KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Marius Kėsas

Duomenų atnaujinimo lygiagretumo konfliktų sprendimas prekybos ir klientų aptarnavimo sistemose

Magistro darbas

Darbo vadovas
doc. dr. V. Pilkauskas

Kaunas, 2004

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. habil. dr. H. Pranevičius
2004 05

Duomenų atnaujinimo lygiagretumo konfliktų sprendimas prekybos ir klientų aptarnavimo sistemose

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
dr. J. Mikelionienė
2004 05 20

Vadovas

doc. dr. V. Pilkauskas
2004 05 25

Recenzentas

doc. dr. R.Butleris
2004 05 24

Atliko
IFM-8/2 gr. stud.
M.Kėsas
2004 05 20

Kaunas, 2004

Turinys

1. Įvadas

1.1. Dokumento paskirtis

Vienas svarbiausių objektų *ADO.NET'e* yra *DataSet* objektas. Tai objektas, atskirtas nuo duomenų šaltinio (išskyrus tuos atvejus, kai skaitome ar rašome į duomenų šaltinį), gali būti laikomas komponentu - konteineriu, saugojančiu duomenis atmintyje ir leidžiančiu atlikti veiksmus su jais. Be abejo, *DataSet* objektas turi dar eilę privalumų, tokių kaip duomenų integralumo taisyklių taikymas atmintyje, ne pačiame duomenų šaltinyje, ryšių tarp lentelių sukūrimas atmintyje ir kt. Pristatoma duomenų abstrakcija leidžia naudoti tą patį programinį kodą veiksmams su *DataSet* objektu atlikti, neatsižvelgiant į duomenų šaltinio prigimtį.

Atskirtas nuo duomenų šaltinio objekto modelis nėra naujiena. Šis modelis sėkmingai buvo pristatytas *ADO 2.0* versijoje, kartu su atskirtų įrašų objektais (*disconnected recordsets*). Tačiau duomenų atnaujinimo lygiagretumo konfliktinių situacijų valdyme ir kitur pastebimas lankstumo trūkumas, informacijos stoka. Duomenų atnaujinimo lygiagretumo konfliktinių situacijų valdymas buvo daug resursų reikalaujantis uždavinys.

DataSet objekto atsiradimas ir jo panaudojimas kartu su *DiffGram* formatu išsprendžia daugelį sunkumų, kilusių duomenų atnaujinimo lygiagretumo konfliktinių situacijų valdyme.

Tinkamo *DataSet* objekto panaudojimo negalime įsivaizduoti be *ADO.NET* supratimo. *ADO.NET , DataSet* ir *Recordset* objektų apibrėžimus, skirtumus ir naudojimą aptarsime 2 skyriuje „*ADO.NET* ir *DataSet* prigimtis".

Data update concurrency conflict sollutions in commerce and customer service systems

Summary

There are many benefits to upgrading your data access layer to ADO.NET, most of which involve using the intrinsic DataSet object. The DataSet object is basically a disconnected, in-memory replica of a database. DataSets provide many benefits, but also present a few challenges. Specifically, you can run into problems related to data concurrency exceptions. I've created a simple Windows® Forms customer service application that illustrates the potential pitfalls of this particular problem. I'll walk you through my research and show you ways to overcome the data concurrency issues that arose.

DataSets provide a number of benefits. For example, you gain the ability to enforce rules of integrity in memory rather than at the database level.

The most important benefit of using DataSets, however, is improved performance. Since the DataSet is disconnected from the underlying database, your code will make fewer calls to the database, significantly boosting performance.

As with most performance optimizations, this one comes with a price. Since the DataSet object is disconnected from the underlying database, there is always a chance that the data is out of date.

Since a DataSet doesn't hold live data, but rather a snapshot of live data at the time the DataSet was filled, problems related to data concurrency can occur.

2. *ADO.NET* ir *DataSet* prigimtis

Vienas svarbiausių objektų *ADO.NET'e* yra *DataSet* objektas. *DataSet* objektas sukurtas pagal atskirtų duomenų nuo fizinio duomenų šaltinio modelį su prielaida, kad bus naudojamas pagal nutylėjimą optimistinis lygiagretumas.

Aplinkoje, kur daug vartotojų, optimistinis lygiagretumas įvyksta, kai vartotojo taikomoji programa leidžia skaityti vienu metu tuos pačius duomenis kitiems vartotojams. Pesimistinė lygiagretumo forma priešingai - situacija, kai vartotojo skaitomi, redaguojami duomenys yra neprieinami kitiems vartotojams, šiuo atveju vartotojo duomenų pakeitimai neturi jokios įtakos kitiems vartotojams.

Norint geriau suprasti atskirtų duomenų atnaujinimo lygiagretumo konfliktų sprendimo būdus reikia nuodugniai išsiaiškinti *ADO.NET* prigimtį, *DataSet* ir panašių į jį egzistuojančių objektų skirtumus.

2.1. OLE DB

Kuriant universalų būdą taikomosioms programoms pasiekti duomenų šaltinius buvo sukurta *ODBC (Open Database Connectivity)* sąsaja. *ODBC* turėjo būti bendra, abstrakti taikomųjų programų sąsaja su duomenų šaltiniais, nepriklausomai nuo duomenų šaltinių vidinės architektūros. Kiek vėliau sukuriama *OLE DB* – ją galime laikyti tam tikra *ODBC* modifikacija.

OLE DB - sąsaja, kuri praktiškai įgyvendina teorines *Microsoft Universal Data Access (UDA)* strategijos koncepcijas. *UDA* koncepcija apibrėžia taikomųjų programų galimybę pasiekti bet kokius duomenis naudojant vieną bendrą *COM* sąsają.

OLE DB modelyje išskiriame pagrindines dalis:

- duomenų šaltinio komponentai - *OLE DB* duomenų Tiekėjai (*OLE DB data providers*)
- kliento komponentai, kuriuos naudodami prisijungiame prie duomenų šaltinio, gauname/modifikuojame duomenis - *OLE DB* Vartotojai (*OLE DB consumers*)

Duomenų gavimo iš duomenų šaltinio scenarijų galima nusakyti tokia veiksmų seka:

- klientas prisijungia prie duomenų šaltinio
- sukuriama sesija
- įvykdome komandą (-as) duomenims gauti iš duomenų šaltinio
- gaunami įvykdytos komandos rezultatai

- atsijungiama nuo duomenų šaltinio

OLE DB yra dar labiau nepriklausomas nuo fizines duomenų šaltinio struktūros nei *ODBC*. *OLE DB* komandos gali būti *SQL* sakiniai, tačiau nebūtinai, tai gali būti ir tekstas, kurio sintaksę supranta duomenų šaltinis. Kaip ir *ODBC*, *OLE DB* sukurta turint omenyje *C++* (dėl spartos). Tačiau yra nemažai sistemų, naudojančių *Visual Basic* komponentų kūrimui. Situacijos sprendimui *Microsoft* pristatė *ActiveX® Data Objects (ADO)* biblioteką.

2.2. ActiveX® Data Objects (ADO)

ADO paremta *OLE DB*, tačiau tiesioginiai kreipiniai (per *OLE DB* sąsają) ir kreipiniai per *ADO* vykdomi skirtinga sparta (*C++* spartesnė už *Visual Basic*).

Šioje situacijoje reikia rinktis:

- *C++* ir *OLE DB* - spartesnis darbas
- *Visual Basic* ir *ADO* - paprastesnis, tačiau lėtesnis darbas

Be Tiekėjų ir Vartotojų , *OLE DB* modelyje reiktų išskirti ir trečią elementą - *OLE DB* servisus. *OLE DB s*ervisas yra *COM* komponentas, kuris valdo duomenų eilutes, grąžinamas Vartotojui iš duomenų šaltinio. *ADO* labai priklauso nuo *OLE DB* servisų, nes servisai suteikia papildomų funkcijų: duomenų formavimą, atskirtus nuo duomenų šaltinio įrašus (*disconnected recordsets*)...

Kuriant *COM* naudojančias taikomąsias programas, atskirose srityse sukurta daug technologijų, rekomendacijų. Interneto taikomosiose programose svarbus atskirto duomenų šaltinio modelis.
Trumpai modelį galime apibūdinti taip: Vartotojas ir Tiekėjas visą laiką nėra susijungę. Vartotojui susijungus su duomenų šaltiniu, Vartotojas įvykdo vieną ar daugiau užklausų, gauna duomenis, kuriuos išsaugo lokalioje atmintyje, tada Vartotojas atsijungia nuo duomenų šaltinio. Vartotojas apdoroja gautus duomenis atsijungęs nuo duomenų šaltinio ir, jei reikia, prisijungia prie duomenų šaltinio bei atlieka lokaliai paruoštus duomenų pakeitimus. Šis modelis labai svarbus siekiant spartos ir informacinės sistemos išplečiamumo.

OLE DB architektūros lankstumas sėkmingai leidžia panaudoti *OLE DB* atskirtų duomenų modelyje, tačiau tai nereiškia, kad ši architektūra yra pati geriausia realizuojant atskirtų duomenų

modelį. Tuo labiau, kad patys *ADO* įrašų rinkiniai (*Recordsets*) turi tiek daug atlikti, kad net kyla įtarimų, ar jie tai atlieka efektyviai.

Taip evoliuciniame duomenų priėjimo technologijoje atsirado *ADO.NET*. Kaip ir pats pavadinimas leidžia suprasti, *ADO.NET* yra *ADO* atmaina.

2.3. ADO.NET

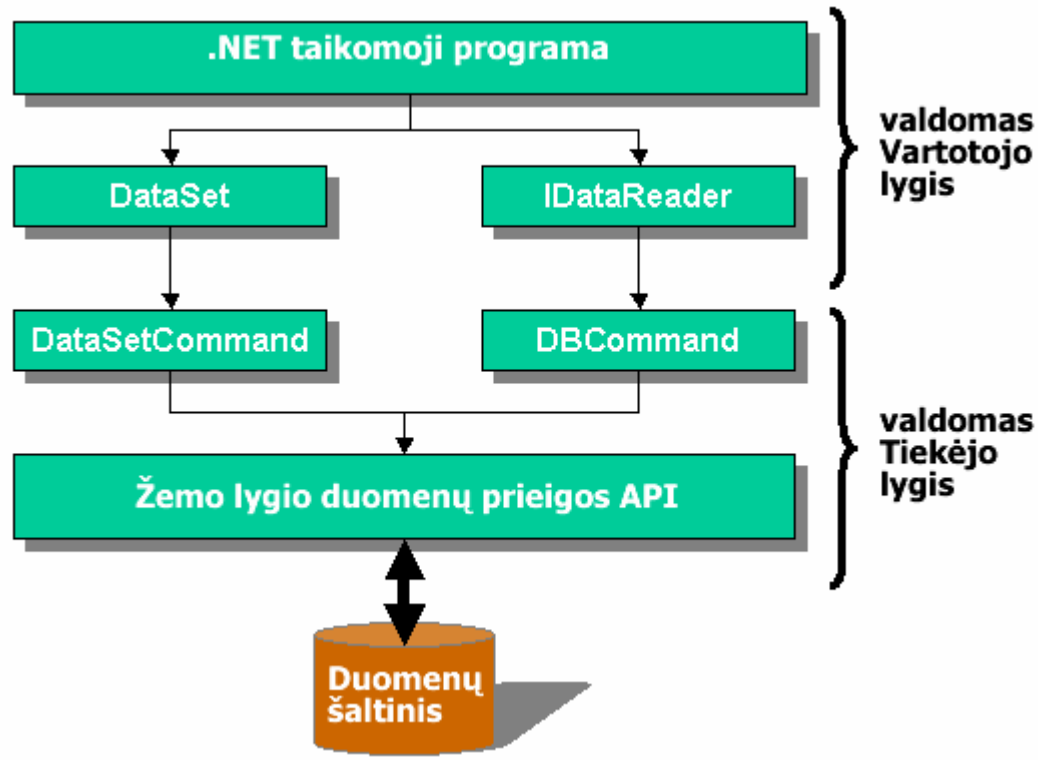
Apibūdinant *ADO.NET* objektinį modelį, būtų sunku tai padaryti nelyginant jo su *ADO* (*ActiveX® Data Objects*).

Pagrindiniai klausimai, kurie iškyla pristatant *ADO.NET* naujoves:

- Atskirtų duomenų modelis? Modelis, kurio dėka padidinamas taikomųjų programų išplečiamumas, sparta?
- *Tai jau buvo pristatyta kartu su ADO 2.0 versija...*
- Glausta sąsaja su *XML*?
- *XML ir ADO bendradarbiauja jau nuo ADO 2.1 versijos...*
- Nepastebima integracija į *.NET Framework'ą*?
- *Framework*'as?

Iš pirmo žvilgsnio atrodo, kad *ADO.NET* nepristato nieko naujo, lyginant su ankstesnėmis *ADO* versijomis, tačiau *framework* yra naujovė, kuri nebuvo pristatyta anksčiau. *ADO.NET* nepastebimai integruojasi sistemos *SDK (software development kit)* aplinkoje (*.NET Framework'e*), ir, be abejo, suteikia papildomo funkcionalumo.

Paprastai interneto taikomoji programa yra atskirta nuo duomenų šaltinio. *.NET Framework'e* yra keletas pagalbinių klasių, skirtų darbui su duomenimis. Šios klasės, ypač *ADO.NET* ir *XML* vardų srityse, suteikia daugiau funkcionalumo apdorojant duomenis. *ADO.NET* ir *XML* praktiškai pakeičia *ADO* ir *OLE DB SDK* - turime bendrą, nuo kalbos nepriklausomą būdą gauti ir keisti duomenis.
.NET'e OLE DB tiekėjai vadinami valdomais tiekėjais (*managed providers*).



1 pav. Valdomų Tiekėjų ir Vartotojų pasiskirstymas ADO.NET'e

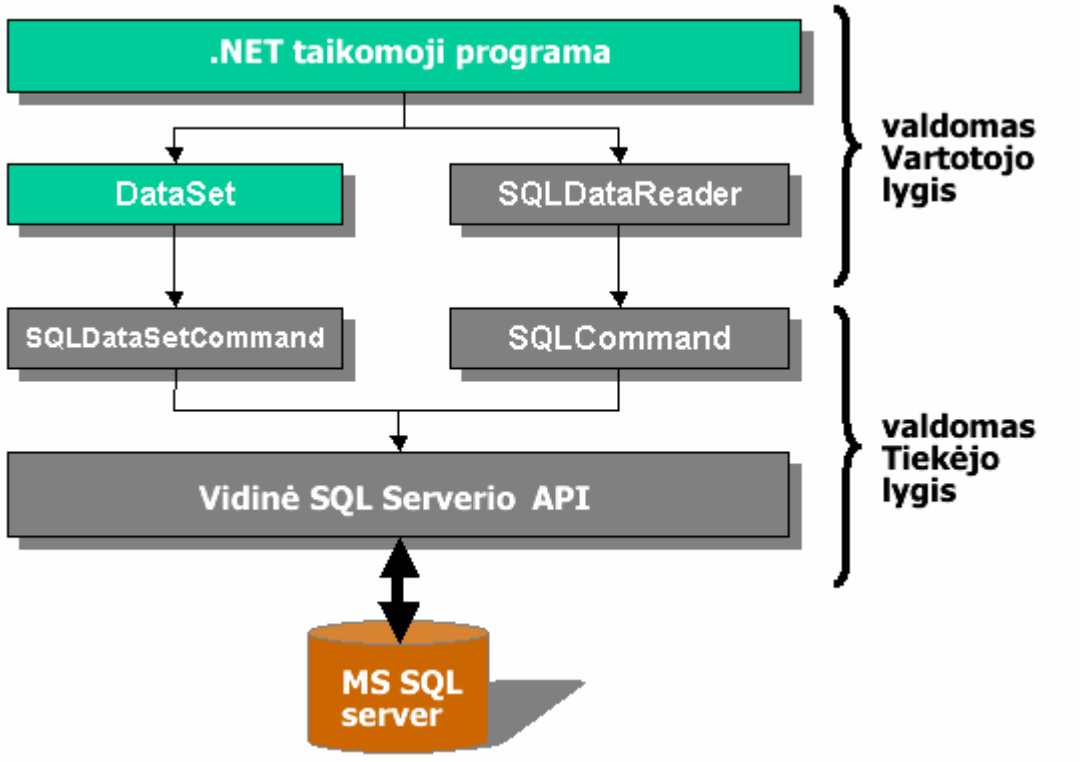
ADO.NET, kaip ir *OLE DB*, turi du lygius: valdomą Tiekėjo ir valdomą Vartotojo lygį.

1 lentelė. OLE DB ir *.NET* duomenų Tiekėjų palyginimas

	OLE DB Tiekėjai	**.NET valdomi Tiekėjai**
Identifikatorius	*COM progID*	Įdiegta *command* klasė
Rezultatai	*Rowset* arba *ADO Recordset*	*DataSet* arba *DataReader*
Atnaujinimas	Tiekėjo specifinės komandos	Tiekėjo specifinės komandos
Formatas	Binarinis	*XML*

Iš esmės, abu variantai panašūs. Tačiau valdomų Tiekėjų panaudojimas daug paprastesnis ir specializuotas. Taip pat pasiekiama geresnė sparta, labiausiai dėl dviejų priežasčių: valdomiems Tiekėjams nereikia naudoti *COM Interop bridge* duomenų gavimui ir pakeitimams atlikti. Be to, valdomi Tiekėjai geriau išnaudoja žinomų duomenų šaltinių veikimo principus (kaip *MS SQL server* atveju, 2 pav.), todėl gauti ir modifikuoti duomenis gali daug greičiau. Tai daro ir *OLE DB* Tiekėjai, tačiau naudojant juos *.NET* aplinkoje, *OLE DB* Tiekėjams maišo jų *COM* prigimtis.

Atliekant tiriamąjį darbą taip pat buvo naudojama *MS SQL server* duomenų bazė. Jos lokalų palaikymą *ADO.NET*'e apibrėžia žemiau esantis paveikslas:



2 pav. *MS SQL* serverio sąsajų, metodų pasiskirstymas *ADO.NET'e*

Kaip matyti 2 paveiksle, *ADO.NET'e* yra netgi atskiri objektai, metodai, savybės optimizuoti *MS SQL server* duomenų bazei.

2.4. DataSet ir Recordset objektų palyginimas

DataSet objektas yra į duomenų bazę panašus duomenų konteineris, saugojamas atmintyje. *DataSet* objektų naudojimas panašus į *ADO Recordset* objektų: mes sukuriame objektus, nustatome kelias argumentų reikšmes , įvykdome specialias komandas, užpildome objektus duomenimis, atliekame veiksmus su duomenimis, ir, jei reikia, atnaujiname duomenis duomenų šaltinyje. Taigi, *DataSet* objektas laikosi panašaus veikimo principo kaip ir *ADO Recordset* objektas.

ADO.NET'e Recordset objekto nėra, nėra ir atgalinio suderinamumo su ankstesnėmis *ADO* versijomis. Vietoj *Recordset* objekto *ADO.NET'e* pristatomas *DataSet* objektas.

ADO Recordset ir *DataSet* objektų panašumai:

- gali būti atskirti nuo duomenų šaltinio
- gali būti serializuoti į *XML*

- gali būti sugeneruoti, nenaudojant jokio duomenų šaltinio
- palaiko duomenų formatavimą
- palaiko duomenų filtravimą
- palaiko duomenų išlyginimą pagal reikšmes
- palaiko duomenų grupinį atnaujinimą

Tačiau šių dviejų objektų architektūra žymiai skiriasi. *DataSet* objektai orentuoti į duomenis, tuo tarpu *Recordset* objektai - į duomenų bazes. *DataSet* objektai veikia atmintyje, nėra jokio sąryšio su duomenų baze, bendriau šnekant, duomenų šaltiniu. *DataSet* objektai galime įsivaizduoti, kaip duomenų lentelių saugyklas atmintyje, ir beje, viename objekte gali būti saugojama daug lentelių, kai tuo tarpu *Recordset* objektas gali turėti tik vieną.

Be to, *DataSet* objektas gali savo duomenis serializuoti į *XML* , išsaugoti turinio byloje nebūtina. *DataSet* turinio atvaizdą *XML* formatu galima gauti bet kada, pasinaudojus atitinkamomis savybėmis ir metodais, tam nereikia naudoti atskiro *XML* DOM objekto.

DataSet objektą sudaro vienas ar daugiau *DataTable* objektų, pastarieji dažniausiai atitinka duomenų bazės lenteles (paprastai *DataSet* objekto duomenų šaltiniai ir yra duomenų bazės). *DataTable* objektai gali turėti vieną ar daugiau *DataRow* objektų.

Būdamas atskirtų duomenų konteineriu, *DataSet* objektas leidžia atlikti veiksmus su duomenimis, kurie yra pirma vykdomi atmintyje ir duomenų šaltiniui tol, kol neiškviečiama atskira duomenų atnaujinimo operacija. Galimi veiksmai *DataSet* objekte su duomenimis: sukūrimas, keitimas, šalinimas. Prieš vykdant duomenų atnaujinimo operaciją (*„Update“*), būtina iš naujo susijungti su *DataSet* objekto duomenų šaltiniu, kur vienu ypu bus atliekama eilė pakeitimų. Dėl šios savybės *DataSet* objekto duomenų atnaujinimo operacija dar vadinama grupiniu atnaujinimu *(„batch update”)*. Sėkmingai atlikus grupinį atnaujinimą, duomenų pakeitimai pačiame *DataSet objekte* yra automatiškai priimami, *DataSet* objekto turinys po pakeitimų vėl atitinka duomenų šaltinį.

Kiekviena duomenų eilutė(*DataRow* objektas), esanti *DataSet* objekte, gali turėti savo pakeitimų istoriją, kuria taikomosios programos gali pasinaudoti. Visa ši informacija yra prarandama serializavus *DataSet objektą* turinį į *ADO.NET* normalinę formą, pasinaudojus *WriteXml* metodu.

Alternatyvi XML schema serializuoti *DataSet* objekto turinį yra *DiffGram*. *WriteXml* metodo *DiffGram* formatas išsaugo *DataSet* objekto turinio būklę (priešingai nei normalinė forma, kuri būklės neišsaugo).

Kadangi *DiffGram* formatas gali išsaugoti duomenų eilučių būklę, jį naudojant galime valdyti duomenų atnaujinimo lygiagretumo situacijas. Žvilgtelėsime į *DiffGram* struktūrą iš arčiau 4-ame skyriuje „*DiffGram* formatas ir jo taikymas“.

3. Prekybos ir klientų aptarnavimo sistema

Pristatomas projektas aprašo prekybos ir klientų aptarnavimo informacinės sistemos realizaciją internete. Galutinis produktas – tai sistema, funkcionuojanti interneto aplinkoje, leidžianti įmonei efektyviau bendrauti su klientais, valdyti dalį verslo procesų, veiklos informaciją saugojanti centralizuotoje duomenų bazėje.

3.1. Projekto tikslas

Projekto tikslas – suprojektuoti, realizuoti ir atiduoti užsakovui naudoti sukurtą produktą. Vis dažniau ir dažniau interneto svetainės kuriamos ne tik tam, „kad būtų", bet siekiant naudos. Internetas įgalina įmones bendrauti su didele auditorija ir kartu su kiekvienu tos auditorijos dalyviu asmeniškai. To neleido nė vienas kitas iki interneto versle naudotas telekomunikacijos kanalas, todėl vis daugiau šalies įmonių ima vertinti ir e-verslo galimybes. Šiandien pasaulyje interneto svetainių tikslai orientuojami į verslą ir formuluojami labai paprastai: veiklos sąnaudų mažinimas; pardavimų didinimas; kuo didesnės lojalių vartotojų bendruomenės sukūrimas; klientų lojalumo ugdymas; pridėtinę vertę turinčių informacinių paslaugų suteikimas partneriams ir klientams – geresnis jų informacijos poreikių patenkinimas; turtingos ir interaktyvios informacinės aplinkos likusioms tikslinėms auditorijoms (ypač žiniasklaidai) suformavimas, ir pan. Tokiu būdu sukuriamas aktyvus virtualus įmonės informacinis pasaulis, tampriai susijęs su vidine ir išorine įmonės veikla ir naudingas ne tik išorei, tačiau ir patiems įmonės darbuotojams. Svarbu nepamiršti, kad tokia svetainė bus funkcionali tik tada, kai su ja bus nuolat dirbama.

Šiuo metu rinkoje tai būtų originalus projektas, labiau surišantis paprastus interneto vartotojus su įmonės veikla. Dabar egzistuojantys interneto projektai yra labiau *B2B* tipo, mažiau orientuoti į paprastus vartotojus.

Pagrindiniai veiksniai, lemiantys *Interneto* svetainės sėkmę:

- aiškus jos egzistavimo tikslas;
- gerai parengta projektinė dalis, įskaitant ir estetinę išvaizdą;
- patogus, lankstus, nedaug kaštų reikalaujantis svetainės valdymas;
- marketingas.

Projekto taikomoji sritis yra labai plati: elektroninis verslas bei vartotojų aptarnavimas internete.

Kurdami interneto svetainę daug dėmesio skiriame kuo efektyvesniam sprendimui pateikti. Prieš kuriant svetainę, reiktų nuspręsti, kokie *Interneto* svetainės tikslai. Jeigu įmonės gaminamos ar

parduodamos prekės, suteikiamos paslaugos yra plataus vartojimo, jų pasirinkimą lemia kokybinės ar techninės charakteristikos, prekei pasirinkti reikalingas aprašymas ar pan., naudinga prekių asortimentą pateikti interneto kataloge. Patogu tuo, kad vartotojas gali ne tik pamatyti prekių vaizdą, paskaityti aprašymus bei charakteristikas, bet ir pasiieškoti prekių pagal norimus kriterijus, gauti rekomendacijas prekėms pasirinkti.

Jau įsibėgėja Lietuvos komercinių bankų planai teikti atsiskaitymo mokėjimo kortelėmis internetu paslaugas. Klientai galės užsisakyti prekių neišeidami iš namų, o tai reiškia, kad keisis ir prekių pardavimo bei reklamos būdai. Kelių pelytės spustelėjimų atstumas nuo konkurentų verčia skirti daugiau dėmesio interneto svetainės marketingui - įtikinti vartotojus tapti klientais.

Ruoštis elektroninės komercijos erai galima pradėti jau dabar - sukuriant užsakymų *Internetu* sistemą, kurią vėliau būtų galima išplėsti atsiskaitymų kreditinėmis kortelėmis moduliu.

Projekto adresatas – įmonės bei organizacijos, suinteresuotos keisti savo vykdomo verslo strategiją internete bei keliančios sau tokias užduotis:

- plėtoti į vartotoją orientuotą kompanijos *Interneto* svetainę (informacijos prieinamumas ir skleidimas / žinių vadyba / globalus pasiekiamumas);
- pergalvoti verslo procesus (nauji ryšiai su vartotoju / kai kurių procesų sinchronizavimas su *Internetu* / elektroninė infrastruktūra);
- kurti naujus prekių/paslaugų realizavimo kanalus (nauji ryšiai su partneriais ir tiekėjais / nauji pajamų šaltiniai / užsakymų vadyba, pardavimai ir pristatymai);
- sukurti elektroninio verslo modelį (didysis pajamų svorio centro kitimas / visi procesai susiję su *Internetu* / elektroninė infrastruktūra 24x7).

Yra trys pagrindiniai suinteresuotų verslininkų tipai:

- tie, kurie jau turi normalų verslą ir *Internetą* nori panaudoti kaip dar vieną pardavimo kanalą;
- tie, kurie svajoja apie išgrynintą virtualų verslą su virtualiu ofisu, leisiantį minimaliomis investicijomis uždirbti milijonus;
- tie, kurie nepatenkinti esamo verslo kokybe ir internete norėtų savo pozicijas sustiprinti gal net radikaliai keičiant verslo modelį.

3.2. Problemos sprendimas pasaulyje

Jungtinių Tautų prekybos ir ekonominės plėtros (*UNCTAD*) ekspertų teigimu, e-komercijos pasaulinės apimtys sėkmingai auga, nors besivystančiose ekonomikos valstybėse jos tebėra nereikšmingos. Būtina e-komercijos plėtros prielaida – *B2B* (verslas verslui) bei *B2C* (verslas klientui) sektoriaus augimas. Ši perspektyva vertinama palankiai, ypač atsižvelgiant į spartų e-komercijos ir *Interneto* naudojimo apimčių augimą. Per praėjusius metus *Interneto* naudojimas pasaulyje išaugo 30 proc., ir trečdalis naujų vartotojų atsirado besivystančiose šalyse. Nors ir lėtesniais tempais, jose auga ir e-komercijos apimtys.

Pagal *B2B* apimčių augimą dominuoja Azijos ir Ramiojo vandenyno regiono šalys. Maždaug 50-70 proc. Lotynų Amerikos bendrovių turi *Interneto* prieigą, ir prognozuojama sparti *B2B* ir *B2C* sektoriaus plėtra. Svarbus vaidmuo tenka tarptautinėms korporacijoms, ypač veikiančiomis automobilių pramonėje, mažmeninės tinklo prekybos srityje klesti automobilių prekyba *Internetu*, be to, toliau populiarėja *Interneto* bankininkystė. Centrinės bei Rytų Europos šalyse tiek *B2B*, tiek *B2C* sektoriai sparčiai auga, nors 2001 m. Europos B2B prekyba *Internete* pasižymėjo gana kukliomis apimtimis – sudarė tik 77 mlrd. *EUR*, arba mažiau nei 1 proc. visų verslo sandorių, tačiau „*Forrester Researc*“ prognozuoja, kad 2006 m. šios apimtys sieks jau 2.2 trl. *EUR* ir sudarys penktadalį – 22 proc. – visų verslo sandorių. Augimas tiek Šiaurės Amerikos, tiek Vakarų Europos rinkose tęsiasi, ypač mažmeninės *Interneto* prekybos srityje. Vien per pirmąjį šių metų ketvirtį JAV B2C sektorius išaugo 19 proc., ir nors B2C dalis bendroje šalies mažmeninės prekybos struktūroje išlieka maža (3 proc.), kai kuriuose sektoriuose *Interneto* pardavimai sudaro net iki 18 proc. visų pardavimų.

Internetu sudaromų sandėrių dalis tarp visų *B2B* pardavimų auga abiejose Atlanto pusėse, ir per ateinančius dvejus-ketverius metus gali pasiekti 20 proc. Be to, prognozuojama, kad JAV ir Europos *Interneto* pardavimų lygio skirtumai iki 2006 m. sumažės 30 proc., nes pastaruoju metu Europos bendrovės gerokai didesnę IT biudžeto dalį skiria e-verslo sprendimams.

Natūraliai, kartu su e-komercijos vystymosi tendencijomis, išsivysčiusiose valstybėse panašūs portalai yra pakankamai populiarūs (www.nike.com, www.yahoo.com, www.amazone.com ir pan.), atneša nemažą dalį pelno įmonėms – yra pakankamai efektyvūs, tuo pačiu yra labiau prižiūrimi, o jų sukūrimas - brangus.

Galima būtų išskirti tris stambias *Interneto* svetainių grupes:

- vizitinės kortelės;
- korporatyvinės svetainės;
- portalai.

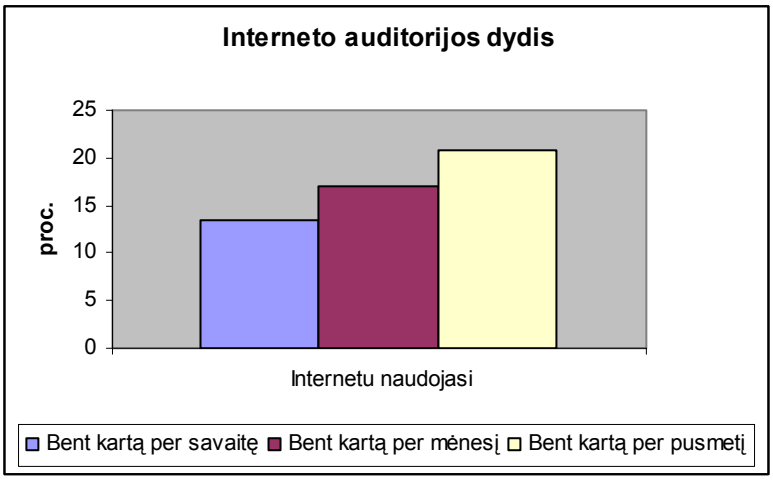
3.3. Situacijos Lietuvoje įvertinimas

Tyrimo, atlikto "SIC *Gallup Media*" 2002m. birželio-rugpjūčio mėnesiais, metu buvo apklausti 1.469 žmonės (apklausiamųjų amžius - 15-74 m.) ir nustatyta, jog *Interneto* vartotojai Lietuvoje pasiskirsto į tokias amžiaus grupes:

2 lentelė. Interneto vartotojų amžių grupės.

Procentai	**Amžius (metai)**
50,2	15-24
24,3	25-34
19,2	35-49
6,3	50-74

Tarptautinei "*Taylor Nelson Sofres*" rinkos tyrimų grupei priklausančios įmonės "SIC *Gallup Media*" duomenimis viso šalyje potencialių *Interneto* vartotojų yra 2,625 mln. žmonių.



3 pav. Interneto auditorijos dydis Lietuvoje 2002m. birželio-rugpjūčio mėn.

Pirmosios komercinės svetainės lietuviškame internete pradėjo kurtis apie 1995-1996 metus, tuo metu, kai savo veiklą pradėjo vienas pirmųjų interneto tiekėjų "Omnitel", bei pirmosios interneto svetainių kūrimo kompanijos "Skaitmeninės komunikacijos" ir "*Infosistema*".

Šiuo metu dauguma didžiausių Lietuvos įmonių ir bendrovių jau turi savo puslapius *Internete* (Iš 200 įmonių, esančių leidinyje "Lietuvos verslo lyderių katalogas 2001 / 2002", sud. UAB "Verslo žinios", savo *Interneto* svetaines turi 121-a., t.y. 60,5 proc.).

Tačiau pats jų buvimo faktas dar negarantuoja tinkamo kompanijos atstovavimo virtualioje erdvėje – daugeliu atvejų menkai išnaudojamos naujų rinkų, klientų ir darbuotojų paieškos galimybės,

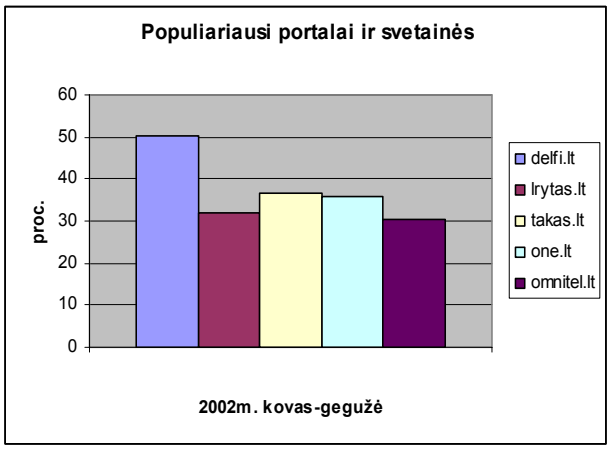
vis dar silpnai internete kuriami ir pristatomi firmos įvaizdis ir prekinis ženklas, dar menkiau išnaudojamos elektroninės prekybos ir interakcijos galimybės.

„*Ebiz.lt*“ atliktas tyrimas (2001 m. gruodį, 2002 m. sausį ir 2002 m. vasarį) parodė, kad Lietuvos įmonių puslapiai kokybiškai smarkiai atsilieka ne tik nuo Vakarų, bet ir nuo Centrinės bei Rytų Europos kompanijų *Interneto* puslapių. Dažnai aiškėja, kad kompanijų marketingo skyriai turi dar daug kur padirbėti: skurdžiai pateikiama kontaktinė ir ryšių su visuomene informacija, tikslūs paslaugų ir produktų aprašymai, bendrovės naujienų prezentacijos. Dažnai stinga specifinės *korporatyvios* ir/ar tiesiogiai klientui skirtos informacijos. Šis tyrimas leidžia prognozuoti, kad artimiausiu metu Lietuvos kompanijos (ypač aktyviai eksportuojančios) turės perimti Vakarų kompanijų *Interneto* puslapiuose jau kurį laiką įdiegtas naujoves – informaciją apie darbo vietas, kompanijos pristatymą potencialiems investuotojams, išsamesnį bei interaktyvų prekių ir paslaugų pristatymą.

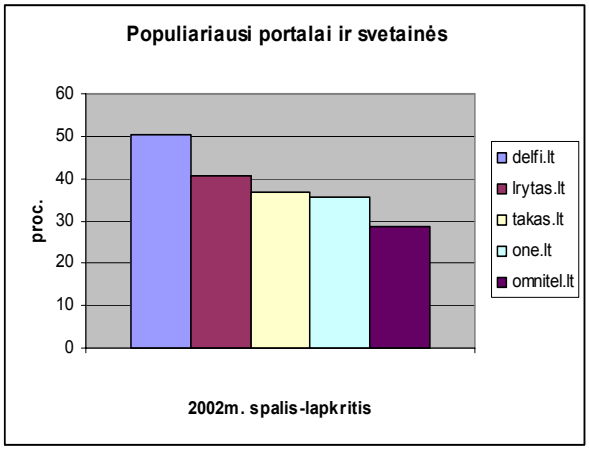
Taigi, menkas *Interneto* potencialo išnaudojimas – vis dar silpnoji Lietuvos stambiojo verslo vieta.

Vis dėl to, “Interaktyviosios komunikacijos sprendimai” 2003 m. pabaigoje atliktas tyrimas rodo, kad dauguma įmonių vadovų (net 54 proc. apklaustųjų) yra patenkinti savo svetaine (užsibrėžtus uždavinius įvykdo net 70 proc. svetainių, vienas iš pagrindinių uždavinių – pagalba verslui), 51 proc. įsitikinę, kad svetainė atsipirko finansiškai.

Tuo tarpu "*SIC Gallup Media*" vykdytų tyrimų duomenimis, populiariausių šalies *Interneto* portalų penketuke yra tik dvi e-komercinę veiklą vykdančių įmonių svetainės (Pav.4, 5).



4 pav. Populiariausi portalai 2002m. kovo-gegužės mėn.



5 pav. Populiariausi portalai 2002m. birželio-rugpjūčio mėn.

Ir nors vis daugiau šalies įmonių ima vertinti ir e-verslo galimybes, “SIC rinkos tyrimai” atlikto tyrimo duomenimis, iš 75 proc. apklaustųjų įmonių, per artimiausius 12 mėn. ketinančių investuoti į IT, tik 10 proc. įmonių artimiausiu metu pradės arba toliau plėtos verslą internete. 16,7 proc. įmonių e-verslą ketina pradėti arba vystyti tuomet, kai atsiras palankios galimybės.

3.4. Sistemos kontekstas

Produktas naudojamas vienas. Nėra sistemos dalis. Produktas yra pats kaip sistema, į kurį įeina kiti komponentai. Visi sistemos nustatymai atliekami per internete, išskyrus tik patį sistemos įdiegimą.

3.5. Vartotojo charakteristikos

Sistema naudojasi kelių tipų vartotojai, kurie gali atlikti skirtingas funkcijas sistemoje. Jie turi mokėti naudotis kompiuteriu, Internetu bei suvokti kas yra e-verslas. Jokios papildomos kompiuterinės žinios jiems nebūtinos.

3.6. Projekto apribojimai

Pagal nutylėjimą imama, kad informacinės sistemos vartotojų skaičius gali siekti ir viršyti 10000 iš kurių 300 nuolatiniai vartotojai. Reikalaujama, kad sistema palaikytų šį krūvį, būtų numatyta reikalui esant sistemą išplėsti - papildyt atskirais moduliais, kitaip didinti sistemos efektyvumą. Informacinė sistema neturi būti sudėtinga – ją turi greitai perprasti nekvalifikuoti darbuotojai.

3.6.1. Vartotojai

Produkto vartotojas – tai sistemos paslaugomis besinaudojantis bet kuris interneto prieigą turintis asmuo, kadangi produktas bus integruotas interneto aplinkoje. Sistema naudojasi dviejų tipų vartotojai (klientai), kurie gali atlikti skirtingas funkcijas. Jie turi mokėti naudotis kompiuteriu, Internetu bei suvokti kas yra e-verslas. Jokios papildomos kompiuterinės žinios jiems nebūtinos.

- Paprastas vartotojas turi galimybę peržiūrėti įmonės teikiamus pasiūlymus, įsigyti naujų prekių/paslaugų, bei susimokėti už jau įsigytas paslaugas ar prekes kurios jam yra suteiktos, tikrinti užsakymo būklę, prekės/paslaugos garantinio aptarnavimo būklę, apmokėjimus už prekes/paslaugas, registruotis naujienoms, siųsti portalo nuorodą draugams. Taip pat siųsti pranešimus ir paklausimus.

- Autorizuotas vartotojas turi galimybę įvesti naują prekę arba paslaugą bei ją redaguoti, atsiųsti prekės nuotrauką per interneto sąsają ir talpinti *serveryje*, nustatyti standartinius prekės ar paslaugos parametrus bei siųsti naujienas vartotojams. Taip pat turi visas paprasto vartotojo teises.

3.6.2. Apribojimai sprendimui

Kadangi ši projektas yra pirmasis žingsnis užsakovo įmonės elektroninės komercijos ir aptarnavimo srityje, todėl apribojimų kuriamai programinei įrangai nedaug.

Pagrindiniai reikalavimai:

- Kuriama Informacinės Sistemos kliento pusės programinė įranga turi veikti *Windows 98SE* ir vėl. operacinėse sistemose.
- Informacine Sistema vienu metu gali naudotis daugiau nei vienas vartotojas.

3.6.3. Diegimo aplinka

- Informacinė sistema bus diegiama Windows *2003 Server* OS terpėje.

- Bus taikoma *ASP.NET* (*Active Server Pages*) technologija.
- Naudojama *MS SQL 2000 server* DBVS.
- Reikalingas lokalus bei *FTP* priėjimas prie *serverio*.

Produktas naudojamas vienas. Nėra egzistuojančios sistemos dalis. Produktas yra pats kaip sistema, į kurį įeina kiti komponentai. Visi sistemos nustatymai atliekami per *INTERNETO* sąsają, išskyrus tik patį sistemos įdiegimą.

Programų sistemos diegimo reikalavimais siekiama minimizuoti išlaidas, reikalingas naudotojams ir aptarnaujančiam personalui apmokyti bei sistemą darbui parengti.

Instaliuojant programų sistemą, iš distributyvinės tos sistemos versijos yra generuojama jos funkcinė versija. Darbo sąnaudos reikalingos programų sistemai instaliuoti priklauso nuo dviejų pagrindinių veiksnių:

- distributyvinės sistemos versijos patikimumo;
- instaliavimo procedūros.

Norint užtikrinti distributyvinės versijos patikimumą bei sumažinti problemas diegimo aplinkai, dažniausiai yra reikalaujama, kad kartu su programų sistemos distributyvine versija būtų pateikti detalūs tos versijos ir funkcinės versijos konfigūracijų aprašai, programa, leidžianti patikrinti, ar distributyvinėje sistemos versijoje tikrai yra visi reikalingi komponentai, ir testas, tikrinantis sugeneruotos funkcinės sistemos versijos veiklą. Be to, reikalaujama taip ženklinti fizinius duomenų nešiklius (pvz., instaliacinius diskelius), į kuriuos įrašyta distributyvinė versija, kad jie būtų lengvai skiriami vienas nuo kito.

Bendruoju atveju instaliavimo procedūros apribojimai nesiskiria nuo bet kokios kitos programų sistemos procedūros apribojimų, t.y. ši procedūra turi būti:

- maksimaliai automatizuota;
- reikalauti minimalaus operatoriaus įsikišimo;
- naši;
- patikima;
- paprasta ir maksimaliai standartizuota.

Jokiu kitu specifiniu apribojimu, kliudančių diegimui, numatomoje diegimo aplinkoje nėra.

3.6.4. Įtaka jau instaliuotoms sistemoms

- Sistema turi būti įmanoma pritaikyti operacinės aplinkos pokyčiams.
- Sistema turėtų būti lanksti kompiuterinės platformos pasikeitimams.

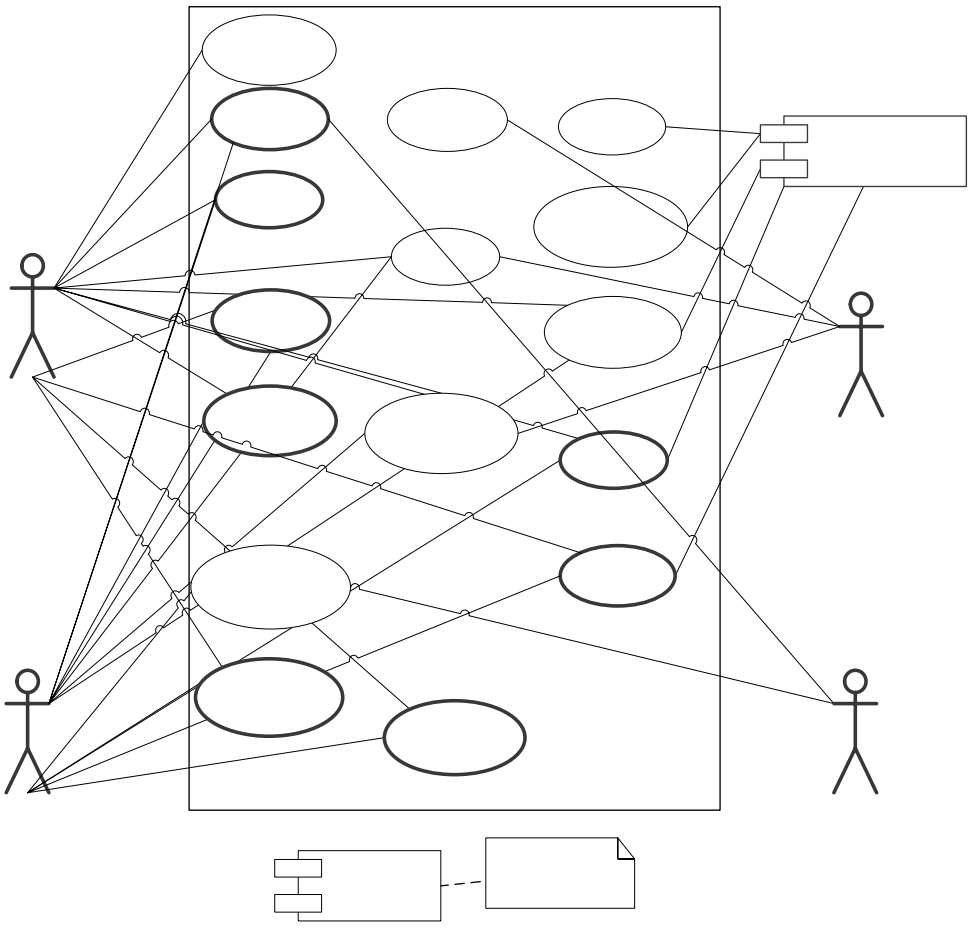
- Sistema turi būti lanksti minimaliems praplečiamumams, t.y. kuo mažesnis sąnaudų lygis atliekant jos modernizavimo darbus, kitu sistemų atžvilgiu.
- Sistemos suvartojamu resursu kiekis turi būti suderintas su kitomis instaliuotomis sistemomis. Sistema turi veikti taip jog nesutrikdytu kitu sistemų veikimo, resursu atžvilgiu, bet tuo pačiu garantuotu sėkminga savo pačios funkcionavimo veikla.
- Sistema turi būti pritaikoma. Sistema neturėtų naudoti kažkokių specifiniu tvarkyklių pažeidžiančių kitos instaliuotos sistemos funkcionavimo veikla.

3.7. Architektūra

Detaliai sistemos architektūrai atvaizduoti, naudojami 4+1 perspektyvos: panaudojimų atvejų, loginė, procesų, diegimo aplinkos ir įdiegimo.

3.7.1. Panaudojimų atvejai

Paveikslėlyje pateikiama svarbesnius sistemos panaudojimo atvejus. Išskiriami panaudojimo atvejų aktoriai: vartotojas, tiekėjas, vadybininkas, administratorius, informacinė sistema, duomenų bazė.

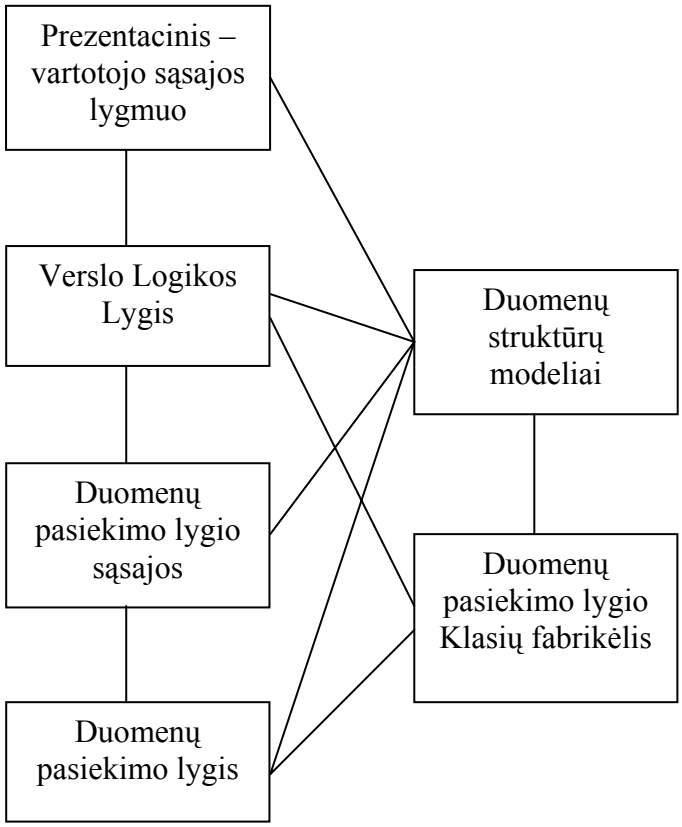


6 pav. Svarbūs architektūros panaudojimo atvejai.

3.7.2. Loginis sistemos požiūris

Loginiu požiūriu, projekte išskiriami šie objektai:

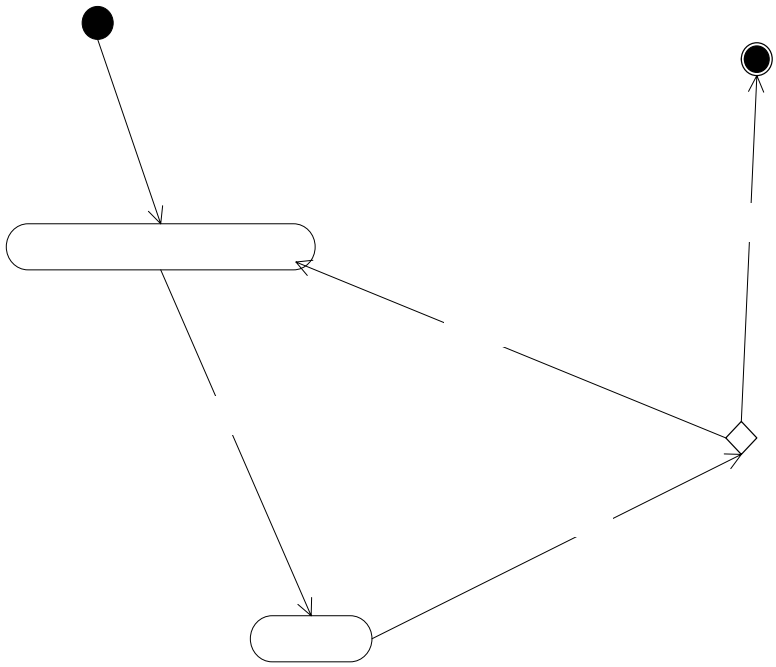
- Prezentacinis – vartotojo sąsajos lygmuo
- Verslo Logikos Lygis
- Duomenų struktūrų modeliai
- Duomenų pasiekimo lygio sąsajos
- Duomenų pasiekimo lygis
- Duomenų pasiekimo lygio klasių fabrikėlis



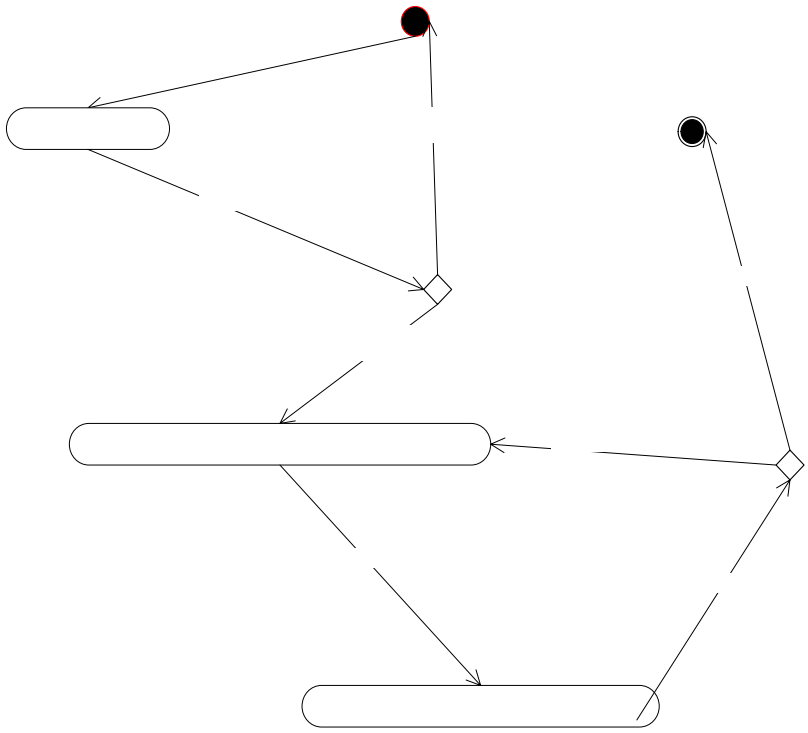
7 pav. Kuriamos sistemos struktūra.

3.7.3. Pagrindiniai sistemos procesai

Procesų perspektyvos architektūros aprašymas apibrėžia užduotis (procesų ir gijų) kurios yra įtrauktos į sistemos įgyvendinimo, jos sąveikos ir derinimo planus. Taip pat parodo objektų ir klasių priklausomumą nuo užduočių.



8 pav. Procesų veikla katalogo peržiūrai.

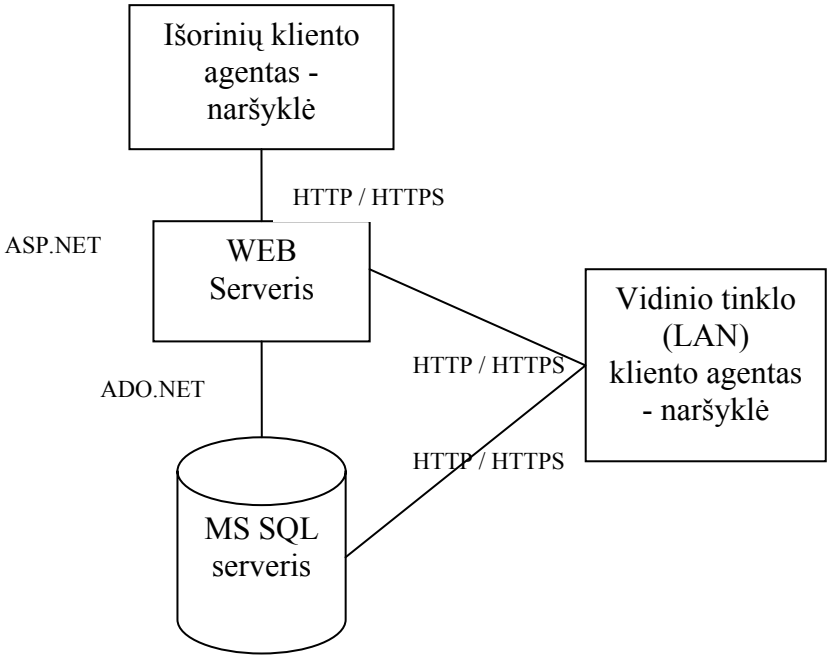


9 pav. Procesų veikla naujos prekės/paslaugos įvedimui ar koregavimui jau esamos

3.7.4. Diegimo aplinka

Programinės įrangos išdėstymo perspektyvos aprašymas apibrėžia įvarius fizinius mazgus daugumos platformų suderinamumui. Taip pat apibrėžia užduočių paskirstymą fiziniams mazgams.

Žemiau pateikiamas programinės įrangos išdėstymas.



10 pav. Įrangos išdėstymas.

4. Duomenų atnaujinimo lygiagretumo konfliktų tyrimas ir realizacijos būdai prekybos ir klientų aptarnavimo sistemose

Panagrinėsime duomenų atnaujinimo lygiagretumo konfliktus ir realizacijos būdus *.NET Frameworke*.

4.1. Bendra *DiffGram* formato apžvalga

DiffGram yra *XML* serializacijos formatas, kuriame yra originalios ir einamosios visų duomenų lentelių (*DataTable* objektų) eilučių (*DataRow* objektų) reikšmės, t. y. *DiffGram'oje* yra einamosios eilutes su naujausiais duomenimis bei skyriai, kuriuose visos originalios pakeistų duomenų eilučių reikšmės ir klaidos yra saugojamos.

Kiekviena eilutė turi unikalų identifikacinį numerį, pagal kurį atsekami pakeitimai tarp šių dviejų *DiffGram'os* skyrių. Tai labai primena ryšius duomenų bazėse tarp lentelių su pirminiais raktais. Žemiau pateikiama *DiffGram'os* struktūros pavyzdys:

```
<diffgr:diffgram
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
<DataSet>
...
</DataSet>
<diffgr:before>
...
</diffgr:before>
<diffgr:errors>
...
</diffgr:errors>
</diffgr:diffgram>
```

<diffgr:diffgram> mazgas gali turėti iki trijų vaikų.

Pirmasis yra *DataSet* objekto einamųjų duomenų skyrius, įskaitant naujai sukurtas, pakeistas duomenų eilutes. Pašalintos eilutės čia nesaugojamos. Tikrasis pomedžio pavadinimas yra *DataSet* objekto *DataSetName* savybė. Jei *DataSet* objektui nėra nurodytas pavadinimas, jam priskiriamas pagal nutylėjimą „*NewDataSet"*.

<*diffgr:before*> pomedyje saugoma informacija, kurios pakanka atstatyti atitinkamų pakeistų eilučių originalias būkles. Čia išsaugoma kiekviena eilutė, kuri buvo pašalinta, pakeistų eilučių originalai. Viskas, kas yra pakeista, saugoma <*diffgr:before*> skyriuje.

Paskutinis aptariamas pomedis yra <*diffgr:errors*>, kuriame yra laikoma visa informacija apie klaidas, kurios įvyko tam tikrose eilutėse. *DataRow* klasėje yra keletas metodų ir savybių, kurias naudodami programuotojai gali nustatyti klaidą bet kuriai duomenų eilutei, eilutės stulpeliui. Klaidos gali būti nustatomos bet kuriuo metu, nebūtinai tik tada, kai duomenys yra įvedami. Pavyzdžiui, tam tikrose taikomosiose programose vienas vartotojas gali įvesti duomenis ir nusiųsti kitam vartotojui patikrinti. Pastarasis gali nustatyti klaidą bet kuriam eilutės stulpeliui, informuoti įvestų duomenų autorių, kad kažkas yra blogai su įvestais duomenimis. *Microsoft Windows Forms DataGrid* objektas duomenų eilutėse, kurios rodomos vartotojui, susiranda priskirtas klaidas ir išskiria duomenų eilutes su klaidomis, taip informuojamas vartotojas, kad duomenų eilutės tam tikrame stulpelyje yra klaida.

Žemiau pateikiama *DiffGram'os* struktūros pavyzdys, kuriame pirma eilute buvo pakeista, antra eilutė pašalinta, trečioje yra klaida bei buvo sukurta nauja eilutė:

```
<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
  <StrategasDS>
    <Vartotojai diffgr:id="Vartotojai1" msdata:rowOrder="0"
diffgr:hasChanges="modified">
      <UserName>Jonas</UserName>
      <FirstName>Jonas</FirstName>
      <Surname>Jonaitis</Surname>
      <PublicEmail />
      <Approved>true</Approved>
    </Vartotojai>
    <Vartotojai diffgr:id="Vartotojai3" msdata:rowOrder="2"
diffgr:hasErrors="true">
      <UserName>Antanas</UserName>
      <FirstName>Antanas</FirstName>
      <Surname>Baranauskas</Surname>
      <PublicEmail />
      <Approved>true</Approved>
    </Vartotojai>
    <Vartotojai diffgr:id="Vartotojai4" msdata:rowOrder="3"
diffgr:hasChanges="inserted">
```

```
      <UserName>Petras</UserName>
      <FirstName>Petras</FirstName>
      <Surname>Petraitis</Surname>
      <PublicEmail>pastas@petro.lt</PublicEmail>
      <Approved>true</Approved>
    </Vartotojai>
  </StrategasDS>
  <diffgr:before>
    <Vartotojai diffgr:id="Vartotojai1" msdata:rowOrder="0">
      <UserName>a</UserName>
      <FirstName>Antanas</FirstName>
      <Surname>Baranauskas</Surname>
      <PublicEmail />
      <Approved>true</Approved>
    </Vartotojai>
    <Vartotojai diffgr:id="Vartotojai2" msdata:rowOrder="1">
      <UserName>Admin</UserName>
      <FirstName>Antanas</FirstName>
      <Surname>Baranauskas</Surname>
      <PublicEmail>marius@strategas.lt</PublicEmail>
      <Approved>true</Approved>
    </Vartotojai>
  </diffgr:before>
  <diffgr:errors>
    <Vartotojai diffgr:id="Vartotojai3" diffgr:Error="Klaida !" />
  </diffgr:errors>
</diffgr:diffgram>
```

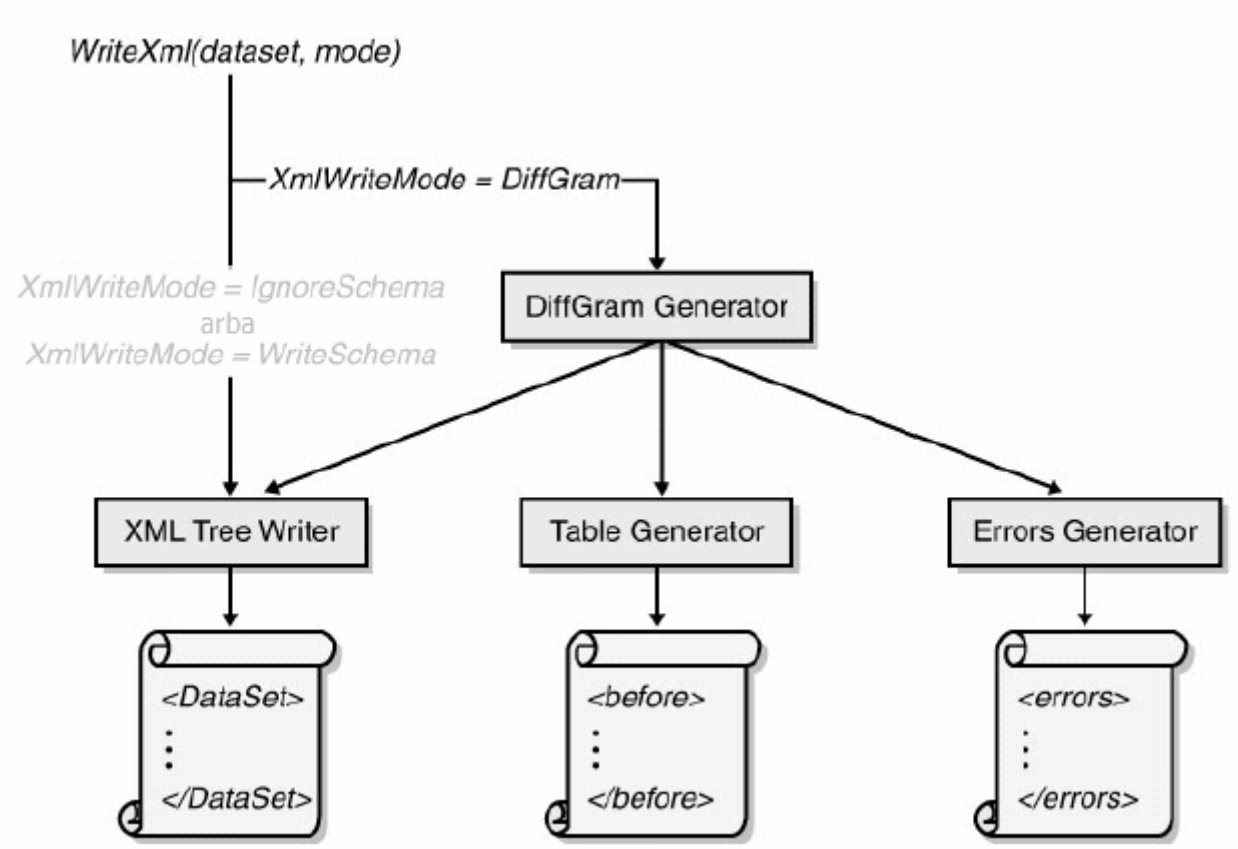
Kai kurie atributai ir mazgai, kurie formuoja *DiffGram'ą,* yra imami iš vardų sričių, tokių, kaip *msdata* ir *diffgr*.

4.2. Einamieji duomenys

Pirmame *DiffGram'os* skyriuje yra einamieji duomenys. *DiffGram'a* be duomenų yra tuščio *DataSet* objekto atvaizdavimas. <*diffgr:before*> ir <*diffgr:errors*> pomedžių nėra, jei *DataSet* objekte esantys duomenys nesikeitė ir neįvyko klaidų.

DiffGram'a išsaugo būklę ir yra labai panaši į *ADO.NET* XML normalinę formą. Duomenys yra beveik identiški kaip ir normalinės formos, tačiau pastaroji neišsaugo duomenų būklės. Didžiulis skirtumas tarp *DiffGram* ir normalinės formos yra tai, kad *DiffGram* formate nėra schemos informacijos.

Būtina sąlyga duomenų būklei *DiffGram* formate išsaugoti yra duomenų skyriaus, originalių duomenų bei klaidų skyrių egzistavimas. Pasinaudojus šiais trim pomedžiais galima atstatyti originalius *DataSet* objekte saugojamus duomenis.



11pav. *DiffGram'os* generavimas kviečiant *WriteXml* metodą

Kitas didelis skirtumas tarp *ADO.NET XML* normalinės formos ir *DiffGramo's* duomenų yra tai, kad pastaroji papildomai turi *id*, *hasChanges*, *hasErrors*, ir *rowOrder* atributus. Šie papildomi atributai aprašyti vardų srityse, pateiktose *DiffGram'os* pradžioje. Šie specialūs atributai naudojami mazgams pažymėti, taip susiejant elementus duomenų, pakeitimų bei klaidų skyriuose.

4.3. DiffGram'os specialūs atributai

3 lentelė. DiffGram'os žymėjimai / atributų aprašymai

diffgr:error	Klaidos tekstas, kuris apibudina klaidą eilutėje ar eilutės stulpelyje
diffgr:hasChanges	Informuoja, kad eilutė buvo pakeista
diffgr:hasErrors	Informuoja, kad eilutėje yra klaida
diffgr:id	Unikalus ID, naudojamas eilutėms susieti tarp skyrių
diffgr:parentId	Grąžina "tėvines" eilutės unikalų ID
msdata:hiddenXXX	Atributas stulpeliams paslėpti *XXX* yra tikrasis stulpelio pavadinimas
msdata:rowOrder	Nurodo eilutės vietą *DataSet* objekte

4.4. Duomenų pakeitimo atvaizdavimas

diffgr:hasChanges atributas nurodo, kad duomenys eilutėje buvo pakeisti. Atributas gali įgyti žemiau nurodytas reikšmes:

4 lentelė. *diffgr:hasChanges* atributo reikšmės / reikšmių aprašymai

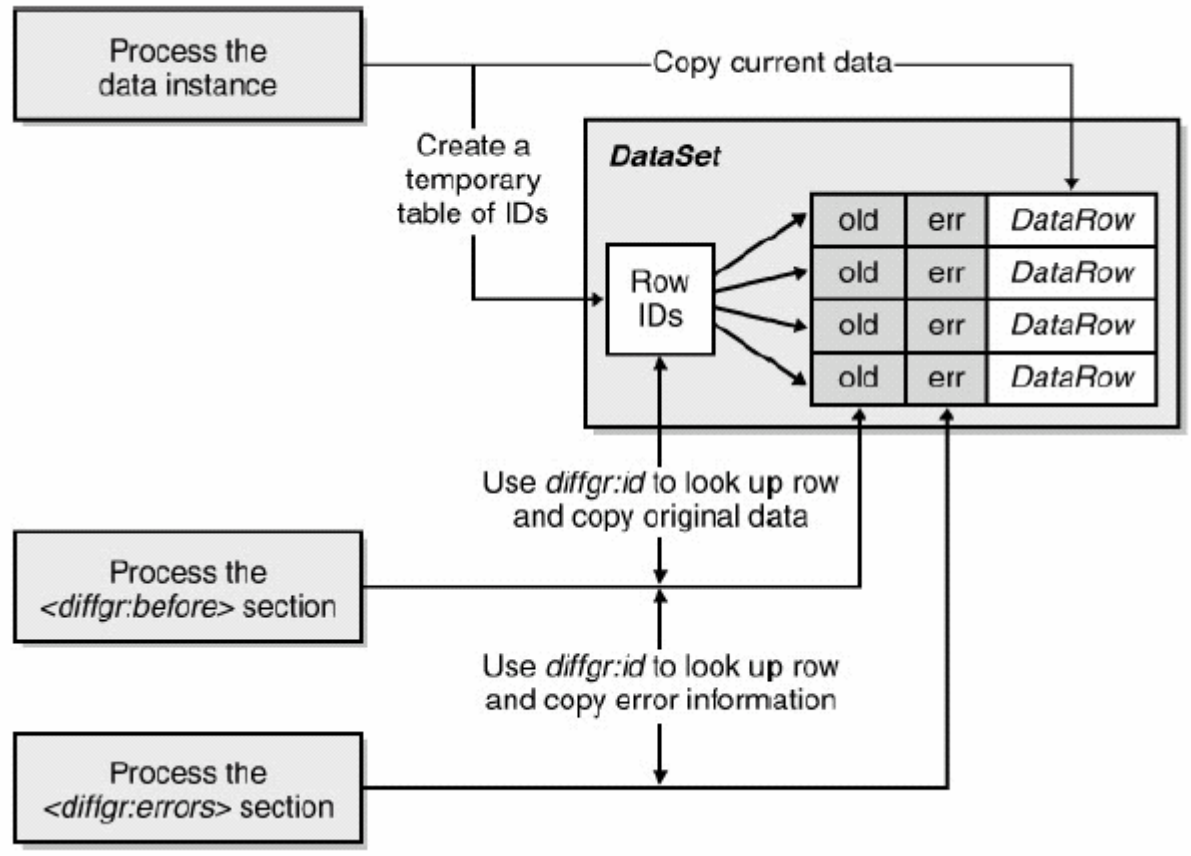
descent	Informuoja, kad yra pakeitimų susijosiose tėvo/vaiko eilutėse
inserted	Informuoja, kad buvo sukurta nauja duomenų eilutė
modified	Informuoja, kad eilutė buvo pakeista. Eilutės originalas išsaugotas atitinkamoje <*diffgr:before*> skyriaus eilutėje

Naujai sukurta eilutė neturi atitinkamo elemento <*diffgr:before*> skyriuje. Pašalinta eilutė neturi atitinkamo elemento duomenų skyriuje, tačiau rasime įrašą <*diffgr:before*> skyriuje. Peržiūrint duomenų skyrių galima greitai ir paprastai identifikuoti, kokios eilutės sukurtos ar modifikuotos –

kiekviena eilutė turi *diffgr:hasChanges* atributą su paaiškinama reikšme. Pašalintas eilutes nurodo *msdata:rowOrder* reikšmės – jei yra tarpų šioje numeracijos sekoje, tos eilutės buvo pašalintos.

4.5. Duomenų skaitymas iš *DiffGram*'os

Skaitant duomenis iš *DiffGram*'os į *DataSet* objektą, *DataSet* objekto *ReadXml* metodas visų pirma skaito duomenis į duomenų skyrių ir sukuria visus reikalingas lenteles ir eilutes. Kiekvienai eilutei yra atitinkamai nustatoma naujai sukurtos ar pakeistos eilutės būklė. Visos d*iffgr:id* reikšmės yra laikinai nukopijuojamos į vidinę *hash* lentelę, programuojant ją matome kaip *DataSet* objekto savybę. Kiekvienas *hash* lentelės įrašas susijęs su *DataRow* objekte kuriamu įrašu. *ReadXml* metodas toliau pildo <*diffgr:before*> skyrių ir nuskaito senas lentelių reikšmes, jei tokios yra. Jei rastas atitikmuo tarp eilutės <*diffgr:before*> skyriuje ir eilutės, jau nuskaitytos į lentelę, ką tik nuskaityta duomenų eilutė yra saugoma kaip lentelės duomenų eilutės originalas. *ReadXml* metodas atitikmens ieško pagal *diffgr:id* atributą <*diffgr:before*> skyriuje ir hash lentelėje.
Žemiau pateikta, kaip skaitoma *DiffGram'a* į *DataSet* objektą.



12 pav. *DiffGram'os* duomenų skaitymas į *DataSet* objektą

DataSet objekto turinys sukuriamas skaitant *DiffGram*'os vieną po kito skyrius ir ieškant duomenų eilučių atitikmenų pagal eilučių ID numerius įvairiuose skyriuose.
Jei atitikmens nepavyko rasti, *ReadXml* supranta tai kaip eilutės, kuri yra <*diffgr:before*> skyriuje, pašalinimą. Pašalintos eilutės buvimo vietą nurodo *msdata:rowOrder* atributo reiksmė. Metodas įterpia naują eilutę į *DiffGram'os* einamųjų duomenų skyriuje buvusią vietą ir užpildo nuskaitytomis iš <*diffgr:before*> skyriaus atitinkamomis reikšmėmis. Tada eilutė, iškviečiant *DataRow* objekto *Delete* metodą, pažymima pašalinimui.

Galiausiai skaitome reikšmes iš <*diffgr:errors*> skyriaus ir atnaujiname atitinkamai *DataRow* objekto *RowError* savybę atitinkamoje lentelėje.

4.6. Įvykių valdymas duomenų eilutėje

DataSet, *DataTable* ir *DataRow* objektai turi savo būklę apibrėžiančius atributus. Kai duomenų eilutė pakeičiama, pašalinama ar sukuriama, jos būklė pakeičiama į vieną iš *DataRowState* enumeratoriaus reikšmių.

Jei duomenų eilutė pakeičiama, pašalinama ar sukuriama lentelėje, visa vidinė lentelės (*DataTable*) būklė pasikeičia. Priimti ar atmesti pakeitimus galima *DataSet*, *DataTable*, ar *DataRow* lygmenyje. Priimti pakeitimus reiškia, kad lentelės duomenų eilutėje bus atlikti pakeitimai (eilutės visuomet dalyvauja pakeitimuose). Pakeitimų atmetimas reiškia, kad lentelės būklė atstatoma į pradinę, jokių pakitimų lentelėje neįvyksta. *DiffGram'oje* fiksuojami paruošti eilučių pakeitimai, kurie dar yra nepriimti ir neatmesti.

5 lentelė. DataRow objekto būklės reikšmės / reikšmių aprašymai

Added	Lentelėje sukurta eilutė, bet metodas *AcceptChanges* dar nėra iškviestas.
Deleted	Eilutė pažymėta pašalinti.
Detached	Sukurta eilutė, bet dar nepriskirta lentelei, arba eilutė pašalinta iš eilučių rinkinio.
Modified	Eilutėje buvo atlikti pakeitimai.
Unchanged	Eilutėje pakeitimų nebuvo nuo to laiko, kai *AcceptChanges* metodas buvo iškviestas.

AcceptChanges metodas priima ir vykdo pakeitimus, tada einamosios reikšmės tampa originaliomis. *RejectChanges* atšaukia paruoštus vykdyti pakeitimus.

4.7. Originalių duomenų skyrius <diffgr:before>

DiffGram'os struktūra suskirstyta į skyrius, kur einamosios reikšmės , originalios modifikuotų eilučių reikšmės ir klaidos saugojami atskirai. *DataSet* objekto duomenų eilučių originalios reikšmės saugojamos <*diffgr:before*> skyriuje.

DataRow objektas turi keletą savo versijų, kurios saugojamos eilučių masyve. Šios versijos apibrėžtos *DataRowVersion* enumeratoriuje.

14 lentelė. DataRow versijų *e*numeratoriaus reikšmės / reikšmių aprašymas

Current	Einamosios eilutės reikšmės
Default	Eilutės versijos reikšmė pagal nutylėjimą, atsižvelgiant į jos būklę. Kai *DataRowState* reikšmė yra *Added, Modified* arba *Current*, versijos reikšmė yra *Current*. Kai *DataRowState* reikšmė *Deleted*, versijos reikšmė bus *Original*. Kai *DataRowState* reikšmė *Detached*, versijos reikšmė bus *Proposed*.
Original	Originalios eilutės reikšmės, reikšmės po paskutinio *AcceptChanges* metodo iškvietimo
Proposed	Siūlomos reikšmės eilutei

Tik *Current* ir *Original* eilučių versijos yra saugojamos *DataRow* objekte pastoviam laikui. *Proposed* versijų gyvavimo trukmė apima tik eilučių redagavimo laiką. Eilutės redagavimo laikas yra tarp *BeginEdit* ir *EndEdit* metodų iškvietimų. Kai dirbame su *DataRow* objektu, mes galime nurodyti, kokių versijų mes norime, kaip parodyta pavyzdyje:

```
if(row[0] == row[0, DataRowVersion.Original])
{
...
}
```

ReadXml metodas naudoja <*diffgr:before*> skyriaus informaciją atstatyti *Original* eilučių versijoms. Naujai sukurtos eilutės neturi ankstesnės būklės, todėl atitinkamo rašo <*diffgr:before*>

skyriuje nerasime. Pašalintos eilutės yra tik <*diffgr:before*> skyriuje, nes neturi duomenų einamųjų duomenų skyriuje. Pašalintos eilutės ieškomos pagal originalių eilučių *diffgr:id* atributo atitikmenį *DiffGram'os* einamųjų duomenų skyriuje. Eilutės <*diffgr:before*> skyriuje, kurios neturi savo atitikmens einamųjų duomenų skyriuje, yra vėl įterpiamos į einamųjų duomenų skyriaus lentelę ir vėl pašalinamos.

Nesvarbu kiek duomenų eilutėje stulpelių pakeista, <*diffgr:before*> skyriuje saugojama visa originalios eilutės informacija. Eilutės <diffgr:before> skyriuje pavyzdys:

```
<diffgr:before>
  <Vartotojai diffgr:id="Vartotojai3" msdata:rowOrder="2">
    <UserName>Antanas</UserName>
    <FirstName>Antanas</FirstName>
    <Surname>Baranauskas</Surname>
    <PublicEmail />
    <Approved>true</Approved>
  </Vartotojai>
</diffgr:before>
```

Pastebime duomenų stulpelių dubliavimą (nepakeisti duomenų stulpeliai vis tiek atvaizduojami <*diffgr:before*> skyriuje, nors tos pačios stulpelių reikšmės yra ir *DiffGram'os* duomenų skyriuje), tačiau taip greičiau atstatomos *DataRow* objekto originalios reikšmes.

4.8. Paruoštų klaidų valdymas

DataRow klasėje eilučių klaidoms valdyti yra keletas metodų. Galima nustatyti bendrą klaidą visai eilutei, galima nustatyti specifinę klaidą stulpeliui.
Bendrai eilutės klaidai nustatyti naudojame *DataRow* objekto *RowError* savybę. Specifinei stulpelio klaidai naudojame metodus *SetColumnError* ir *GetColumnError*. Yra dar ir pagalbiniai *GetColumnsInError* ir *ClearErrors* metodai.

Nėra jokio skirtumo tarp stulpelio ar eilutės su klaidomis ir stulpelio ar eilutės be klaidų. *Klaidą* suprantame kaip papildomą informaciją vartotojui ar taikomajai programai, kad einamajame kontekste yra nesuderinamumų. Dirbti su duomenimis galime, nors ir yra klaidos atributų eilutėse. Niekas nedraudžia naudoti klaidos savybes visai kitiems tikslams (pavyzdžiui galime laikyti savo programos kintamųjų reikšmes). Be abejo, reikia turėti omeny, kad kai kurie vartotojo sąsajos komponentai, naudojantys *DataSet* objektus kaip savo duomenų šaltinį, gali automatiškai pakeisti savo

vartotojo sąsajos išvaizdą, informuoti vartotojus apie klaidą duomenų eilutėje, nors mes naudojame eilutės klaidos savybes saviems tikslams.

7 lentelė. *DataRow* klaidų savybės / savybių aprašymas

HasErrors	Informuoja, ar eilutėje yra klaidų
RowError	Gauna arba nustato klaidos tekstą

HasErrors reikšmė yra *true,* kai *RowError* savybė turi reikšmę arba nors vienas stulpelis turi klaidos aprašymą. Stulpelius su klaidomis gauname pasinaudojus *GetColumnsInError* metodu.

8 lentelė. *DataRow* klaidų metodai / metodų aprašymas

ClearErrors	Pašalina eilutės klaidas. Nėra svarbu, kokiu metodu klaidos buvo nustatytos (*RowError* ar *SetColumnError*)
GetColumnError	Gauname stulpelio klaidos aprašymą
GetColumnsInError	Grąžina *DataColumn* objektų su klaidomis masyvą
SetColumnError	Pateiktam stulpeliui nustato klaidos tekstą

4.9. <diffgr:errors> skyriaus turinys

<*diffgr:errors*> skyriuje yra duomenų eilutė, jei jos savybė *HasErrors* grąžina reikšmę *true*. Šiuo atveju eilutė einamųjų duomenų skyriuje turi papildomą atributą *diffgr:hasErrors*, kaip parodyta žemiau:

```
<Vartotojai diffgr:id="Vartotojai1" msdata:rowOrder="0" diffgr:hasErrors="true"
UserName="a" FirstName="Antanas" Surname="Baranauskas">
      <PublicEmail />
      <Approved>true</Approved>
</Vartotojai>
```

Aprašytas elementas yra susiejamas su atitinkamu elementu <*diffgr:errors*> skyriuje, kur yra sekamos klaidos:

```
  <diffgr:errors>
```

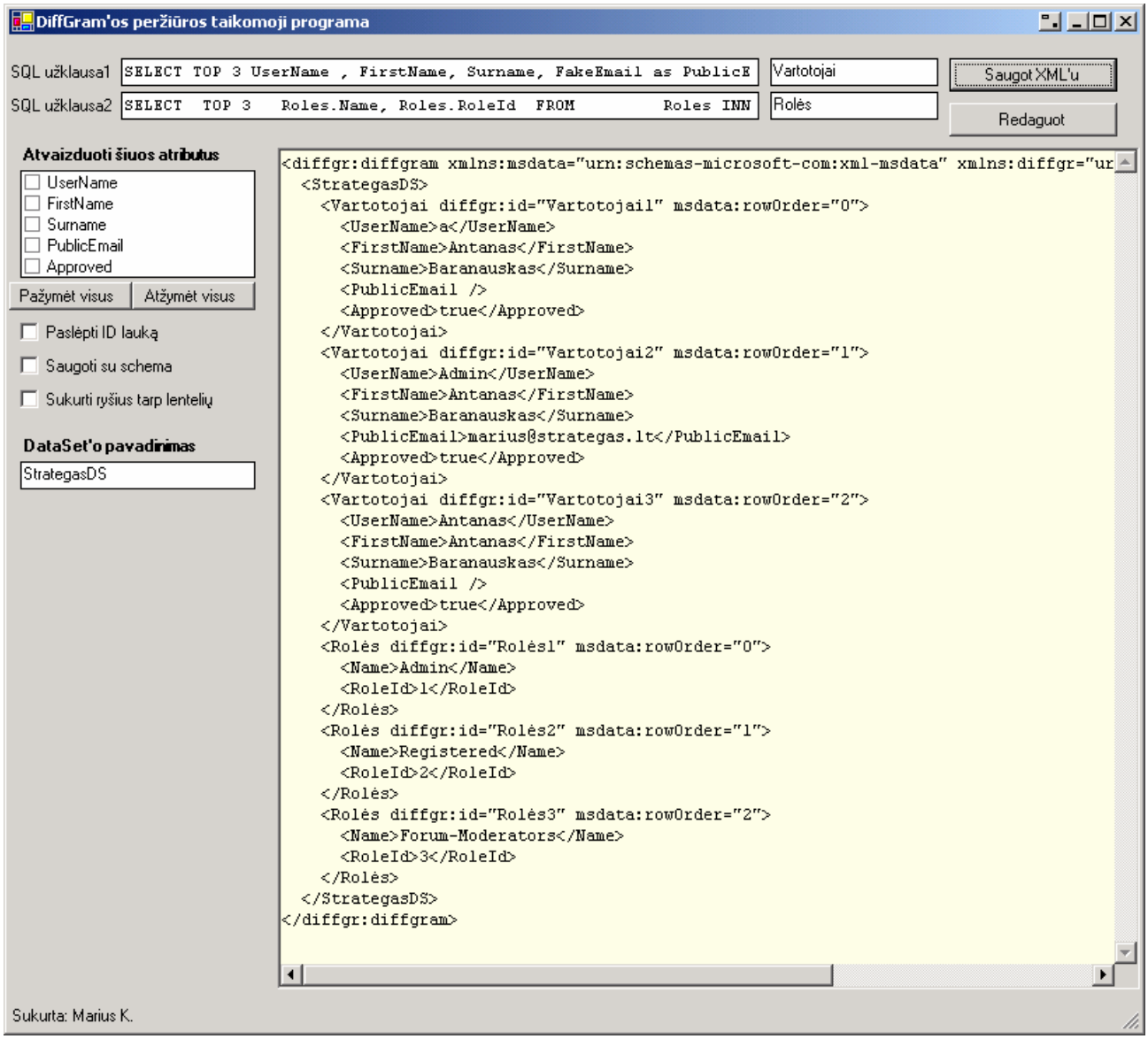
```
  <Vartotojai diffgr:id="Vartotojai1">
    <UserName diffgr:Error="Klaida! Vartotojo vardas per trumpas!" />
  </Vartotojai>
</diffgr:errors>
```

diffgr:error atributo reikšmė (klaidos tekstas) eilutėje „Vartotojai“ yra ir *RowError* savybės reikšmė. Kiekvienams stulpeliui su savo klaidos tekstu, sukuriamas naujas elementas su stulpelio pavadinimu ir *diffgr:error* atributu. Pavyzdyje *UserName* stulpelis yra su klaidomis. *RowError* savybės reikšmė nėra automatiškai sukuriama, jos reikšmę reikia nurodyti patiems.

5. *DiffGram'os peržiūros taikomoji programa*

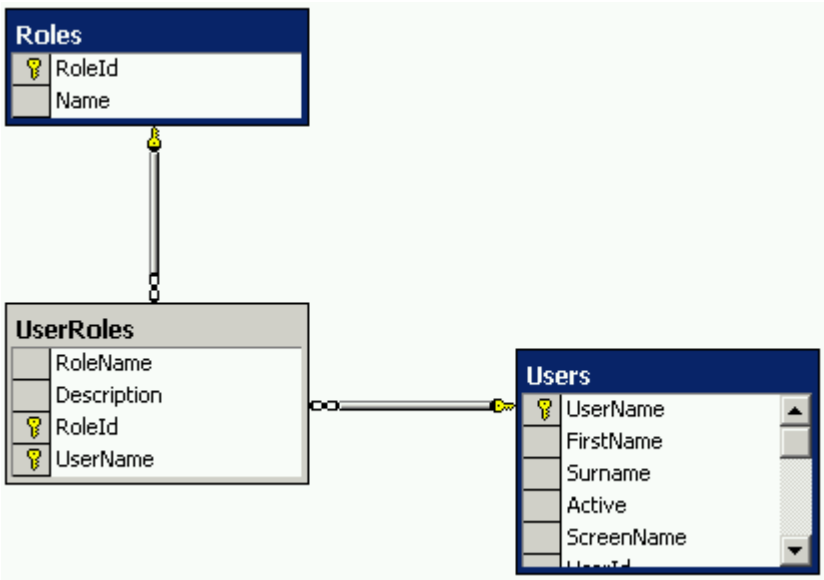
Šiame skyriuje apžvelgsime *XML DiffGram*'os realizaciją praktikoje. Tam tikslui sukurta specializuota "*DiffGram'os* peržiūros taikomoji programa".

Eksperimento metu atliksime veiksmus su *DataSet* objektu ir stebėsime, kaip keičiasi *DiffGram'os* vaizdas.



13 pav. *DiffGram'os* peržiūros taikomoji programa.

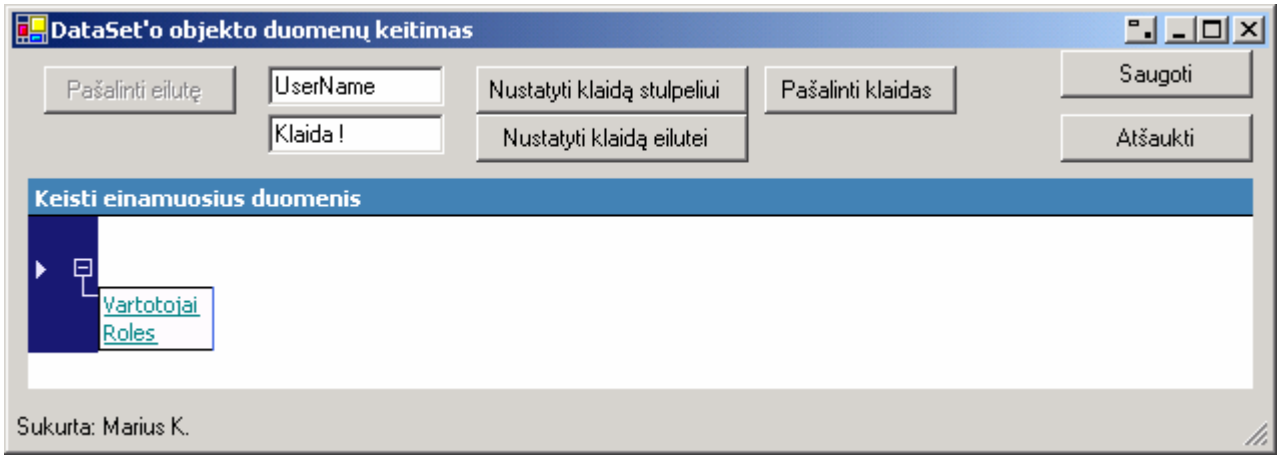
"DiffGram'os peržiūros taikomosios programos" *DataSet* objekto duomenų šaltinis yra *MS SQL Server* duomenų bazė.



14 pav. *MS SQL Server* duomenų bazės fragmentas - *DataSet* objekto duomenų šaltinis

Programa įvykdo dvi SQL užklausas („SQL užklausa1" ir „SQL užklausa2"), sukuria iš gautų duomenų *DataSet* objektą. Įvykdžius šias užklausas, gaunamas duomenų rinkinys, kurio turinį atvaizduojame.

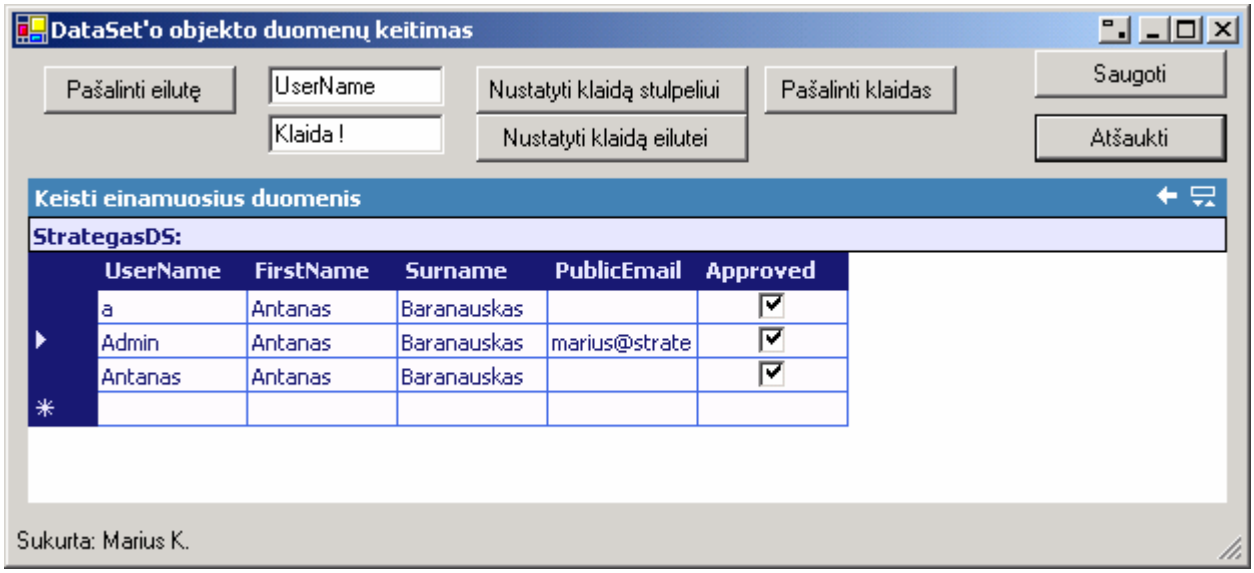
DataSet objektas turi dvi lenteles: „Vartotojai" ir „Roles". *DataSet* objekto atvaizdavimui naudojame *Windows Forms DataGrid* komponentą.



15 pav. Duomenų lentelių „Vartotojai" ir „Roles" atvaizdavimas *DataGrid* komponente

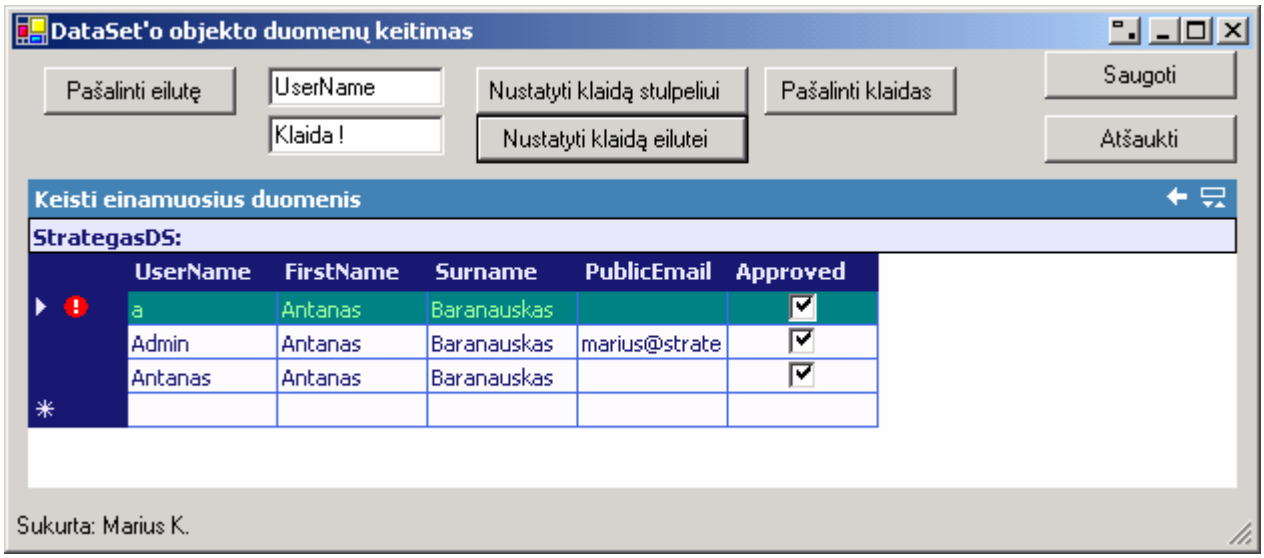
Sekančiame žingsnyje pasirenkame *DataSet* objekto lentelę „Vartotojai", išvedami lentelės „Vartotojai" duomenys, juos galima redaguoti. Mūsų atveju lentelė, kuri yra *DataTable* objektas, turi

tris eilutes, jos atitinka *DataRow* objektus, pastarosios turi duomenų stulpelius: „UserName“, „FirstName“, „Surname“, „PublicEmail“, „Approved“.



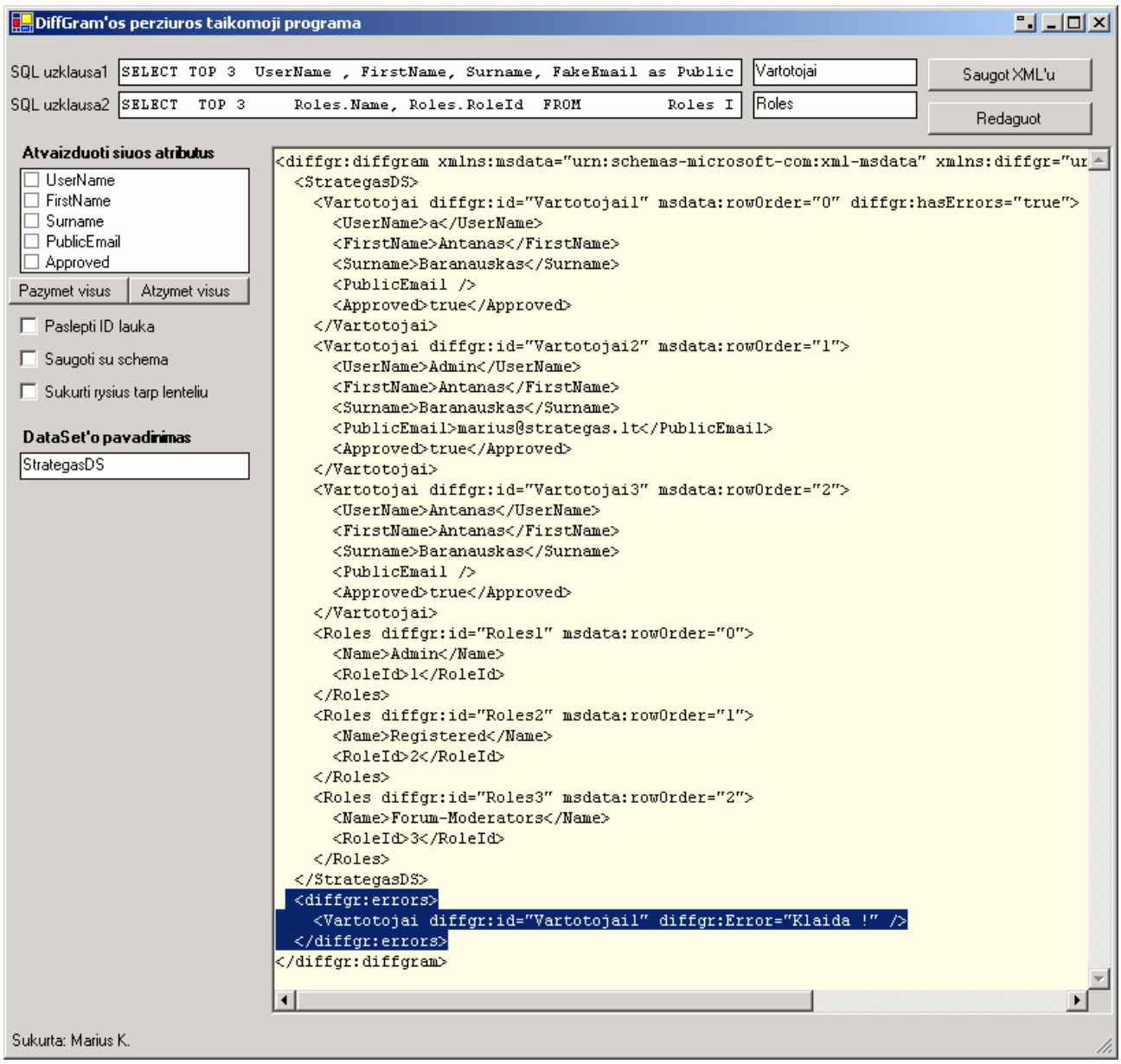
16 pav. Lentelės „Vartotojai“ turinys

Pabandykime nustatyti klaidą pirmajai duomenų eilutei. Duomenų atvaizdavimo objektas DataGrid automatiškai pakeičia savi išvaizdą ir informuoja vartotoją, kad atitinkamoje eilutėje yra klaidingi duomenys.



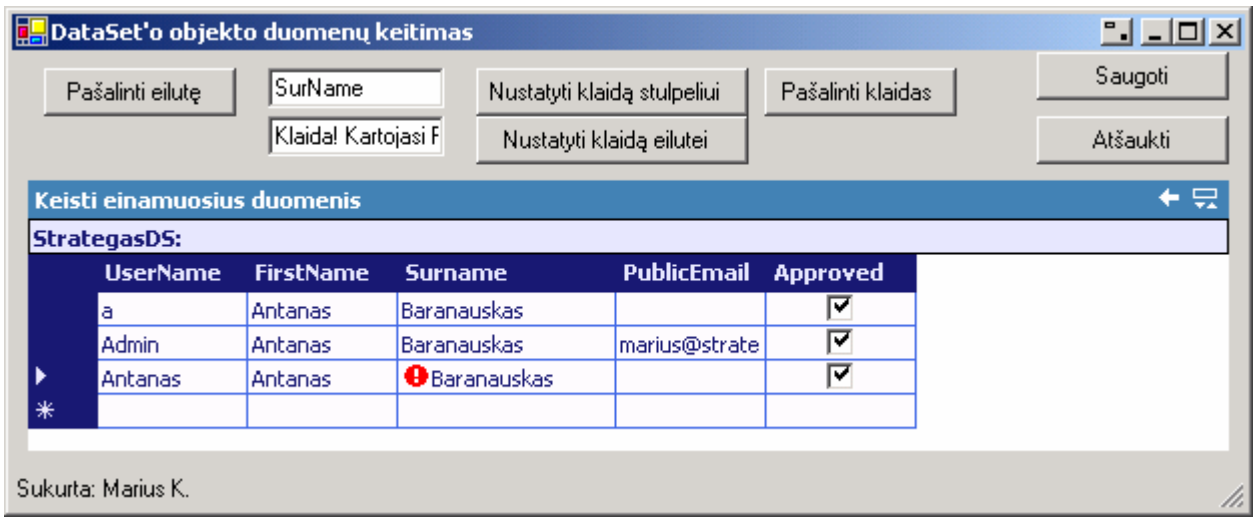
17 pav. Nustatoma klaida pirmajai eilutei lentelėje „Vartotojai“

Nustačius klaidą, galime peržiūrėti pasikeitusią *DataSet* objekto *DiffGram*‘os struktūrą. <diffgr:errors> skyriuje pastebime naują įrašą su klaidos aprašymo tekstu (jį mes nustatome patys, klaidos tekstas nėra automatiškai sugeneruojamas) bei nuoroda į klaidingą eilutę duomenų skyriuje.



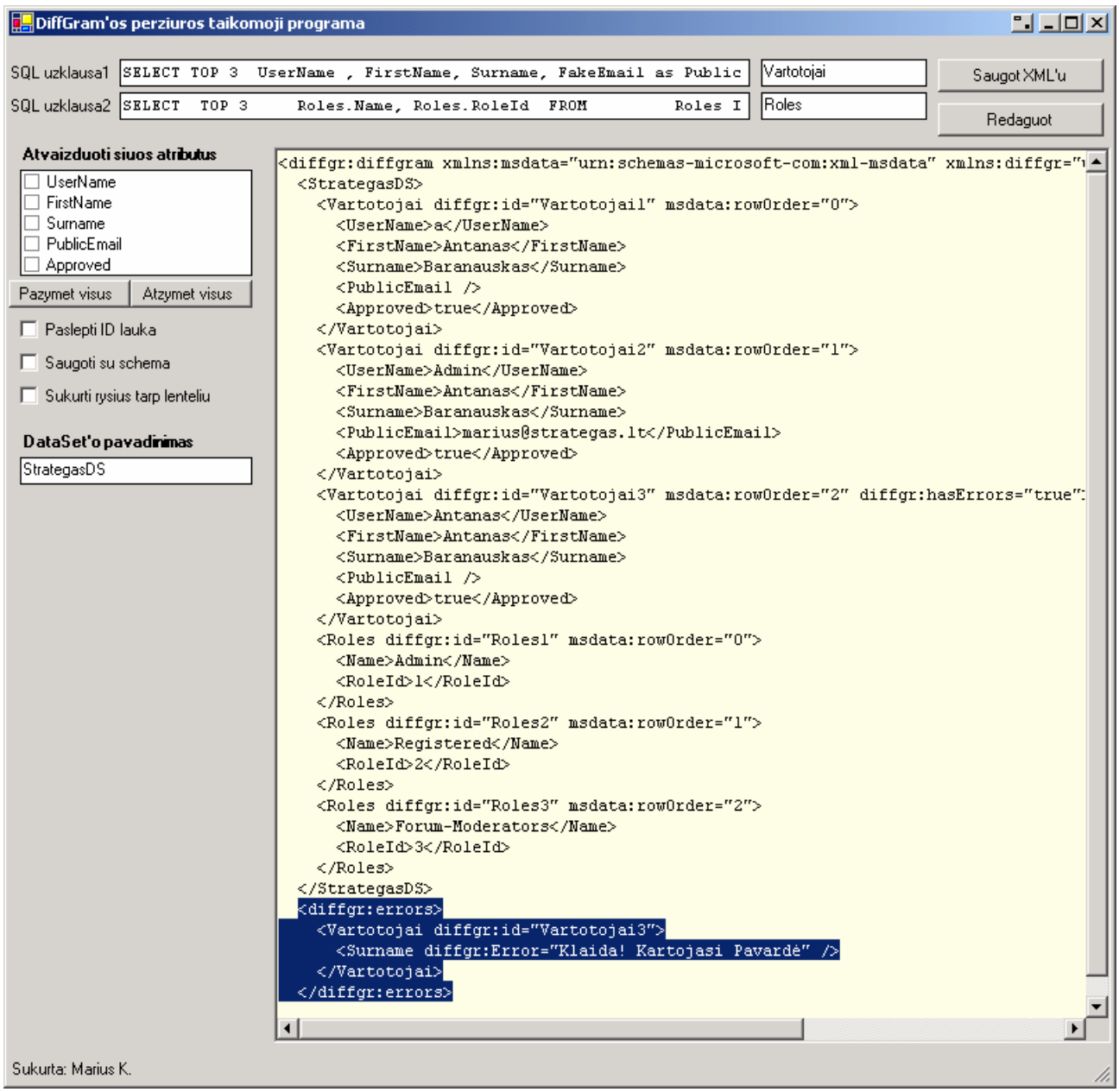
18 pav. Išvedama pakitusi *Diffgram'a* po to, kai nustatėme klaidą pirmajai eilutei lentelėje „Vartotojai“

Kaip minėta ankstesniame skyriuje, mes turime galimybę nustatyti klaidą ne tik visai duomenų eilutei, tačiau ir konkrečiam duomenų eilutės stulpeliui. Nustačius klaidą stulpeliui vėlgi DataGrid objektas pakeičia savo išvaizdą, kaip ir parodyta žemiau:



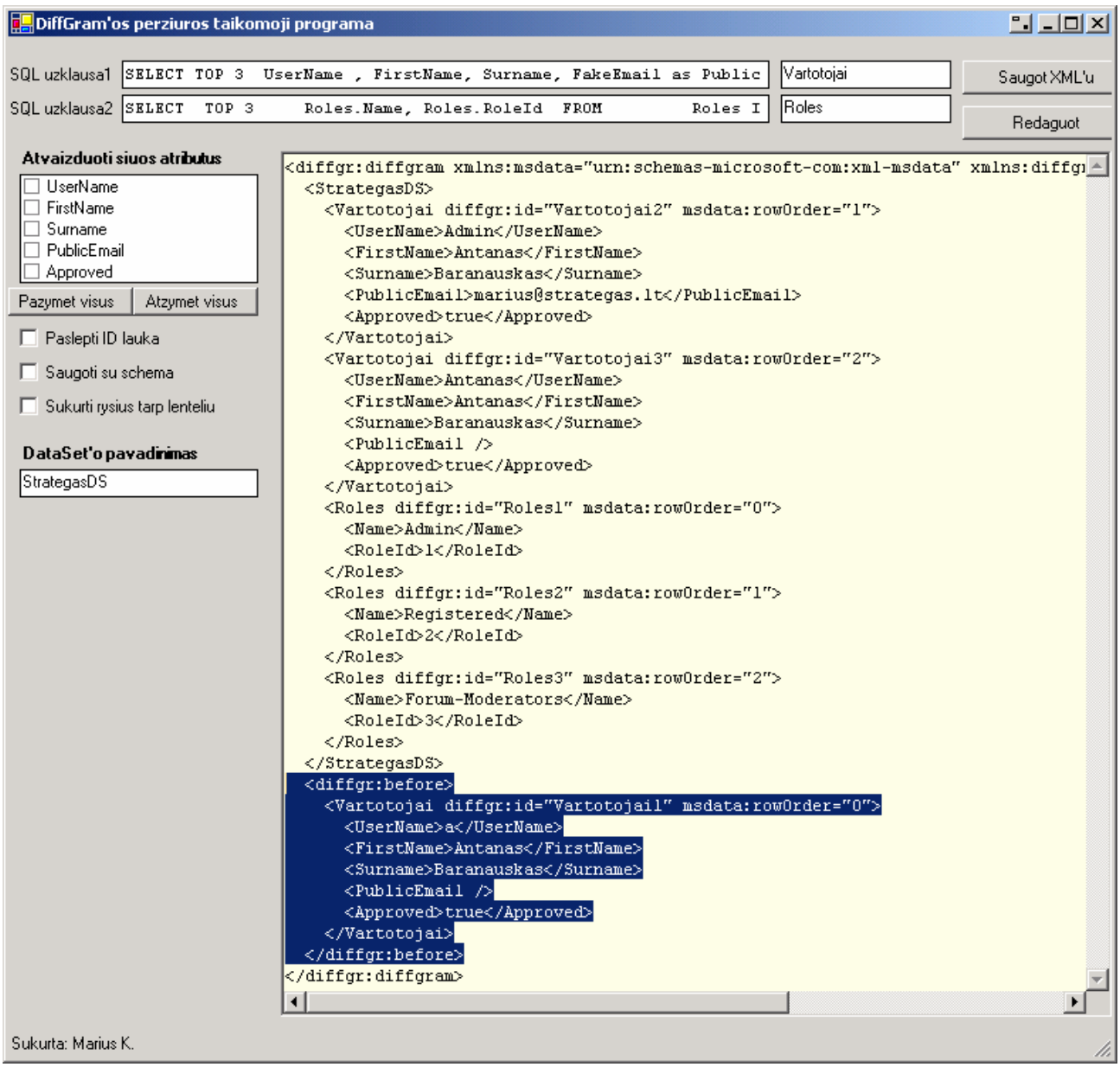
19 pav. Klaidos nustatymas konkrečiam duomenų eilutės stulpeliui

Nustačius klaidą stulpeliui, pasikeičia ir *DataSet* objekto *DiffGram*'os struktūra. <diffgr:errors> skyriuje pastebime naują įrašą su klaidos aprašymo tekstu (jį mes nustatome patys, klaidos tekstas nėra automatiškai sugeneruojamas) konkrečiam stulpeliui, be abejo yra ir nuoroda į klaidingą eilutę duomenų skyriuje.



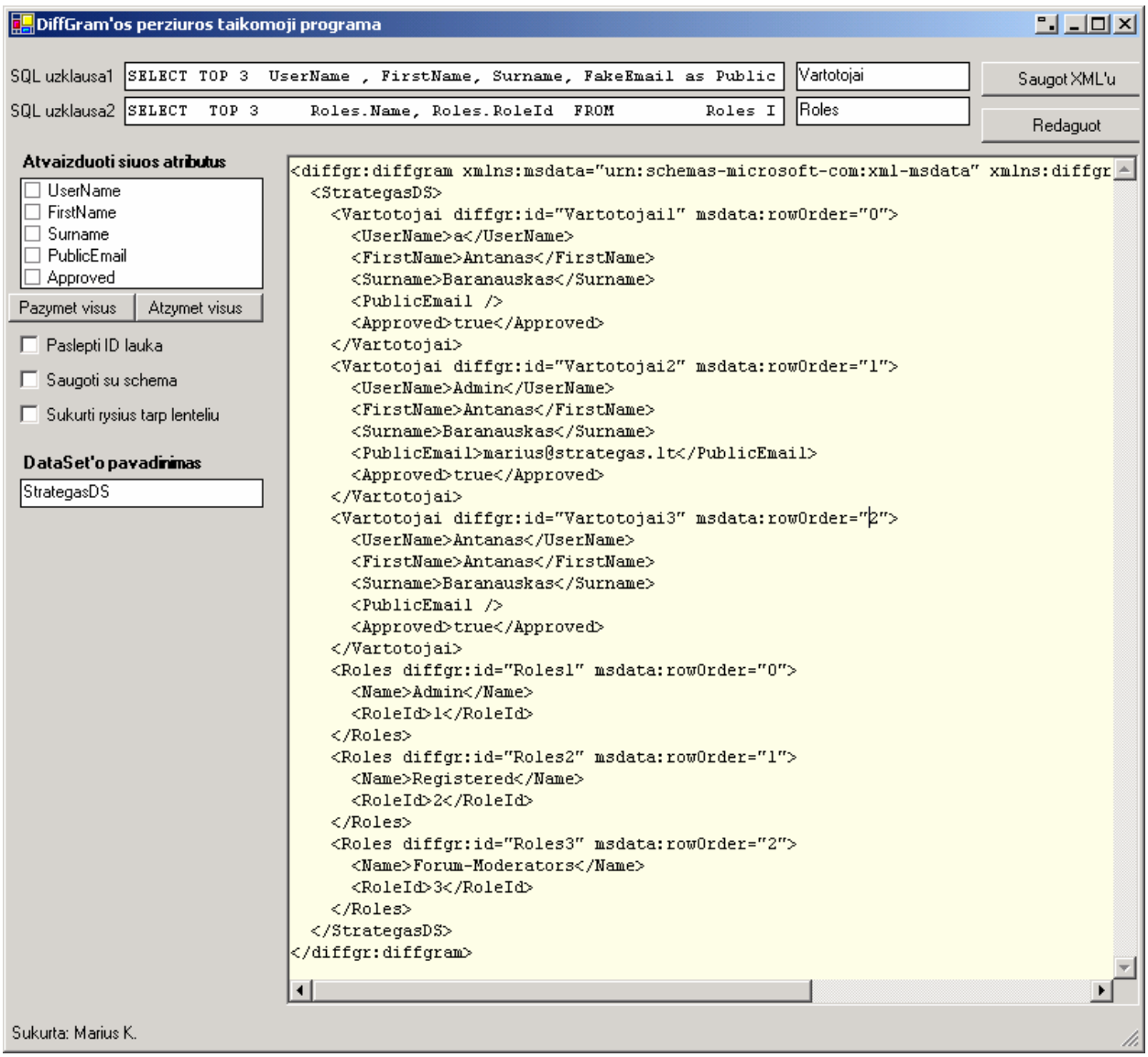
20 pav. .Išvedama pakitusi *Diffgram'a* po to, kai nustatėme klaidą konkrečiam eilutės stulpeliui

DataSet objektas naudojamas ne tik duomenims atvaizduoti, tačiau leidžia ir atlikti veiksmus su duomenimis. Atlikus pakeitimus, kaip minėta ir ankstesniame skyriuje, atsiranda pakeistų eilučių įrašai <diffgr:before> skyriuje, čia saugomi originalai. Duomenų skyriuje taip pat atsiranda pakitimų. Žemiau esančiame pavyzdyje duomenų eilutė buvo pašalinta, todėl duomenų skyriuje atitinkamos eilutės įrašo nerasime.



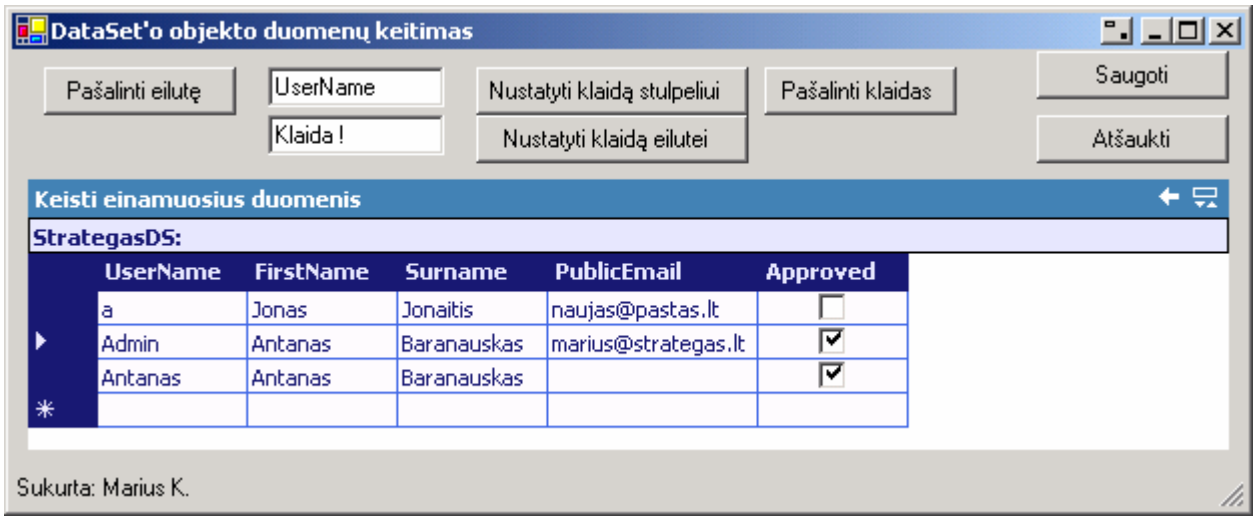
21 pav. Pašalinta pirma eilutė, išvedama pasikeitusi DiffGram‘a

Duomenų pakeitimai atmintyje, kol neatliekamas lokaliai paruoštų duomenų pakeitimas duomenų šaltinyje, gali būti atšaukti. Atšaukus ankstesniame pavyzdyje duomenų eilutės pašalinimą, DiffGram‘os turinys vėl grįžta į pradinę savo būseną.



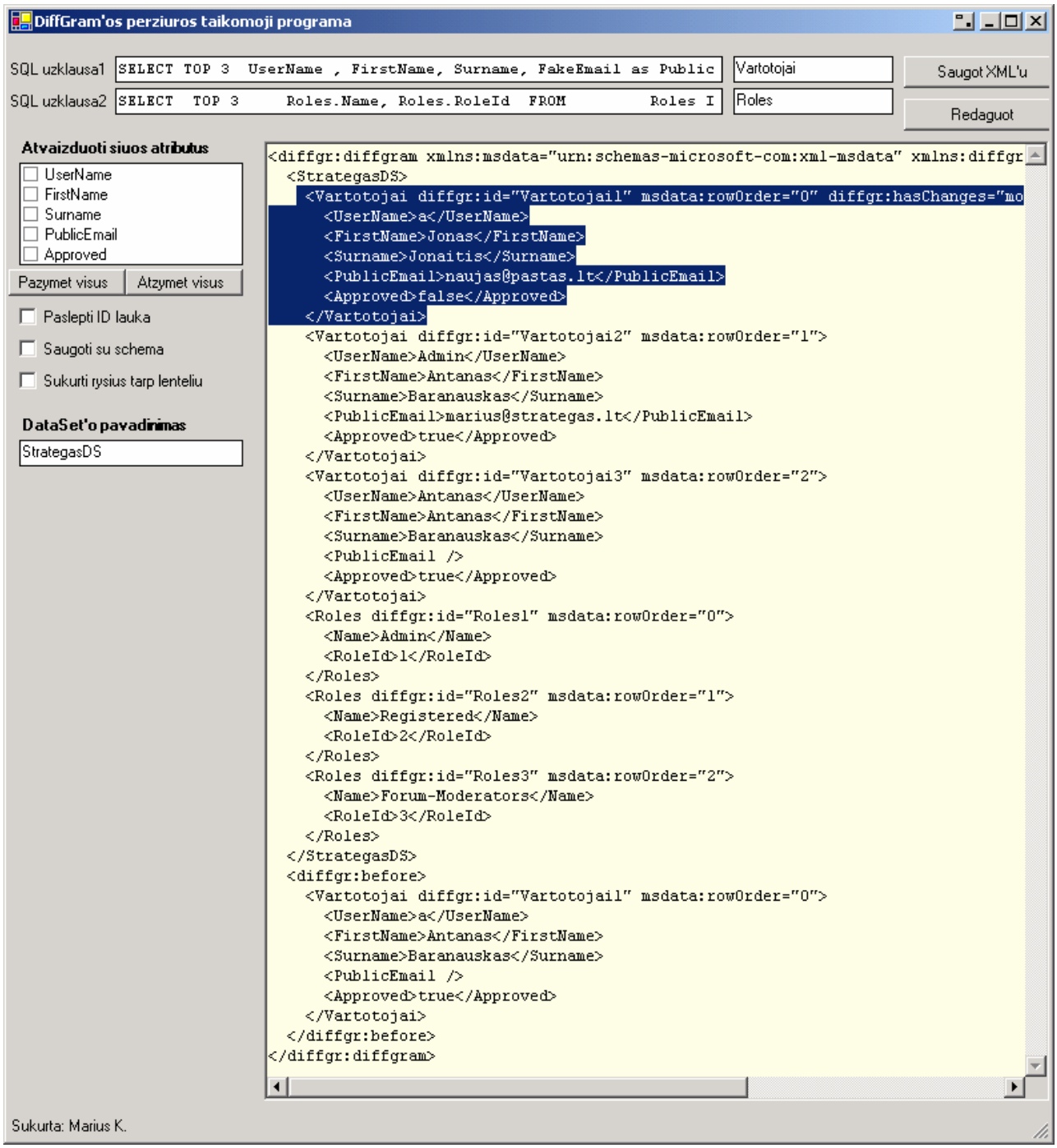
22 pav. Atšaukti lokaliai paruošti pakeitimai.

Duomenų eilutėms DataSet'e galime ne tik nustatyti klaidas. Galime keisti ir pačius duomenis. Tai ir atliekame sekančiame pavyzdyje. Pakeičiame pirmos eilutės stulpelių „*FirstName*“, „Surname“, „PublicEmail“, „Approved“ reikšmes.



23 pav. Atliekami duomenų pakeitimai.

Išvedus DiffGram‘os turinį, pastebime, kad pasikeitė einamųjų duomenų ir *<diffgr:before>* skyrių turinys. Nepaisant to, kad mes stulpelio „UserName“ reikšmės nekeitėme, jis vistiek atvaizduojamas *<diffgr:before>* ir einamųjų duomenų skyriuose. *<diffgr:before>* skyriuje saugomas duomenų eilutės originalas.



24 pav. Duomenų pakeitimų atvaizdavimas *DiffGram‘*oje

6. DiffGram‘os panaudojimas taikomosiose programose

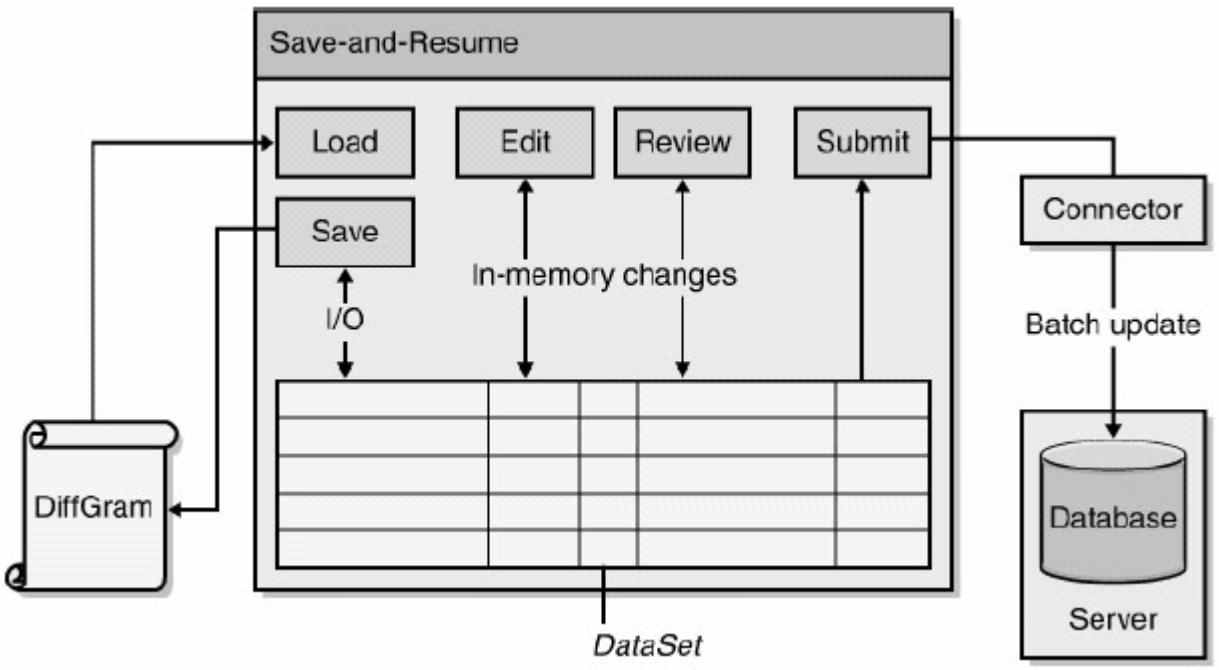
DiffGram‘os ypatybė išsaugoti duomenų būklę plačiai naudojama kuriant *save-and-resume* taikomąsias programas. *Save-and-resume* taikomosios programos yra *desktop* arba Interneto taikomosios programos, kurios iš dalies veikia ir *on-line,* ir *off-line* režime. Šių programų duomenys dažniausiai saugojami nutolusiose duomenų bazėse. Tokioms taikomosioms programoms pastovus ryšys su duomenų šaltiniu yra nebūtinas, tai ypač naudinga, kai stabilus ryšys su duomenų šaltiniu yra negarantuotas.

Save-and-resume taikomųjų standartiniai bruožai:taikomoji programa prisijungia prie duomenų bazės (duomenų šaltinio), parsisiunčia reikalingus duomenis ir atsijungia. Nuo šio momento taikomoji programa veikia atsiskyrusi nuo duomenų šaltinio, duomenys, kurių jai reikėjo, yra saugomi lokalioje atmintyje.Visi lokalių duomenų pakeitimai yra sekami. Atlikus reikiamus pakeitimus, taikomoji programa iš naujo jungiasi prie duomenų bazės (duomenų šaltinio) ir atlieka lokalių duomenų pakeitimus duomenų šaltinyje.

Be abejo, pakeitimus duomenų šaltinyje nebūtina atlikti, tai priklauso nuo pačios taikomosios programos pobūdžio, ką ji nori su duomenimis daryti.

Iš s*ave-and-resume* taikomųjų programų reikalaujama šių pagrindinių funkcijų:

- Taikomoji programa turi veikti atskirai nuo duomenų šaltinio (lokaliai saugojami duomenys iš duomenų šaltinio)
- Taikomoji programa turi leisti peržiūrėti, atšaukti duomenų pakeitimus
- Taikomoji programa bet kada gali vėl prisijungti prie duomenų šaltinio ir atlikti pakeitimus



25 pav. Sudedamosios *save-and-resume* taikomosios programos dalys

Pakeitimų siuntimas

Pakeitimų siuntimas yra eilė veiksmų, kai lokalioje atmintyje atlikti duomenų pakeitimai siunčiami duomenų šaltiniui lokaliai paruoštų pakeitimų vykdymui.

Grupinis pakeitimas („*Batch Update*“)

DataSet objektas gali pateikti lokalius pakeitimus duomenų bazei atlikti „grupinio pakeitimo“ režimu, pasinaudojus *DataAdapter* objekto *Update* metodu, kaip vėliau parodysiu pavyzdyje. Vienu metu galima pateikti pakeitimus tik vienai lentelei iš *DataSet* objekto. Jei kviečiame *Update* metodą nenurodydami lentelės pavadinimo, imama, kad lentelės pavadinimas yra „*Table*“. Jei lentelės su tokiu pavadinimu rasti nepavyksta, įvyksta klaida.

```
manoDataAdapter.Update(dataSet, tableName);
```

Iškviestas *Update* metodas visų pirma tikrina eilučių *RowState* savybes. Tada kviečiamos jau iš anksto aprašytos atitinkamos *INSERT*, *UPDATE* ar *DELETE* komandos (SQL užklausos) pakeitimams atlikti duomenų šaltinyje kiekvienai sukurtai, pakeistai ar pašalintai *DataTable* objekto eilutei. Be abejo, dar prieš atliekant pakeitimus būtina susijungti su duomenų šaltiniu.

Duomenų eilutės apdorojamos pagal jų vietą lentelės *Rows* rinkinyje. Jei būtina specifinė eilučių apdorojimo tvarka, reiktų duomenų keitimą išskirti į kelis veiksmus, tokiu atveju į atskirus veiksmus įeina atskirų eilučių keitimas.

Žemiau pateiktas programos išvesties kodas, kuris parodo, kaip atlikti pakeitimus tik naujai sukurtoms eilutėms.

```
DataRow[] arrayOfRows = table.Select("", "",DataViewRowState.Added);
manoDataAdapter.Update(arrayOfRows);
```

Kaip matome, *Update* metodas labai lankstus: galime paduoti per parametrus *DataRow* objektų masyvą, ne tik patį *DataTable* objektą.

Duomenų atnaujinimo lygiagretumo klaidų sekimas ir šalinimas

Duomenų atskyrimas nuo tikrojo duomenų šaltinio paremtas optimistiniu lygiagretumu. Kas atsitinka, kai bandant atlikti lokaliai paruoštus pakeitimus duomenų šaltinyje pasirodo, kad bandome pakeisti duomenų eilutės duomenis, eilutes, kurią neseniai pašalino iš duomenų šaltinio kitas vartotojas, kol modifikavome duomenis lokaliai? Įvyksta duomenų atnaujinimo lygiagretumo konfliktas-klaida.

Kaip išspręsti susidariusias klaidas iš esmės priklauso nuo pačios taikomosios programos, tačiau apibendrinant, išskiriame tris variantus:

Kas pirmesnis, tas gudresnis („*First-Win*“)

Duomenų pakeitimas, kurį norime atlikti, atšaukiamas. Norint pasinaudoti šiuo metodu, reiktų nustatyti *DataAdapter* objekto *ContinueUpdateOnError* savybei reikšmę *true*. Jei savybės *ContinueUpdateOnError* reikšmė yra *true*, klaidos pranešimas nėra išvedamas vartotojui, nors ir įvyksta klaida keičiant duomenis. Toliau vykdomas „grupinis keitimas“ (jei toks keitimas buvo vykdomas) su sekančiomis duomenų eilutėmis.

Klaidos informavimo įrašas išsaugomas atitinkamos eilutės *RowError* savybėje. Pasinaudojus *RowError* savybe galime informuoti vartotoją, su kokiomis eilutėmis iškilo problemos.

Skubėk lėtai (*Last-win*)

Pakeitimai atliekami nepaisant eilutės būklės. Būtina užtikrinti, kad *SQL* pakeitimų atlikimo komanda yra įvykdoma. Pavyzdžiui, *SQL* komandai neradus eilutės, kuriai turi būti atlikti pakeitimai, gali įvykti klaida.

Microsoft Visual Studio turi *ADO.NET* komandų generatorių, kurie automatiškai sukuria konfliktų išvengiantį *SQL* kodą: *WHERE* srityje naudojant parametrus tikrinamas einamųjų ir originalių reikšmių suderinamumas (tikrinimas vyksta duomenų šaltinio pusėje).

Klausk Vartotojo (Ask-the-user)

Šis metodas pasirenkamas, kai aukščiau išvardinti metodai netinka klaidų valdyme. Pagal nutylėjimą, duomenų konfliktas iškviečia *DBConcurrencyException* situaciją. Situacija neiškviečiama , jei *ContinueUpdateOnError* savybės reikšmė nustatoma į *true*. Situacijos klasės *Row* savybė grąžina eilutės su klaida nuorodą. *DataRow* objekto savybių dėka, mes galime gauti *proposed* ir *original* eilutės reikšmes. Galime įvykdyti naują užklausą duomenų šaltiniui ir sužinoti, kodėl įvyko klaida, tada galima generuoti atitinkamą pranešimą vartotojui, pateikiame galimus būdus įvykusiai klaidai ištaisyti. Mūsų tikslas yra pateikti pakankamai informacijos, kad vartotojas galėtų pasirinkti tinkamą klaidos ištaisymo-išvengimo būdą.

Reikia turėti omenyje, kad kartais duomenų atnaujinimo konfliktų sprendimas gali sumažinti visos sistemos veikimo spartą. Gali būti, kad kartais būti visą laiką susijungus su duomenų šaltiniu yra geresnis variantas, nei atskirto duomenų šaltinio modelis. Kokį modelį reikėtų taikyti, priklauso nuo to, kiek vartotojų vienu metu ir kaip dažnai atnaujina duomenis. Jei duomenys atnaujinami retai, be abejo, reiktų rinktis atskirtų duomenų modelį.

7. Išvados

Atskirtų duomenų modelis yra labai svarbus Interneto taikomosiose programose. Tokio modelio dėka padidinama taikomosios programos sparta, išplečiamumas. Atskirtų duomenų modelyje, efektyvus lokalių duomenų panaudojimas atmintyje yra labai svarbus. *.NET* taikomosiose programose *DataSet* objektas yra idealus duomenų konteineris atskirtiems duomenims nuo duomenų šaltinio saugoti ir atlikti veiksmus.

Lyginant senesnes *ADO versijas su ADO.NET* galime rasti daug panašumų, tačiau yra ir didelių skirtumų. Palyginus *Recorset* objektus, pristatytus senesnėse *ADO* versijose, su jų atitikmeniu *ADO.NET* aplinkoje *DataSet* objektu, skirtumas didžiulis spartos, paprastumo naudoti, lankstumo prasme.

Viena esminių *DataSet* objekto savybių yra duomenų būklės išsaugojimas. To nebūtų lengvai galima pasiekti be *DiffGram* formato. Duomenų būklės išsaugojimas yra svarbiausia savybė duomenų atnaujinimo lygiagretumo konfliktų sprendimui užtikrinti. Lanksti *DiffGram*‘os struktūra leidžia taikyti kelis metodus duomenų atnaujinimo lygiagretumo konfliktams spręsti, aiškiai vartotojui suformuluoti įvykusių klaidų aprašymus.

DiffGram formato galbūt vienintelis trūkumas, kad jis neturi schemos informacijos, o ši yra būtina *DataSet* objekto sukūrimui iš *XML* duomenų.

8. Literatūra

Wahlin D. The DataSet Alternative [žiūrėta 20040520]. Prieiga per internetą: http://www.fawcette.com/xmlmag/2002%5F06/magazine/columns/net/default.aspx

„*Devguru.com*“. Using Disconnected Recordsets [žiūrėta 20040520]. Prieiga per internetą: http://www.devguru.com/features/tutorials/DisconnectedRecordsets/tutDisconnRS.html

Grimaldi Sh. Using Disconnected Recordsets [žiūrėta 20040520]. Prieiga per internetą: http://www.4guysfromrolla.com/webtech/080101-1.shtml

Azarbani Ch. Using Disconnected Recordsets [žiūrėta 20040520]. Prieiga per internetą: http://www.webpronews.com/it/database/wpn-20-20011106UsingDisconnectedRecordsets.html

Esposito D. Cutting Edge - Build a Variety of Custom Controls Based on the DataGrid Control [žiūrėta 20040520]. Prieiga per internetą: http://www.wintellect.com/resources/articles

Narayanaswamy A. Introducing ASP.NET [žiūrėta 20040520]. Prieiga per internetą: http://www.codeguru.com/Csharp/.NET/net_asp/tutorials/article.php/c5353/

Esposito D. Applied XML Programming for Microsoft .NET. – Microsoft Press, 2003. – 329 p.

Platt D. Introducing Microsoft.NET 2nd Edition. – Microsoft Press, 2003. – 247 p.

Blum R. C# Network Programming. – Sybex, 2003. - 324 p.

Burgett D., Baute M., Pickett J., Brown E., Sullivan G. A. .Net e-Business Architecture. - Sams Publishing, 2002. – 358p.

9. Terminų ir santrumpų žodynas

- ***Projekto varovai*** - užsakovai, pirkėjai ir kiti sistema suinteresuoti asmenys, jų tarpe projekto autoriai.
- ***WWW*** arba ***WEB*** – viena iš *Interneto* paslaugų – pasaulinė hipertekstinių dokumentų valdymo sistema.
- ***B2B –*** verslas verslui (*Business To Business*), Web projektų tipas.
- ***WEB svetainė*** – hipertekstinių puslapių visuma, susijusi bendru kontekstu, informacija.
- ***IIS –*** informacinis interneto serveris (*Internet Information Sever*).
- ***ASP.NET –*** *Active Server Pages* technologija.
- ***DBVS –*** duomenų bazių valdymo sistema.
- ***DB*** – duomenų bazė.
- ***SQL*** – duomenų bazių valdymo programavimo kalba.
- ***WEB naršyklė*** – programa, skirta peržiūrinėti *Web* tiekiamus hipertekstinius dokumentus.
- ***HTTP*** – protokolas, skirtas perduoti duomenis *Web*'e.
- ***Web svetainės puslapis*** – tam tikro formato hipertekstinis dokumentas.
- ***HTML*** – hipertekstinių dokumentų tipas (plačiai naudojamas *Web*).
- ***Paprastas vartotojas*** – bet kokios sistemos vartotojas, neturintis galimybių keisti sistemos konfigūraciją ir informaciją.
- ***Administratorius*** – bet kokios sistemos vartotojas, turintis visas ar dalines teises keisti sistemos konfigūraciją ir informaciją
- ***IS*** - informacinė sistema
- ***TCP/IP*** - *Trasfer Comunication Protocol/Internet Protocol* – informacijos perdavimo protokolas *Internet* tinkle.
- ***ODBC*** - *Open DataBase Connectivity* – susijungimo su atvirosiomis duomenų bazėmis tvarkyklės.
- ***OS*** - operacinė sistema
- ***LAN*** - *Local Area Network* – lokalus tinklas
- ***UML*** *(Unified Modeling Language)* – modeliavimo kalba, naudojama objektiškai orientuotame projektavime.
- ***ITP*** – Interneto paslaugų tiekėjas.
- ***PĮ*** – programinė įranga.
- ***ADO*** – *Microsoft ActiveX® Data Objects* biblioteka
- ***XML*** – *Extended Markup Language*