KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Ramutė Vyšnauskaitė

XML SCHEMŲ SUDARYMO IR NORMALIZAVIMO METODIKA

Magistro darbas

Vadovas
doc. dr. L. Nemuraitė

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas
doc. R. Butleris

XML SCHEMŲ SUDARYMO IR NORMALIZAVIMO METODIKA

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių k. katedros lekt.
dr. J. Mikelionienė

Vadovas
doc. dr. L. Nemuraitė

Recenzentas
doc. dr. V. Pilkauskas

Atliko
IFM 9/1 gr. stud.
R. Vyšnauskaitė

SUMMARY

The vast majority of systems use different data formats. This raises a variety of difficulties and problems while transferring data between those systems. At this time, XML (*eXtensible Markup Language*) is the most developed and popular data description standard. XML may be used for data exchange as well as for data storage.

As all the other text based notations, XML schema document may often be complicated and difficult to comprehend. Because of that, the designing of XML schema should always begin with the construction of a conceptual model, witch is the base for artifacts of subsequent design stages. A transition process from UML class diagram to XML schema is analyzed in this paper. The conducted research (UML CASE tool MagicDraw XML schema modeling possibility study) clearly illustrates that described extensions are accessible and supported by UML language. All major problems are explored. They are related to the fact, that UML does not include all the features required to describe a XML schema. It's useful to automate the process of XML schema transformation to UML class diagram (the programmer should not be required to supplement it).

Normalization process is one of possible ways to ensure good relational data base design – exclusion of inconsistencies in data expressions, minimized redundancy and easier way of data logic preservation. The same problems and solution methods may be applied to a XML document. The right structure of XML schema would prevent data redundancy or inconsistency, and therefore provide effective data use. XML schema normalization principles, presented in this paper, are similar to relational data base normalization principles. Similarities and distinctions of applying normalization to relational data bases and XML documents are also examined here.

TURINYS

Lentelių sąrašas

Paveikslų sąrašas

ĮVADAS

Dauguma sistemų naudoja skirtingų formatų duomenis, todėl atsiranda įvairių problemų ir sunkumų perduodant duomenis iš vienos sistemos į kitą. Šiai problemai spręsti reikia naudoti tokį duomenų struktūros aprašą, kurį suprastų visos sistemos – toks aprašas turėtų būti universalus. Šiuo metu labiausiai išvystytas ir plačiai naudojamas duomenų aprašymo standartas yra XML (*eXtensible Markup Language*). Dabartinė (antroji) XML versija 1.0 yra W3C (*World Wide Web Consortium*) pasiūlyta rekomendacija [22]. XML gali būti panaudota tiek keistis duomenimis, tiek duomenims saugoti. XML dokumentai gali būti naudojami pačiose įvairiausiose srityse: elektroninėje komercijoje, bendraujant su verslo partneriais ar organizacijos viduje integruojant programinę įrangą bei duomenų bazes.

Kaip ir visos tekstinio pobūdžio notacijos, XML schemos dokumentas dažnai būna gana painus ir sunkiai suvokiamas. Dėl šios priežasties XML schemų projektavimas turėtų prasidėti nuo konceptualiojo modelio, kuris yra tolesnių projektavimo fazių artefaktų pagrindas. Pagrindinis konceptualiojo modeliavimo tikslas yra atskirti projektavimo ir įgyvendinimo etapus. UML panaudojimas suteikia objektinių programų sistemų ir XML dokumento struktūrų sujungimo galimybes, padeda išlaikyti schemų suderinamumą su kitais projekto elementais. Modeliui būdingas vizualinis vaizdas padidina projekto suprantamumą ir aiškumą – daro aiškesnę schemų semantiką. Be to, UML konceptualusis modeliavimas padeda patobulinti pakartotinį projektavimą, atskleisti dokumento struktūrinius trūkumus ir pagerina XML schemų projektavimo procesą.

Kadangi UML yra patikimas ir išbandytas objektinių sistemų modeliavimo įrankis ir lengviausias verslo procesų modeliavimo būdas, todėl naudinga automatizuoti XML schemų transformacijos iš UML klasių diagramos procesą taip, kad transformacija būtų visiškai baigta, t. y., programuotojui nereikėtų jos papildyti.

Efektyvus XML dokumento panaudojimas priklauso nuo šio dokumento metaduomenų kokybės – DTD ar XML schemos. Gera XML schemos struktūra neleistų susidaryti duomenų pertekliui ir nelogiškumui, kurie gali būti neefektyvaus duomenų panaudojimo priežastimi [15]. Programos gali talpinti didelius duomenų kiekius į XML dokumentą ir dažnai vykdyti XML dokumento duomenų atnaujinimo operacijas. Esant netinkamai XML schemos struktūrai, XML dokumente galimi tokie trūkumai kaip duomenų perteklius ir dubliavimasis. Kaip ir sąryšinės duomenų bazėse, duomenų dubliavimas gali sąlygoti didelę saugomų duomenų apimtį ir netikslumus vykdant operacijas su duomenimis.

Duomenų bazių projektavimas ir norminės formos – tai esminiai sąryšinių duomenų bazių technologijos konceptai. Normalizavimas yra vienas iš būdų, kuris leidžia užtikrinti gerą sąryšinių duomenų bazių projektavimą – pašalinti nevienareikšmiškumus duomenų išraiškose, minimizuoti duomenų perteklių ir palengvinti duomenų logiškumo išsaugojimą [7]. Tas pačias problemas ir jų

pašalinimo metodus galima pritaikyti ir XML dokumentui. Normalizuotos XML schemos struktūra garantuoja, kad XML dokumentas atitinka pageidaujamas savybes.

Gero projektavimo kriterijai, kurie yra pakankamai intuityvūs ir lengvai pritaikomi sąryšinėse duomenų bazėse, tampa sunkiai suprantami, kai bandoma juos pritaikyti XML dokumentui.

Sąryšinių duomenų bazių projektavimas yra plačiai išnagrinėtas remiantis funkcinėmis priklausomybėmis. Norminės formos taip pat analizuojamos remiantis funkcinių priklausomybių, kurios įtakoja duomenų perteklių, eliminavimu. Kaip ir sąryšinių duomenų bazių atveju, XML dokumentui norminės formos apibrėžiamos panaudojant funkcines priklausomybes, kurios sudaro normalizavimo algoritmo pagrindą.

Šio darbo analizės dalyje pateikta XML schemų modeliavimo UML metodologijos analizė. Aprašyti pagrindiniai XML schemos ir UML klasių diagramos konceptai ir elementai. Susipažinta su XML schema ir UML klasių diagrama, jų panaudojimo galimybėmis ir sintakse. Kadangi XML dokumento struktūrai apibrėžti gali būti naudojama XML schema arba DTD, atlikta šių dviejų formatų lyginamoji analizė. Taip pat pateikta programinės įrangos paketų, kurie leidžia XML schemas (ir DTD) modeliuoti panaudojant grafinį ir medžio struktūra paremtą vaizdavimą, analizė. Nurodyti du galimi grafinio XML schemų kūrimo variantai – XML specifinės savitos kalbos ir bendresnės modeliavimo kalbos (UML) panaudojimas. Išnagrinėti šių XML schemos modeliavimo būdų privalumai ir trūkumai.

Pagrindinėje dalyje aprašyta XML schemų kūrimo metodika panaudojant UML klasių modelį. Parodyta galimybė grafiškai aprašyti XML schemos struktūrą UML klasių diagrama. Pradedant nuo abstraktaus modeliavimo išanalizuotas perėjimas nuo UML klasių diagramos prie XML schemos. Aptarti galimi variantai transformuojant UML modelį į XML schemą. Kai kurių XML schemos savybių nėra UML diagramoje, todėl dažniausiai projektuotojas turi papildyti XML schemą reikiamomis savybėmis [14]. Detalaus projektavimo ir automatinio pilnų schemų generavimo galimybių užtikrinimui galimas UML profilio XML schemai panaudojimas. UML profilis XML schemai suteikia galimybes sukurti teisingą XML schemą iš bet kokio UML klasių modelio, išplėsti konceptualųjį modelį iki pritaikyto XML schemoms ir palaikyti dvipusį sujungimą tarp UML ir XML schemų, įtraukiant egzistuojančių XML schemų atvirkštinę inžineriją į UML objektinius modelius.

Kelių autorių (Elmasri R., Wu Y.Ch., Hojabri B. ir Li Ch., Fu J.) pateiktame straipsnyje „Conceptual Modeling for Customized XML Schemas“ aprašyta XML schemos projektavimo metodologija, kuri yra paremta gerai žinomu konceptualiuoju projektavimu [11]. Metodologija aprašyta ir pritaikyta vienam iš gerai žinomų konceptualiųjų duomenų modelių – EER (*Extended Entity Relationship*) schemai. Straipsnyje pateiktas algoritmas, kuris apima hierarchinio vaizdo generavimą iš EER modelio, XML schemos ir XML dokumento sukūrimą. Pasiūlytas metodas nėra pritaikytas konkrečiai sistemai, nes jis yra paremtas konceptualiojo modeliavimo technika, kuri nėra

priklausoma nuo specifinės komercinės sistemos. Kadangi EER ir UML klasių diagramos turi daug bendro, tai atsiranda galimybė generuoti XML schemas iš abiejų modelių.

Šiame darbe minėtame straipsnyje pateiktas algoritmas, kuris pritaikytas iš UML klasių diagramos XML schemų automatinio generavimo procesui. Šis algoritmas apima ciklų eliminavimą klasių diagramoje, hierarchinio vaizdo formavimą ir automatinį XML schemos generavimą klasių diagramos darinius transformuojant į atitinkamus schemos elementus. Pagrindinis algoritmo panaudojimo privalumas yra tas, kad transformacija būtų visiškai baigta – programuotojui nereikėtų jos papildyti.

Taip pat šioje darbo dalyje aprašytos ir XML schemai pritaikytos norminės formos, kurios naudojamos sąryšinėse duomenų bazėse duomenų atnaujinimo anomalijoms išvengti. XML norminių formų nustatymas remiasi XML dokumentų funkcinių priklausomybių, kurios iššaukia duomenų perteklių, eliminavimu. Išanalizuotos ir XML dokumentui pritaikytos nuo pirmosios iki ketvirtosios norminės formos. Parodyti skirtumai ir panašumai pritaikant normines formas sąryšinėms duomenų bazėms ir XML dokumentams.

Arenas M. ir Libkin L. straipsnyje „A Normal Form for XML Documents“ pateiktas XML norminės formos XNF aprašas, kuris remiasi funkcinėmis priklausomybėmis [2]. Šiame straipsnyje pateiktas algoritmas, kuris transformuoja XML dokumento DTD į XML norminę formą (XNF). Straipsnyje taip pat parodyta, kad XNF yra ekvivalenti sąryšinių duomenų bazių BCNF (*Boyce–Codd Normal Form*), o tai įrodo, kad XNF yra griežta norminė forma. Remiantis straipsniu, šiame darbe aprašytas ir pateiktas normalizavimo algoritmas, pritaikant jį XML schemai. Šio normalizavimo algoritmo pagrindą sudaro funkcinės priklausomybės, kurias panaudojant apibrėžiamos XML dokumento norminės formos. Nurodyti ir aprašyti esminiai algoritmo žingsniai – atributų perkėlimas ir naujų elemento tipų sukūrimas.

Eksperimentinėje dalyje atliktas UML CASE įrankio MagicDraw XML schemų modeliavimo galimybių tyrimas. MagicDraw paketas turi galimybę grafiškai pavaizduoti XML schemos struktūrą UML klasių diagrama, įvedus reikiamus stereotipus, ir palaiko tiesioginę ir atvirkštinę XML kodo inžineriją. MagicDraw paketo palaikoma kodo inžinerija leidžia egzistuojančias UML klasių diagramas transformuoti į XML schemos kodą ir atvirkščiai. Tam, kad būtų įmanoma kodo inžinerija, klasių diagrama turi būti pertvarkyta į XML schemos diagramą. Praktiškai pavaizduotas XML schemos sudarymas iš jau egzistuojančios klasių diagramos ir XML schemos modeliavimas nuo nulio, naudojant stereotipais pažymėtus XML schemos diagramos elementus. Naudojantis MagicDraw paketu, parodytas aprašyto XML schemų generavimo iš UML klasių diagramos algoritmo praktinis pritaikymas.

Darbe nagrinėjama tema buvo pristatyta 10-joje tarpuniversitetinėje magistrantų ir doktorantų konferencijoje „Informacinės technologijos‘05“, vykusioje KTU š. m. balandžio 29 d. Kaune [21].

1. XML SCHEMŲ PROJEKTAVIMO METODŲ ANALIZĖ

1.1. XML schemų tipai ir pagrindiniai konceptai

Naudojant XML galima sukurti duomenų struktūras, kurios aprašo jose esančių duomenų turinį, neatsižvelgiant į tai, kaip turinys bus pavaizduotas. Nors XML griežtai specifikuoja sintaksę, tačiau leidžia laisvai apibrėžti XML dokumentų prasmę. XML leidžia susikurti savitą gramatiką arba naudotis jau sukurta gramatika, pavyzdžiui, tokią, kuri naudojama konkrečioje probleminėje srityje (pvz., prekyboje, matematikoje, chemijos pramonėje ir kt.). Gramatikai apibrėžti naudojami du formatai: XML schema arba dokumento tipo apibrėžtis – DTD [23] [12]. Dabartinis XML schemos standartas yra W3C pasiūlyta rekomendacija. W3C kuria standartus ir technologijas, kurios atitinka informacijos pateikimo internete keliamus reikalavimus.

Nors XML schemos ir DTD paskirtis ta pati, XML schema labiau pritaikyta programų, kurios yra orientuotos į duomenis ir naudoja XML, reikalavimams palaikyti. XML schemos formatas yra pranašesnis ir patogesnis nei DTD. Galima paminėti tokius pagrindinius XML schemos privalumus:

- § XML schemoje naudojama XML sintaksė. Tai reiškia, kad XML schemoms apdoroti gali būti naudojamos įprastos XML programos. DTD turi specifinę sintaksę, todėl XML redagavimo programinės įrangos kūrėjai turi papildomai rūpintis kitokios nei XML sintaksės palaikymu.
- § XML schema palaiko platesnę duomenų tipų aibę nei DTD. DTD leidžia naudotis tik pirminių duomenų tipų aibe, kuri pagrįsta tekstinio tipo elementais. XML schema palaiko duomenų tipus (*numeric*, *date/time*, *binary*, *boolean*, *URIs* ir kt.), kurie naudojami ir kitose kalbose, pavyzdžiui, SQL ar JAVA. Panaudojant kitų duomenų tipų kompozicijas, yra galimybė sudaryti sudėtinį tipą. Be to, XML schema naudoja paveldėjimo mechanizmą, kuris leidžia apriboti ar išplėsti duomenų tipo apibrėžimą.
- § Kita svarbi XML savybė – terminų mechanizmas (*namespace*). Turintys tą patį pavadinimą XML schemos elementai gali būti panaudoti skirtingais kontekstais. Be to, XML elementai ir elementų tipai gali būti įtraukti (ar importuoti) iš kitos XML schemos naudojant tą patį (ar kitą) terminų mechanizmą.
- § XML schema leidžia naudoti apribojimus, kurių nepalaiko DTD. Į šiuos apribojimus įeina formatais pagrįstų šablonų apribojimai, raktai, unikalumo apribojimai, sudėtiniai raktai ir kt.

Taigi XML schema aprašo XML dokumentų struktūrą. Ji nusako elementus, kurie gali būti naudojami XML dokumentuose. Jei XML dokumentas naudoja schemą, tai neaprašytų elementų negalima vartoti. Kitaip tariant, XML schema aprašo gramatiką, kurios laikydamiesi XML dokumentai yra teisingi. XML dokumentų tikrinimas reiškia, kad išoriniai duomenys turi laikytis taisyklių, kurios

yra aprašytos XML schemoje. Schema užtikrina, kad XML dokumentas yra tokio formato, kokio tikimasi. XML schema yra galinga ir sudėtinga XML dokumentų struktūros kūrimo ir tos struktūros gramatikos patvirtinimo priemonė.

XML schemos dokumentai aprašo konkretų XML dokumento tipą, nurodo apribojimus šio tipo dokumento duomenims, pateikia žymes ir atributus, naudojamus to tipo dokumentuose, sąryšius tarp šių XML dokumento elementų. XML schema, kaip ir XML dokumentas, yra hierarchinės medžio tipo struktūros. Šakninis kiekvienos XML schemos elementas yra <schema>. Jis gali turėti keletą atributų. Paprastai schemos paskelbimas atrodo taip:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
...
...
</xsd:schema>
```

Šakniniame schemos elemente saugomi visi schemos žemesnio lygio elementai. Kiekvienas schemos elementas turi prefiksą *xsd*: (gali būti naudojamas ir kitoks prefiksas), kuris susietas su XML schemos vardų erdve, kurį aprašoma tokia eilute:

xmlns: xsd=http://www.w3.org/2001/XMLSchema.

Vardų erdvė yra naudojama apriboti ir įvardinti XML schemos elementų grupę. Naudojant vardų erdves tampa įmanoma į XML schemos aprašą įjungti kitus XML schemos aprašus, kai juose naudojamų tipų ir elementų pavadinimai yra jau vartojami kituose.

XML schemos pagrindinės konstrukcijos gali būti išreikštos šešiomis skirtingomis formomis:

- Elementas (*xsd:element*);
- Atributas (*xsd:attribute*);
- Paprastas (tekstinis) elemento tipas (*xsd:simpleType*);
- Sudėtinis elemento tipas (*xsd:complexType*);
- Elementų grupė (*xsd:group*);
- Atributų grupė (*xsd:attributeGroup*).

Elementai yra naudojami apibrėžti duomenų esybes XML dokumente. Jie tarnauja kaip XML dokumento duomenų metainformacija. Elementai dokumente išdėstomi hierarchiškai, iš vieno šakninio elemento. Elementai XML schemoje apibrėžiami naudojant <element> žymę. Elementui būtina nurodyti jo vardą ir duomenų tipą:

<xsd:element name=“elementoVardas“ type=“elementoTipas“/>.

Duomenų tipai nurodo vidinę elementinę struktūrą. Galimi trys elemento duomenų tipai: pirminis (pvz., *integer*), paprasto ir sudėtinio turinio.

Paprasto tipo elementas schemoje apibrėžiamas panaudojant <simpleType> žymę. Galimi keli paprasto tipo elemento sudarymo būdai:

- Paprasto tipo elementas gaunamas panaudojant jau egzistuojančių paprasto tipų elementų arba pirminių tipų apribojimą. Tam, kad nurodyti jau egzistuojantį tipą, kuris bus apribotas, naudojamas „*restriction*“ atributas. Be to, ši konstrukcija identifikuoja žymes, kurios apriboja formuojamos naujos reikšmės sritį.
- Paprastas tipas sudaromas iš pirminių (atominių) duomenų tipų reikšmių sąrašo. Šiuo atveju naudojama <list> žymė ir *length, minLength, maxLength, enumeration, pattern* atributai, kurie apriboja <list> konstrukcijos reikšmę.
- Paprastas tipas apibrėžiamas kaip kitų paprasto tipo elementų sąjunga panaudojant <union> žymę.

Sudėtinio tipo elementas schemoje apibrėžiamas naudojant <complexType> žymę, kuri paprastai apimą aibę elementų ir atributų. Sudėtinis tipas naudoja <sequence>, <choice> ar <all> žymes, kurios apsprendžia subelementų išrikiavimą:

- <sequence> – subelementai būtinai turi būti išdėstyti eilės tvarka;
- <choice> – iš sąrašo gali būti pasirinktas tik vienas subelementas;
- <all> – subelementų eilės tvarka nesvarbi.

Be elementų, XML schema dar aprašo atributus, kurie gali būti naudojami dokumente. Atributai XML schemoje apibrėžiami naudojant <attribute> žymę. Atributai negali turėti kitų elementų ar atributų savo sudėtyje. Galimi du atributo duomenų tipai: pirminis ir paprasto turinio. Atributui, kaip ir elementui, būtina nurodyti jo vardą ir duomenų tipą:

<xsd:attribute name=”vardas” type=”tipas” use=”kaip-naudojamas” default/fixed=”reikšmė”/>.

Be to, kas jau buvo paminėta, XML schema apibrėžia:

- galima subelementų skaičių (*minOccure, maxOccure*);
- ar elementas tuščias, ar į jį gali būti įrauktas tekstas;
- pradines ir fiksuotas elementų ir atributų reikšmes (*default, fixed*);
- elementų ir atributų raktines reikšmes, leidžia sudaryto sudėtinį raktą (sujungia kelias raktines reikšmes į vieną raktą) (<key>, <keyref>);
- sudėtinio ir paprasto elementų tipų išplėtimą ir apribojimą (<extension>, <restriction>);
- schemų įtraukimą į kitas XML schemas (<import>, <include>);
- kitas priemones, leidžiančias apriboti tipų panaudojimą, deklaruoti schemos anotacijas, išorinių schemų panaudojimą kitos schemos sudaryme ir kt.

1.2. UML klasių diagrama

Šiandieninis informacinių sistemų (IS) projektavimas neįsivaizduojamas be automatizuoto projektavimo (CASE) įrankiu, kurių daugelis naudoja unifikuotą modeliavimo kalbą UML, skirtą vizualizuoti, specifikuoti, konstruoti ir dokumentuoti objektines sistemas. UML turi devynių tipų – klasių, objektų, panaudojimo atvejų, sekos, bendradarbiavimo, būsenų, veiklos, komponentų ir įdiegimo – diagramas [19].

UML klasių diagrama yra statinio struktūrinio modelio grafinis atvaizdavimas. Klasių diagrama neparodo laikinos informacijos, ji aprašo tik klasifikaciją. UML statinės struktūros klasių diagramą sudaro objektų tipai (klasės) ir jų tarpusavio ryšiai (asociacijos). Klasės tarpusavyje gali būti susietos keliais skirtingais būdais: klasės susiejamos tarpusavyje asociacijos ryšiais, viena klasė priklauso nuo kitos klasės, klasė gali būti kitos klasės potipis arba klasės gali būti sugrupuotos į vieną vienetą – paketą. Klasių diagrama neišreiškia konkrečių duoto objekto ryšių – ji tik abstrakčiai aprašo objekto galimus ryšius su kitais objektais. Klasių diagrama yra tarsi loginis egzistuojančio ar būsimo programos kodo žemėlapis.

Pagrindinis UML klasių diagramos elementas yra klasė. Objektų klasė aprašo grupę objektų, kurie turi panašias savybes – atributus, ryšius su kitais objektais, panašią elgseną – operacijas, ir bendrą prasmę. Klasė yra abstrakti sąvoka, kuri turi vardą, apibrėžimą ir egzempliorių (objektų) aibę. Klasės aprašymas susideda iš trijų dalių: klasės vardo (papildomai dar gali būti naudojami stereotipai ir jų savybės), atributų ir klasės operacijų. Kiekviena klasė gali turėti atributą – identifikatorių, kuris diagramose nerodomas.

Atributas yra įvardinta klasės savybė ir apibrėžia reikšmes, kurias gali įgyti klasės egzempliorius. Atributo aprašymas susideda iš: matomumo (*public, protected, private*), atributo pavadinimo, kuris turi būti unikalus klasės ribose, atributo kardinalumo (*multiplicity*), jo duomenų tipo (pirminis arba kita klasė), iš anksto nustatytos (*default*) reikšmės ir kitų galimų savybių. Išvestiniai atributai gali būti gauti iš kitų atributų, t. y., jų reikšmė gali būti apskaičiuota pagal kitų atributų reikšmes. Jie pažymini „/“ ženklu priešais pavadinimą.

UML atributų įgyjami duomenų tipai gali būti pirminiai (pvz., *integer, string*), išvardijami iš anksto (*enumeration*) ir programavimo kalbų duomenų tipai, kurie atitinka konkrečios programavimo kalbos semantiką.

Paketas naudojamas klasėms grupuoti. Kiekviena klasių diagrama turi turėti savo paketą, kuris gali būti įtrauktas į kitus paketus. Paketai taip pat gali turėti tarpusavio ryšius, kurie paprastai išvedami iš į paketus įtrauktų klasių ryšių.

Klasių tarpusavio ryšių tipai:

- Asociacija
- Klasifikacija

- Apibendrinimas
- Agregavimas
- Kompozicija
- Rekursyvus ryšys

Asociacijos aprašo UML klasių semantinius tarpusavio ryšius, kurie vaizduojami linijomis tarp klasių. Pavyzdžiui, asociacija tarp klasės A ir B. Ryšio kardinalumas yra ryšio egzempliorių skaičius, kurį vienos klasės objektas gali turėti su kitos klasės objektais. Pavyzdžiui, ryšio kardinalumas r..s klasės B pusėje nusako, kad A klasės objektas gali turėti ne mažiau kaip r ir ne daugiau kaip s ryšio egzempliorių su B klasės objektu.

Asociacijos gali apimti daugiau nei dvi klases. Šie daugiamačiai ryšiai klasių diagramoje vaizduojami rombais. Asociacijos gali būti pažymėtos kaip kryptinės, o tai reiškia, kad ryšys skaitomas tik viena kryptimi. Dažniausiai naudojamos dvikryptės asociacijos. Asociacijos atributų apibrėžimui papildomai gali būti panaudojama ryšio klasė.

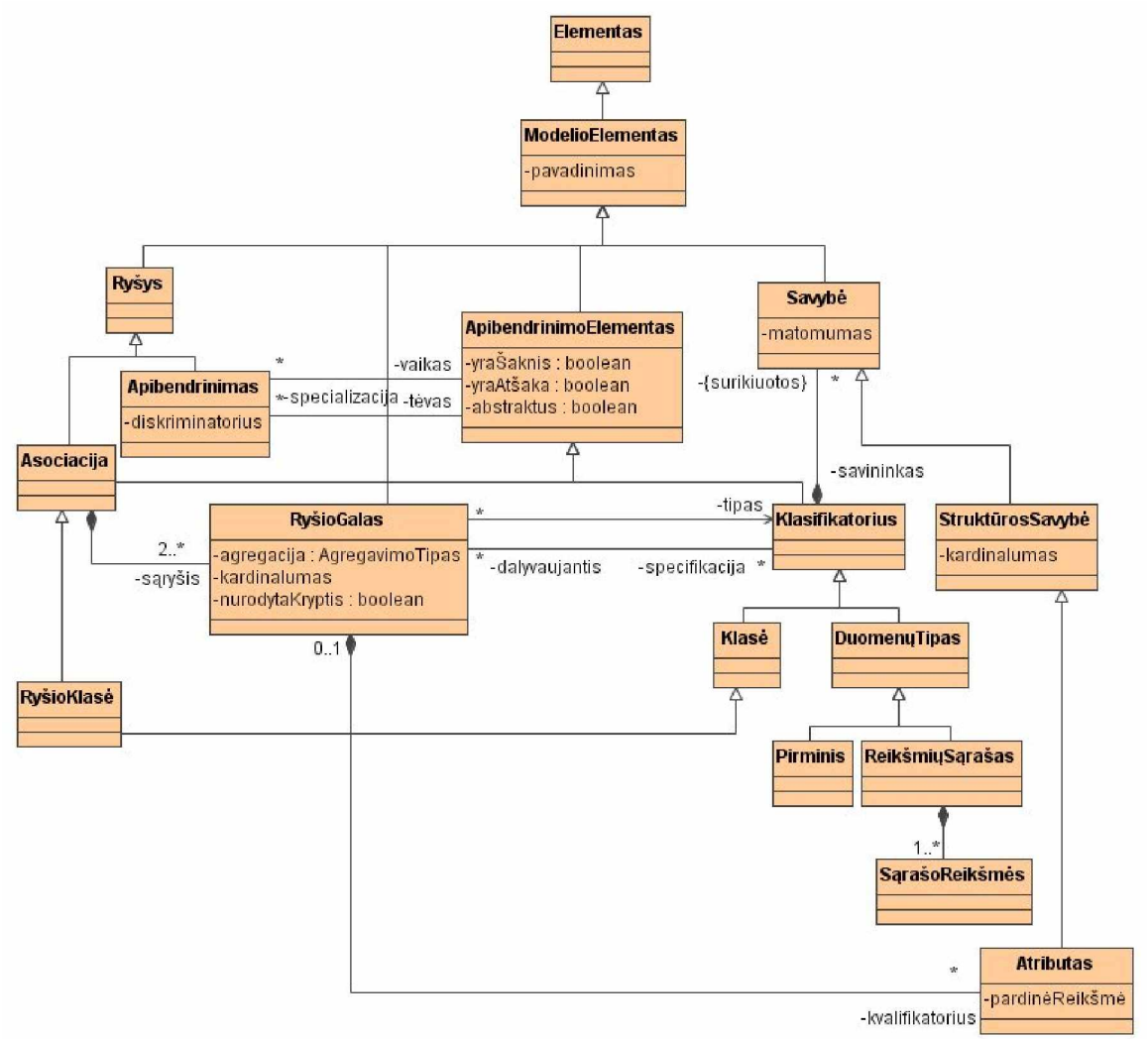
Apibendrinimas sukuria ryšius tarp žemesnio lygio esybių egzempliorių ir aukštesnio lygio esybių. Apibendrinimo hierarchija užtikrina paveldėjimo savybę (pvz., buhalteris paveldi darbuotojo savybes).

Panašių objektų grupavimas į klases ar aibes vadinamas klasifikavimu.

Be pagrindinių asociacijų UML klasių diagramoje galima naudoti specialaus tipo asociacijas. Viena iš tokių asociacijų – agregavimo ryšys. Agregavimo mechanizmas tam tikrų objektų pagrindu suformuoja aukštesnio lygio objektus (agregatus). Tarp jų egzistuoja *part_of* ryšys. Griežtas agregavimas – kompozicija. Klasė gali turėti daugiausiai vieną kompozicijos ryšį su superklase, ir jos gyvavimo trukmė priklauso nuo superklasės gyvavimo trukmės. Kompozicijos ryšys vaizduojamas užpildytu rombu ryšio gale.

Rekursyvus ryšys – ryšys tarp tos pačios klasės egzempliorių.

1.1 paveiksle pateikta UML metamodelio dalis, kuri susijusi su klasių diagrama:



1.1 pav. UML metamodelio dalis, apimanti klases ir jų savybes

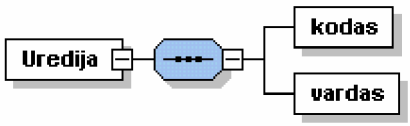
Elementas yra atominė sudėtinė modelio dalis. Elementas yra šakninė metaklasė šioje hierarchijoje. Modelio elementas – įvardinta modelio esybė. Tai visų UML metaklasių pagrindas. Kitos metaklasės yra modelio elemento tiesioginės ar netiesioginės subklasės. Savybė yra abstrakti klasė. Struktūrinės savybės nurodo statines modelio elemento savybes – atributus. Klasifikatorius yra elementas, kuris aprašo elgsenos ir struktūros savybes – klases, duomenų tipus, komponentus. Ryšys yra sąjunga tarp modelio elementų. UML apibrėžia keletą ryšio (asociacija, apibendrinimas) ir duomenų (pirminis, reikšmių sąrašas) tipų.

1.3. XML schemų kūrimo įrankių analizė

Šiuo metu egzistuoja programinės įrangos paketai, kurie leidžia XML schemas (ir DTD) modeliuoti panaudojant grafinį ir medžio struktūra paremtą vaizdavimą. Grafinį XML schemų kūrimą galima realizuoti dviem būdais. Vienu atveju XML schemoms modeliuoti galima naudoti XML specifinę savitą kalbą, kitu – bendresnę modeliavimo kalbą.

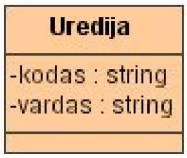
XML schemos modeliavimas paketais, kurie naudoja savitą kalbą ir grafinį vaizdavimą, yra paprastesnis ir padidina modelio suprantamumą ir aiškumą lyginant su tekstinio tipo redagavimo

programomis. Vienas iš geriausiai žinomų XML schemų modeliavimo paketų yra XMLSpy [1]. Paketai kaip XMLSpy XML schemoms modeliuoti naudoja specifinį grafinio medžio vaizdavimą:



1.2 pav. XML schemos modeliavimo pavyzdys XMLSpy pakete

Kita alternatyva – standartinės modeliavimo kalbos panaudojimas, pavyzdžiui, UML:



1.3 pav. XML schemos modeliavimo pavyzdys panaudojant UML

Abu pasiūlyti būdai turi savitų privalumų ir trūkumų. XML specifiniai simboliai puikiai tinka XML schemos darinių atvaizdavimui. Lengva identifikuoti tokias XML žymes kaip <sequence>, <choice>, XML elementus, atributus ir kt. Tačiau modeliavimas su tokias įrankiais dažnai nesuderinamas su likusiais projektuotojų poreikiais. Modeliavimas šiais įrankiais yra tinkamas mažiems projektams. Tačiau dirbant su didesniais sudėtiniais projektas, pvz., kurie apima XML, JAVA ar SQL, atsiranda sunkumų. Be to, projektuotojai yra priverstas per anksti priimti sprendimus, susijusius su XML dokumento struktūra. Tinkamesnis ir patogesnis būdas – generuoti fizinę duomenų struktūrą, prieš tai suprojektavus atitinkamą konceptualųjį modelį.

UML klasių diagramos kaip pagrindo duomenų struktūrai apibrėžti panaudojimas taip pat turi savų privalumų. Pirmiausia, lyginant su kitomis notacijomis ar programavimo kalbomis, UML grafinis atvaizdavimas yra daug lengviau skaitomas ir suprantamas. Grafinė notacija yra neutrali realizacijos atžvilgiu – apibrėžtos struktūros gali būti koduojamos į bet kokią konkrečią realizacijos kalbą – ne tik į XML. UML modelius galima tiesiogiai pavaizduoti įvairiose programavimo kalbose (JAVA, C++, Visual Basic ir kt.), taip palengvinant projektuotojų ir programuotojų darbą. Be to, UML yra standartas, kuris aprašytas daugelyje knygų ir palaikomas daugelio programinės įrangos įrankių. UML diagramos gali parodyti tiek informacijos, kiek reikia – galima pasiruošti keletą reikiamų modelių naudojantis vienu įrankiu.

MagicDraw – UML modeliavimo ir CASE įrankis [20]. Sukurtas verslo ir programinės įrangos analitikams, programuotojams ir dokumentacijai, šis dinamiškas ir įvairiapusis įrankis palengvina objektinių (OO) sistemų ir duomenų bazių analizę ir projektavimą. Be to, jis palaiko kodo inžinerijos mechanizmą (Java, C#, C++, WSDL, XML Schema ir CORBA IDL) ir atvirkštinės inžinerijos galimybes. O tai labai naudinga XML schemų modeliavimui UML.

MagicDraw paketo palaikoma kodo inžinerija leidžia egzistuojančias UML klasių diagramas transformuoti į XML schemos kodą ir atvirkščiai:

§ Tiesioginė kodo inžinerija: XML schemos diagrama → XML schemos kodas.

§ Atvirkštinė kodo inžinerija: XML schemos kodas → XML schemos diagrama.

Tam, kad būtų įmanoma kodo inžinerija, klasių diagrama turi būti pertvarkytą į XML schemos diagramą. Šios diagramos paskirtis – sudaryti XML schemos bylos struktūrą. Diagramos privalumas – greitas ir gana paprastas XML schemos elementų atvaizdavimas. XML schemos elementai – tai stereotipais praplėsti UML klasių ir realizacijos diagramų elementai.

1.4. XML schemų kūrimo įjungimas į informacinių sistemų projektavimo procesą

Šiandieninis informaciniu sistemų (IS) projektavimas neįsivaizduojamas be automatizuoto projektavimo (CASE) įrankių, kurių daugelis naudoja unifikuotą modeliavimo kalbą UML, skirtą vizualizuoti, specifikuoti, konstruoti ir dokumentuoti objektines sistemas. UML turi devynių tipų – klasių, objektų, panaudojimo atvejų, sekos, bendradarbiavimo, būsenų, veiklos, komponentų ir įdiegimo – diagramas.

Projektuotojams būtų naudinga turėti būdą, kuris XML schemų kūrimą įjungtų į informacinės sistemos projektavimo procesą. XML dokumentams projektuoti naudojant UML, XML dokumentų struktūrų projektavimą galima integruoti į bendrą informacinių sistemų projektavimo procesą [10]. UML modelius galima tiesiogiai atvaizduoti įvairiose programavimo kalbose (JAVA, C++, Visual Basic ir kt.), taip palengvinant projektuotojų ir programuotojų darbą. Be to, UML yra standartas, kuris aprašytas daugelyje knygų ir palaikomas daugelio programinės įrangos įrankių. UML diagramos gali parodyti tiek informacijos, kiek reikia – galima pasiruošti keletą reikiamų modelių naudojantis vienu įrankiu.

Loginė XML dokumento struktūra sudaryta iš hierarchiškai susietų elementų. XML elementas turi pavadinimą, gali turėti atributų ir tam tikrą turinio modelį. Tokia XML elemento struktūra atitinka UML klasės pavadinimą, atributus ir klasių tarpusavio ryšius [18].

1.5. Analizės išvados

Šio darbo analizės dalyje pateikti XML schemos ir UML klasių diagramos konceptai ir elementai. Lyginant XML schemos ir UML klasių diagramos pagrindinius struktūrinius komponentus – elementą ir klasę, galima pastebėti daug panašumų. XML elementas turi pavadinimą, gali turėti atributų ir tam tikrą turinio modelį. Tokia XML elemento struktūra atitinka UML klasės pavadinimą, atributus ir klasių tarpusavio ryšius. Tačiau XML schemų kūrimas panaudojant UML klasių diagramas nėra paprastas, nes UML neapima visų savybių, kurių reikia modeliuojant XML schemas.

Kadangi XML dokumento struktūrai apibrėžti gali būti naudojama XML schema arba DTD, atlikta šių dviejų formatų lyginamoji analizė. Pagal pateiktus XML schemos privalumus, tokius kaip naudojama XML sintaksė, platesnė duomenų tipų aibė, terminų mechanizmas ir DTD nepalaikomi apribojimai, galima spręsti, kad XML schemos formatas yra pranašesnis ir patogesnis nei DTD. Nors XML schemos ir DTD paskirtis ta pati, XML schema labiau pritaikyta programoms, kurios yra orientuotos į duomenis ir XML naudojimą.

Taip pat darbe pateikta programinės įrangos paketų, kurie leidžia XML schemas (ir DTD) modeliuoti panaudojant grafinį ir medžio struktūra paremtą vaizdavimą. Nurodyti du galimi grafinio XML schemų kūrimo variantai – XML specifinės savitos kalbos ir bendresnės modeliavimo kalbos (UML) panaudojimas. Abu XML schemos modeliavimo būdai turi savitų privalumų ir trūkumų. Atsižvelgiant į tai, kad projektuotojams būtų patogu turėti būdą, kuris XML schemų kūrimą įjungtų į informacinės sistemos projektavimo procesą, naudingiau XML dokumentus projektuoti panaudojant UML klasių diagramas. Tokiu būdu galima susieti sistemų projektavimą su XML schemos kūrimu.

2. XML SCHEMŲ KŪRIMO METODIKA

2.1. XML schemų vaizdavimas UML

Vaizduojant XML schemas UML klasių diagrama pagrindinė problema yra ta, kad UML neapima visų savybių, kurių reikia aprašant XML schemas. Šiame poskyryje išanalizuotas kiekvienas UML klasių diagramos komponentas ir pavaizduotas jo transformavimas į XML schemą. Tokiu būdu aprašytas beveik pilnas perėjimas nuo UML klasių diagramos prie XML schemos. Aptarti galimi variantai transformuojant UML modelį į XML schemą. UML klasių diagrama nepalaiko kai kurių XML schemos savybių, todėl projektuotojas turi papildyti XML schemą reikiamomis savybėmis. Ši problema išsprendžiama panaudojant UML išplėtimo profilius. UML profilį sudaro trys esminės esybės: stereotipai, pažymėtos reikšmės (*tagged values*) ir apribojimai. Kiekvienas stereotipas susietas su viena ar keliomis UML konstrukcijomis, kurios yra modifikuojamos. Stereotipas tiksliai nurodo, su kokia XML schemos struktūra susiejamas UML modelio elementas. Kiekvienas stereotipas gali būti tikslinamas pridedant vieną ar kelias savybes, kurios sustiprina jų reikšmę ar įtaką modelyje. Pavyzdžiui, stereotipas, kuris susietas su UML klase, išplečia „klasės“ prasmę, o stereotipo savybės – detalizuoja klasę modelyje. Apribojimai apibrėžia sąlygas, kurios turi būti patenkintos norint priskirti stereotipą UML elementui [5] [4].

2.1.1. UML klasės

UML klasė yra struktūrinių ir elgsenos savybių saugykla. Bet tik struktūrinės savybės svarbios XML schemos sudarymui. Struktūrinės savybės susideda iš atributų, ryšių ir jų kardinalumų [6].

Klasės susiejimas su XML yra gana paprastas, nes kiekvienas UML klasės egzempliorius atitinka XML elementą. XML dokumente elementas laikomas kaip atributų ir subelementų saugykla. Kaip ir UML klasė, XML elementas yra atributų ir ryšių saugykla. UML klasės ir XML elementai turi daug bendro: abu turi vardą ir tam tikrą skaičių atributų. Vadinasi, klases patogiausia vaizduoti XML elementais. 2.1 paveiksle pavaizduota UML klasių diagramos klasė „Uredija“:

2.1 pav. UML diagramos klasė

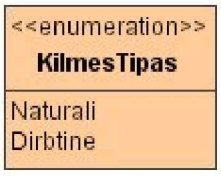
XML dokumente ši klasė atitinką XML elementą, kuris turi tą patį vardą:

<Uredija> ... </Uredija>

XML schemoje UML klasė apibrėžiama kaip sudėtinė duomenų struktūra ir susiejama su XML schemos *complexType* tipu:

<xsd:element name="Uredija" type="UredijaType"/>

Klasės su <<enumeration>> stereotipais XML schemoje vaizduojamos kitaip. Visoms <<enumeration>> klasėms sukuriamas *simpleType* tipo elementas su sąrašu reikšmių, kurios atitinka <<enumeration>> klasės atributus:




```
<xsd:simpleType name="KilmesTipas">
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="Naturali"/>
   <xsd:enumeration value="Dirbtine"/>
  </xsd:restriction>
</xsd:simpleType>
```

2.2 pav. UML <<enumeration>> klasės transformavimas į XML schemą

2.1.2. UML atributai

Klasių atributai neatvaizduojami į XML schemą taip paprastai kaip klasės. UML klasės atributas gali būti transformuojamas į XML dokumento atributą arba elementą. UML atributų automatinis transformavimas į XML elementus ar atributus priklauso nuo XML schemos pobūdžio.

Paprasčiausias būdas yra susieti kiekvieną UML klasės atributus su tą klasę atitinkančio XML elemento subelementais. Tokiu atveju klasė „Uredija“ ir jos atributai atitiktų tokį XML schemos aprašą:

```
<xsd:element name="Uredija" type="UredijaType"/>
  <xsd:complexType name="UredijaType">
    <xsd:sequence>
      <xsd:element name="Uredija_vardas" type="xsd:string"/>
      <xsd:element name="Uredija_kodas" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Jei UML klasės atributas aprašo pirminį duomenų tipą (pvz., *integer*, *string*), tai jį geriausia transformuoti į tą klasę atitinkančio elemento atributą, kuris taip pat aprašo pirminį duomenų tipą ir negali būti aprašytas kaip sudėtinio (*complexType*) tipo elementas:

```
<xsd:complexType name="UredijaType">
  <xsd:attribute name="vardas" type="xsd:string"/>
  <xsd:attribute name="kodas" type="xsd:string"/>
</xsd:complexType>
```

Yra keletas UML konstrukcijų, kurios neturi atitikimų XML schemose. Vienas iš tokių pavyzdžių yra UML atributų matomumo ir *frozen* savybė, kuri nurodo, kad atributo reikšmė gali būti priskirta vieną kartą ir išlieka statinė. Šios savybės negali būti atvaizduotos į visiškai ekvivalenčią XML schemos konstrukciją. Tik aplinkiniais sprendimais galima apibrėžti pradinę XML schemos atributo reikšmę panaudojant *fixed* savybę.

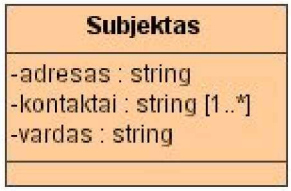
Kitas atvejis – išvestiniai atributai. Išvestinių atributų transformacijos įgyvendinimas reikalauja prisijungti prie kito XML dokumento dalių, kurios suteikia galimybę išvestines išraiškas transformuoti

į *XPath* išraišką. Paprasčiausias pasiūlymas – ignoruoti išvestinius atributus, nes jie nesaugo papildomos informacijos.

Prieš transformuojant UML atributą į XML schemos elementą ar atributą, būtina atsižvelgti į tai, kokio pobūdžio duomenys bus priskirti UML klasės atributo reikšmei. XML atributo reikšmė turi atitikti tam tikrus reikalavimus – XML redaktorius pašalina visus papildomus atributo reikšmės ženklus (pvz., perkėlimas į kitą eilutę ar keletas tarpo ženklų iš eilės). XML schemos atributui nereikėtų priskirti ir ilgos tekstinės eilutės. Tai ir yra pagrindinės UML klasės atributų pakeitimo XML elementais (o ne atributais) priežastys.

Galima pastebėti, kad keičiant UML atributą XML atributu, jo vardas nebūtinai turi turėti priekyje prirašytą klasės vardą, kuris buvo reikalingas vaizduojant UML atributą XML elementu. Pavyzdžiui, galima paprasčiausiai naudoti „*kodas*“ vietoj „*Uredija_kodas*“. Tai įmanoma todėl, kad XML elementas savo atributams sukuria atskirą rezervuotų pavadinimų mechanizmą (*namespace*).

Susiduriama su dar vienu sudėtingu transformavimo atveju, kai klasės atributas yra apibrėžtas kaip galintis turėti kelias reikšmes. XML schemoje negalimi daugiareikšmiai atributai, todėl tokie UML klasės atributai transformuojami į XML elementus. Pavyzdžiui, klasės „*Subjektas*“ atributas „*kontaktai*“gali įgyti nuo 1 iki n reikšmių. Todėl privalu šį atributą generuoti kaip XML schemos elementą:




```
<xsd:complexType name="SubjektasType">
 <xsd:sequence>
  <xsd:element name="Subjektas_adresas"
type="xsd:string"/>
  <xsd:element name="Subjektas_kontaktas"
type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
 </xsd:sequence>
 <xsd:attribute name="Subjektas_vardas"
type="xsd:string"/>
</xsd:complexType>
```

2.3 pav. UML diagramos klasė su daugiareikšmiu atributu

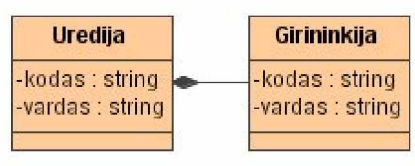
2.1 lentelėje pateiktas XML atributų ir elementų palaikomų savybių, kurios reikalingos atvaizduojant UML klasių diagramos elementus į XML schemą, palyginimas [13]:

1 lentelė. UML ir XML palaikomų savybių palyginimas

UML elementas	XML atributas	XML elementas
Pirminis duomenų tipas	Taip	Taip
Sudėtinis duomenų tipas	Ne	Taip
Kardinalumas	[0..1] ir [1..1] arba visi naudojami duomenų tipai iš sąrašo (reikšmė negali būti tuščia)	Visi
Pradinė reikšmė	Pradinės reikšmės (*default*) savybė	Pradinės reikšmės (*default*) savybė
Fiksuota reikšmė	Fiksuotos reikšmės (*fixed*) savybė	Fiksuotos reikšmės (*fixed*) savybė
Reikšmių sąrašas	Sąrašo palaikymas	Sąrašo palaikymas
Aprašo galiojimo sritis	Vietinė	Vietinė arba globali

2.1.3. UML kompozicijos ryšys

Kompozicija UML diagramose vaizduojama tamsiu rombu ryšio gale. Šis ryšys nurodo, kad susieti objektai valdomi pagal reikšmę. Panašiai XML elementas pagal reikšmes valdo jam priklausančius subelementus. Iš kitos pusės, jei savininkas ištrinamas, tai visi jam priklausę elementai taip pat ištrinami. Kompozicijos tipo ryšys XML schemoje gali būti realizuotas hierarchiniais ryšiais, nes nuo elemento egzistavimo priklauso subelementų egzistavimas, o tai ir atitinka kompozicijos ryšio semantiką. 2.4 paveiksle parodytas kompozicijos ryšio atvejis klasių diagramoje ir XML schemoje:



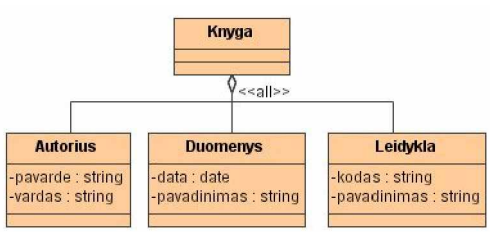

```
<xsd:element name="Uredija" type="Uredija"/>
<xsd:complexType name="Uredija">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0"
  name="Girininkija" type="Girininkija"/>
  <xsd:element name="Uredija_vardas"
type="xsd:string"/>
  <xsd:element name="Uredija_kodas"
type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Girininkija">
 <xsd:sequence>
  <xsd:element name="Girininkija_kodas"
type="xsd:string"/>
  <xsd:element name="Girininkija_vardas"
type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
```

2.4 pav. Kompozicijos ryšys UML diagramoje ir XML schemoje

2.1.4. UML agregavimo ryšys

UML agregavimo ryšys gali būti išreikštas keliomis skirtingomis XML dokumento palaikomomis struktūromis: *sequence, choice* ir *all* [16].

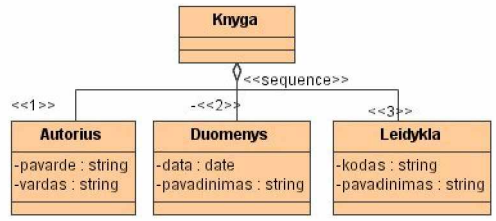
All darinys XML schemoje reiškia, kad elementas gali turėti visus *all* darinyje aprašytus subelementus, ir jie gali būti pateikti bet kokia tvarka – jų išrikiavimo tvarka XML dokumente nesvarbi. Jei klasių diagramoje agregavimo ryšiu susietų klasių tvarka nesvarbi, ši XML schemos konstrukcija labiausiai tinka ryšio realizavimui XML dokumente. Tarkime, kad knygos apraše nesvarbu kuria tvarka bus pateikti autorius, pavadinimas, leidykla ir kiti duomenys. Tokiu atveju agregavimo ryšys XML schemoje gali būti realizuojamas panaudojant *all* konstrukciją:




```
<xsd:element name="Knyga">
 <xsd:complexType>
  <xsd:all>
   <xsd:element name="Autorius" type="AutoriusType"/>
   <xsd:element name="Duomenys" type="DuomenysType"/>
   <xsd:element name="Leidykla" type="LeidyklaType"/>
  </xsd:all>
 </xsd:complexType>
</xsd:element>
```

2.5 pav. Agregavimo ryšys UML diagramoje ir XML schemoje

UML stokoja klasių ir atributų rikiavimo, nes nėra poreikio programiniuose objektuose turėti surikiuotus kompozicijos ir agregavimo ryšiais sujungtus objektus. Jei nenurodyti jokie stereotipai, agregavimo ryšys transformuojamas į *sequence* XML schemos konstrukciją. Taip gaunama agregavimo ryšio objektų seka. Jei norima, kad agregavimo ryšys būtų transformuojamas tiktai į XML *sequence* konstrukciją, UML klasių diagramoje turėtų būti naudojamas <<*sequence*>> stereotipas. Šiuo atveju būtina tiksliai nurodyti elementų tvarką naudojant papildomus stereotipus: <<1>>, <<2>>, ..., <<n>> (n – elementų skaičius). 2.6 paveiksle pateiktas tas pats knygos klasių diagramos pavyzdys, tik šiuo atveju jau svarbus agregavimo ryšiu susietų klasių išrikiavimas:

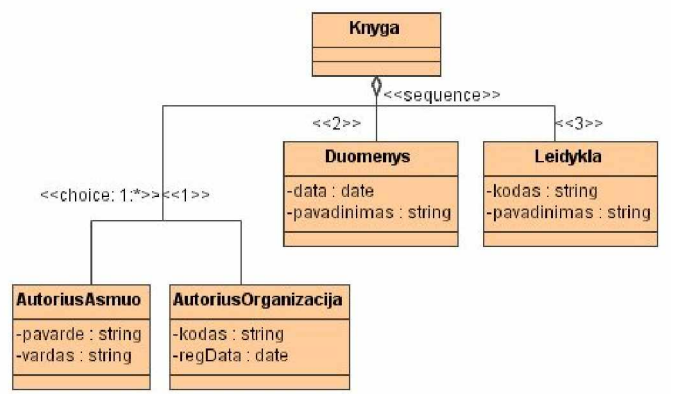



```
<xsd:element name="Knyga">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Autorius" type="AutoriusType"/>
   <xsd:element name="Duomenys" type="DuomenysType"/>
   <xsd:element name="Leidykla" type="LeidyklaType"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

2.6 pav. <<sequence>> stereotipo panaudojimas agregavimo ryšyje

Dažniausiai *choice* konstrukcija naudojama apibendrinimo ryšio atveju. Agregavimo ryšio atveju XML schemos *choice* konstrukcijos realizavimui siūloma UML klasių diagramą praplėsti <<*choice*>> stereotipu. Ši konstrukcija taip pat gali turėti savo kardinalumą, pavyzdžiui, <<choice:1..*>>.

Nėra sunku UML modelyje kartu suderinti *sequence* ir *choice* konstrukcijas:



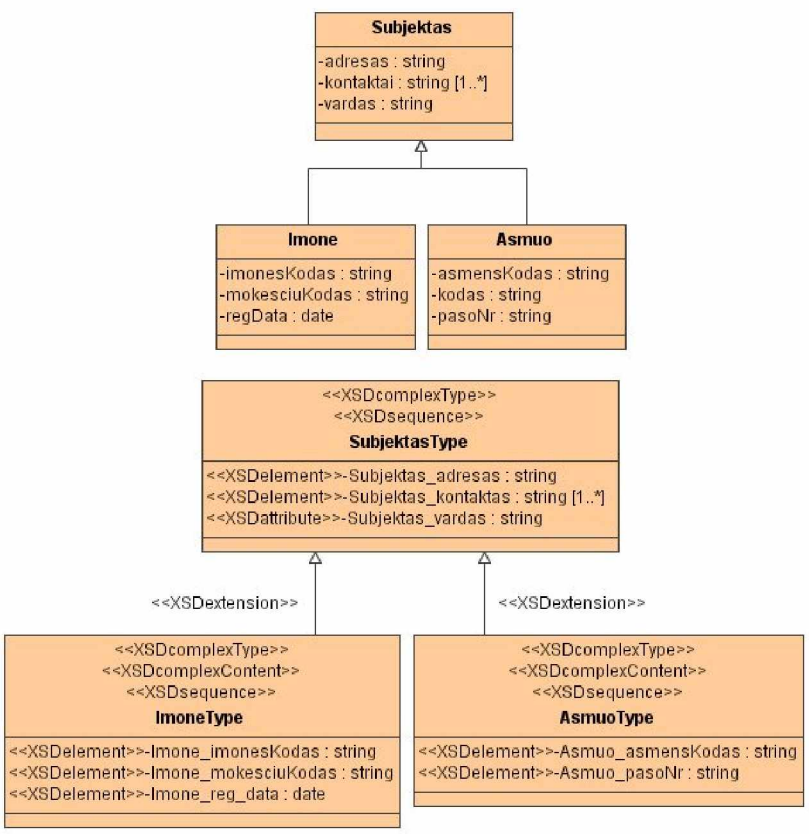

```
<xsd:element name="Knyga">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:choice>
    <xsd:element name="AutoriusAsmuo"
type="AutoriusAsmuoType"/>
    <xsd:element name="AutoriusOrganizacij"
type="AutoriusOrganizacijaType"/>
   </xsd:choice>
   <xsd:element name="Duomenys"
type="DuomenysType"/>
   <xsd:element name="Leidykla"
type="LeidyklaType"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

2.7 pav. Agregavimo ryšys praplėstas <<sequence>> ir <<choice>> stereotipais

2.1.5. UML apibendrinimo ryšys

XML schemose nėra apibendrinimo konstrukcijų. Svarbiausias apibendrinimo aspektas yra superklasės atributų paveldėjimas. Vienas iš siūlomų būdų, kuriuo galima realizuoti paveldėjimą XML schemose, yra *extension* konstrukcijos panaudojimas XML schemoje. Kiekvienai klasei sukuriamas sudėtinio turinio elementas. Subklasę atitinkantis sudėtinio tipo elementas yra panaudojamas kaip superklasės sudėtinio turinio elemento išplėtimas. Tokiu būdu subklasės elementas paveldi visas

superklasės elemento savybes. 2.8 paveiksle pavaizduotas apibendrinimo ryšio atvejis ir jo išplėtimas stereotipais:



2.8 pav. Apibendrinimo ryšys UML diagramoje

```
     <xsd:complexType name="SubjektasType">
      <xsd:sequence>
       <xsd:element name="Subjektas_adresas" type="xsd:string"/>
       <xsd:element maxOccurs="unbounded" minOccurs="1" name="Subjektas_kontaktas"
type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="Subjektas_vardas" type="xsd:string"/>
     </xsd:complexType>
     <xsd:complexType name="AsmuoType">
      <xsd:complexContent>
       <xsd:extension base="SubjektasType">
        <xsd:sequence>
         <xsd:element name="Asmuo_asmensKodas" type="xsd:string"/>
         <xsd:element name="Asmuo_pasoNr" type="xsd:string"/>
        </xsd:sequence>
       </xsd:extension>
      </xsd:complexContent>
     </xsd:complexType>
     <xsd:complexType name="ImoneType">
      <xsd:complexContent>
       <xsd:extension base="SubjektasType">
        <xsd:sequence>
         <xsd:element name="Imone_imonesKodas" type="xsd:string"/>
         <xsd:element name="Imone_mokesciuKodas" type="xsd:string"/>
         <xsd:element name="Imone_reg_data" type="xsd:date"/>
        </xsd:sequence>
       </xsd:extension>
      </xsd:complexContent>
     </xsd:complexType>
```

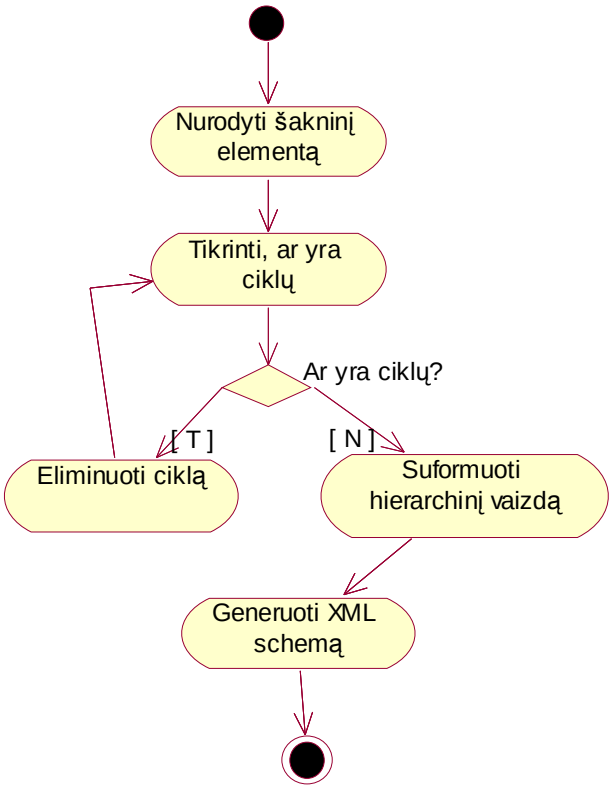
2.9 pav. Apibendrinimo ryšio realizacija XML schemoje

2.2. XML schemų kūrimo algoritmas

Dauguma sistemų naudoja skirtingų formatų duomenis, todėl atsiranda sunkumų keičiantis duomenimis. XML yra standartinis formatas, kurį įvairios programos supranta vienodai, todėl duomenys gali būti transformuoti į XML ir laisvai perduoti kitai sistemai ar programai. XML palaiko savybes, kurios būdingos įvairiems duomenų modeliams (tarp jų ir UML klasių diagramai). Todėl svarbu sudaryti iš UML klasių diagramos XML schemų automatinio generavimo metodologiją.

Kadangi XML dokumento struktūra yra hierarchinė, keletas skirtingų XML schemų (ar XML dokumentų) gali būti sugeneruotos pagal tą pačią konceptualiąją UML klasių diagramą. Iš UML klasių diagramos XML schemos generavimo algoritmas susideda iš keleto žingsnių:

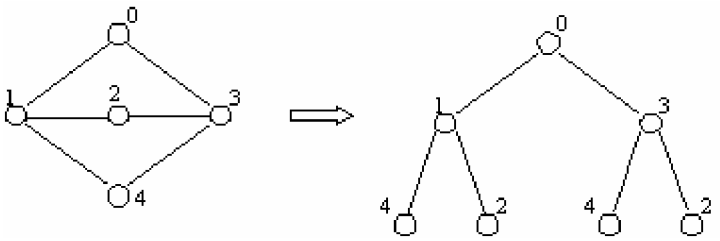
- Ne visada UML klasių diagramoje klasės ir jų tarpusavio ryšiai sudaro hierarchinį vaizdą – dažnai diagramose pasitaiko vienas ar keli ciklai. Kadangi XML dokumentas yra hierarchinė struktūra, tai modeliuojant XML schemą UML klasių diagramoje turi būti pašalinami ciklai.
- Pašalinus UML klasių diagramoje ciklus, seka antrasis algoritmo žingsnis – hierarchinio vaizdo formavimas. Hierarchinio vaizdo formavimas susijęs su ryšių ir jų kardinalumų pertvarkymu klasių diagramoje.
- Trečiasis ir paskutinis algoritmo žingsnis – XML schemos automatinis generavimas pagal sudarytą hierarchinį vaizdą.



2.10 pav. XML schemos generavimo algoritmo veiklos diagrama

2.2.1. Ciklų eliminavimas

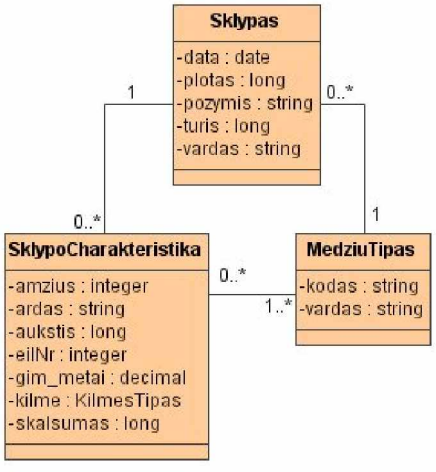
Dažnai pasitaiko, kad UML klasių diagramoje klasės ir jų ryšiai sudaro vieną ar kelis ciklus. XML dokumentas yra hierarchinė struktūra. Dėl šios priežasties pirmiausia UML klasių diagramoje turi būti pašalinami ciklai ir suformuojamas hierarchinis vaizdas. Hierarchinio vaizdo struktūra priklauso nuo to, kuri diagramos klasė bus pasirinkta ir traktuojama kaip šakninis XML dokumento elementas. Šiame algoritmo žingsnyje pirmiausia visos klases, išskyrus tą, kuri pažymėta kaip šakninė, suskirstomos į atskiras klasių grupes pagal tai, kokių atstumu jos nutolusios nuo šakninės klasės. Tada bandoma rasti labiausiai nuo šakninės klasės nutolusią klasę, kuri įeina į kurį nors ciklą. Šis ciklas pašalinamas dubliuojant šią klasę taip, kad išliktų jos prieš tai buvę ryšiai su kitomis klasėmis. Pašalinus ciklą, šis procesas kartojamas tol, kol klasių diagramoje nebelieka ciklų. Ciklų eliminavimas parodyta 2.11 paveiksle:



2.11 pav. Ciklų eliminavimas

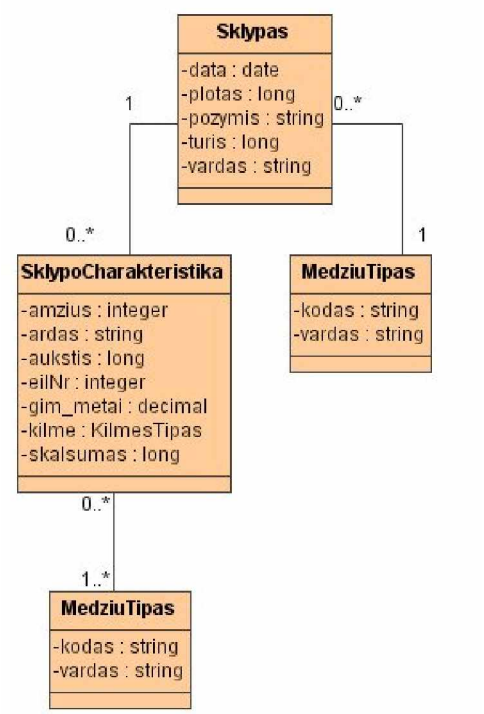
2.11 paveiksle pavaizduotame grafe matyti, kad 2-oji ir 4-oji viršūnės yra labiausiai nutolusios nuo šakninės (nulinės) viršūnes ir sudaro ciklą su kitomis viršūnėmis. Pagal prieš tai aprašytą algoritmą, bus surasta 2-oji viršūnė, kuri sudaro ciklą su 0-ne, 1-ąja ir 3-ąja viršūnėmis. Šis ciklas pašalinamas dubliuojant 2-ąją viršūnę – grafe atsiranda dvi viršūnės su tuo pačiu numeriu. Analogiškai surandama ir dubliuojama 4-oji viršūnė.

2.12 ir 2.13 paveiksluose parvaizduotas ciklo eliminavimas klasių diagramoje. Čia pateikta trijų UML klasių diagrama, kurioje klasės ir jų ryšiai tarpusavyje sudaro ciklą:



2.12 pav. Ciklas, kurį sudaro klasės ir jų ryšiai

Pirmiausia pasirenkama šakninė klasė (šiuo atveju - klasė „*Sklypas*“). Norint pašalinti ciklą, reikia dubliuoti vieną iš dviejų likusių klasių („*SklypoCharakteristika*“ arba „*MedziuTipas*“), nes atstumai nuo kiekvienos iš jų iki šakninės klasės yra vienodi. Tarkime, ciklo pašalinimui pasirenkama klasė „*MedziuTipas*“. Ši klasė dubliuojama ir ciklas pašalinamas taip, kad ryšiai tarp klasių liktų tokie patys.



2.13 pav. Klasių diagrama, po ciklo pašalinimo

2.2.2. Hierarchinio vaizdo formavimas

Nurodžius UML klasių diagramoje šakninį elementą ir pašalinus ciklus, atliekamas antrasis algoritmo žingsnis – hierarchinio vaizdo formavimas. Hierarchinis vaizdas formuojamas atsižvelgiant į UML klasių tarpusavio ryšius ir jų kardinalumus.

- Jei tarp klasių A ir B egzistuoja 1:N arba 1:1 ryšiai, nieko keisti nereikia. Tokiu atveju XML schemoje B klasė tampa A klasę atitinkančio elemento subelementu.
- M:N ryšio atveju, šiame algoritmo žingsnyje ryšys pakeičiamas 1:N tipo ryšiu.
- Jei 1:N ar M:N ryšių atveju egzistuoja ryšio klasė, tai ji XML schemoje tampa A klasę atitinkančio elemento subelementu.
- Jei klasių tarpusavio ryšys yra N:1, formuojant hierarchinį vaizdą šis ryšys pakaičiamas 1:1 tipo ryšiu.

Be 1:1, 1:N, N:1 ir M:N tipo asociacijų, klasių diagramoje galimi agregavimo, kompozicijos ir apibendrinimo ryšiai.

XML schemose nėra apibendrinimo konstrukcijų. Svarbiausias apibendrinimo aspektas yra superklasės atributų paveldėjimas. Vienas iš prieinamų sprendimų atvaizduojant paveldimumą XML

schemose yra savybių paveldėjimas panaudojant *extension* atributą XML schemoje. Formuojant hierarchinį vaizdą, apibendrinimo ryšys klasių diagramoje nekeičiamas, tačiau į jį reikia atsižvelgti trečiajame algoritmo žingsnyje generuojant XML schemą.

UML stokoja klasių ir atributų rikiavimo, nes nėra poreikio programiniuose objektuose turėti surikiuotus kompozicijos ir agregavimo ryšiais sujungtus objektus. Pagal nutylėjimą, agregavimo ir kompozicijos ryšiai transformuojami į *sequence* XML schemos konstrukciją. Taip gaunama šių ryšių objektų seką. Jei norima, kad agregavimo ar kompozicijos ryšys tiksliai būtų transformuojamas į XML *sequence*, *all* ar *choice* konstrukcijas, UML klasių diagramoje turėtų būti naudojami stereotipai.

2.2.3. XML schemos generavimas

Nurodžius šakninį UML diagramos elementą (klasę), eliminavus visus diagramos ciklus ir suformavus hierarchinį vaizdą pereinama prie XML schemos generavimo. Šis algoritmas aprašo pagrindinius standartinės transformacijos aspektus, kurie panaudojami XML schemai modeliuoti.

UML klasės su atitinkamomis savybėmis transformuojamos į sudėtinio tipo (*complexType*) XML elementus. Klasės vardas tampa elemento vardu ir pridėjus prie klasės vardo „*Type*“ galūnę – sudėtinio tipo vardu. Klasės atributai tampa šią klasę aprašančio *complexType* subelementais, kurių duomenų tipai atitinka atributų duomenų tipus. Subelemento vardas sudaromas iš klasės ir atributo vardo, atskiriant juos tašku. UML klasės atributų kardinalumai XML schemoje išreiškiami kardinalumą aprašančiais *minOccure* ir *maxOccure* atributais. Asociacijos, agregavimo ir kompozicijos ryšiai transformuojami į *sequence* XML schemos konstrukcijas. Taip gaunama šių ryšių objektų sekos. Ryšio kardinalumas tiesiogiai išreiškiamas *minOccure* ir *maxOccure* atributais. Apibendrinimo ryšio atveju generuojamas *complexType* su *extension* subelementu, kurio *base* atributas susiejamas su superklasės vardu.

Šakninio klasės diagramos elemento vardas bus panaudotas pagrindinio XML schemos elemento pavadinimui. Šio elemento tipas bus “*rootType*“, kuris yra sudėtinio tipo (*complexType*) elementas. Tarkime, jei šakninės klasės vardas „*Uredija*“, tai XML schemos šakninio elemento pavadinimas bus „*UredijaDoc*“, o tipas „*rootType*“. Nuo šio elemento ir pradedamas XML schemos generavimas, kuriam naudojama rekursinė procedūra. Tarkime, E1 XML schemos generavimo procedūros įėjimo parametras (pradžioje E1 – šakninis UML klasių diagramos elementas):

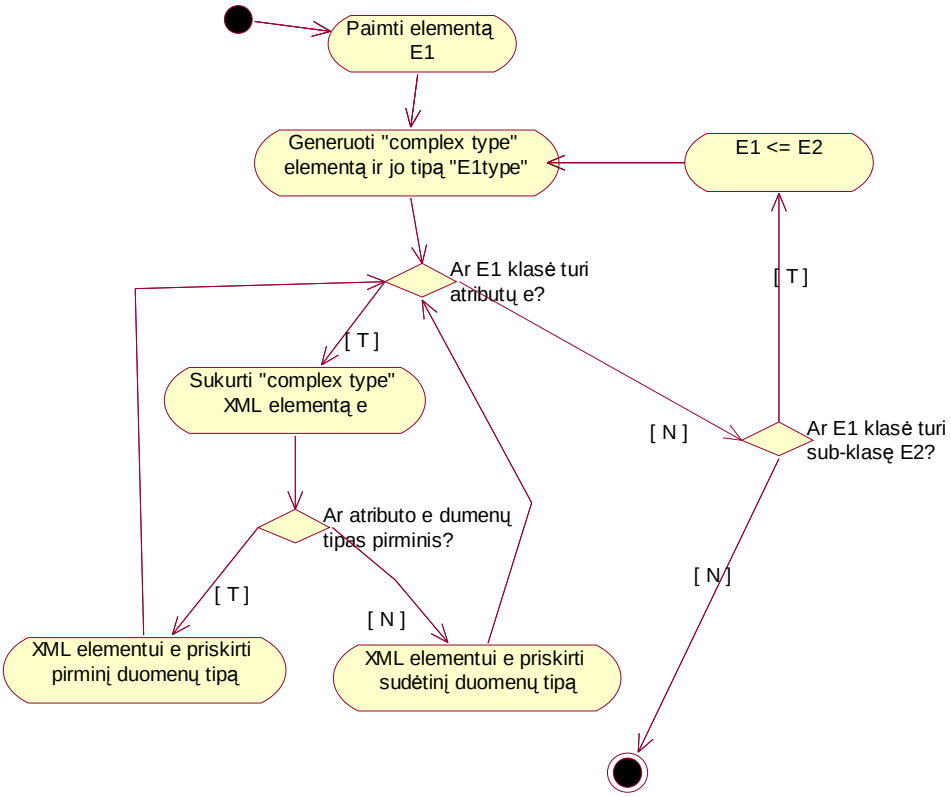
1. Generuojamas sudėtinio tipo (*comlexType*) elementas E1 ir jo tipas „E1Type“.
2. Kiekvienam šį elementą atitinkančios klasės atributui e generuojamas:
   2.1. Jei atributo tipas pirminis (*integer, string*), tai generuojamas elementas E1.e ir jo pirminis tipas (*xsd:integer, xsd:string*);
   2.2. Kitu atveju, generuojamas sudėtinio tipo (*complexType*) elementas E1.e, kurio tipas atitinka atributo tipą. Pavyzdžiui, jei atributo e tipas t, tai generuojamas *complexType* elemetas E1.e ir jo tipas „t“.
3. Kiekvienai klasei E2, kuri yra E1 klasės subklasė hierarchiniame vaizde, kartojami visi ankstesni algoritmo žingsniai.

Kadangi ši procedūra yra rekursinė, tai baigus ją vykdyti, bus sugeneruoti XML schemos elementai kiekvienai UML klasei. Pagal šį algoritmą sugeneruojama *griežta* XML schema:

§ Kardinalumai yra apribojami pagal UML klasių modelį panaudojant *minOccure* ir *maxOccure* atributus kiekvienai klasių asociacijai aprašyti.

§ Visuose sudėtinio tipo elementuose naudojama *sequence* konstrukcija subelementams patalpinti.

§ Generuojami tik XML elementai (nėra XML atributų).



2.14 pav. XML schemos automatinio generavimo veiklos diagrama

Detalesni *griežtos* XML schemos generavimo kriterijai:

Pavadinimų mechanizmas – visas UML klasių modelis pateikiamas vienoje XML schemoje.

Elemento vardo unikalumas – generuojant atributą atitinkantį elementą, jo vardas sudaromas iš klasės ir atributo vardo, atskiriant juos tašku (pavyzdžiui, Klasė.atributas). Jei taip sudarytas vardas nėra unikalus, dar pridedamas paketo vardas (pavyzdžiui, Paketas.Klasė.atributas).

Elementas ar atributas – generuojamas elementas kiekvienam klasės atributui

Kardinalumas – kiekvieno elemento aprašyme *minOccure* ir *maxOccure* atributams priskiriamos reikšmės atitinkančios UML klasių modelį.

Paveldėjimas – jei UML klasė turi apibendrinimo ryšį su viena superklase, tai generuojamas *complexType* su *extension* subelementu, kurio *base* atributas susiejamas su superklasės vardu. Jei UML klasė turi apibendrinimo ryšį su keliomis superklasėmis, tai dar kartą aprašomi superklasės atributai jos subklasę atitinkančio elemento turinyje.

complexType **turinys** – kiekviena UML klasė yra generuojama kaip *complexType* elementas, kuriame naudojama *sequence* konstrukcija.

Duomenų tipas – jei atributo tipas pirminis (pvz., *integer, string*), tai generuojamas elementas, kurio tipas taip pat pirminis. Kitu atveju, generuojamas sudėtinio tipo (*complexType*) elementas, kurio tipas atitinka atributo tipą.

2.3. XML schemų norminės formos

Efektyvus XML dokumento panaudojimas priklauso nuo šio dokumento metaduomenų kokybės – XML schemos arba DTD. Programos gali talpinti didelius duomenų kiekius į XML dokumentą ir dažnai vykdyti XML dokumento duomenų atnaujinimo operacijas. Esant netinkamai XML schemos struktūrai, XML dokumente galimi tokie trūkumai kaip duomenų perteklius ir dubliavimasis. Kaip ir sąryšinės duomenų bazėse, duomenų dubliavimas gali sąlygoti didelę saugomų duomenų apimtį ir netikslumus vykdant operacijas su duomenimis. Todėl naudinga XML schemų struktūros kūrimui suformuoti ir pritaikyti nurodymus ir principus, kuriais remiantis galima sukurti tinkamą XML schemos struktūrą [8].

Normalizacija yra vienas iš būdų, kuris leidžia užtikrinti gerą sąryšinių duomenų bazių projektavimą – pašalinti nevienareikšmiškas duomenų išraiškas, minimizuoti perteklių ir palengvinti duomenų logiškumo išsaugojimą. Tas pačias problemas ir jų pašalinimo metodus galima pritaikyti ir XML dokumentui [9].

XML norminių formų aprašas paremtas funkcinėmis priklausomybėmis. Pateiktas pagrindinių funkcinių priklausomybių savybių aprašymas. Apibrėžtos *neteisingos* funkcinės priklausomybės, kurios ir yra duomenų dubliavimosi XML schemoje priežastis. Be to, pateiktas algoritmas, kuris XML schemą transformuoja į XML norminę formą – XNF.

2.3.1. XML formalus aprašas

Sąryšinės duomenų bazės nagrinėjamos remiantis funkcinėmis priklausomybėmis. Norminės formos taip pat analizuojamos remiantis funkcinių priklausomybių, kurios įtakoja duomenų perteklių, eliminavimu. Kaip ir sąryšinių duomenų bazių atveju, XML dokumento norminės formos apibrėžiamos panaudojant funkcines priklausomybes, kurios sudaro normalizavimo algoritmo pagrindą.

XML schemos atveju galima įvesti tokius žymėjimus ir apibrėžimus:

- *El* – elemento vardai;
- *Att* – atributo vardai;
- *Str* – tekstinio (*string*) tipo atributų galimos reikšmės;
- *Vert* – viršūnių identifikatoriai;
- Visi atributų vardai prasideda @ simboliu.
- *S* ir $\perp$ (*null*) – rezervuoti nei vienoje aibėje nenaudojami simboliai.

2.3.1.1. XML medis

XML medis *T* apibrėžiamas kaip medis (*V, lab, ele, att, root*), kur:

- $V \subseteq Vert$ yra baigtinė viršūnių aibė;
- $lab: V \to El$;
- $ele: V \to Str \cup V$;
- *att* yra dalinė funkcija $V \times Att \to Str$;
- $root \in V$ vadinama medžio *T* šaknimi.

XML medžiui *T* eilutė $w=w_1...w_n$, kur $w_1,..., w_n\text{-}1 \to El$, o w_n yra *El, Att* ir *{S}*, yra *T* kelias, jei aibėje *V* yra tokios viršūnės $v_1,...,v_{n-1}$, kad:

- $v_1=root$, v_{i+1} yra viršūnės v_i subelementas $(1 \le i \le n-2)$, $lab(v_i)=w_i(1 \le i \le n-2)$;
- Jei $w_n \in El$, tai yra tokia viršūnei v_{n-1} priklausanti žemesnio lygio viršūnė v_n, kad $lab(v_n)=w_n$. Jei $w_n=S$, tai v_{n-1} turi subelementą *Str*.

Galima pažymėti *paths(T)* kaip XML medžio *T* kelių aibę.

2.3.1.2. XML schema

XML schema apibrėžiama kaip *D = (E, A, P, R, r)*, kur:

- $E \subseteq El$ baigtinė elementų tipų aibė;
- $A \subseteq Att$ baigtinė atributų aibė;
- *P* – *E* susiejimas su elemento tipo aprašais;
- *R* – *E* susiejimas su *A*;
- $r \in E$ ir yra vadinamas šakninio elemento tipu.

XML schemos *D = (E, A, P, R, r)* eilutė $w = w_1...w_n$ yra XML schemos *D* kelias, jei $w_1=r$, w_i priklauso $P(w_{i-1})$ alfabetui arba $w_n = @l$, kur $@l \to R(w_n\text{-}1)$. Galima apibrėžti *length(w)* kaip kelio ilgį *n*, *last(w)* – kaip paskutinį kelio elementą w_n, *paths(D)* – visų XML schemos *D* kelių aibę, o *EPaths(D)* – aibę visų kelių, kurie baigiasi elemento tipu.

XML medis *T* atitinka XML schemą *D (D|=T)*, jei:

- *lab* yra sąryšis nuo *V* prie *E*;
- Kiekvienam $v \in V$, jei $P(lab(v))=S$, tai $ele(v)=[s]$, kur $s \in Str$;

- *att* yra tokia dalinė funkcija nuo $V \times A$ prie *Str*, kad kiekvienam $v \in V$ ir $@l \in A$, *att(v,@l)* yra apibrėžta, jei $@l \in R(lab(v))$;
- *lab(root)=r*;
- $paths(T) \subseteq paths(D)$.

2.3.1.3. XML medžio sąrašai

Kad išplėsti XML funkcinių priklausomybių supratimą, XML medžiai pavaizduojami kaip sąrašų aibės. Sąryšinėse duomenų bazėse sąrašas yra funkcija, kuri kiekvienam atributui priskiria reikšmę iš atitinkamos srities. XML schemos *D* medžio sąrašas *t* kiekvienam medžio keliui priskiria reikšmę iš $Vert \cup Str \cup \{\perp\}$ tokiu būdu, kad *t* atitinka baigtinį medį su keliais iš *D*, kuris turi daugiausia vieną atvejį iš kiekvieno kelio.

XML medis gali būti išreikštas kaip medžio sąrašų aibė.

Duotos XML schemos *D = (E, A, P, R, r)* medžio sąrašas yra tokia funkcija nuo *paths(D)* iki $Vert \cup Str \cup \{\perp\}$, kad, jei $p \in EPaths(D)$, tai $t(p) \in Vert \cup \{\perp\}$ ir $t(r) \neq \perp$.

T(D) apibrėžiama kaip visų *D* medžio sąrašų aibė. Medžio sąrašas *t* ir kelias *p* aprašomas kaip *t.p*.

2.3.1.4. XML funkcinės priklausomybės

XML schemos *D* funkcinės priklausomybės (*FP*) yra $S1 \rightarrow S2$ formos išraiškos, kur *S1* ir *S2* yra baigtiniai netušti aibės *paths(D)* poaibiai. Visų XML schemos *D* funkcinių priklausomybių aibė pažymima kaip *FP(D)*.

2.3.2. Pirmoji norminė forma

Pirmoji norminė forma reikalauja, kad visos atributų reikšmės būtų atominės. Atomiškumą apibrėžti yra gana sunku, nes ši sąvoka yra gana reliatyvi. Reikšmė, kuri yra atomas vienu atveju, gali būti ne atomu kitame taikyme. Bendras principas – reikšmė yra ne atominė, jei ji vartojama dalimis, t. y., jei ji vartojama dalimis sąryšiuose su kitomis reikšmėmis ar jų dalimis.

Sąryšinės duomenų bazės lentelė *l(L)* yra 1-joje NF, jei visų jos atributų reikšmių aibių elementai yra atomai (nedalomi), t. y., $a \in L$: *a* reikšmių aibė yra atomai, kur:

- *l* – eilučių rinkinys (nesutvarkyta aibė) $\{e_1, e_2, ..., e_m\}$, kur visos eilutės skirtingos, t. y., $e_i \neq e_j$, *jei* $i \neq j$ *(i, j=1,..,n)*;
- *L* – lentelės *l* schema (struktūra) – aibė $\{a_1, a_2, ..., a_n\}$, kur a_i *(i=1,..,n)* – atributai (lentelės stulpelių vardai).

XML schemai *D* ir XML medžiui $T \models D$, jei kiekvienos viršūnės $v \in V_e$ (t. y., $ele(v) = \perp$ ar $v \in A$) reikšmė yra atominė, tai XML schema *D* atitinka pirmąją XML norminę formą (1XNF).

1XNF reikalauja, kad nebūtų galima priskirti daugiau nei vienos reikšmės XML dokumento elementui ar atributui. Jei XML schema nėra pirmojoje norminėje formoje, ji gali būti transformuota į 1-ąją norminę formą pridedant papildomus subelementus ar atributus elementams. 2.15 paveiksle pateiktas XML schemos pavyzdys, kuris netenkina 1-osios norminės formos reikalavimų, o 2.16 paveiksle pavaizduota XML schema po transformavimo į 1-ąją norminę formą.

```
<xsd:element name="Asmuo">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Asmens kodas" type="xsd:string" />
      <xsd:element name="VPavardė" type="xsd:string" />
      <xsd:element name="Adresas" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

2.15 pav. XML schema, kuri neatitinka 1XNF

```
<xsd:element name="Asmuo">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Asmens kodas" type="xsd:string" />
      <xsd:element name="Pavardė" type="xsd:string" />
      <xsd:element name="Vardas" type="xsd:string" />
      <xsd:element name="Miestas" type="xsd:string" />
      <xsd:element name="Gatve" type="xsd:string" />
      <xsd:element name="Namo_nr" type="xsd:string" />
      <xsd:element name="Buto_nr" type="xsd:integer" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

2.16 pav. XML schema, kuri atitinka 1XNF

2.3.3. Antroji norminė forma

Antroji ir trečioji norminės formos yra apribotos funkcinėmis priklausomybėmis ir susijusios su sąryšiais tarp raktinių ir neraktinių atributų.

Atributų aibė B vadinama *pilnai* priklausančia nuo atributų aibės A funkcinių sąryšių aibės F atžvilgiu, jei B priklauso nuo visos aibės A, bet nepriklauso nuo jokio aibės A poaibio. Formaliai, $A \rightarrow B \in F^+$ ir $A' \subset A : A' \rightarrow B \notin F^+$.

Antroji norminė forma reikalauja, kad kiekvienas neraktinis atributas būtų funkcionaliai priklausomas nuo visų raktinių atributų, t. y., negali priklausyti nuo rakto dalies (vieno iš raktinių atributų). Taigi 2NF gali būti pažeista tik turint sudėtinį raktą.

Sąryšinės duomenų bazės lentelė $l(L)$ yra 2-joje NF funkcinių sąryšių aibės F atžvilgiu, jei ji yra 1-joje NF, ir kiekvienas jos neraktinis atributas pilnai priklauso nuo kiekvieno lentelės l rakto. Bendru atveju, tam, kad lentelė būtų 2-joje NF, reikia, kad kiekvienas jos neraktinis atributas pilnai priklausytų nuo kiekvieno jos rakto. Jei taip nėra, tai reikia atributus, dalyvaujančius dalinėje priklausomybėje,

iškelti į kitą lentelę. Schematiškai tai galima pavaizduoti taip: jei turime lentelę l(***A, B***, *C*, *D*), kurioje galioja: $\{A, B\} \rightarrow \{C, D\}$ ir $A \rightarrow D$, tai lentelę l reikia skaidyti į dvi: l_1(***A, B***, *C*) ir l_2(***A***, *D*).

Išskaidymo etapai:

1. Sukuriam nauja lentelė, kurios atributai yra pradinės lentelės atributai, netenkinantys funkcinės priklausomybės sąlygos. Funkcinės priklausomybės determinantas (atributas, kuris yra kairėje FP pusėje) tampa naujos lentelės raktu.
2. Atributas, esantis dešinėje FP pusėje, išmetamas iš pradinės lentelės.
3. Jeigu yra daugiau negu viena FP, kuri pažeidžia 2NF, tai 1 ir 2 etapai kartojami kiekvienai FP.
4. Jeigu vienas determinantas įeina į kelias FP, tai visi nuo jo funkcionaliai priklausanti atributai talpinami kaip neraktiniai atributai į lentelę, kurios raktu bus determinantas.

XML schemoje taip pat galima apibrėžti dalinę ir tranzityvinę priklausomybes.

Jei funkcinei priklausomybei $\{S_1.S_2.S_3.\sigma_2\} \rightarrow S_1.S_2.S_3.\sigma$ egzistuoja kita funkcinė priklausomybė $S_1.\sigma_1 \rightarrow S_1.S_2.S_3.\sigma$, kur $S_1 \neq \perp$ *ir* $S_1 \neq r$, $\sigma_1, \sigma_2, \sigma$ yra elemento reikšmė ar atributas, tai pirmoji funkcinė priklausomybė vadinama daline funkcine priklausomybe.

Jei funkcinei priklausomybei $S_1.\sigma_1 \rightarrow S_1.S_2.\sigma_2$ egzistuoja kita funkcinė priklausomybė $S_1.S_2.\sigma_2 \rightarrow S_1.S_2.S_3.\sigma_3$, tai funkcinė priklausomybė $S_1.\sigma_1 \rightarrow S_1.S_2.S_3.\sigma_2$ vadinama tranzityvine funkcine priklausomybe.

XML schemai transformuoti į 2-ąją norminę formą galima panaudoti submedžio perkėlimą į viršų. Tai atliekama tol, kol nelieka dalinių funkcinių priklausomybių.

XML schemos pavyzdys prieš 2NF pritaikymą:

```
<xsd:elemente name="Miškas">
 <xsd:sequence>
  <xsd:element minOccurs="1" maxOccurs="unbounded" name="Uredija" type="UredijaType"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UredijaType">
 <xsd:sequence>
  <xsd:element name="Kodas" type="xsd:string"/>
  <xsd:element name="Pavadinimas" type="xsd:string"/>
  <xsd:element minOccurs="0" maxOccurs="unbounded" name="Girininkija"
type="GirininkijaType"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="GirininkijaType">
 <xsd:sequence>
  <xsd:element name="Kodas" type="xsd:string"/>
  <xsd:element name="Pavadinimas" type="xsd:string"/>
  <xsd:element name="Uredas" type="UredasType"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UredasType">
 <xsd:sequence>
  <xsd:element name="AsmensKodas" type="xsd:string"/>
  <xsd:element name="Pavarde" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
```

Galima nurodyti tokias funkcines priklausomybes:

F1: {Miskas.Uredija, Miskas.Uredija.GirininkijaType.Girininkija.Uredas.@AsmensKodas →Miskas.Uredija.GirininkijaType.Girininkija.Uredas.@Pavarde}

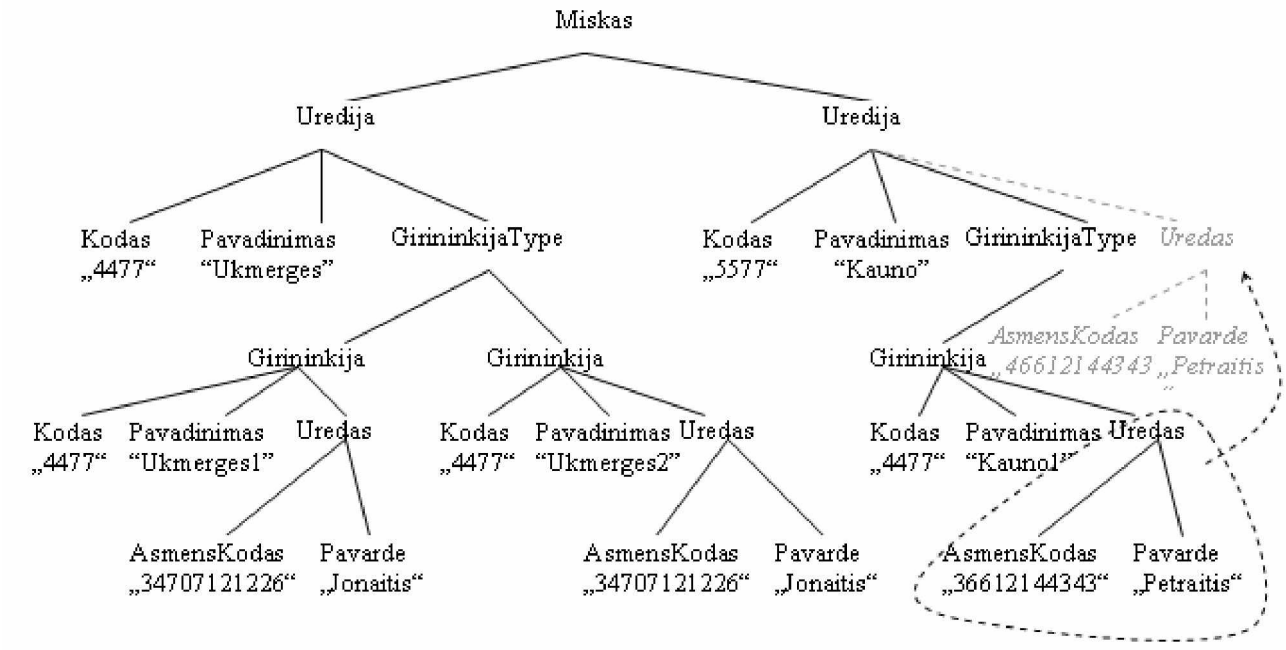
F2: {Miskas.Uredija.@Kodas→Miskas.Uredija.GirininkijaType.Girininkija.Uredas.@Pavarde}

Funkcinės priklausomybės F2 egzistavimas parodo, kad F1 funkcinė priklausomybė yra dalinė. Taigi XML schema nėra antrojoje norminėje formoje. Tam, kad XML schema atitiktų 2-ąją norminę formą, reikia pašalinti F1 funkcinę priklausomybę. Šiuo atveju reikia submedį *Uredas* perkelti į viršų, t. y., jis tampa šakos *Miskas.Uredija* elementu.

XML schema po F1 dalinės funkcinės priklausomybės pašalinimo:

```
<xsd:elemente name="Miškas">
 <xsd:sequence>
  <xsd:element minOccurs="1" maxOccurs="unbounded" name="Uredija" type="UredijaType"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UredijaType">
 <xsd:sequence>
  <xsd:element name="Kodas" type="xsd:string"/>
  <xsd:element name="Pavadinimas" type="xsd:string"/>
  <xsd:element minOccurs="0" maxOccurs="unbounded" name="Girininkija"
type="GirininkijaType"/>
  <xsd:element name="Uredas" type="UredasType"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="GirininkijaType">
 <xsd:sequence>
  <xsd:element name="Kodas" type="xsd:string"/>
  <xsd:element name="Pavadinimas" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UredasType">
  <xsd:sequence>
    <xsd:element name="AsmensKodas" type="xsd:string"/>
    <xsd:element name="Pavarde" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

2.17 paveiksle pavaizduotas XML schemos transformavimas į antrąją norminę formą perkeliant XML submedį *Uredas* į viršų:



2.17 pav. Submedžio perkėlimas į viršų

2.3.4. Trečioji norminė forma

Trečioji norminė forma reikalauja, kad kiekvienas determinantas būtų raktas. Yra ir kitas 3NF kriterijus, bet jis logiškai yra silpnesnis. Pagal šį kriterijų lentelė tenkina 3NF, jeigu ji neturi tranzityvinių priklausomybių.

Sąryšinės duomenų bazės lentelė $l(L)$ yra 3-joje NF funkcinių sąryšių aibės F atžvilgiu, jei ji yra 1-joje NF, ir nėra neraktinių atributų, kurie tranzityviai priklauso nuo kurio nors rakto. Tranzityvinė priklausomybė atsiranda, kai neraktinis atributas funkcionaliai priklauso nuo vieno ar daugiau neraktinių atributų.

Jei kurioje nors lentelėje yra tranzityvi priklausomybė nuo kurio nors rakto, tai dalį atributų, dalyvaujančių tranzityvioje priklausomybėje, reikia iškelti į kitą lentelę, t. y., jei turim lentelę $l(\boldsymbol{A}, B, C)$, kurioje galioja: $A \rightarrow \{B, C\}$ ir $B \rightarrow C$, tai lentelę l reikia skaidyti į dvi: $l_1(\boldsymbol{A}, B)$ ir $l_2(\boldsymbol{B}, C)$.

XML schemos transformavimą į trečiąją norminę formą galima aprašyti tokiu algoritmu:

1. XML schema $D=(E, A, P, R, r)$ atitinka 2 XNF.
2. Pakartotinai submedis perkeliamas į viršų transformuojant D į D' tol, kol D' pradeda atitikti 3XNF:
   2.1. $D' = D$;
   2.2. Kol D' neatitinka 3XNF:
      2.2.1. Randamos tranzityvinės funkcinės priklausomybės $S_1.\sigma_1 \rightarrow S_1.S_2.S_3.\sigma_2$, ir kitos dvi funkcinės priklausomybės $S_1.\sigma_1 \rightarrow S_1.S_2.\sigma_2$ ir $S_1.S_2.\sigma_2 \rightarrow S_1.S_2.S_3.\sigma_3$ eliminuojamos sukuriant naujus elementus.

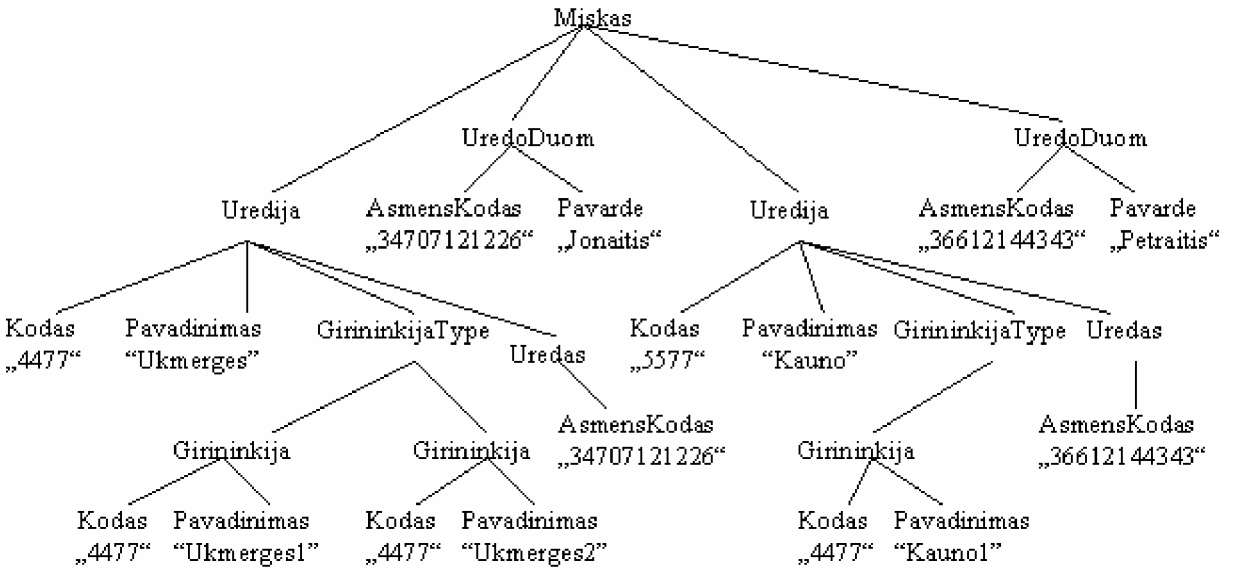
Pagal anksčiau pavaizduotą pavyzdį galima nurodyti tokias funkcines priklausomybes:

F3: {Miskas.Uredija.@Kodas →Miskas.Uredija.Uredas.@AsmensKodas}

F4: {Miskas.Uredija.Uredas.@AsmensKodas →Miskas.Uredija.Uredas.Pavarde.S}

F5: {Miskas.Uredija.@Kodas →Miskas.Uredija.Uredas.Pavarde.S}

F5 funkcinė priklausomybė yra tranzityvinė, todėl ji turi būti pašalinta norint, kad XML schema atitiktų trečiąją norminę formą. Tuo tikslu sukuriamas naujas elementas *UredoDuom*.



2.18 pav. 3NF atitinkanti XML schema

```
<xsd:elemente name="Miškas">
 <xsd:sequence>
  <xsd:element minOccurs="1" maxOccurs="unbounded" name="Uredija" type="UredijaType"/>
  <xsd:element minOccurs="1" maxOccurs="unbounded" name="UredasDuom"
type="UredasDuomType"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UredijaType">
 <xsd:sequence>
  <xsd:element name="Kodas" type="xsd:string"/>
  <xsd:element name="Pavadinimas" type="xsd:string"/>
  <xsd:element minOccurs="0" maxOccurs="unbounded" name="Girininkija"
type="GirininkijaType"/>
  <xsd:element name="Uredas" type="UredasType"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="GirininkijaType">
 <xsd:sequence>
  <xsd:element name="Kodas" type="xsd:string"/>
  <xsd:element name="Pavadinimas" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UredasType">
 <xsd:sequence>
  <xsd:element name="AsmensKodas" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UredasDuommType">
 <xsd:sequence>
  <xsd:element name="AsmensKodas" type="xsd:string"/>
  <xsd:element name="Pavarde" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
```

2.3.5. Ketvirtoji norminė forma

Ketvirtajai norminei formai apibrėžti funkcinio sąryšio sąvokos nepakanka. Tam naudojama daugiareikšmio sąryšio (MV) sąvoka. Daugiareikšmė priklausomybė – sąlyga, užtikrinanti daugiareikšmių atributų nepriklausomybę tarpusavyje [17].

Lentelė atitinka ketvirtąją norminę formą, kai ji tenkina 3NF ir neturi daugiareikšmių priklausomybių. Sąryšinės duomenų bazės lentelė $l(L)$ yra 4-joje NF funkcinių ir daugiareikšmių sąryšių aibės F atžvilgiu, jei kiekvienam MV-sąryšiui $A \rightarrow\rightarrow B$ ($A, B \subseteq L$) yra arba $A \cup B = L$, arba $A \cup B$ yra lentelės L raktas, t. y., jei kiekvienas MV-sąryšis yra trivialus.

Ketvirtoji norminė forma – tai pirmoji forma, kuri susijusi su daugiareikšmėmis priklausomybėmis. RDB lentelė, kurioje bandoma realizuoti keletą „vienas su daug“ ryšių, netenkina ketvirtosios norminės formos reikalavimų. Ji turi būti išskaidyta į kelias lenteles kiekvienam „vienas su daug“ ryšio atvejui. Pavyzdžiui, jei norima priskiri kelis telefono numerius ir kelis elektroninio pašto adresus vienam asmeniui, vienos lentelės realizacija atrodytų taip:

Kontaktai		
	AsmensID	integer
PK	TelefonoNr	string
PK	EPastoAdresas	string

2.19 pav. RDB lentelė prieš normalizavimą

Tokia sąryšinės duomenų bazės lentelė turi daug trūkumų – reikalauja nereikalingos vietos dėl tuščių reikšmių arba duomenų pertekliaus. Be to, tuščios reikšmės pažeidžia duomenų vientisumą, nes visi atributai kartu yra sudėtinis raktas. Akivaizdu, kad atributai *TelefonoNr* ir *EPastoAdresas* yra tarpusavyje nepriklausomi. 4-jai norminei formai RDB lentelė išskaidoma į dvi lenteles:

TelefonoNr		
	AsmensID	Integer
PK	TelefonoNr	String

EpastoAdresas		
	AsmensID	Integer
PK	EpastoAdresas	String

2.20 pav. RDB lentelė po normalizavimo

Tačiau XML atveju tai netinka. XML dokumente daugiareikšmės priklausomybės neiššaukia tokių trūkumų kaip nereikalinga vieta tuščioms reikšmėms ar duomenų perteklius. XML įrašas gali lengvai valdyti nepriklausomas daugiareikšmes funkcines priklausomybes. Ketvirtoji norminė forma praranda savo realiąją prasmę, kai ji pritaikoma XML medžiui.

```
<xsd:complexType name="Kontaktai">
 <xsd:sequence>
  <xsd:element name="AsmensID" type="xsd:integer" />
  <xsd:element name="TelefonoNr" type="xssd:string" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element name="EPastoAdresas" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
```

2.21 pav. Daugiareikšmiai sąryšiai XML schemoje

2.4. Normalizavimo algoritmas

XML schemos dekompocizijos į XNF algoritmas XML schemą pakeičia į XML norminę formą XNF, kuri yra ekvivalenti sąryšinių duomenų bazių BCNF (Boyce–Codd norminei formai) [3].

XML schemos D ir $\sum \subseteq FP(D)$ atveju, $(D,\sum)$ atitinka XML norminę formą XNF, jei kiekvienai netrivialiai funkcinei priklausomybei $X \to p.@l$ ar $X \to p.S \in (D,\sum)^+$ yra atvejis, kai $X \to p \in (D,\sum)^+$.

Taigi *neteisinga* funkcinė priklausomybė yra tokia netriviali funkcinė priklausomybė, kuri netenkina XNF reikalavimų, t. y., $X \to p.@l$ ar $X \to p.S \in (D,\sum)^+$, bet $X \to p \notin (D,\sum)^+$.

Jei abi funkcinės priklausomybės S→p.@a (ar S→p.S) ir S→p priklauso XML dokumento funkcinių priklausomybių aibei, vadinasi nėra tokių dviejų skirtingų viršūnių v_1 ir v_2, kad $lab(v_1)=lab(v_2)=e$ ir $e=last(p)$ su ta pačia p.@a ar p.S reikšme (vadinasi nėra ir duomenų dubliavimosi).

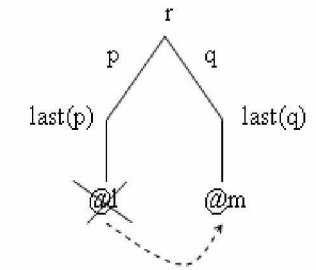
Iš kitos pusės, jei funkcinės priklausomybės $S \to p.@a$ (ar $S \to p.S$) priklauso XML dokumento funkcinių priklausomybių aibei, o $S \to p$ nepriklauso, tai yra dvi skirtingos viršūnės v_1 ir v_2 su ta pačia $p.@a$ ar $p.S$ reikšme (duomenys dubliuojasi).

Funkcinė priklausomybė, kuri pažeidžia XNF, būtinai turi būti netriviali. Trivialios funkcinės priklausomybės egzistavimas, kur p.@a→p.@a priklauso FP aibei, o p.@a→p nepriklauso, nebūtinai parodo, kad XML dokumento struktūra bloga.

XML schemos dekompocizijos į XNF algoritmas pašalina neteisingas funkcines priklausomybes panaudojant naujų elementų tipų kūrimą ir atributų perkėlimą. Šis algoritmas aprašo XML schemos ir FP aibės ∑ transformavimą į naują specifikaciją $(D',\sum')$, kuri atitinka XNF ir palaiko tą pačią informaciją kaip $(D,\sum)$.

2.4.1. Atributų perkėlimas

Pasirenkama XML schema $D=(E, A, P, R, r)$ ir jos funkcinių priklausomybių FP aibė ∑ tokios, kad $(D,\sum)$ turi neteisingų FP $q \to p.@l$, kur $q \in EPaths(D)$. Norint pašalinti neteisingą funkcinę priklausomybę atributas $@l$ perkeliamas iš XML schemos kelio p paskutiniojo elemento $last(p)$ atributų aibės į XML schemos kelio q paskutiniojo elemento $last(q)$ atributų eibę. Atributo $@l$ pavadinimas pakeičiamas į $@m$. Atributų perkėlimo realizacija pavaizduota 2.22 paveiksle:



2.22 pav. Atributo perkėlimas

Naujoji XML schema *D[p.@l:=q.@m]* yra apibrėžiama kaip *(E, A', P, R', r)*, kur:

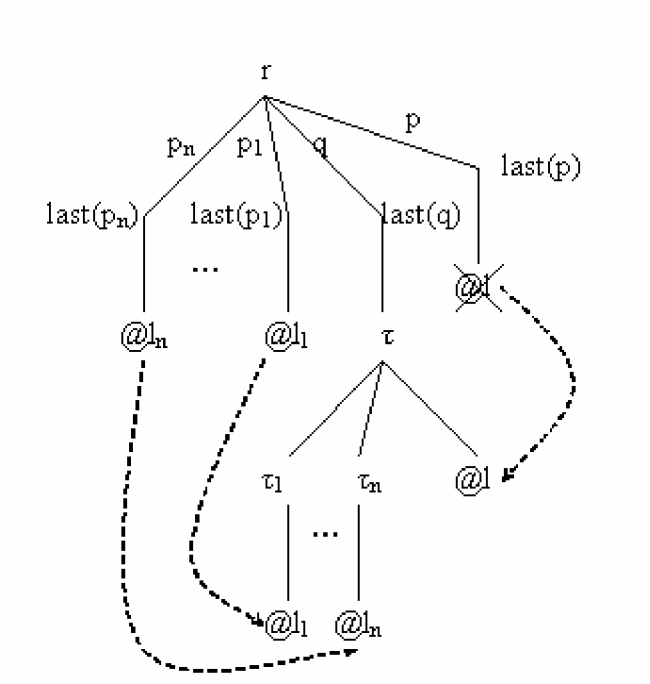
- *A'=A∪{@m}*;
- *R'(last(q))=R(last(q))∪{@m}*;
- *R'(last(p))=R(last(p))-{@l}*.

2.4.2. Naujų elementų tipų sukūrimas

Pasirenkama XML schema *D=(E, A, P, R, r)* ir jos funkcinių priklausomybių FP aibė ∑ tokios, kad *(D,∑)* turi neteisingų FP $\{q,p_1.@l_1,\ldots,p_n.@l_n\}\rightarrow p.@l$, kur $q\in EPaths(D)$ ($n\geq 1$). Norint pašalinti neteisingą funkcinę priklausomybę:

- sukuriamas naujas elemento tipas τ kaip XML schemos kelio q paskutiniojo elemento subelementas;
- sukuriami $\tau_1,\ldots,\tau_n$ kaip naujojo elemento tipo τ elementai;
- atributas $@l$ pašalinamas iš XML schemos kelio p paskutiniojo elemento *last(p)* atributų sąrašo ir padaromas naujojo elemento tipo τ atributu;
- atributai $@l_1,\ldots,@l_n$ padaromi naujai sukurtų elementų $\tau_1,\ldots,\tau_n$ atributais (nepašalinant jų iš $last(p_1),\ldots,last(p_n)$ atributų aibės).

Visi šie neteisingos funkcinės priklausomybės ir naujo elemento tipo sukūrimo žingsniai pavaizduoti 2.23 paveiksle:



2.23 pav. Naujų elemento tipų sukūrimas

2.4.3. Algoritmo aprašas

XML schemos normalizacijos algoritmas susijęs su dviem transformacijomis:

§ Naujų elementų tipų sukūrimu;

§ Atributų perkėlimu.

Algoritmo žingsniai:

1. Jei $(D,\sum)$ atitinka XNF, tai grąžinama $(D,\sum)$, kitu atveju – pereinama į antrą žingsnį.
2. Jei yra neteisingų FP $X \to p.@l$, ir XML schemos kelias $q \in EPaths(D)$ yra toks, kad $q \in X$ ir $q \to X \in (D,\sum)^+$, tai reikia:
   2.1. Pasirinkti naują atributą $@m$;
   2.2. $D:=D[p.@l:=q.@m]$;
   2.3. $\sum:=\sum[p.@l:=q.@m]$;
   2.4. Grįžti į pirmą žingsnį.
3. Pasirinkti neteisingą FP $X \to p.@l$, kur $X=\{q,p_1.@l_1,\ldots,p_n.@l_n\}$ ir:
   3.1. Sukurti naują elemento tipą τ, $\tau_1,\ldots,\tau_n$;
   3.2. $D:=D[p.@l:=q.\tau[\tau_1.@l_1,\ldots,\tau_n.@l_n]]$;
   3.3. $\sum:=\sum[p.@l:=q.\tau[\tau_1.@l_1,\ldots,\tau_n.@l_n]]$;
   3.4. Grįžti į pirmą žingsnį.

Tokiu būtu iš XML dokumento pašalinamos netinkamos funkcinės priklausomybės. Visos identifikuotos netinkamos funkcinės priklausomybės yra netinkamo duomenų išdėstymo elementuose rezultatas.

Jei elementas *B* yra *A* elemento subelementas, tai elemento *A* atributų *@a* ir elemento *B* reikšmės turi apimti informaciją, kuri susijusi tik su *A* ir *B* elementų sąryšiu. Priešingu atveju atsiranda informacijos dubliavimasis:

§ Jei atributas *@a* ir elemento *B* reikšmė apima informaciją, kuri susijusi tik su *B* elementu, tai sukuriamas naujas elementas, kuris palaikys informaciją apie *B* elementą.

§ Jei atributas *@a* ir elemento *B* reikšmė apima informaciją, kuri susijusi tik su *A* elementu, tai jie perkeliami į *A* elementą.

3. XML SCHEMŲ KŪRIMO METODIKOS TAIKYMAS CASE ĮRANKIUOSE

3.1. XML schemų modeliavimas MagicDraw paketu

MagicDraw – UML modeliavimo ir CASE įrankis. Sukurtas verslo ir programinės įrangos analitikams, programuotojams ir dokumentacijai, šis dinamiškas ir įvairiapusis įrankis palengvina objektinių sistemų ir duomenų bazių analizę ir projektavimą. Be to, jis palaiko kodo inžinerijos mechanizmą (Java, C#, C++, WSDL, XML Schema ir CORBA IDL) ir atvirkštinės inžinerijos galimybes. O tai labai naudinga XML schemų modeliavimui UML.

MagicDraw paketo palaikoma kodo inžinerija leidžia egzistuojančias UML klasių diagramas transformuoti į XML schemos kodą ir atvirkščiai:

§ Tiesioginė kodo inžinerija: XML schemos diagrama → XML schemos kodas.

§ Atvirkštinė kodo inžinerija: XML schemos kodas → XML schemos diagrama.

Kad būtų įmanoma pasinaudoti kodo inžinerija, klasių diagrama turi būti pertvarkyta į XML schemos diagramą. Šios diagramos paskirtis – sudaryti XML schemos bylos struktūrą. Diagramos privalumas – greitas ir gana paprastas XML schemos elementų atvaizdavimas. XML schemos elementai yra stereotipais praplėsti UML klasių ir realizacijos diagramų elementai.

Galimi du XML schemos diagramos sudarymo atvejai:

§ XML schema sudaroma iš jau egzistuojančios klasių diagramos. Tokiu atveju klasių diagrama turi būti tik papildoma atitinkamais stereotipais ir jų žymėmis.

§ XML schema sudaroma ne iš egzistuojančios klasių diagramos, o modeliuojama nuo nulio panaudojant stereotipais pažymėtus XML schemos diagramos elementus.

Abiem atvejais galutinis rezultatas – XML schema – atrodys vienodai, bet pats modeliavimo procesas gali skirtis.

3.1.1. XML schemos diagramos elementai

Šioje darbo dalyje pateikiami XML schemos komponentai aprašant jų modeliavimą UML klasių diagramoje.

Atributai

XML atributas modeliuojamas UML klasės atributą pažymėjus <<XSDattribute>> stereotipu.

§ XML atributo pavadinimą (*name*) atitinka UML klasės atributo pavadinimas;

§ XML atributo duomenų tipas (pirminis ar paprasto tipo) – UML atributo tipas;

§ Pradinę reikšmę (*default*) atitinka UML klasės atributui priskirta pradinė reikšmė;

§ Aprašas (*annotation*) – UML atributo aprašas.

XML atributo *ref* reikšmė generuojama iš *ref* ar *refString* pažymėtos reikšmės. Diagramoje gali būti naudojama *ref* arba *name* reikšmė (bet ne abi kartu).

3.1 pav. XML schemos atributų modeliavimas

Elementai

XML elementas siejamas su UML atributu arba ryšio galu panaudojant <<XSDelement>> stereotipą.

§ XML elemento pavadinimas (*name*) – UML atributo ar ryšio galo pavadinimas;

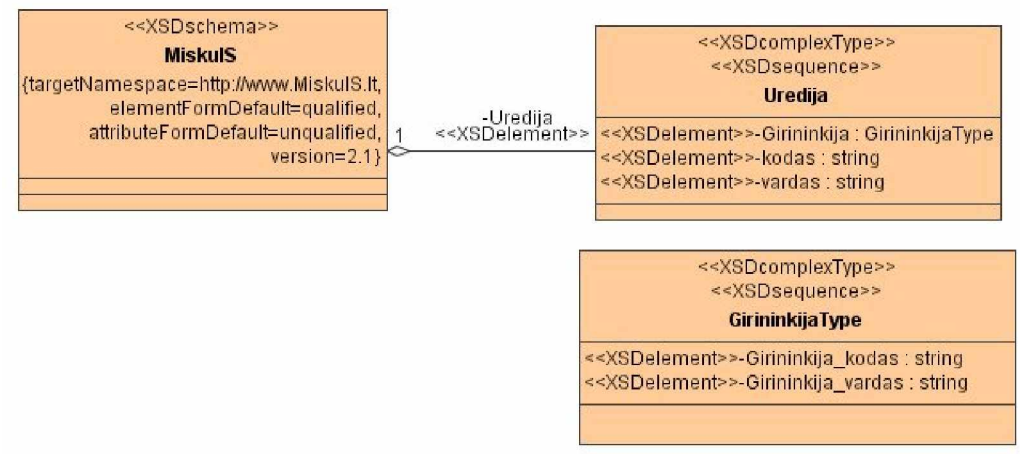
§ Elemento tipas – UML atributo ar ryšio galo tipas;

§ Pradinė reikšmė (*default*) – UML atributui ar ryšio galui priskirta pradinė reikšmė;

§ *maxOccurs* – viršutinė kardinalumo reikšmė. „*unbounded*“ reikšmė atitinka UML diagramoje naudojamą žvaigždutę;

§ *minOccurs* – apatinė kardinalumo reikšmė;

§ Aprašas (*annotation*) – UML atributo ar ryšio galo aprašas.



3.2 pav. XML schemos elementų modeliavimas

Sudėtinis duomenų tipas (*complexType*)

Sudėtinis XML elemento duomenų tipas modeliuojamas UML klasę pažymėjus <<XSDcomplexType>> stereotipu.

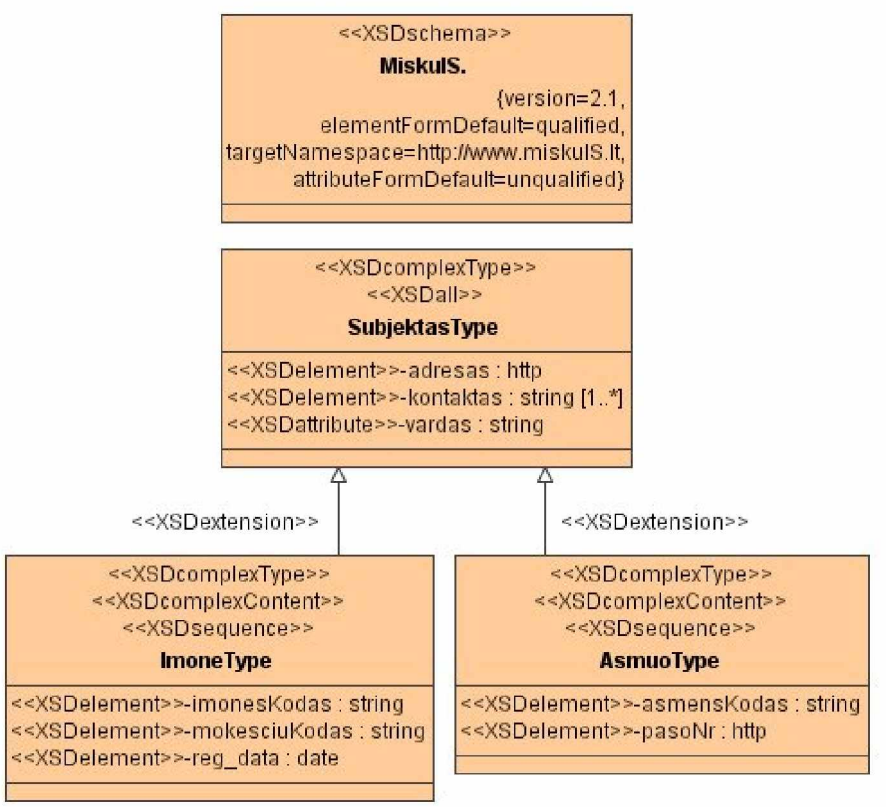
§ Pavadinimas (*name*) – UML klasės pavadinimas;

§ XML elemento atributas – vidinis UML klasės atributas pažymėtas <<XSDattribute>> stereotipu;

§ Atributų grupė (*attributeGroup*) – UML ryšio galas ar UML atributas, kuris aprašytas <<XSDattributeGroup>> stereotipu;

§ Aprašas (*annotation*) – UML klasės parašas.

XML schemos *complexType* komponentą atitinkanti klasė papildomai gali turėti *XSDsimpleContent, XSDcomplexContent, XSDall, XSDchoice, XSDsequence* stereotipus. Jei antrasis stereotipas nenurodytas, pagal nutylėjimą parenkamas <<XSDsequence>> stereotipas. Apibendrinimo ryšiai, jungiantys <<complexType>> stereotipu pažymėtą klasę su kita klase, pažymini stereotipais <<XSDrestriction>> arba <<XSDextension>>. Jei stereotipas nepažymėtas, pagal nutylėjimą parenkamas <<XSDextension>> stereotipas.



3.3 pav. Darinio *complexType* modeliavimas

Atributų grupė (*attributeGroup*)

XML schemos atributų grupė modeliuojama UML klasę pažymėjus <<XSDattributeGroup>> stereotipu.

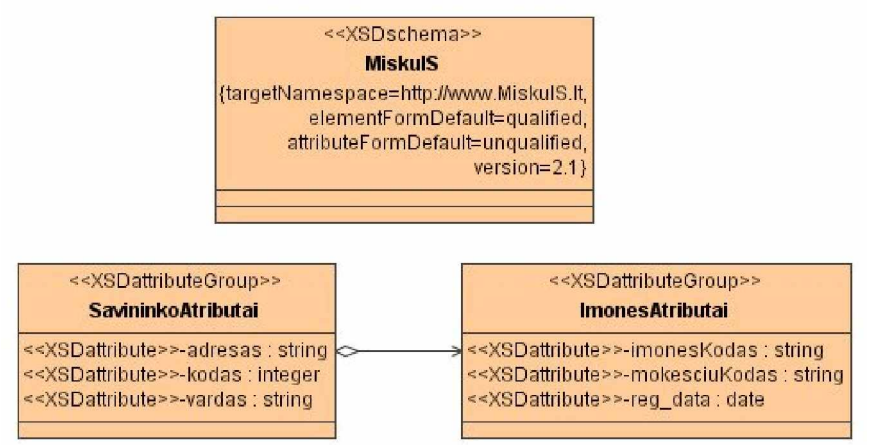
§ Grupės pavadinimas (*name*) – UML klasės pavadinimas;

§ Grupės atributai – vidiniai UML klasės atributai ar UML ryšio galas, kuris pažymimas <<XSDattribute>> stereotipu;

§ *anyAttribute* – vidiniai UML atributai, kurie pažymimi <<XSDanyAttribute>> stereotipais;

§ Aprašas (*annotation*) – UML klasės parašas.

Vidinei atributų grupei apibrėžti visada naudojama tik nuoroda į tą grupę. Tokia nuoroda susiejama su atributu ar ryšio galu, kurio tipas atitinka nurodytą atributų grupę (*attributeGroup*). Priešingas ryšio galas turi būti agregavimo tipo ir kryptinis.



3.4 pav. Atributų grupės modeliavimas

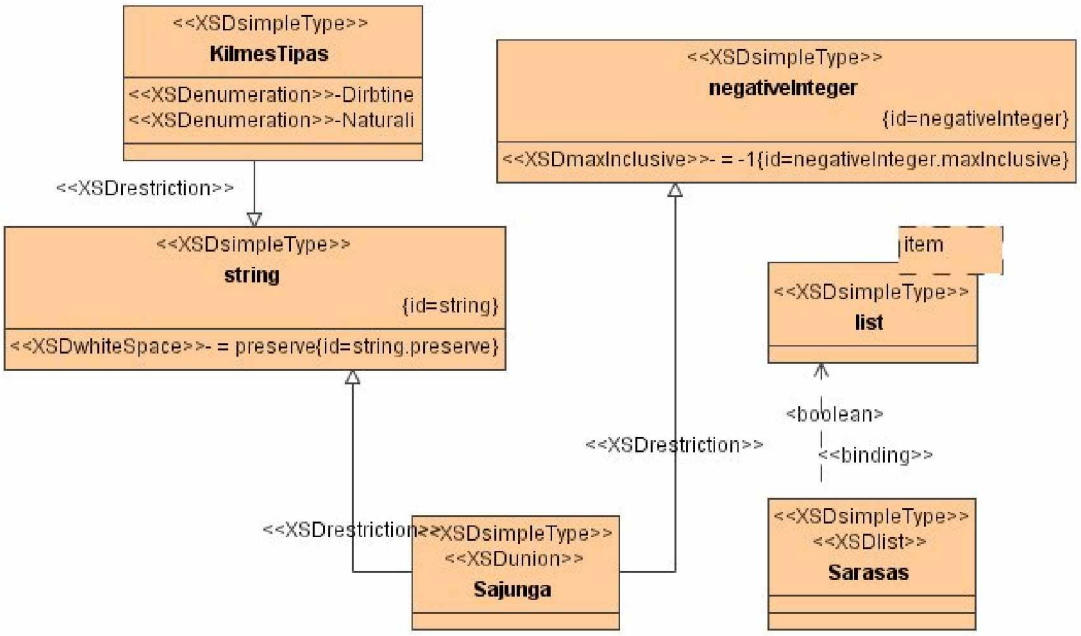
Paprastas duomenų tipas (*simpleType*)

XML schemos *simpleType* komponentas modeliuojamas UML klasę pažymėjus <<XSDsimpleType>> stereotipu. *simpleType* komponentas sudaromas apribojant kitą komponentą (*restriction*), sudarant reikšmių sąrašą (*list*) arba sujungiant du ar daugiau *simpleType* komponentų (*union*).

§ *restriction* – apribojimą apibrėžia apibendrinimo ryšys, kuris sieja klasę ir jos superklasę. Šis apibendrinimo ryšys gali būti pažymėtas (nebūtinai) <<XSDrestriction>> stereotipu. Apribojimo identifikatorius (*ID*) ir aprašas atitinka apibendrinimo ryšio savybes;

§ *list* – UML klasė papildomai pažymima <<XSDlist>> stereotipu;

§ *union* – UML klasė papildomai pažymima <<XSDunion>> stereotipu.



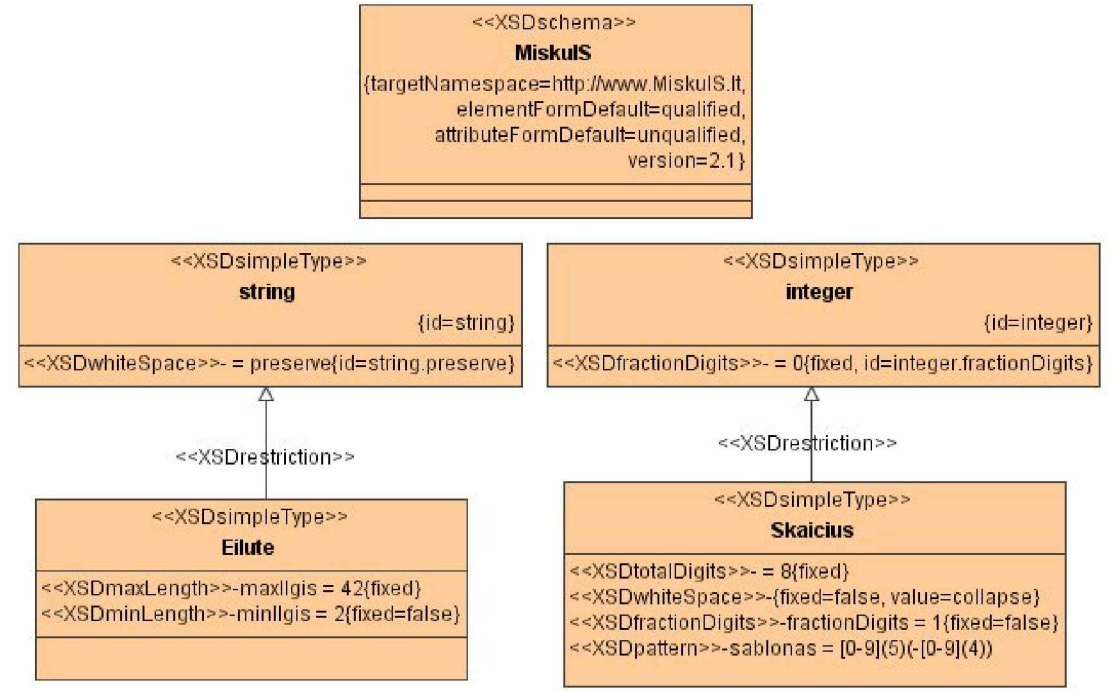
3.5 pav. Darinio *simpleType* modeliavimas

Reikšmės šablonas (*pattern*)

XML atributo ar elemento reikšmės šablonas modeliuojamas UML klasės atributą pažymint <<XSDpattern>> stereotipu. Tokio UML atributo pavadinimas ir tipas nebeturi reikšmės. Svarbiausia yra pradinė atributo reikšmė arba pažymėta reikšmė (*TaggedValue*).

Kiti XML schemoje naudojami atributai (*minExclusive, maxExclusive, minInclusive, maxInclusive, totalDigits, factionDigits, length, minLength, maxLength, whiteSpace*)

Šie XML schemos komponentai modeliuojami iš UML klasės atributų juos pažymint <<XSDminExclusive>>, <<XMLmaxExclusive>>, <<XSDminInclusive>>, <<XSDmaxInclusive>> , <<XSDtotalDigits>>, <<XSDfactionDigits>>, <<XSDlength>>, <<XSDminLength>>, <<XSDmaxLength>> ir <<XSDwhiteSpace>> stereotipais. Tokiu atveju komponentus aprašančio atributo vardas ir tipas jau nebeturi reikšmės. Komponento reikšmė (*value*) – pradinė UML klasės atributo reikšmė.



3.6 pav. XML schemoje naudojamų atributų modeliavimas

Reikšmių išvardijimas (*enumeration*)

Šios konstrukcijos realizavimui UML klasės atributas pažymimas <<XSDenumeration>> stereotipu. Reikšmė – UML klasės atributo pavadinimas.

Unikalumas (*unique*)

Šis XML schemos komponentas modeliuojamas UML klasės atributą pažymint <<XSDunique>> stereotipu.

Raktas (*key*)

Šis XML schemos komponentas modeliuojamas UML klasės atributą pažymint <<XSDkey>> stereotipu:

§ *key* komponento pavadinimas (*name*) – UML klasės atributo pavadinimas;

§ identifikatorius (*id*) – pažymėta atributo reikšmė (*TaggedValue*).

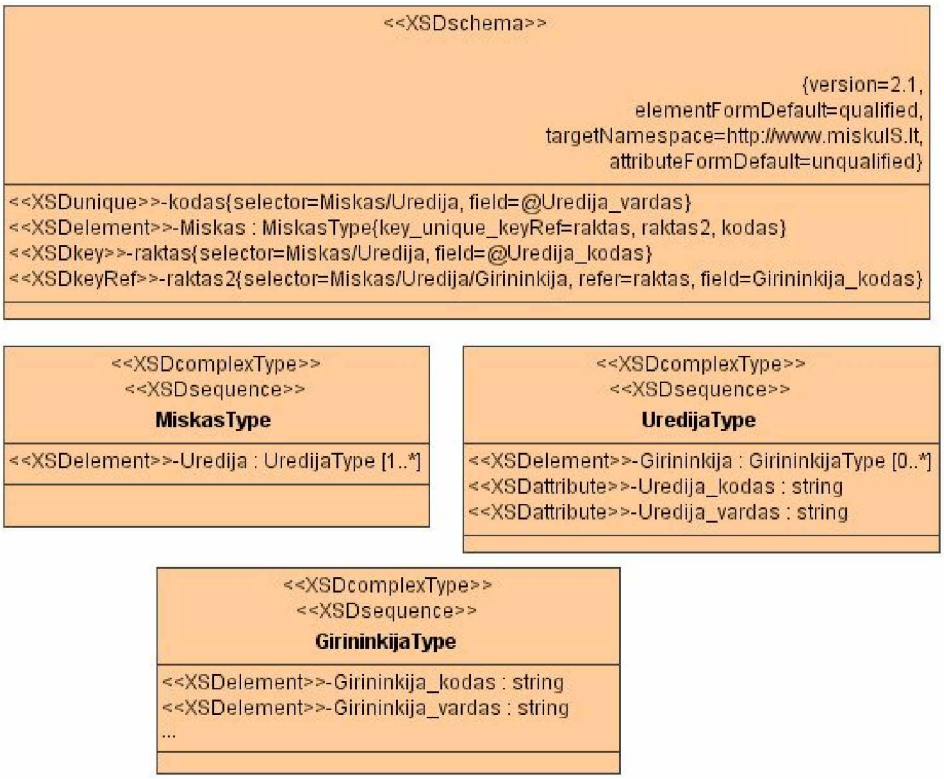
Išorinis raktas (*keyref*)

Šis XML schemos komponentas modeliuojamas UML klasės atributą pažymint <<XSDkeyref>> stereotipu:

§ *refer* – „*refer*“ ar „*referString*“ pažymėtos reikšmės (*TaggedValue*);

§ Pavadinimas (*name*) – klasės atributo pavadinimas;

§ Identifikatorius (*id*) – pažymėta atributo reikšmė.

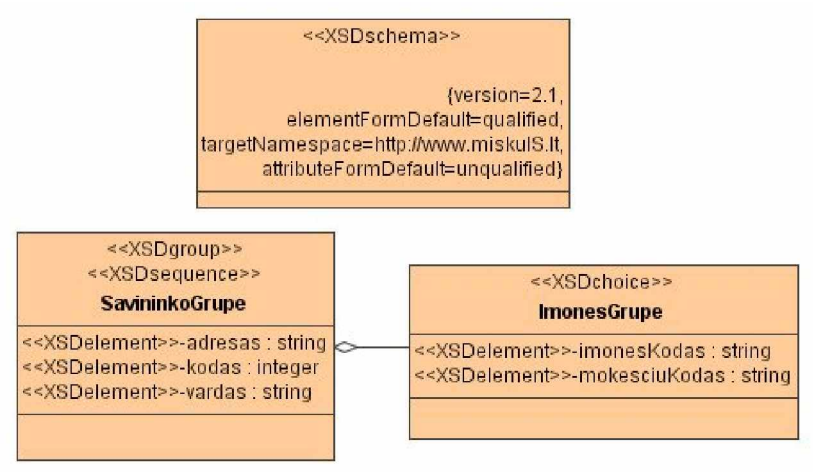


3.7 pav. Darinių *unique, key, keyref, selector* ir *field* modeliavimas

XML schemos **aprašams** (pastaboms) modeliuoti naudojamas stereotipas <<XSDannotation>>.

Grupė (*group*)

Šis XML schemos komponentas modeliuojamas UML klasę pažymint <<XSDgroup>> stereotipu. Ši klasė papildomai gali turėti <<XSDall>>, <<XSDchoice>> ir <<XSDsequence>> stereotipus.



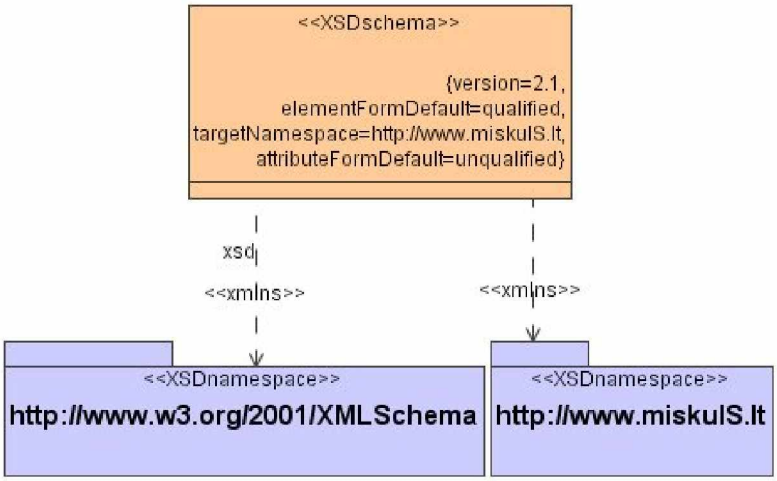
3.8 pav. Darinio *group* modeliavimas

Schema (*schema*)

Šis šakninis XML schemos komponentas modeliuojamas UML klasę pažymint <<XSDschema>> stereotipu. Visi XML schemos globalūs elementai ir atributai atitinka šios klasės atributus. Šios klasės vardas nurodo XML schemos bylos vardą, arba turi būti susietas su komponentu, kuris atitinką šią bylą.

Pavadinimų mechanizmas (*namespace*)

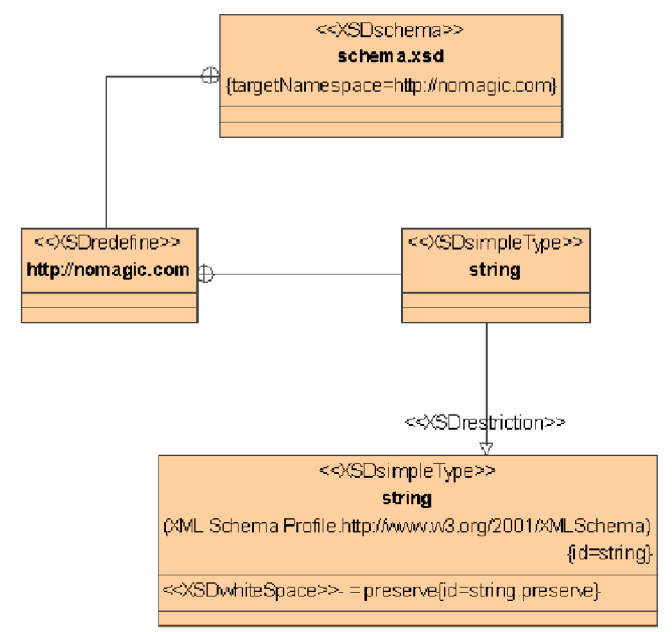
XML schemos pavadinimų mechanizmas modeliuojamas UML paketui priskiriant <<XSDnamespace>> stereotipą.



3.9 pav. Darinių *schema* ir *namespace* modeliavimas

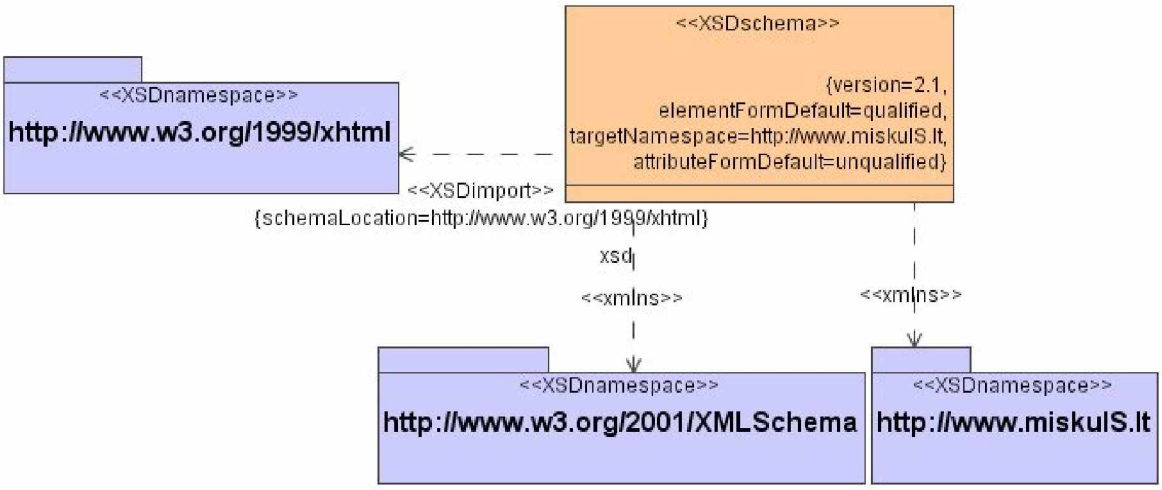
Kiti XML schemos komponentai

Komponentas *redefine* modeliuojamas UML klasei priskiriant <<XSDredefine>> stereotipą. Kitos klasės atitinka naujai apibrėžtus elementus. Kiekvienas naujai apibrėžtas elementas turi būti išvestas iš klasės, pažymėtos <<XSDsimpleType>>, <<XSDcomplexType>>, <<XSDgroup>> ar <<XSDattributeGroup>> stereotipu. Be to, šios klasės vardas apibrėžia XML schemos „*schemaLocation*“ reikšmę.



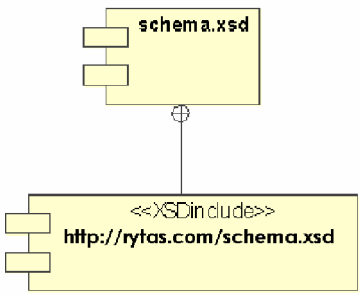
3.10 pav. Darinio *redefine* modeliavimas

Komponentas *import* modeliuojamas ryšiui priskiriant <<XSDimport>> stereotipą. Šis ryšys sieja <<XSDschema>> klasę ir pavadinimų mechanizmą atitinkantį paketą, kuris pažymėtas <<XSDnamespace>> stereotipu.



3.11 pav. Darinio *import* modeliavimas

Komponentas *include* modeliuojamas UML komponentui priskiriant <<XSDinclude>> stereotipą.



3.12 pav. Darinio *include* modeliavimas

3.1.2. XML schemos modeliavimas iš jau egzistuojančios klasių diagramos

Norint pavaizduoti XML schemos modeliavimą iš jau egzistuojančios klasių diagramos, bus panaudota 3.12 paveiksle pateikta klasių diagrama. Kad būtų galima sugeneruoti XML schemos kodą, klasių diagrama turi būti praplėsta stereotipais ir jų žymėmis bei naujais elementais. Klasėms, jų atributams ir ryšių galams stereotipus ir žymes projektuotojas turi priskirti savo nuožiūra remdamasis savo patirtimi, UML profiliu XML schemai ir XML schemų modeliavimo taisyklėmis. Vieną klasių diagramą gali atitikti keletas XML schemos skirtingų struktūrų. Toliau pateiktas tik vienas iš galimų XML schemos modeliavimo iš UML klasių diagramos variantas:

§ Pirmiausia visi klasių diagramos komponentai turi būti įtraukti į <<XSDnamespace>> stereotipu pažymėtą paketą. Šis paketas atitinka XML schemos rezervuotų pavadinimų mechanizmą.

§ Pagrindiniam XML schemos komponento *schema* modeliavimui sukuriama nauja UML klasė su <<XSDschema>> stereotipu. Visi XML schemos globalūs elementai ir atributai atitinka šios klasės atributus. Be to, šios klasės vardas apibrėžia XML schemos bylos vardą.

§ Kadangi XML schema – hierarchinė struktūra, tai klasių diagramoje būtina pasirinkti šakninį elementą – klasę ir ją susieti su <<XSDschema>> klase. Patogiausia pritaikyti 2.2 poskyryje aprašyto algoritmo pirmą ir antrą žingsnius ir, eliminavus klasių diagramoje ciklus, sudaryti hierarchinį vaizdą.

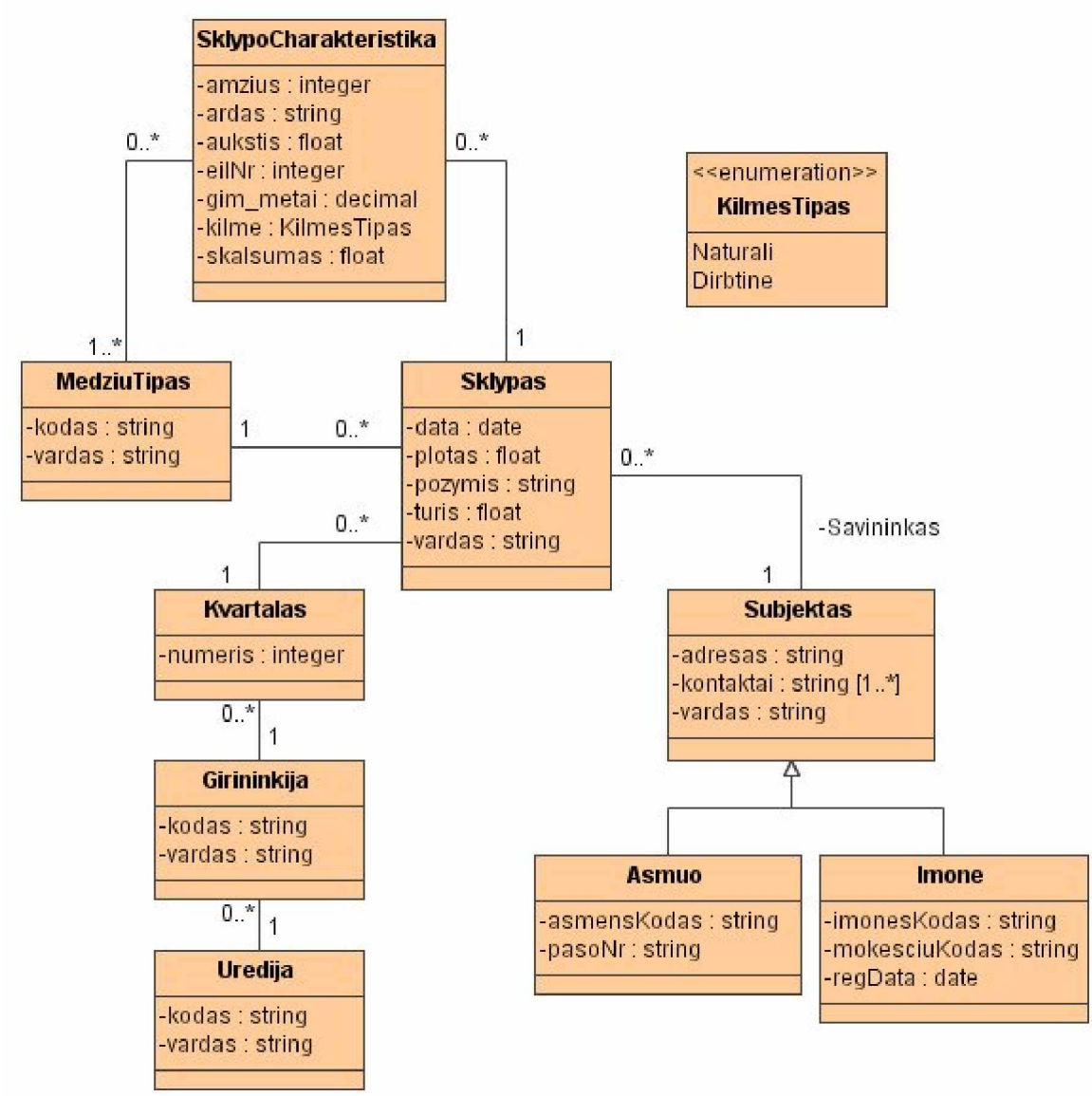
§ MagicDraw paketo XML schemos diagramos nepalaiko asociacijų ryšių tarp klasių. Kiekviena asociacija, jei ją norima atvaizduoti XML schemoje, turi būti pakeista agregavimo ryšiu. Ryšių kardinalumai gali likti tie patys. Be to, kiekvienam ryšiui (kaip ir visiems kitiems klasės komponentams) priskiriamas <<XSDelement>> stereotipas ir vardas.

§ Klasei, kuri pažymėta <<enumeration>> stereotipu, reikia priskirti <<XSDsimpleType>> stereotipą, o jos atributams – <<XSDenumeration>> stereotipus.

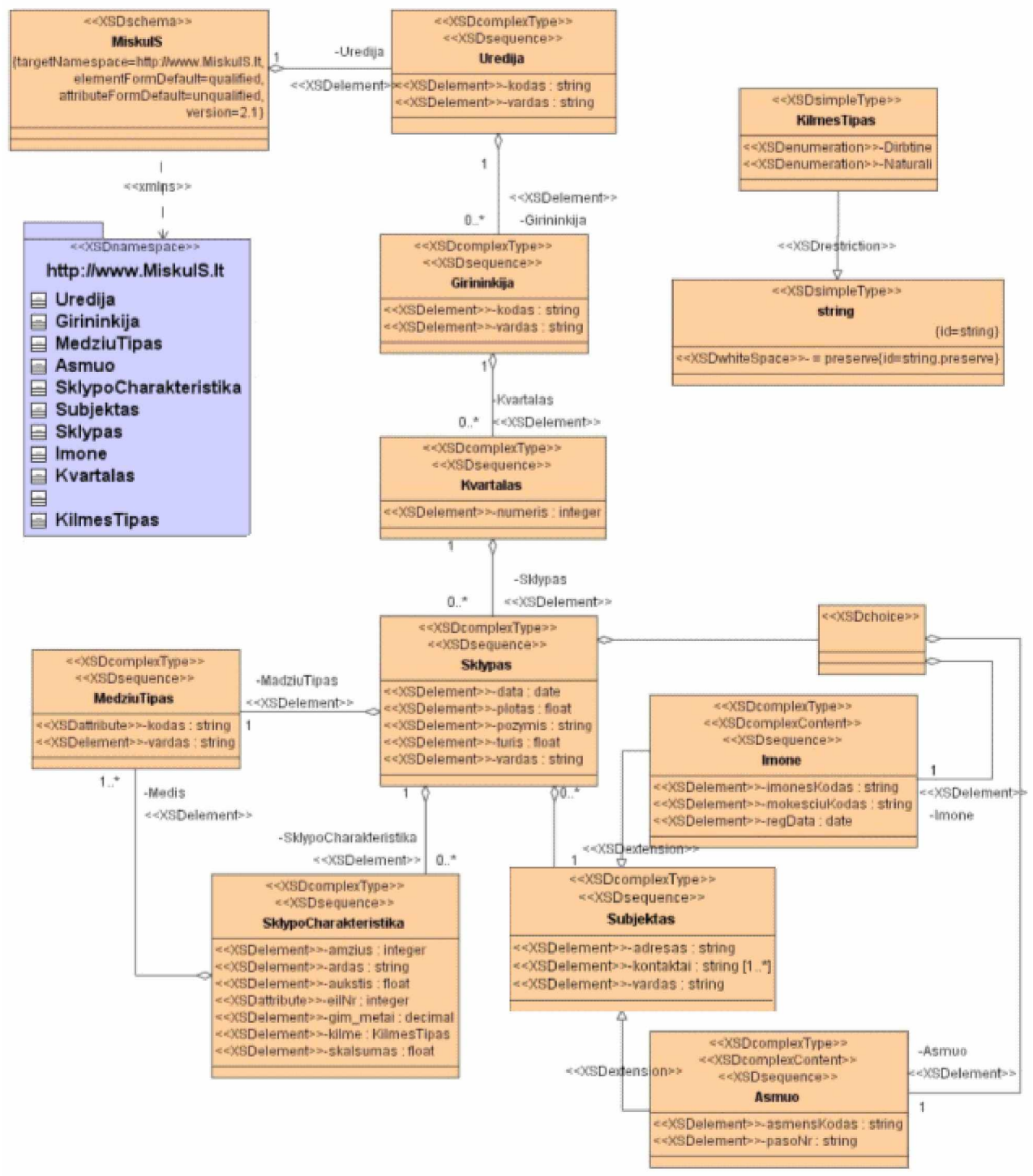
§ Apibendrinimo ryšio atveju superklasei sukuriama bevardė <<XSDchoice>> klasė, kuri agregavimo ryšiais susiejama su subklasėmis.

§ Kiekvienai klasei priskiriami <<XSDcomplexType>> ir <<XSDsequence>> stereotipai.

§ Klasių atributams priskiriami <<XSDelement>> stereotipai.



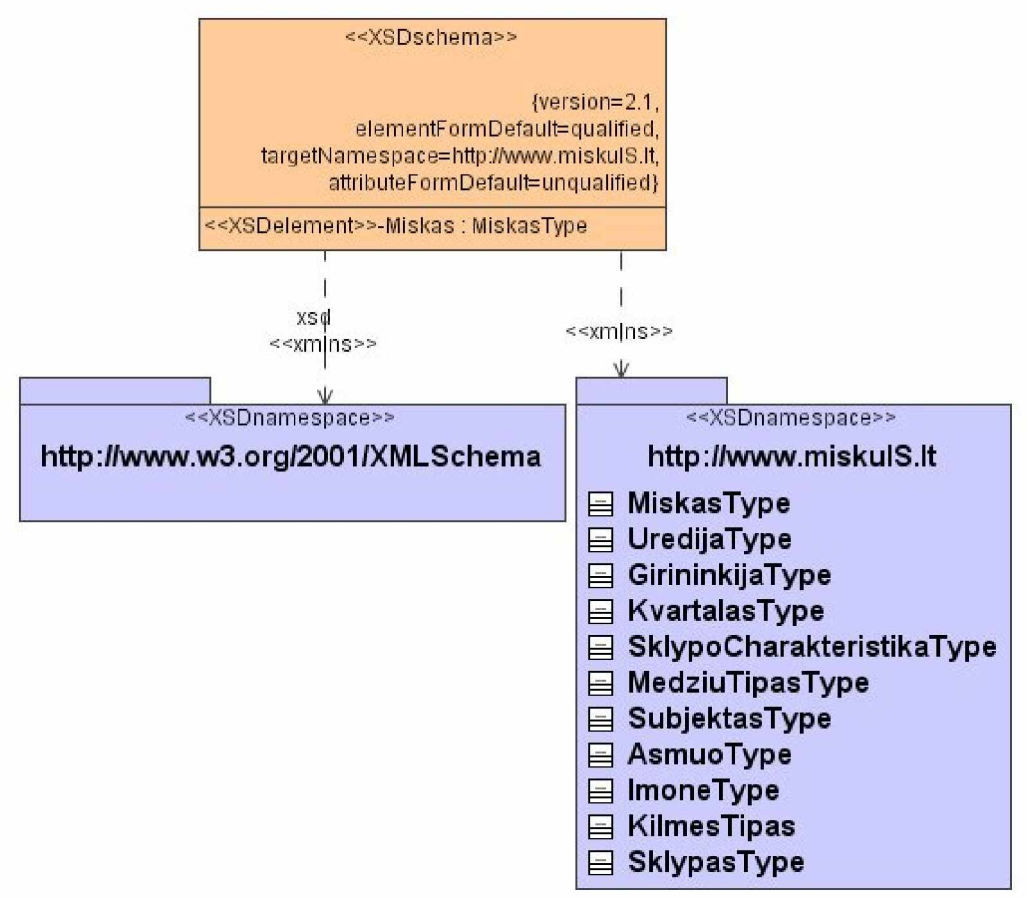
3.13 pav. UML klasių diagrama



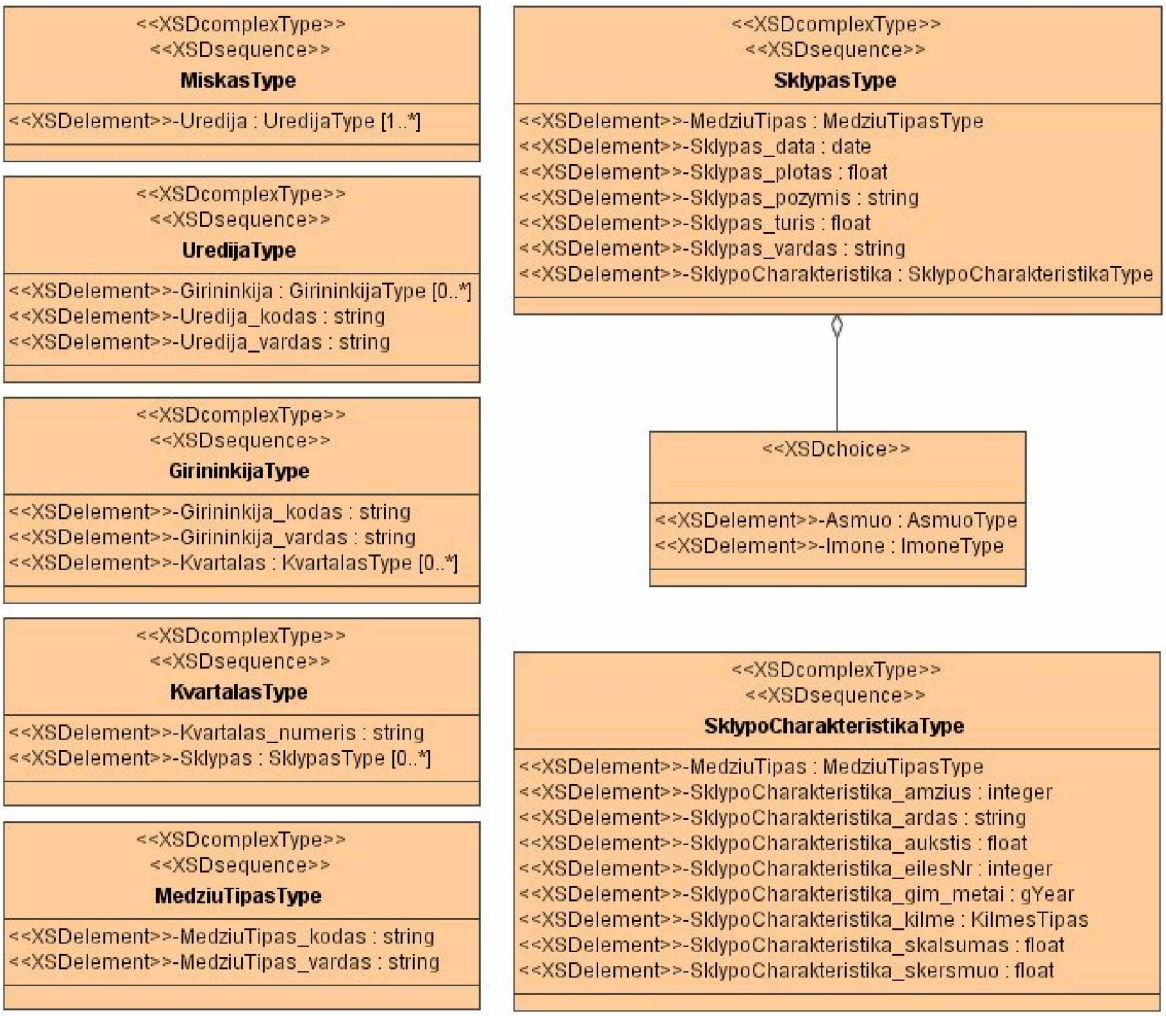
3.14 pav. XML schemos diagrama

3.1.3. XML schemos modeliavimas nuo nulio

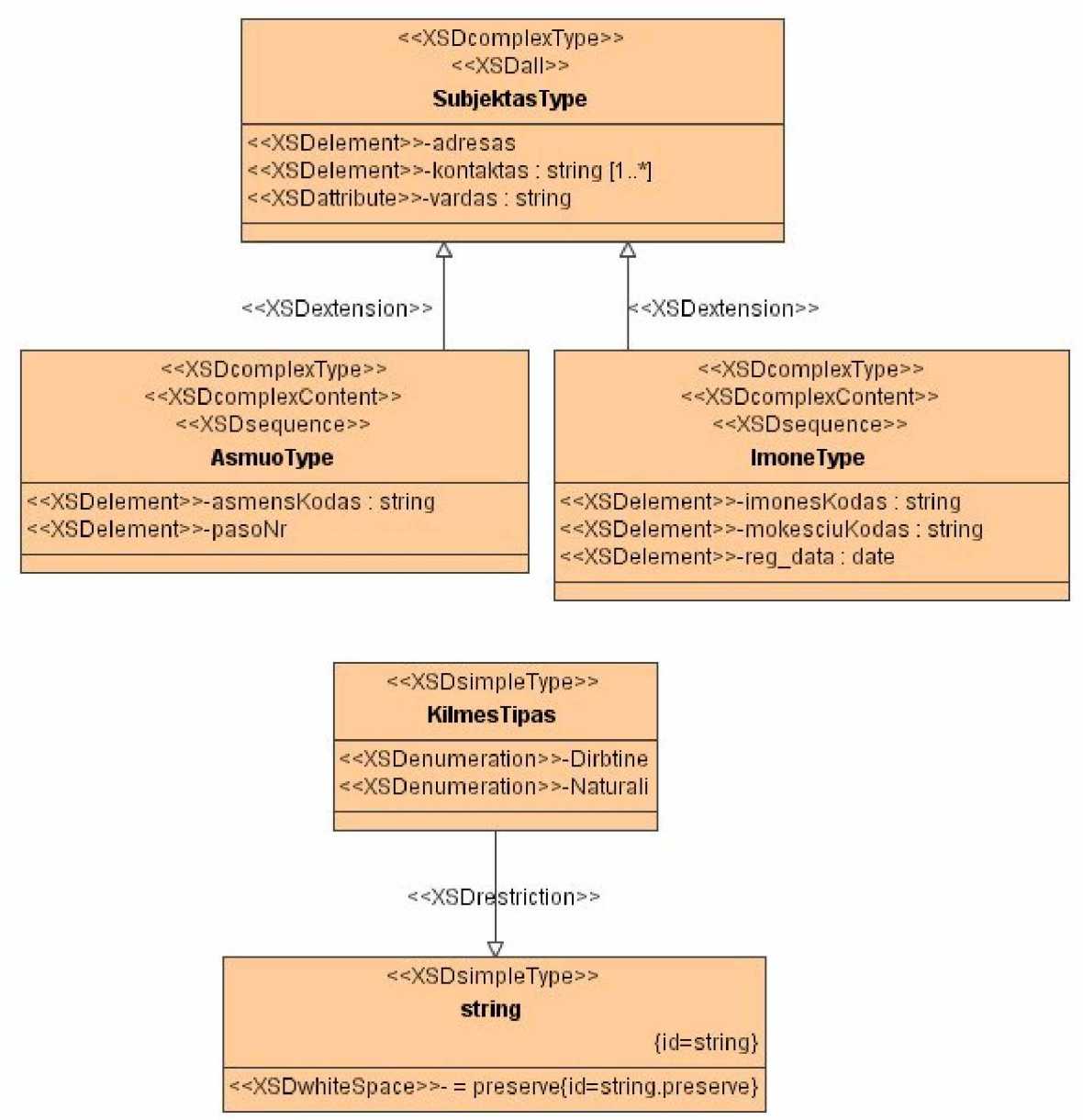
XML schemos modeliavimas nuo nulio gali skirtis nuo jos sudaromo iš egzistuojančios klasių diagramos. XML schema modeliuojama panaudojant stereotipais pažymėtus XML schemos diagramos elementus. Norint pavaizduoti praktinį XML schemos modeliavimą nuo nulio, bus panaudota 3.13 paveiksle pateikta klasių diagrama. Čia pateiktas tik vienas iš XML schemos modeliavimo galimų būdų, nes vieną klasių diagramą gali atitikti kelios XML schemos.



3.15 pav. XML schemos diagrama



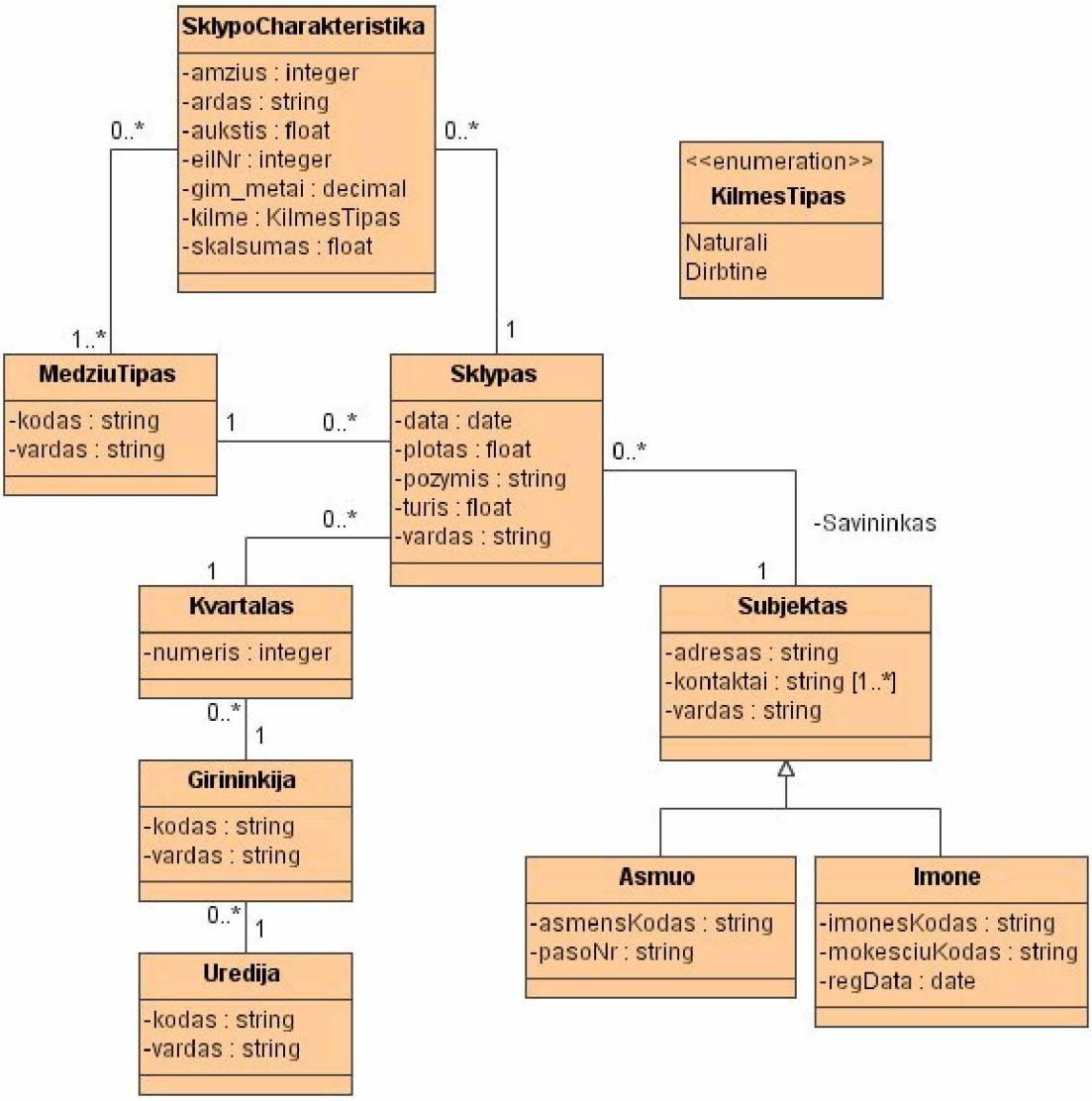
3.16 pav. XML schemos diagrama (tęsinys)



3.17 pav. XML schemos diagrama (tęsinys)

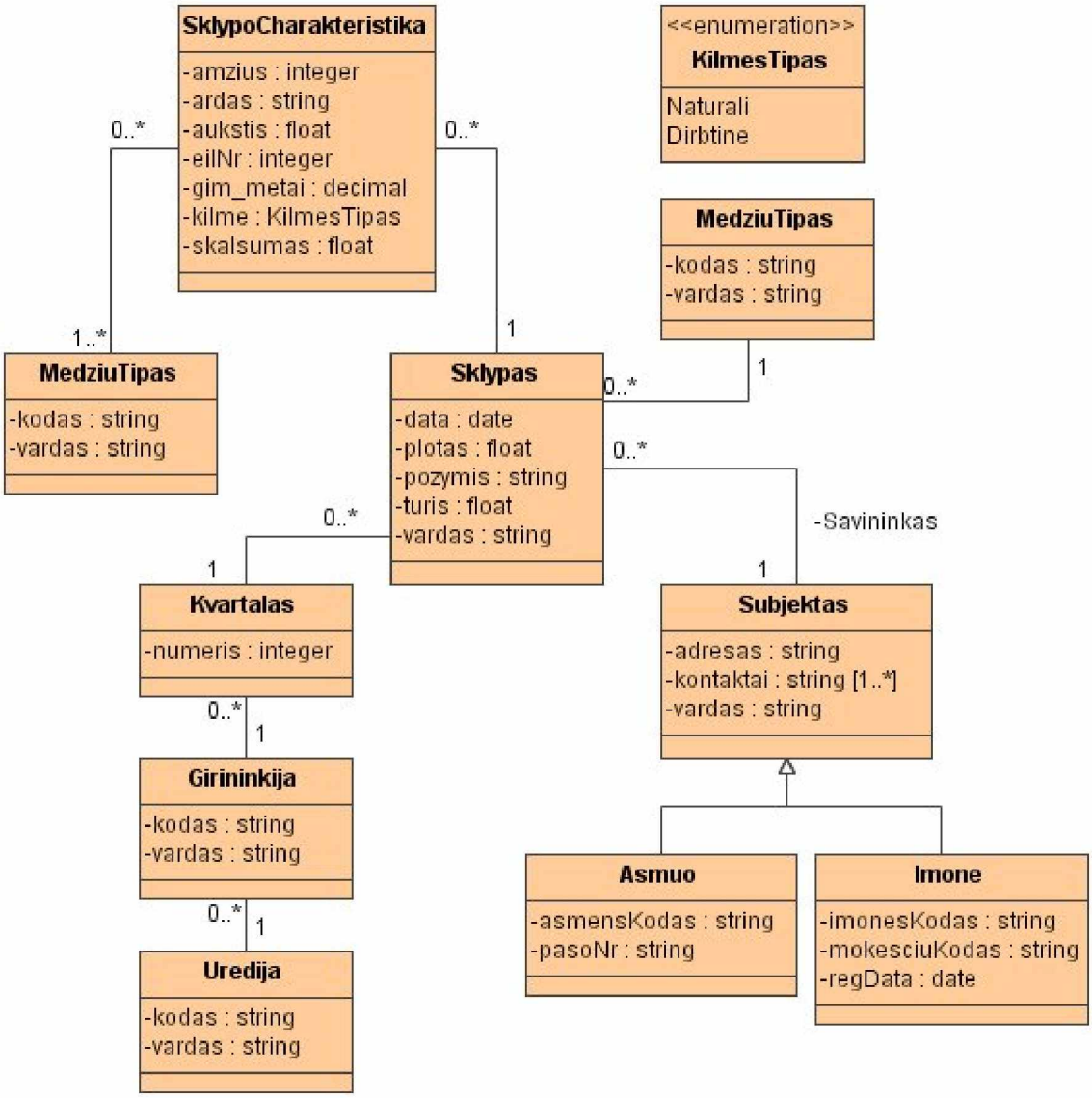
3.2. Praktinis XML schemos generavimo algoritmo pritaikymas

Algoritmo, kuris aprašo XML schemos generavimą iš UML klasių diagramos, visiems žingsniams pavaizduoti naudojama 3.18 paveiksle pavaizduota klasių diagrama:



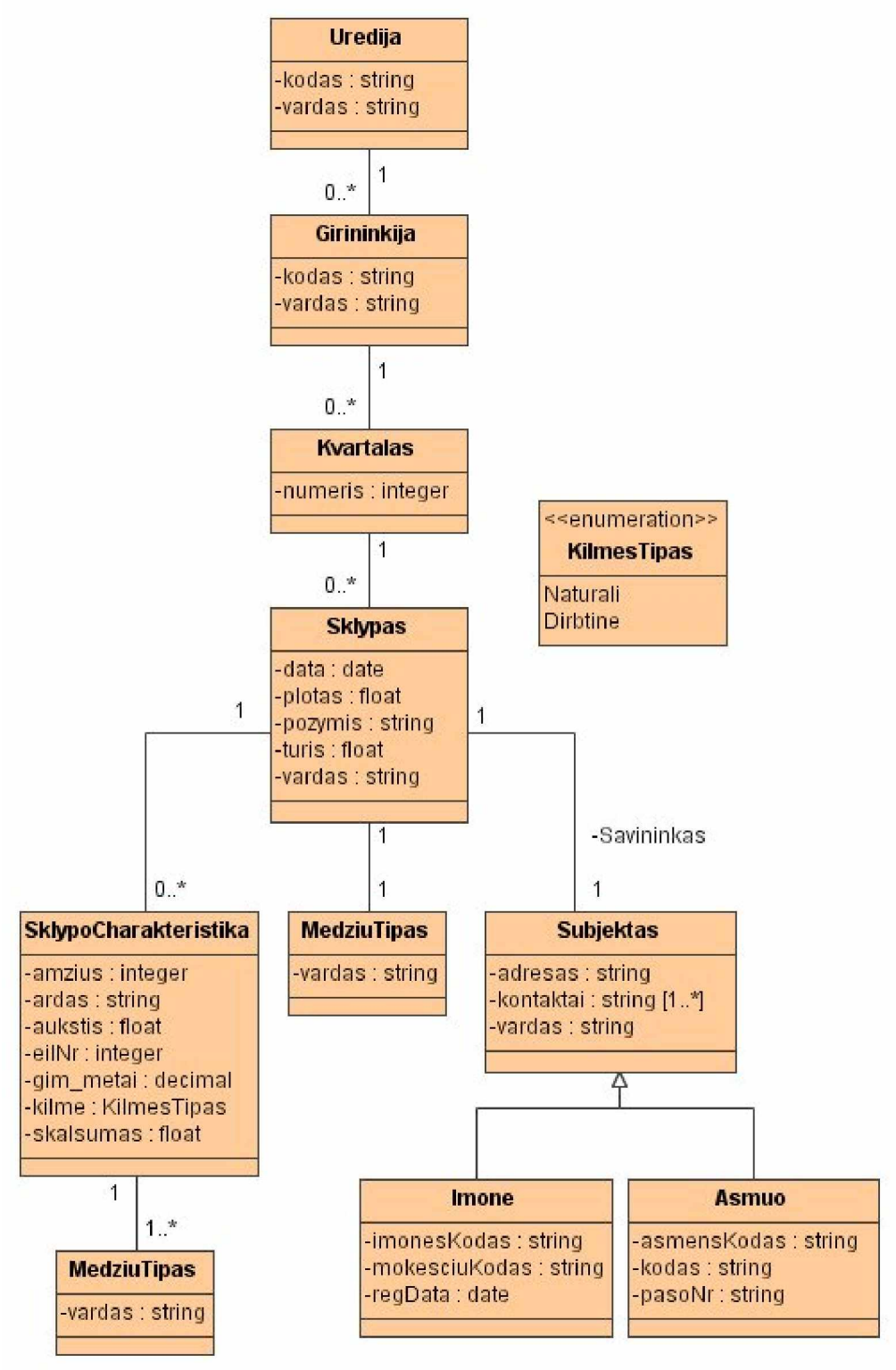
3.18 pav. UML klasių diagrama

Pirmas žingsnis – ciklų klasių diagramoje eliminavimas. Šakniniu elementu pasirinkta klasė „*Uredija*“. Diagramoje matyti, kad klasės „*Sklypas*“, „*SklypoCharakteristika*“ ir „*MedziuTipas*“ sudaro ciklą. Ciklui pašalinti klasė „*MedziuTipas*“ dubliuojama.



3.19 pav. Klasių diagrama po ciklo eliminavimo

Antrasis žingsnis – hierarchinio vaizdo formavimas. Ryšiai „*Sklypas*“ – „*MedziuTipas*“ ir „*Sklypas*“ – „*Subjektas*“ yra N:1 tipo, todėl jie pakeičiami į 1:1 tipo ryšiais. Ryšys „*SklypoCharakteristika*“ – „*MedziuTipas*“, kuris yra N:M tipo, pakeičiamas 1:N tipo ryšiu.



3.20 pav. Klasių diagramos hierarchinis vaizdas

Trečiasis žingsnis – iš sudaryto klasių diagramos hierarchinio vaizdo generuojama XML schema.

```
<xsd:schema version="2.1" xmlns:xsd=http://www.w3.org/2001/XMLSchema
attributeFormDefault="unqualified" elementFormDefault="qualified" >
<xsd:element name="UredijaDoc" type="rootType"/>
<xsd:complexType name="rootType">
  <xsd:sequence>
    <xsd:element minOccurs="1" maxOccurs="unbounded" name="Uredija" type="UredijaType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UredijaType">
  <xsd:sequence>
    <xsd:element name="Uredija.kodas" type="xsd:string"/>
    <xsd:element name="Uredija.vardas" type="xsd:string"/>
    <xsd:element minOccurs="0" maxOccurs="unbounded" name="Girininkija"
type="GirininkijaType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="GirininkijaType">
  <xsd:sequence>
    <xsd:element name="Girininkija.kodas" type="xsd:string"/>
    <xsd:element name="Girininkija.vardas" type="xsd:string"/>
    <xsd:element minOccurs="0" maxOccurs="unbounded" name="Kvartalas"
type="KvartalasType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="KvartalasType">
  <xsd:sequence>
    <xsd:element name="Kvartalas.numeris" type="xsd:string"/>
    <xsd:element minOccurs="0" maxOccurs="unbounded" name="Sklypas" type="SklypasType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SklypasType">
  <xsd:sequence>
    <xsd:element name="Sklypas.vardas" type="xsd:string"/>
    <xsd:element name="Sklypas.plotas" type="xsd:float"/>
    <xsd:element name="Sklypas.turis" type="xsd:float"/>
    <xsd:element name="Sklypas.data" type="xsd:date"/>
    <xsd:element name="Sklypas.pozymis" type="xsd:string"/>
    <xsd:element name="SklypoCharakteristika" type="SklypoCharakteristikaType"/>
    <xsd:element name="MedziuTipas" type="MedziuTipasType"/>
    <xsd:choice>
      <xsd:element name="Asmuo" type="AsmuoType"/>
      <xsd:element name="Imone" type="ImoneType"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SklypoCharakteristikaType">
  <xsd:sequence>
    <xsd:element name="SklypoCharakteristika.eilesNr" type="xsd:integer"/>
    <xsd:element name="SklypoCharakteristika.ardas" type="xsd:string"/>
    <xsd:element name="SklypoCharakteristika.amzius" type="xsd:integer"/>
    <xsd:element name="SklypoCharakteristika.aukstis" type="xsd:float"/>
    <xsd:element name="SklypoCharakteristika.skersmuo" type="xsd:float"/>
    <xsd:element name="SklypoCharakteristika.gim_metai" type="xsd:gYear"/>
    <xsd:element name="SklypoCharakteristika.skalsumas" type="xsd:float"/>
    <xsd:element name="SklypoCharakteristika.kilme" type="KilmesTipas"/>
    <xsd:element name="MedziuTipas" type="MedziuTipasType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="MedziuTipasType">
  <xsd:sequence>
    <xsd:element name="MedziuTipas.kodas" type="xsd:string"/>
    <xsd:element name="MedziuTipas.vardas" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SubjektasType">
  <xsd:sequence>
    <xsd:element name="Subjektas.vardas" type="xsd:string"/>
    <xsd:element name="Subjektas.adresas" type="xsd:string"/>
    <xsd:element name="Subjektas.kontaktas" type="xsd:string"/>
```

```
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AsmuoType">
  <xsd:complexContent>
    <xsd:extension base="SubjektasType">
      <xsd:sequence>
        <xsd:element name="Asmuo.asmensKodas" type="xsd:string"/>
        <xsd:element name="Asmuo.pasoNr" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ImoneType">
  <xsd:complexContent>
      <xsd:extension base="SubjektasType">
        <xsd:sequence>
          <xsd:element name="Imone.imonesKodas" type="xsd:string"/>
          <xsd:element name="Imone.mokesciuKodas" type="xsd:string"/>
          <xsd:element name="Imone.reg_data" type="xsd:date"/>
        </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="KilmesTipas">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Naturali"/>
    <xsd:enumeration value="Dirbtine"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

3.21 pav. Sugeneruota XML schema

IŠVADOS

1. Esminis šio darbo tikslas – ištirti XML schemų kūrimo ir normalizavimo metodologiją
2. Atlikta UML klasių modelio ir XML schemos sąvokų atitikimo analizė parodė, kad yra galimybė pereiti nuo UML klasių diagramos prie XML schemos, tačiau kai kuriais atvejais reikia aprašyti papildomas XML schemų savybes, kurių nėra UML klasių modelyje.
3. UML panaudojimas pagerina XML schemų projektavimo procesą, daro aiškesnę schemų semantiką, padeda išlaikyti schemų suderinamumą su kitais projekto elementais. Pagrindinė problema yra ta, kad UML neapima visų savybių, kurių reikia aprašant XML schemas.
4. Darbe pradedant nuo abstraktaus modeliavimo išanalizuotas perėjimas nuo UML klasių diagramos prie XML schemos. Aprašyti sudėtingesni XML schemos modeliavimo atvejai.
5. Išanalizuotos XML schemų norminės formos parodė, kad jos iš esmės atitinka taikomas sąryšinėms ir objektinėms schemoms. Iš normalizuoto UML modelio sugeneruota XML schema taip pat bus normalizuota.
6. Darbe sudaryti algoritmai XML schemoms generuoti iš UML klasių diagramų, kurie leistų integruoti XML schemų kūrimą į bendrą projektavimo procesą.
7. Sudarytas XML schemų pritaikymo XML norminei formai (XNF) algoritmas, kurį reikėtų taikyti projekte naudojant iš kitur gautas XML schemas.
8. Tai, kad visi aprašyti išplėtimai yra prieinami ir palaikomi UML kalbos, atsispindi šiame darbe atliktame UML CASE įrankio MagicDraw XML schemų modeliavimo galimybių tyrime. Panaudojant šį įrankį, aprašyti algoritmai išbandyti ir patikrinti pavyzdžiais.
9. Pasiūlytų metodų taikymas padėtų užtikrinti projekto darnumą, semantinį aiškumą įvairiems projekto dalyviams.
10. Norminių formų taikymas pagerintų XML dokumentų kokybę, automatinis generavimas pagreitintų projektavimo procesą.

LITERATŪRA

[1] *Altova XMLSpy 2005* [interaktyvus], 2005 [žiūrėta 2005-01-14]. Prieiga per internetą: <http://www.altova.com>

[2] Arenas M., Libkin L. *A Normal Form for XML Documents* [interaktyvus], 2003 [žiūrėta 2004-10-15]. Prieiga per internetą: <http://www.cs.ualberta.ca/~yuan/ courses/692/lectures/xnf>

[3] Arenas M., Libkin L. *An Information-Theoretic Approach to Normal Forms for Relational and XML Data* [interaktyvus], 2003 [žiūrėta 2004-04-10]. Prieiga per internetą: <http://xml.coverpages.org/ArenasPODS2003.pdf>

[4] Carlson D. *Modeling XML Applications with UML: Practical E-Business Applications*, Boston, Addison Wesley, 2001, 315 p.

[5] Carlson D. *Modeling XML Vocabularies with UML* [interaktyvus], 2002 [žiūrėta 2004-04-10]. Priega per internetą: <http://www.xml.com>

[6] Conrad R., Scheffner D., Freytag J. Ch. *XML Conceptual Modeling Using UML* [interaktyvus], 2000 [žiūrėta 2004-04-10]. Prieiga per internetą: <http://www.csd.uch.gr>

[7] *Database normalization* [interaktyvus] , Wikipedia, the free encyclopedia, 2001 [žiūrėta 2004-10-15]. Prieiga per internetą: <http://en.wikipedia.org/wiki/Database_normalization>

[8] Davidson S., Fan W., Hara C., Qin J. *Propagating XML Constraints to Relations* [interaktyvus], 2003 [žiūrėta 2004-04-10]. Prieiga per internetą: <http://www.bell-labs.com>

[9] Ding Ch., Yueguo Ch., Weiwei Ch. *A Survey Stuy on XML Schema Design* [interaktyvus], 2002 [žiūrėta 2004-10-15]. Prieiga per internetą: <http://www.comp.nus.edu.sg/~jaga/reports>

[10] Eckstein R., Eckstein S. *Conceptual Modeling XML Schemata Using UML*, CAiSE'04 FORUM PROCEEDINGS, The 16th CAISE, Riga, Latvia, 2004, 122-131p.

[11] Elmasri R., Fu J., Wu Y.Ch., Hojabri B., Li Ch. *Conceptual Modeling for Customized XML Schemasi*: Conceptual Modeling – ER 2002, LNCS 2503, USA, 2002, 429-443p.

[12] *Introduction to DTD* [interaktyvus], [žiūrėta 2004-04-10]. Prieiga per internetą: <http://www.w3schools.com/dtd>

[13] Krumbein T., Kudrass T. *Rule-Based Generation of XML Schemas from UML Class Diagrams* [interaktyvus], 2003 [žiūrėta 2004-04-10]. Prieiga per internetą: <www.imn.htwk-leipzig.de>

[14] Malik A. *Design XML schemas using UML* [interaktyvus], 2003 [žiūrėta 2004-04-10]. Priega per internetą: <http://www-106.ibm.com/developerworks/library/x-umlschem>

[15] Provost W. *Normalizing XML, Part 1* [interaktyvus], 2002 [žiūrėta 2004-04-10]. Prieiga per internetą: <http://www.xml.com/pub/a/2002/11/13/normalizing.html>

[16] Provost W. *UML For W3C XML Schema Design* [interaktyvus], 2002 [žiūrėta 2004-04-10]. Prieiga per internetą: <http://www.xml.com/pub/a/2002/08/07/wxs_uml.html>

[17] Saxton L. V., Tang X. *Tree Multivalued Dependencies for XML Datasets* [interaktyvus], 2004 [žiūrėta 2004-04-10]. Prieiga per internetą: <www2.cs.uregina.ca/~tang112x>

[18] Skogan D. *UML as a Schema Language for XML based Data Interchange* [interaktyvus], 1999 [žiūrėta 2004-04-10]. Prieiga per internetą: <http://heim.ifi.uio.no/~davids>

[19] *Unified Modeling Language (UML) Tutorial* [interaktyvus], 2001 [žiūrėta 2004-02-02]. Prieiga per internetą: <http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial>

[20] *Users manual version 7.2* [interaktyvus], No Magic Inc. 2003 gruodis, [žiūrėta 2004 09 03]. Prieiga per internetą: <http://www.magicdraw.com>

[21] Vyšnauskaitė R., Nemuraitė L. *Metodika XML schemoms modeliuoti UML*: Informacinės technologijos'05, iš konferencijų ciklo „Lietuvos mokslas ir pramonė", [2005 m. balandžio 29 d., Kaunas]: pranešimų medžiaga. Kaunas, 2005, 151–154 p.

[22] *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C XML working group [interaktyvus], 2000 spalis, [žiūrėta 2004-04-10]. Prieiga per internetą: <http://www.w3.org/TR/2000/REC-xml-20001006>

[23] *XML Schema Part 0: Prime* [interaktyvus], W3C, 2001 [žiūrėta 2004-04-10]. Prieiga per internetą: <http://www.w3.org/TR/xmlschema-0/>

TERMINŲ IR SANTRUMPŲ ŽODYNAS

CASE (*Computer Aided Software Engineering*)	programinės įrangos automatinio projektavimo įrankis
BCNF	Boyce–Codd norminė forma
DTD (*Document Type Definition*)	XML dokumento struktūros aprašas
EER diagrama (*Extended Entity Relationship*)	išplėstinė esybių–sąryšių diagrama
FP	funkcinė priklausomybė
MV-sąryšis	daugiareikšmis sąryšis
NF	norminė forma
OO sistemos (*Object-Oriented Systems*)	objektinės sistemos
RDB (*Relation Database*)	sąryšinė duomenų bazė
UML (*Unified Modeling Language*)	modeliavimo kalba, naudojama objektiniame projektavime
W3C (*World Wide Web Consortium*)	WWW konsorciumas, skirtas pasaulinio internetinio tinklo tobulinimui ir protokolų standartizacijai
XML (*eXtensible Markup Language*)	išplėstinė tekstų kūrimo kalba
XNF	XML norminė forma

1 PRIEDAS. Straipsnis „Metodika XML schemoms modeliuoti UML“

METODIKA XML SCHEMOMS MODELIUOTI UML

Ramutė Vyšnauskaitė, vadovė doc. Lina Nemuraitė

Kauno Technologijos Universitetas, Informacijos sistemų katedra

UML konceptai yra artimesni realaus pasaulio objektams, todėl XML schemų projektavimas turėtų prasidėti nuo konceptualiojo modelio, kuris yra tolesnių projektavimo fazių artefaktų pagrindas. UML naudojimas pagerina XML schemų projektavimo procesą, daro aiškesnę schemų semantiką, padeda išlaikyti schemų suderinamumą su kitais projekto elementais. Straipsnyje išanalizuotas perėjimas nuo UML klasių diagramos prie XML schemos. Išnagrinėtos pagrindinės problemos, kurios susijusios su tuo, kad UML neapima visų savybių, kurių reikia aprašant XML schemas. Pateikti XML schemų normalizavimo principai, panašūs į reliacinių duomenų bazių. Parodyti skirtumai ir panašumai taikant normines formas reliacinėms duomenų bazėms ir XML dokumentams.

1 Įvadas

XML schemos intensyviai naudojamos interneto informacinėse sistemose, kalbose bei standartuose. Kaip ir visos tekstinio pobūdžio notacijos, XML schemų dokumentai dažnai būna gana painūs [6]. UML konceptai yra artimesni realaus pasaulio objektams, todėl XML schemų projektavimas turėtų prasidėti nuo konceptualiojo modelio, kuris yra tolesnių projektavimo fazių artefaktų pagrindas. UML naudojimas pagerina XML schemų projektavimo procesą, daro aiškesnę schemų semantiką, padeda išlaikyti schemų suderinamumą su kitais projekto elementais. Be to, naudinga automatizuoti šį transformacijos procesą taip, kad transformacija būtų išbaigta, t.y., programuotojui nereikėtų jos papildyti.

Normalizacija yra vienas iš būdų, padedančių užtikrinti gerą reliacinių duomenų bazių struktūrą. Tas pačias problemas ir jų pašalinimo metodus galima pritaikyti ir XML dokumentams. Gera struktūra neleidžia atsirasti duomenų pertekliškumui ir nelogiškumui, kurie yra neefektyvaus duomenų naudojimo priežastimi.

2 UML klasių diagramos darinių transformavimas į XML schemą

2.1 UML klasės

UML klasės susiejimas su XML yra gana paprastas, nes kiekviena UML klasė atitinka XML elementą [1]. Kaip ir UML klasė, XML elementas turi vardą, atributus ir ryšius. Todėl klasės atvaizdavimas XML elementu nesukelia problemų.

2.2 UML atributai

UML klasės atributas gali būti transformuojamas į XML dokumento atributą arba elementą. Paprasčiausias būdas yra susieti kiekvieną UML klasės atributą su tą klasę atitinkančio XML elemento sub-elementais. Jei UML klasės atributas aprašo pirminį duomenų tipą (pvz., Integer, String), tai jį geriausia keisti į tą klasę atitinkančio elemento atributą, kuris taip pat aprašo pirminį duomenų tipą ir negali būti aprašytas kaip *complexType* tipo elementas. Yra keletas UML konstrukcijų, kurios neturi atitikmenų XML schemose:

- atributų matomumo bei *frozen* savybė – šių savybių realizacija galima panaudojant *fixed* savybę;
- išvestiniai atributai, kuriuos galima transformuoti panaudojant XML schemos *XPath* išraišką.

2.3 UML ryšiai

Kompozicijos tipo ryšys XML schemoje gali būti realizuotas hierarchiniais ryšiais, nes sub-elementų egzistavimas priklauso nuo elemento egzistavimo, o tai ir atitinka kompozicinio ryšio semantiką. Panašiai kaip ir kompozicinis ryšys XML elementas pagal reikšmes valdo jam priklausančius sub-elementus.

Agregavimo ryšys gali būti išreikštas skirtingomis XML dokumento palaikomomis struktūromis.

All. Jei klasių diagramoje agregavimo ryšiu susietų klasių tvarka nesvarbi, *all* XML schemos konstrukcija labiausiai tinka agregavimo ryšio realizavimui XML dokumente.

Sequence. Jei norima, kad agregavimo ryšys tiksliai būtų transformuojamas į XML *sequence* konstrukciją, klasių diagramoje turėtų būti naudojamas <<*sequence*>> stereotipas [2]. Be to, šiuo atveju būtina tiksliai nustatyti elementų tvarką naudojant papildomus stereotipus.

Choice. Agregavimo ryšio atveju XML schemos *choice* konstrukcijos realizavimui siūloma UML klasių diagramą praplėsti <<*choice*>> stereotipu, nes dažniausiai ši konstrukcija naudojama apibendrinimo ryšio atveju.

Svarbiausias **apibendrinimo** aspektas yra super-klasės atributų paveldėjimas. Vienas iš prieinamų sprendimų atvaizduojant paveldėjimą XML schemose – tai *extension* konstrukcijos panaudojimas. Kiekvienai klasei sukuriamas *complexType* tipo elementas. Sub-klasę atitinkantis *complexType* tipo elementas yra panaudojamas kaip super-klasės *complexType* tipo elemento išplėtimas.

3 XML schemos generavimo iš UML klasių diagramos algoritmas

3.1 Ciklų eliminavimas

Dažnai pasitaiko, kad UML klasių diagramoje klasės ir jų ryšiai sudaro vieną ar kelis ciklus. XML dokumentas yra hierarchinė struktūra, todėl generuojant XML schemą iš UML klasių diagramos pirmiausia reikia pašalinti ciklus ir suformuoti hierarchinį vaizdą. Ciklas pašalinamas dubliuojant atitinkamą klasę taip, kad išliktų jos prieš tai buvę ryšiai su kitomis klasėmis.

3.2 Hierarchinio vaizdo formavimas

Hierarchinis UML klasių diagramos vaizdas formuojamas atsižvelgiant į UML klasių tarpusavio ryšius ir jų kardinalumus:

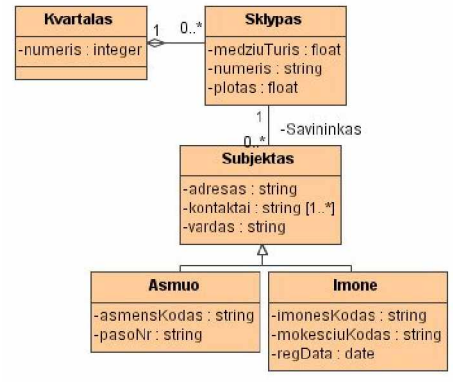
- Jei tarp klasių A ir B egzistuoja 1:N arba 1:1 ryšiai, nieko keisti nereikia. Tokiu atveju XML schemoje B klasė tampa A klasę atitinkančio elemento sub-elementu.
- M:N ryšio atveju šiame algoritmo žingsnyje ryšys pakeičiamas 1:N tipo ryšiu.
- Jei 1:N ar M:N ryšių atveju egzistuoja ryšio klasė, tai ji tampa A klasės elemento sub-elementu.
- Jei klasių tarpusavio ryšys yra N:1, formuojant hierarchinį vaizdą jis pakeičiamas 1:1 tipo ryšiu.

3.3 XML schemos generavimas

UML klasės su atitinkamomis savybėmis transformuojamos į sudėtinio tipo (*complexType*) XML elementus [5]. Klasės vardas tampa elemento vardu, ir pridėjus prie klasės vardo „*Type*“ galūnę – sudėtinio tipo vardu. Klasės atributai tampa šią klasę aprašančio *complexType* sub-elementais, kurių duomenų tipai atitinka atributų duomenų tipus. Sub-elemento vardas sudaromas iš klasės ir atributo vardo, atskiriant juos tašku. Asociacijos, agregavimo ir kompozicijos ryšiai transformuojami į *sequence* XML schemos konstrukcijas. Apibendrinimo ryšio atveju generuojamas *complexType* su *extension* sub-elementu, kurio *base* atributas susiejamas su super-klasės vardu. Atributų ir ryšių kardinalumai išreiškiami *minOccure* ir *maxOccure* atributais.

3.4 Praktinis algoritmo pritaikymas

Visi minėtieji UML plėtiniai XML schemoms modeliuoti yra palaikomi kai kuriuose UML CASE įrankiuose, pavyzdžiui, MagicDraw. Šiame įrankyje galima kurti XML schemų diagramas, kurios apibrėžia tas schemas tenkinančių XML dokumentų struktūrą. XML schemos elementai – tai stereotipais praplėsti UML klasių ir realizacijos diagramų elementai. Taip pat MagicDraw paketas palaiko kodo inžinerijos mechanizmą, kuris leidžia egzistuojančias UML klasių diagramas transformuoti į XML schemos kodą ir atvirkščiai. Praktinis algoritmo pritaikymas pateiktas 1 paveikslėlyje:




```
<xsd:element name="Kvartalas" type="KvartalasType"/>
<xsd:complexType name="KvartalasType"><xsd:sequence>
    <xsd:element minOccurs="0" maxOccurs="unbounded"
name="Sklypas" type="SklypasType"/>
  </xsd:sequence>
  <xsd: attribute name="numeris" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="SklypasType"> <xsd:sequence>
    <xsd:element name="kodas" type="xsd:string"/>
    ...
   <xsd:choice>
      <xsd:element name="Asmuo" type="AsmuoType"/>
      <xsd:element name="Imone" type="ImoneType"/>
    </xsd:choice> </xsd:sequence> </xsd:complexType>
<xsd:complexType name="SubjektasType"> <xsd:sequence>
    <xsd:element name="vardas" type="xsd:string"/>
    ...
  </xsd:sequence> </xsd:complexType>
<xsd:complexType name="AsmuoType"> <xsd:complexContent>
    <xsd:extension base="SubjektasType"><xsd:sequence>
      <xsd:element name="asmensKodas" type="xsd:string"/>
      <xsd:element name="pasoNr" type="xsd:string"/>

</xsd:sequence></xsd:extension></xsd:complexContent></xsd:
complexType>
<xsd:complexType name="ImoneType"> ... </xsd:complexType>
```

1 pav. UML klasių diagrama ir ją atitinkanti XML schema

4 XML schemos normalizavimas

XML medis apibrėžiamas kaip *T = (V, lab, ele, att, root*), kur [3][4]:

- $V \subseteq Vert$ yra baigtinė viršūnių aibė, kur *Vert* – viršūnių identifikatoriai;
- *lab*: $V \to El$ ir *ele*: $V \to Str \cup V$, kur *El* – elemento vardai, o *Str* – tekstinio (*string*) tipo atributų galimos reikšmės;
- *att* yra dalinė funkcija $V \times Att \to Str$, kur *Att* – atributo vardų, kurie prasideda @ simboliu, aibė;
- $root \in V$ – vadinama medžio *T* šaknimi. *paths(T)* galima pažymėti kaip XML medžio *T* kelių aibę.

XML schema apibrėžiama kaip *D = (E, A, P, R, r)*, kur:

- $E \subseteq El$ baigtinė elementų tipų aibė ir $A \subseteq Att$ baigtinė atributų aibė.
- *P – E* susiejimas su elemento tipo aprašais.
- *R – E* susiejimas su *A*, o $r \in E$ ir yra vadinamas šakninio elemento tipu.
- *paths(D)* – visų XML schemos *D* kelių aibę, o *EPaths(D)* aibė visų kelių, kurie baigiasi elemento tipu.

XML medis *T* atitinka XML schemą *D* *(D|=T)*, jei:

- *lab* yra sąryšis nuo *V* prie *E*.
- Kiekvienam $v \in V$, jei *P(lab(v))=S*, tai *ele(v)=[s]*, kur $s \in Str$. Kitaip, *ele(v)=[v_1,...,v_n]*.
- *att* yra dalinė funkcija nuo $V \times A$ prie *Str* tokia, kad kiekvienam $v \in V$ ir $@l \in A$, *att(v,@l)* yra apibrėžta, jei $@l \in R(lab(v))$.
- *lab(root)=r* ir $paths(T) \subseteq paths(D)$.

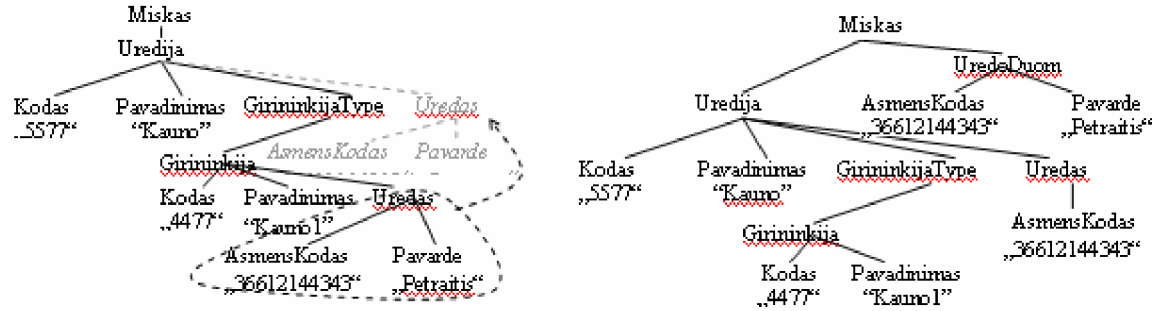
XML schemos *D* funkcinės priklausomybės (*FP*) yra $S1 \rightarrow S2$ formos išraiškos, kur *S1* ir *S2* yra baigtiniai netušti aibės *paths(D)* poaibiai. Visų XML schemos *D* funkcinių priklausomybių *FP* aibė pažymima kaip *FP(D)*.

4.1 XML norminės formos

Pirma norminė forma. Reliacinės duomenų bazės lentelė yra 1-je norminėje formoje (NF), jei visų jos atributų reikšmių aibių elementai yra atomai (nedalomi). XML schema *D* ir medis $T \models D$ atitinka pirmą norminę formą (1XNF) ir reikšmė *v* yra atominė, jei kiekviena viršūnė $v \in V_e$ (t.y. $ele(v) = \perp$ ar $v \in A$). XML schema transformuojama į 1XNF pridedant papildomus sub-elementus ar atributus elementams.

Antra norminė forma. 2NF reikalauja, kad kiekvienas neraktinis atributas būtų funkcionaliai priklausomas nuo pilno rakto (visų raktinių atributų), t.y. negali priklausyti nuo rakto dalies (vieno iš raktinių atributų). Jei XML schemos atveju funkcinei priklausomybei $\{S_1.S_2.S_3.\sigma_2\} \rightarrow S_1.S_2.S_3.\sigma$ egzistuoja kita funkcinė priklausomybė $S_1.\sigma_1 \rightarrow S_1.S_2.S_3.\sigma$, kur $S_1 \neq \perp$ ir $S_1 \neq r$, $\sigma_1, \sigma_2, \sigma$ yra elemento reikšmė ar atributas, tai pirmoji funkcinė priklausomybė vadinama *daline funkcine priklausomybe* (DFP). XML schemos transformavimui į 2-ąją norminę formą galima panaudoti sub-medžio perkėlimą į viršų. Tai atliekama tol, kol nelieka dalinių funkcinių priklausomybių.

Trečia norminė forma (Boiso-Koddo). 3NF reikalauja, kad kiekvienas determinantas būtų raktas. Jei XML schemos atveju funkcinei priklausomybei $S_1.\sigma_1 \rightarrow S_1.S_2.\sigma_2$ egzistuoja kita funkcinė priklausomybė $S_1.S_2.\sigma_2 \rightarrow S_1.S_2.S_3.\sigma_3$, tai funkcinė priklausomybė $S_1.\sigma_1 \rightarrow S_1.S_2.S_3.\sigma_2$ vadinama *tranzityvine funkcine priklausomybe* (TFP). XML schemos transformavimui į trečią norminę formą randamos TFP $S_1.\sigma_1 \rightarrow S_1.S_2.S_3.\sigma_2$ ir kitos dvi FP $S_1.\sigma_1 \rightarrow S_1.S_2.\sigma_2$ ir $S_1.S_2.\sigma_2 \rightarrow S_1.S_2.S_3.\sigma_3$ eliminuojamos sukuriant naujus elementus.



4.

2 pav. XML medžio transformavimas į 2XNF ir 3XNF

Ketvirta norminė forma. 4NF atveju RDB lentelė tenkina 3NF ir neturi daugiareikšmių funkcinių priklausomybių (DFP). Neatitinčios 4-ą NF lentelė turi daug trūkumų – reikalauja nereikalingos vietos dėl tuščių reikšmių arba duomenų pertekliaus. Be to, tuščios reikšmės pažeidžia duomenų vientisumą, nes visi atributai kartu yra sudėtinis raktas. Tačiau XML atveju daugiareikšmės priklausomybės neiššaukia tokių trūkumų kaip nereikalinga vieta tuščioms reikšmėms ar duomenų perteklius. XML įrašas gali lengvai valdyti nepriklausomas DFP. Ketvirtoji norminė forma praranda savo realiąją prasmę, kai ji pritaikoma XML medžiui.

4.2 Normalizavimo algoritmas

XML schemos *D* ir $\sum \subseteq FP(D)$ atveju, $(D,\sum)$ atitinka XML norminę formą XNF, jei kiekvienai netrivialiai FP $X \rightarrow p.@l$ ar $X \rightarrow p.S \in (D,\sum)^+$ yra atvejis, kai $X \rightarrow p \in (D,\sum)^+$. XNF atžvilgiu, *neteisinga* funkcinė priklausomybė yra tokia netriviali FP, kuri netenkina XNF reikalavimų, t.y. netriviali FP $X \rightarrow p.@l$ ar $X \rightarrow p.S \in (D,\sum)^+$, bet $X \rightarrow p \notin (D,\sum)^+$.

XML schemos normalizavimo algoritmas pašalina neteisingas netranzityvines FP panaudojant atributų perkėlimą ir naujų elementų tipų kūrimą.

Pašalinimui neteisingos FP $q \rightarrow p.@l$, kur $q \in EPaths(D)$, atributas $@l$ perkeliamas iš XML schemos kelio p paskutiniojo elemento $last(p)$ atributų aibės į XML schemos kelio q paskutiniojo elemento $last(q)$ atributų eibę ir atributo $@l$ pavadinimas pakeičiamas į $@m$.

Neteisingos FP $\{q,p_1.@l_1,\ldots,p_n.@l_n\} \rightarrow p.@l$, kur $q \in EPaths(D)$ ir $n \geq 1$, pašalinimui:
- sukuriamas naujas elemento tipas τ kaip XML schemos kelio q paskutiniojo elemento sub-elementas,
- sukuriami $\tau_1,\ldots,\tau_n$ kaip naujojo elemento tipo τ elementai,
- atributas $@l$ pašalinamas iš XML schemos kelio p paskutiniojo elemento $last(p)$ atributų sąrašo ir padaromas naujojo elemento tipo τ atributu,
- atributai $@l_1,\ldots,@l_n$ padaromi atitinkamai naujai sukurtų elementų $\tau_1,\ldots,\tau_n$ atributais nepašalinant jų iš $last(p_1),\ldots,last(p_n)$ atributų aibės.

5 Išvados

Kadangi XML schemos dokumentas yra tekstinio pobūdžio ir dažnai gana painus, tai XML schemų projektavimas turėtų prasidėti nuo konceptualiojo modelio. UML naudojimas pagerina XML schemų projektavimo procesą, daro aiškesnę schemų semantiką, padeda išlaikyti schemų suderinamumą su kitais projekto elementais. Be to, straipsnyje išnagrinėtos pagrindinės problemos, kurios susijusios su tuo, kad UML neapima visų savybių, kurių reikia aprašant XML schemas. Aprašyti algoritmai patikrinti pavyzdžiais ir išbandyti MagicDraw paketu. Taip pat staripsnyje pateikti XML schemų normalizavimo principai, bei parodyti skirtumai ir panašumai taikant normines formas reliacinėms duomenų bazėms ir XML dokumentams.

Literatūra

[1] **D. Carlson** Modeling XML Vocabularies with UML, 2002 [žiūrėta 2004-04-10]. Priega per internetą: http://www.xml.com

[2] **D. Carlson** Modeling XML Applications with UML: Practical E-Business Applications, Boston, Addison Wesley, 2001.

[3] **Chen Ding, Chen Yueguo, Cheng Weiwei** A Survey Stuy on XML Schema Design, 2002 [žiūrėta 2004-10-15]. Prieiga per internetą: www.comp.nus.edu.sg/~jaga/reports

[4] **Marcelo Arenas, Leonid Libkin** A Normal Form for XML Documents, 2002 [žiūrėta 2004-10-15]. Prieiga per internetą: www.cs.ualberta.ca/~yuan/ courses/692/lectures/xnf

[5] **Rainer Eckstein, Silke Eckstein** Conceptual Modeling XML Schemata Using UML//CAiSE'04 FORUM PROCEEDINGS, The 16th CAISE, Riga, Latvia, 2004. – 122-131p.

[6] **Ramez Elmasri, Yu-Chi Wu, Babak Hojabri, Charley Li, Jack Fu** Conceptual Modeling for Customized XML Schemas//Conceptual Modeling – ER 2002, LNCS 2503, USA, 2002 - 429-443p.

Modeling metodology XML schema using UML

In this article analyzed the transition from UML class diagrams to XML schema. The main problems, such as - UML does not include all the features required to describe a XML schema - are explored. Also, the principles of XML schema normalization, resemblances and differences between applying normal forms to XML documents and relational data bases are presented.