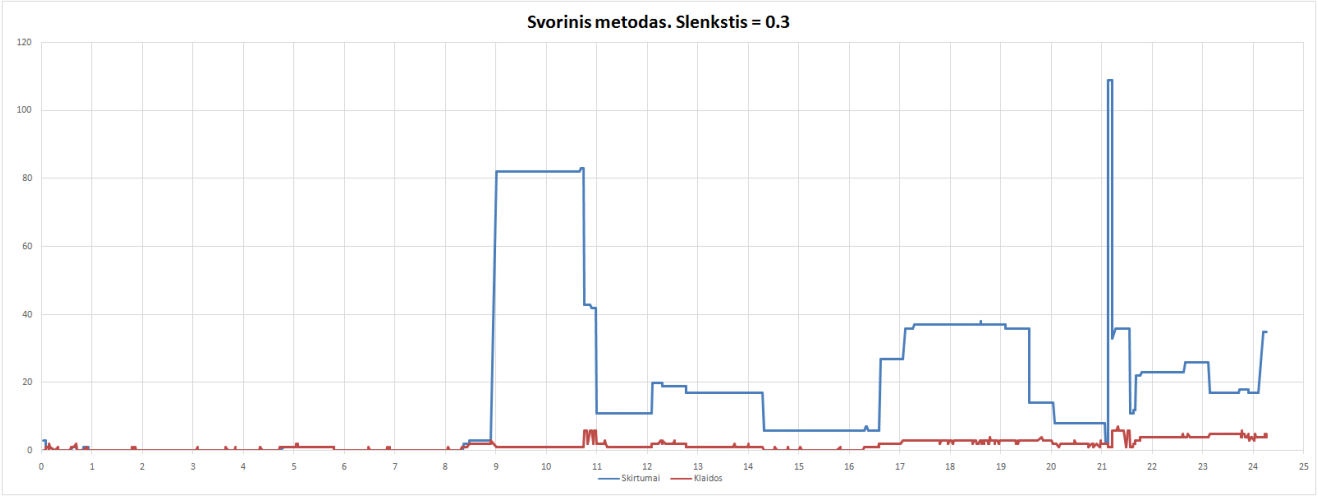
7.12 pav. XSD evoliucija keli bandymai

Metodas išbandomas generuojant kelias skirtingas imtis. Rezultatai (7.12 pav.) rodo tą pačia skirtumų kaupimo tendenciją. Gaunama vidutiniškai 21,7 skirtumo per laiko vienetą.

XSD evoliucijos metodas praktiškai nereaguoja į kardinalius pasikeitimus. Vykdant dokumento patikrą kardinalūs pasikeitimai tėra dar viena klaidų aibė, kurią reikia ištaisyti.

7.2.2 Svorinis metodas

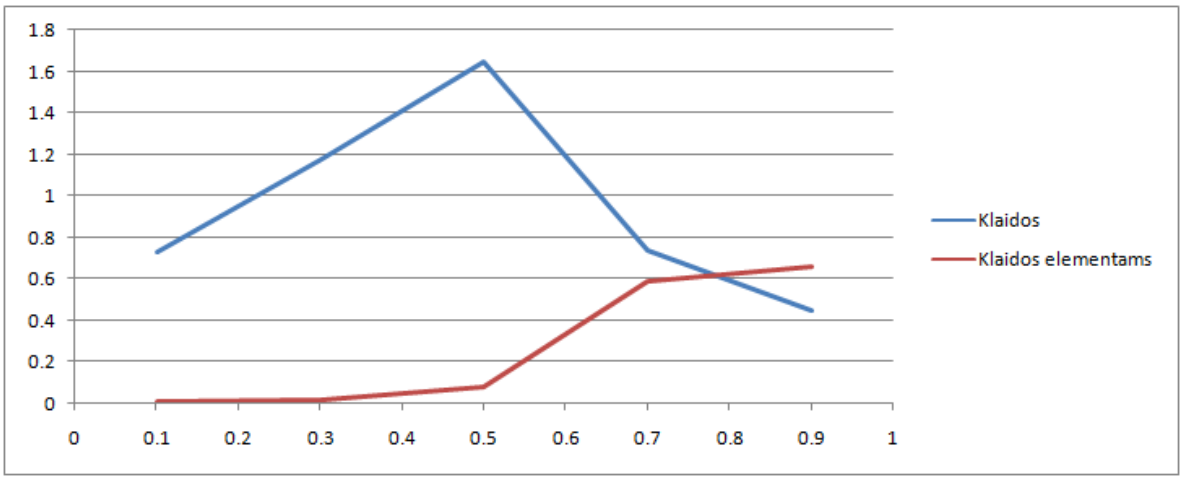
Svorinio metodo metu tikimasi stebėti didelį skirtumų ir klaidų kiekį schemai kardinaliai pasikeitus.



7.13 pav. Kelios schemos. Svorinis metodas. Slenkstis – 0.5

Pateiktame grafike 7.13 pav. galima matyti, kad schemos pasikeitimai buvo atlikti ties 9, 17, 21 laiko vienetu. Gaunama vidutiniškai 17,7 skirtumo ir 1,4 klaidos per laiko vienetą. Klaidų skaičius nėra didelis, skirtumai aiškiai matomi.

Svorinio algoritmo atveju buvo atliktas dar vienas eksperimentas, kurio metu buvo generuojama aibė dokumentų, kurių schema kelis kartus kardinaliai pasikeitė. Stebima, kaip algoritmai su skirtingais svoriais reaguoja į pasikeitimus klaidų prasme. Buvo atliekama 20 bandymų.



7.14 pav. Klaidų ir klaidų santykio elementams grafikas

Iš grafiko 7.14 pav. galima spręsti, kad mažiausią klaidų skaičių duoda svorinis metodas su didžiausiu slenksčiu. Atsižvelgiant į tai, kad schema kardinaliai keičiasi 3 kartus, tuo patikėti yra sunku. Pridedamas klaidų santykis su elementų skaičiumi. Ši kreivė rodo, kad vidutiniškai esant slenksčiui 0,9 vienam schemos elementui ten po 0,659 klaidos. Patikros klaidos. Visi duomenys pateikiami 7.1 lentelėje.

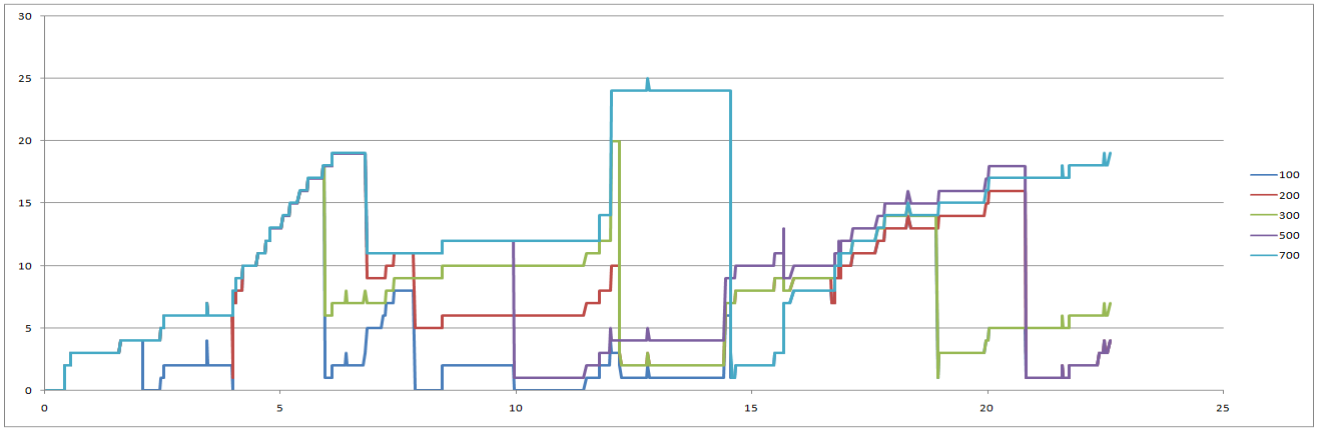
7.1 lentelė. Testavimo rezultatai

Slenkstis	Klaidos	Vid. skirtumai	Klaidos elementams
0.1	0.728194029	10.12712576	0.010447494
0.3	1.175537332	16.72588473	0.02106793
0.5	1.643606517	28.39472715	0.083074099
0.7	0.735424342	24.4195917	0.590859781
0.9	0.445740842	22.13246723	0.659774436

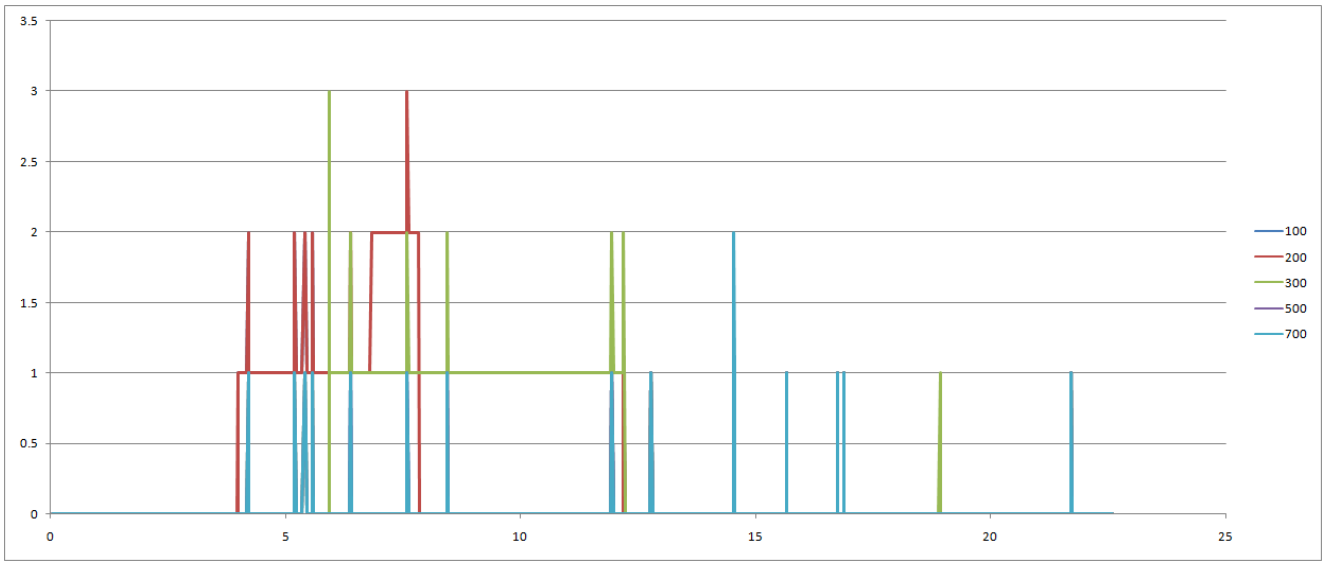
Gauti rezultatai rodo, kad didžiausio svorinio koeficiento metodas nors ir nurodo, kad turi mažiausiai klaidų iš tikro klaidos yra kur kas svarbesnės, nes yra ignoruojami ištisi XML dokumento elementai su vaikais t. y. gaunama patikros klaida yra tik dėl kažkurio elemento nebūvimo, tačiau negaunamos klaidos dėl pastarojo elemento vaikų neegzistavimo.

7.2.3 Maišytas metodas

Maišyto metodo palyginimas keičiant persigeneravimo tašką



7.15 pav. Maišytas metodas skirtumai



7.16 pav. Maišytas metodas. Klaidos

7.15 pav. ir 7.16 pav. Matoma, kaip keičiasi skirtumų ir klaidų kiekis, keičiant persigeneravimo tašką.

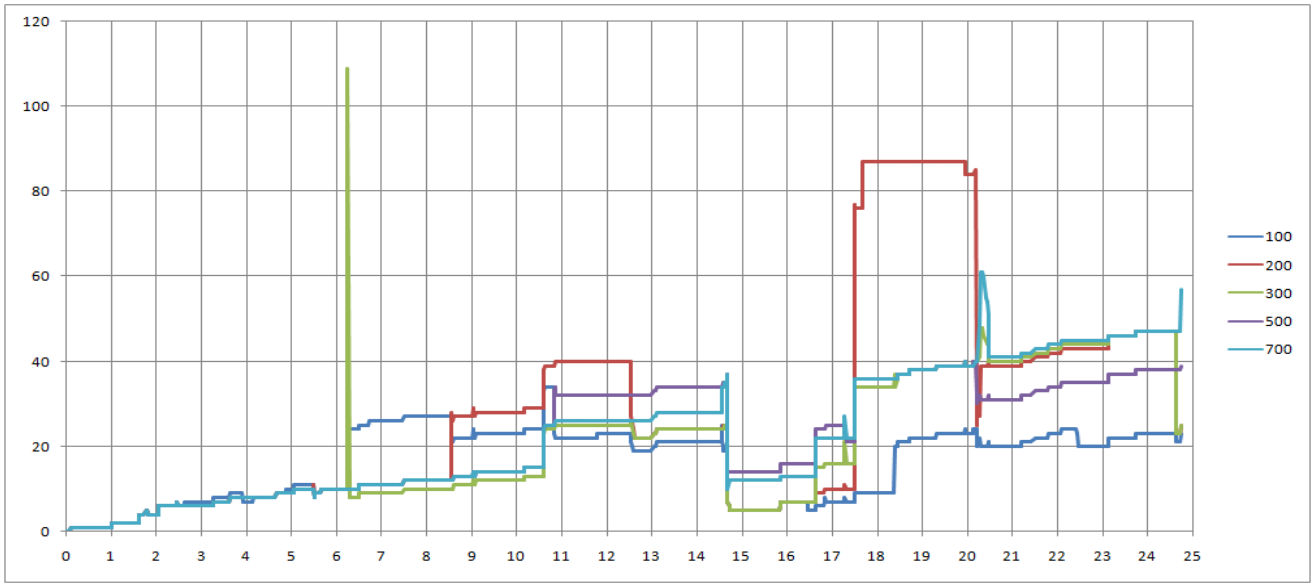
7.2 lentelė. Maišyto metodo rezultatai

Persigeneravimas kas (elementų)	**100**	**200**	**300**	**500**	**700**
Vidutiniškai skirtumų	4.25483	7.839926	7.465501	8.696412	12.18123
Vidutiniškai klaidų	0.103956	0.25023	0.292548	0.01196	0.014719

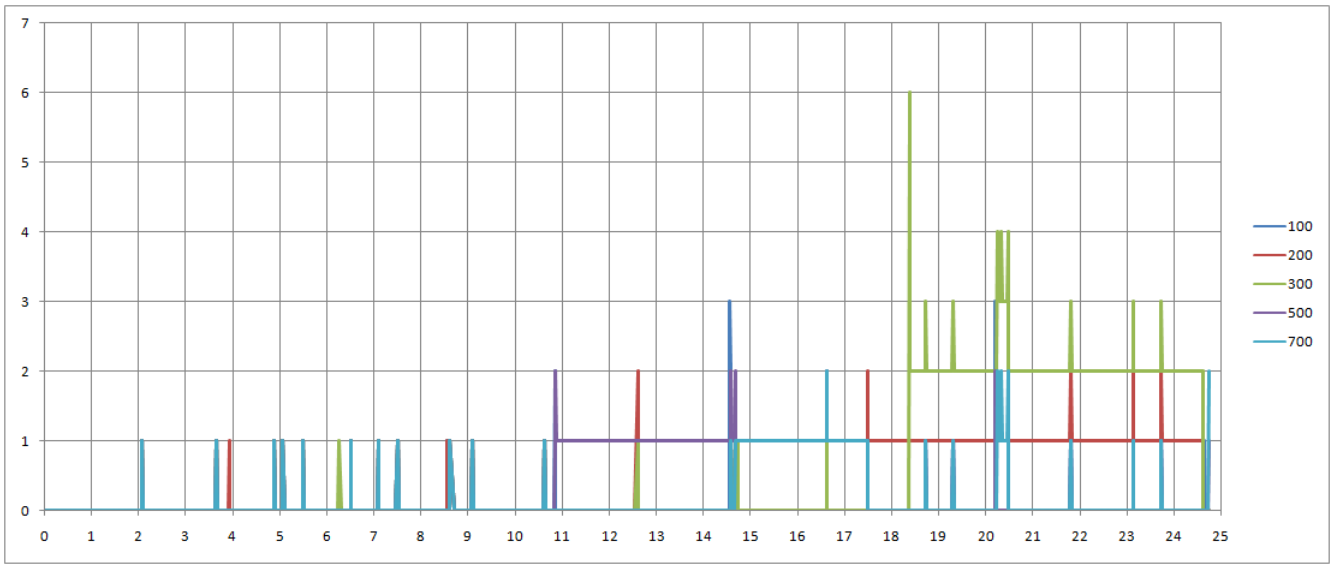
Pagal 7.2 lentelę matoma, kad mažiausiai skirtumų yra, kai persigeneruojama kas 100 elementų, o mažiausiai klaidų, kai pergeneravimas vyksta kas 500 elementų.

7.2.4 Genetinis metodas

Genetinio metodo metu buvo keičiamas persigeneravimo taškas



7.17 pav. Genetinis metodas – skirtumai.



7.18 pav. Genetinis metodas. Klaidos

7.3 lentelė. Genetinio metodo rezultatai

Elementų skaičius	100	200	300	500	700
Vidutiniškai skirtumų	16.0182	28.61621	20.50124	21.95368	22.44169
Vidutiniškai klaidų	0.025641	0.304384	0.526055	0.312655	0.146402

Iš rezultatų matoma, kad metodas nepriklausomai nuo pasirinkto pergeneravimo taško yra linkęs kaupti skirtumus. Mažiausiai klaidų daroma, kai persigeneruota buvo kas 100 elementų. Šia prasme nors skirtumų skaičius yra kur kas didesnis nei maišyto metodo atveju, tačiau klaidų yra gerokai mažiau.

7.3 Metodų taikymo rekomendacijos

- Iš rezultatų matoma, kad XSD evoliucijos metodas suteikia mažiausią galimų klaidų tikimybę, tačiau yra labiausiai linkęs toli nuo realios specifikacijos.
- Maišytas metodas klaidų prasme nedarė smarkiai daugiau klaidų, taip pat sumažino skirtumų kiekį.
- Šio metodo patobulinimas genetiniais principais šiek tiek pagerinti metodo veikimą klaidų atžvilgiu, tačiau padidino skirtumų kiekį.
- Svorinis metodas darė daugiausia klaidų, tačiau skirtumų skaičius nuo realios specifikacijos buvo gana mažas. Svorinio algoritmo slenksčio parinkimas yra didžiausias iššūkis. Parinkus per didelį, galima gauti didelį klaidų skaičių vienam elementui.

8 DARBO REZULTATAI IR IŠVADOS

Pasiūlyti keturi metodai dokumento specifikacijos atkūrimui:

1. Atvirkštine patikra paremta XSD schemos evoliucija;
2. Svoriniais koeficientais besiremiantis metodas;
3. Šių abiejų metodų sudėtinis metodas;
4. Pastarojo metodo versija pasiremianti genetinių (evoliucinių) metodų principais.

Atliktas šių metodų tyrimas, kaip jie veikia esant dokumentams su klaidomis bei kardinaliai besikeičiant specifikacijoms.

Sukurtas programinės įrangos pavyzdys, gebantis reaguoti į dokumento specifikacijos pasikeitimus ir dalį savęs regeneruoti.

Remiantis gautais rezultatais, galima padaryti šias išvadas:

1. Atlikus esamų sprendimų analizė parodė, kad egzistuojanti programinė įranga dažniausiai naudojasi tik vienu XML dokumentu, iš kurio kuriama numanoma specifikacija. O platesnį funkcionalumą suteikianti programinė įranga yra sunkiai integruojama į kitas sistemas.
2. Atvirkštine patikra paremtas XSD evoliucijos metodas, naudojamas dokumento specifikacijos atkūrimui, suteikia mažiausią galimų klaidų tikimybę, tačiau, laikui bėgant, yra labiausiai linkęs tolti nuo realios specifikacijos, jeigu atsiunčiamuose dokumentuose yra trikdžių. Šis metodas geriausiai tinka, jeigu reikia palaikyti kelias dokumentų specifikacijos versijas.
3. Svoriais paremtas metodas, naudojamas dokumento specifikacijos atkūrimui, nors ir negarantuoja visų elementų patekimą į turimą specifikaciją, tačiau skirtumų skaičius nuo realios specifikacijos yra gana mažas, jeigu pastaroji kinta nedažnai ir ne iš esmės. Šio metodo veikimas labai priklauso nuo slenksčio parinkimo. Geriausi rezultatai gauti pasirinkus mažus slenksčiais. Mažos slenksčio reikšmės sudaro galimybę patekti atsitiktiniams dokumento struktūros pokyčiams. Dideli slenksčiai gali ignoruoti didelę specifikacijos dalį pakankamai ilgą laiko tarpą.
4. Maišant XSD evoliucijos bei svorinį metodus gauti rezultatai rodė, kad toks metodas suteikia geresnį elementų patekimą lyginant su svoriniu metodu. Turimos specifikacijos keitimui, naudojant šį metodą, sumažina skirtumų kiekį tarp turimos ir realios specifikacijos.
5. Kai galimas dažnas specifikacijos kitimas ir specifikacija gali kisti iš esmės. Tada, taikant genetinių algoritmų principus, pavyksta pagerinti maišyto metodo veikimą elementų patekimo atžvilgiu.
6. Komponento regeneravimas .NET aplinkoje pavyksta atlikti tik iš dalies. Naudojama domenais paremta programos struktūra leidžia programos veikimo metu įkelti bibliotekas ir jas panaudojus regeneruoti. Tačiau pilnas komponento pašalinimas nėra įmanomas programos veikimo metu, nes .NET leidžia šalinti iš sistemos esamus šalutinius domenus, tačiau neleidžia šalinti komponentų (dll), kurių funkcionalumas buvo panaudotas bent kartą pagrindiniame domene.

Literatūra

[1] Office Open XML Overview [interaktyvus] [žiūrėta 2014-01-28]: http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf

[2] Document management – Portable Document Format Part 1 [interaktyvus] [žiūrėta 2014-01-28]:http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf

[3] OASIS Open Document Format for Office Applications (OpenDocument) TC [interaktyvus] [žiūrėta 2014-01-28]: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office

[4] Stergiou C. Signaos D. Neural Networks [interaktyvus] [žiūrėta 2014-01-28]: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[5] Jones. G. Genetic and Evolutionary algorithms [interaktyvus] [žiūrėta 2014-01-28]: http://www.wiley.com/legacy/wileychi/ecc/samples/sample10.pdf

[6] Budin A.; Golub. M.; Budn L.; Traditional Techniques of Genetic Algorithms Applied to Floating – Point Chrmosome Representations [interaktyvus] [žiūrėta 2014-01-28]: http://www.researchgate.net/publication/228500720_Traditional_Techniques_of_Genetic_Algorithms_Applied_to_Floating-Point_Chromosome_Representations/file/32bfe514234dea522e.pdf

[7] Kurmar A.; Encoding Shemes in genetic algorithm International Journal of Advanced Research in IT and Engineering ISSN: 2278-6244

[8] Kumar R. Blending Roulette Wheel Selection & Rank Selection in Genetic Algorithms; International Journal of Machine Learning and Computing, Vol. 2, No. 4, August 2012

[9] Miller B.L.; Goldberg D.E. Genetic Algorithms, Tournament Selection, and the Effects of Noise; Complex Systems 9 (1995) 193- 212

[10] Deepa S.N.; Sivanandam S.N. Introduction to genetic algorithms. ISBN 978-3-540-73189-4 Springer Berlin Heidelberg New York

[11] Mocato P. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts [interaktyvus] [žiūrėta 2014-01-28]: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.9474&rep=rep1&type=pdf

[12] Shrobe H. Laddga R. Balzer B. Goldman N.; Self-Adaptive Systems for Information Survivability: PMOP and AWDRAT [interaktyvus] [žiūrėta 2014-01-28]: http://dspace.mit.edu/bitstream/handle/1721.1/37151/MIT-CSAIL-TR-2007-023.pdf?sequence=1

[13] Pacheco J.; Aragon A.; Krasnogor N.; Memetic Algorithms [interaktyvus] [žiūrėta 2014-01-28]: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.4618&rep=rep1&type=pdf

[14] Chan S.; Kong C.C. Vincent.Ng. Mapping XML Schema to Relations Using Genetic Algorithm. 8th International Conference, KES 2004, Wellington, New Zealand, September 20-25, 2004, Proceedings, Part III ISBN= 978-3-540-23205-6

[15] Ghosh S.; Nag.A,; Biswas D.;Pal A.; Biswas S.; Sarkar D. XML Mining Using Genetic Algorithm; Volume 2, No. 5, April 2011 Journal of Global Research in Computer Science ISSN-2229-371X

[16] Srinivasa K. G., Sharath S., Venugopal K. R., Patnaik Lalit M.; Indexing XML Documents Using Self Adaptive Genetic Algorithms for Better Retreival 8th Asia-Pacific Web Conference, Harbin, China, January 16-18, 2006. Proceedings, ISBN 978-3-540-31142-3

[17] Havrikov N.; Höschele M. Galeotti J.P.; Zeller A. Evolutionary XML Test Generation [interaktyvus] [žiūrėta 2014-06-10]: https://www.st.cs.uni-saarland.de/testing/xmlmate/XMLMATE.pdf

[18] Tan M., Goh A.; Keeping Pace with Evolving XML-Based Specifications.; Current Trends in Database Technology - EDBT 2004 Workshops p. 280-288 ISBN= 978-3-540-23305-3

[19] Choi B.; What are RealDTDs Like; [interaktyvus] [žiūrėta 2015-05-01]: http://repository.upenn.edu/cgi/viewcontent.cgi?article=1035&context=cis_reports

[20] Ferrandina F.; Schema and Database Evolution in the O2 Object Database [interaktyvus] [žiūrėta 2014-05-28]: http://www.vldb.org/conf/1995/P170.PDF

[21] Sheard T. Accomplishments and Research Challenges in Meta-programming; Second International Workshop, SAIG 2001 Florence, Italy, September 6, 2001 Proceedings ISBN 978-3-540-42558-8

[22] Dassen J.H.M. Dynamic Loading [interaktyvus] [žiūrėta 2014-05-28]: http://linux4u.jinr.ru/usoft/WWW/www_debian.org/Documentation/elf/node7.html

[23] How to: Load and Unload Assemblies [interaktyvus] [žiūrėta 2014-05-28]: http://msdn.microsoft.com/en-us/library/ms173101.aspx

[24] How to: Unload an Application Domain [interaktyvus] [žiūrėta 2014-05-28]: http://msdn.microsoft.com/en-us/library/c5b8a8f9.aspx

[25] The JarClassLoader Class [interaktyvus] [žiūrėta 2014-05-28]: http://docs.oracle.com/javase/tutorial/deployment/jar/jarclassloader.html

[26] Convenience wrappers for __import__ [interaktyvus] [žiūrėta 2014-05-28]: https://docs.python.org/2/library/importlib.html

[27] XML Schema Part 0: Primer Second Edition [interakvytus] [žiūrėta 2014-05-28]: http://www.w3.org/TR/xmlschema-0/

[28] Generating XML Documents from XML Schemas [interakvytus] [žiūrėta 2015-05-20]: https://msdn.microsoft.com/en-us/library/aa302296.aspx

[29] Using the XML Diff and Patch Tool in Your Applications [interakvytus] [žiūrėta 2015-05-20]: https://msdn.microsoft.com/en-us/library/aa302294.aspx

9 PRIEDAI

9.1 Generuotas programinis kodas 1

```
   public partial class Cart {

       private List<CartProduct> itemsField;

       public Cart() {
           this.itemsField = new List<CartProduct>();
       }

       [System.Xml.Serialization.XmlElementAttribute("Product",
Form=System.Xml.Schema.XmlSchemaForm.Unqualified)]
       public List<CartProduct> Items {
           get {
               return this.itemsField;
           }
           set {
               this.itemsField = value;
           }
       }
   }
   public partial class CartProduct {

       private string nameField;

       private string priceField;

       private List<CartProductSpecification> specificationField;

       private string weightField;

       public CartProduct() {
           this.specificationField = new List<CartProductSpecification>();
       }


[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unquali
fied)]
       public string Name {
           get {
               return this.nameField;
           }
           set {
               this.nameField = value;
           }
       }


[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unquali
fied)]
       public string Price {
           get {
               return this.priceField;
           }
           set {
               this.priceField = value;
           }
       }

       public List<CartProductSpecification> Specification {
           get {
               return this.specificationField;
           }
           set {
               this.specificationField = value;
```

```
            }
        }

        [System.Xml.Serialization.XmlAttributeAttribute()]
        public string Weight {
            get {
                return this.weightField;
            }
            set {
                this.weightField = value;
            }
        }
    }
    public partial class CartProductSpecification {

        private string heightField;

        private string colorField;


[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unquali
fied)]
        public string Height {
            get {
                return this.heightField;
            }
            set {
                this.heightField = value;
            }
        }


[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unquali
fied)]
        public string Color {
            get {
                return this.colorField;
            }
            set {
                this.colorField = value;
            }
        }
    }
```

9.2 Generuotas programinis kodas 2

```
    // ------------------------------------------------------------------------------
    //  <auto-generated>
    //    Generated by Xsd2Code. Version 3.4.0.24492 Microsoft Reciprocal License (Ms-RL)
    //
<NameSpace></NameSpace><Collection>List</Collection><codeType>CSharp</codeType><EnableData
Binding>False</EnableDataBinding><EnableLazyLoading>False</EnableLazyLoading><TrackingChan
gesEnable>False</TrackingChangesEnable><GenTrackingClasses>False</GenTrackingClasses><Hide
PrivateFieldInIDE>False</HidePrivateFieldInIDE><EnableSummaryComment>False</EnableSummaryC
omment><VirtualProp>False</VirtualProp><IncludeSerializeMethod>False</IncludeSerializeMeth
od><UseBaseClass>False</UseBaseClass><GenBaseClass>False</GenBaseClass><GenerateCloneMetho
d>False</GenerateCloneMethod><GenerateDataContracts>False</GenerateDataContracts><CodeBase
Tag>Net40</CodeBaseTag><SerializeMethodName>Serialize</SerializeMethodName><DeserializeMet
hodName>Deserialize</DeserializeMethodName><SaveToFileMethodName>SaveToFile</SaveToFileMet
hodName><LoadFromFileMethodName>LoadFromFile</LoadFromFileMethodName><GenerateXMLAttribute
s>True</GenerateXMLAttributes><OrderXMLAttrib>False</OrderXMLAttrib><EnableEncoding>False<
/EnableEncoding><AutomaticProperties>False</AutomaticProperties><GenerateShouldSerialize>F
alse</GenerateShouldSerialize><DisableDebug>False</DisableDebug><PropNameSpecified>Default
</PropNameSpecified><Encoder>UTF8</Encoder><CustomUsings></CustomUsings><ExcludeIncludedTy
pes>False</ExcludeIncludedTypes><EnableInitializeFields>True</EnableInitializeFields>
    //  </auto-generated>
    // ------------------------------------------------------------------------------
    using System;
    using System.Diagnostics;
    using System.Xml.Serialization;
```

```
   using System.Collections;
   using System.Xml.Schema;
   using System.ComponentModel;
   using System.Collections.Generic;
   [System.CodeDom.Compiler.GeneratedCodeAttribute("ConsoleApp", "1.0.0.0")]
   [System.SerializableAttribute()]
   [System.ComponentModel.DesignerCategoryAttribute("code")]
   [System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]
   [System.Xml.Serialization.XmlRootAttribute(Namespace="", IsNullable=false)]
   public partial class Cart {
       private List<CartProduct> itemsField;

       public Cart() {
           this.itemsField = new List<CartProduct>();
       }
       [System.Xml.Serialization.XmlElementAttribute("Product",
Form=System.Xml.Schema.XmlSchemaForm.Unqualified)]
       public List<CartProduct> Items {
           get {
               return this.itemsField;
           }
           set {
               this.itemsField = value;
           }
       }
   }
   [System.CodeDom.Compiler.GeneratedCodeAttribute("ConsoleApp", "1.0.0.0")]
   [System.SerializableAttribute()]
   [System.ComponentModel.DesignerCategoryAttribute("code")]
   [System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]
   public partial class CartProduct {

       private string nameField;

       private double priceField;

       private bool priceFieldSpecified;

       private List<CartProductSpecification> specificationField;

       private int weightField;

       private bool weightFieldSpecified;

       public CartProduct() {
           this.specificationField = new List<CartProductSpecification>();
       }


[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unquali
fied)]
       public string Name {
           get {
               return this.nameField;
           }
           set {
               this.nameField = value;
           }
       }


[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unquali
fied)]
       public double Price {
           get {
               return this.priceField;
           }
           set {
               this.priceField = value;
           }
       }
```

```
        [System.Xml.Serialization.XmlIgnoreAttribute()]
        public bool PriceSpecified {
            get {
                return this.priceFieldSpecified;
            }
            set {
                this.priceFieldSpecified = value;
            }
        }

        [System.Xml.Serialization.XmlElementAttribute("Specification",
Form=System.Xml.Schema.XmlSchemaForm.Unqualified)]
        public List<CartProductSpecification> Specification {
            get {
                return this.specificationField;
            }
            set {
                this.specificationField = value;
            }
        }

        [System.Xml.Serialization.XmlAttributeAttribute()]
        public int Weight {
            get {
                return this.weightField;
            }
            set {
                this.weightField = value;
            }
        }

        [System.Xml.Serialization.XmlIgnoreAttribute()]
        public bool WeightSpecified {
            get {
                return this.weightFieldSpecified;
            }
            set {
                this.weightFieldSpecified = value;
            }
        }
    }
    [System.CodeDom.Compiler.GeneratedCodeAttribute("ConsoleApp", "1.0.0.0")]
    [System.SerializableAttribute()]
    [System.ComponentModel.DesignerCategoryAttribute("code")]
    [System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]
    public partial class CartProductSpecification {

        private int heightField;

        private bool heightFieldSpecified;

        private string colorField;


[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unquali
fied)]
        public int Height {
            get {
                return this.heightField;
            }
            set {
                this.heightField = value;
            }
        }

        [System.Xml.Serialization.XmlIgnoreAttribute()]
        public bool HeightSpecified {
            get {
                return this.heightFieldSpecified;
            }
            set {
                this.heightFieldSpecified = value;
```

```
            }
        }


[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unquali
fied)]
        public string Color {
            get {
                return this.colorField;
            }
            set {
                this.colorField = value;
            }
        }
    }
    // ------------------------------------------------------------------------------
    //  <auto-generated>
    //    Generated by Xsd2Code. Version 3.4.0.24492 Microsoft Reciprocal License (Ms-RL)
    //
<NameSpace></NameSpace><Collection>List</Collection><codeType>CSharp</codeType><EnableData
Binding>False</EnableDataBinding><EnableLazyLoading>False</EnableLazyLoading><TrackingChan
gesEnable>False</TrackingChangesEnable><GenTrackingClasses>False</GenTrackingClasses><Hide
PrivateFieldInIDE>False</HidePrivateFieldInIDE><EnableSummaryComment>False</EnableSummaryC
omment><VirtualProp>False</VirtualProp><IncludeSerializeMethod>False</IncludeSerializeMeth
od><UseBaseClass>False</UseBaseClass><GenBaseClass>False</GenBaseClass><GenerateCloneMetho
d>False</GenerateCloneMethod><GenerateDataContracts>False</GenerateDataContracts><CodeBase
Tag>Net40</CodeBaseTag><SerializeMethodName>Serialize</SerializeMethodName><DeserializeMet
hodName>Deserialize</DeserializeMethodName><SaveToFileMethodName>SaveToFile</SaveToFileMet
hodName><LoadFromFileMethodName>LoadFromFile</LoadFromFileMethodName><GenerateXMLAttribute
s>True</GenerateXMLAttributes><OrderXMLAttrib>False</OrderXMLAttrib><EnableEncoding>False<
/EnableEncoding><AutomaticProperties>False</AutomaticProperties><GenerateShouldSerialize>F
alse</GenerateShouldSerialize><DisableDebug>False</DisableDebug><PropNameSpecified>Default
</PropNameSpecified><Encoder>UTF8</Encoder><CustomUsings></CustomUsings><ExcludeIncludedTy
pes>False</ExcludeIncludedTypes><EnableInitializeFields>True</EnableInitializeFields>
    //  </auto-generated>
    // ------------------------------------------------------------------------------
    using System;
    using System.Diagnostics;
    using System.Xml.Serialization;
    using System.Collections;
    using System.Xml.Schema;
    using System.ComponentModel;
    using System.Collections.Generic;
    namespace Custom
    {
        [System.CodeDom.Compiler.GeneratedCodeAttribute("ConsoleApp", "1.0.0.0")]
        [System.SerializableAttribute()]
        [System.ComponentModel.DesignerCategoryAttribute("code")]
        [System.Xml.Serialization.XmlTypeAttribute(AnonymousType = true)]
        [System.Xml.Serialization.XmlRootAttribute(Namespace = "", IsNullable = false)]
        public partial class Cart
        {
            private List<Product> productField;

            public Cart()
            {
                this.productField = new List<Product>();
            }
            [System.Xml.Serialization.XmlElementAttribute("Product")]
            public List<Product> Product
            {
                get
                {
                    return this.productField;
                }
                set
                {
                    this.productField = value;
                }
            }
        }
        [System.CodeDom.Compiler.GeneratedCodeAttribute("ConsoleApp", "1.0.0.0")]
```

```
        [System.SerializableAttribute()]
        [System.ComponentModel.DesignerCategoryAttribute("code")]
        [System.Xml.Serialization.XmlTypeAttribute(AnonymousType = true)]
        [System.Xml.Serialization.XmlRootAttribute(Namespace = "", IsNullable = false)]
        public partial class Product
        {
            private string nameField;
            private Specification specificationField;
            private double priceField;
            private bool priceFieldSpecified;
            private int weightField;
            public Product()
            {
                this.specificationField = new Specification();
            }
            [System.Xml.Serialization.XmlElementAttribute(Form =
System.Xml.Schema.XmlSchemaForm.Unqualified)]
            public string Name
            {
                get
                {
                    return this.nameField;
                }
                set
                {
                    this.nameField = value;
                }
            }
            public Specification Specification
            {
                get
                {
                    return this.specificationField;
                }
                set
                {
                    this.specificationField = value;
                }
            }
            [System.Xml.Serialization.XmlElementAttribute(Form =
System.Xml.Schema.XmlSchemaForm.Unqualified)]
            public double Price
            {
                get
                {
                    return this.priceField;
                }
                set
                {
                    this.priceField = value;
                }
            }
            [System.Xml.Serialization.XmlIgnoreAttribute()]
            public bool PriceSpecified
            {
                get
                {
                    return this.priceFieldSpecified;
                }
                set
                {
                    this.priceFieldSpecified = value;
                }
            }
            [System.Xml.Serialization.XmlAttributeAttribute()]
            public int Weight
            {
                get
                {
                    return this.weightField;
                }
                set
```

```
            {
                this.weightField = value;
            }
        }
    }
    [System.CodeDom.Compiler.GeneratedCodeAttribute("ConsoleApp", "1.0.0.0")]
    [System.SerializableAttribute()]
    [System.ComponentModel.DesignerCategoryAttribute("code")]
    [System.Xml.Serialization.XmlTypeAttribute(AnonymousType = true)]
    [System.Xml.Serialization.XmlRootAttribute(Namespace = "", IsNullable = false)]
    public partial class Specification
    {
        private int heightField;
        private bool heightFieldSpecified;
        private string colorField;
        [System.Xml.Serialization.XmlElementAttribute(Form =
System.Xml.Schema.XmlSchemaForm.Unqualified)]
        public int Height
        {
            get
            {
                return this.heightField;
            }
            set
            {
                this.heightField = value;
            }
        }

        [System.Xml.Serialization.XmlIgnoreAttribute()]
        public bool HeightSpecified
        {
            get
            {
                return this.heightFieldSpecified;
            }
            set
            {
                this.heightFieldSpecified = value;
            }
        }

        [System.Xml.Serialization.XmlElementAttribute(Form =
System.Xml.Schema.XmlSchemaForm.Unqualified)]
        public string Color
        {
            get
            {
                return this.colorField;
            }
            set
            {
                this.colorField = value;
            }
        }
    }
}
```

9.3 XSD patirkos informacijos saugojimo klasė

```
    public enum SchemaActions
    {
        AddElement = 1,
        AddAttribute,
        ChangeElementType,
        ChangeAttributeType,
        ChangeToList,
        RemoveElement,
        RemoveAttribute,
        Unknown
```

```
        }
        public class SchemaValidationData
        {
            public SchemaActions Action
            {
                get
                {
                    if (Message.Contains("has invalid child element"))
                    {
                        return SchemaActions.AddElement;
                    }
                    if (Message.Contains("attribute is not declared"))
                        return SchemaActions.AddAttribute;
                    if (Message.Contains("element is invalid - The value"))
                        return SchemaActions.ChangeElementType;
                    if (Message.Contains("attribute is invalid - The value"))
                        return SchemaActions.ChangeAttributeType;
                  return SchemaActions.Unknown;
                }
            }
            public XObject Object { get; set; }
            public XElement Element
            {
                get
                {
                    if (Object.NodeType == XmlNodeType.Element)
                        return Object as XElement;
                    return null;
                }
            }
            public string Message { get; set; }
            public string Parent
            {
                get
                {
                    if (Object.Parent == null)
                        return "";
                    return Object.Parent.Name.LocalName;
                }
            }
            public string Text
            {
                get
                {
                    if (Element != null)
                        return Element.FirstNode.NodeType == XmlNodeType.Text ?
(Element.FirstNode as XText).Value : "";
                    if (Object.NodeType == XmlNodeType.Attribute)
                        return (Object as XAttribute).Value;
                   return "";
                }
            }
            public bool IsSimple
            {
                get { return !String.IsNullOrEmpty(Text); }
            }
            public OutputTypes Type
            {
                get
                {
                    int outI = 0;

                    if(int.TryParse(Text, out outI)) return OutputTypes.@int;
                    double outD;

                    if(double.TryParse(Text, out outD)) return OutputTypes.@double;
                    return OutputTypes.@string;
                }
            }
        }
```

9.4 T4 šablonas ir logikos klasė

```
<#@ template language="C#" #>
<#@ assembly name="System.Core" #>
<#@ import namespace="System.Linq" #>
<#@ import namespace="System.Text" #>
<#@ import namespace="System.Collections.Generic" #>
using System;
using System.Xml.Serialization;
using System.Collections.Generic;
namespace TestNameSpace
{
    <#=Head #>
    public partial class <#=Name #>
    {
        <#
        foreach(var attr in GetAttributes())
    {#>
        <#= attr #>
<#
    }#>
    <#    foreach(var item in GetProps())
{
 #>
        <#= item #>
<#    }#>
    }
}
partial class NodeData
    {
        private NodeInfo _info;
        private bool _isRoot;
        public NodeData(NodeInfo info, bool isRoot)
        {
            _info = info;
            _isRoot = isRoot;
        }
        public string Head
        {
            get
            {
                if (_isRoot)
                {
                    return String.Format("[XmlRoot(ElementName = \"{0}\")]",
_info.Name);
                }
                return String.Format("[XmlType(TypeName = \"{0}\")]", _info.Name);
            }
        }
        public string Name
        {
            get
            {
                return Util.UppercaseFirst(_info.Name);
            }
        }
        public IEnumerable<string> GetAttributes()
        {
            foreach (var item in _info.Attributes)
            {
                var t = item.Value;
                var result = String.Format("[XmlAttribute(AttributeName = \"{0}\")]\n",
item.Key);
                var ts = "string";
                if (t == OutputTypes.@int)
                    ts = "int";
                if (t == OutputTypes.@double)
                    ts = "double";
                result += String.Format("public {0} {1} ", ts,
Util.UppercaseFirst(item.Key)) + "{ get; set; }";
```

```
                yield return result;
            }
        }
        public IEnumerable<string> GetProps()
        {
            foreach (var item in _info.Childs)
            {
                yield return ChildToString(item.Value, item.Key);
            }
        }
        private string ChildToString(OutputTypes type, string name)
        {
            var result = String.Empty;
            result = String.Format("[XmlElement(ElementName = \"{0}\")]\n", name);
            if (type < OutputTypes.listint)
            {
                                var ts = Util.UppercaseFirst(name);
                if (type == OutputTypes.@int)
                {
                    ts = "int";
                }
                if (type == OutputTypes.@double)
                {
                    ts = "double";
                }
                if (type == OutputTypes.@string)
                {
                    ts = "string";
                }
                result += String.Format("public {0} {1} ", ts,
Util.UppercaseFirst(name)) + "{ get; set; }";
                            }
            else
            {
                //result = String.Format("[XmlArrayItem(ElementName = \"{0}\")]\n",
name);

                var ts = Util.UppercaseFirst(name);

                if (type == OutputTypes.@listint)
                {
                    ts = "int";
                }
                if (type == OutputTypes.listdouble)
                {
                    ts = "double";
                }
                if (type == OutputTypes.liststring)
                {
                    ts = "string";
                }
                result += String.Format("public List<{0}> {1}s ", ts,
Util.UppercaseFirst(name)) + "{ get; set; }";
            }
            return result;
        }
```