Kauno technologijos universitetas

Informatikos fakultetas

ETL procedūrų vizualizavimas naudojant UML veiklos ir klasių diagramas

Baigiamasis magistro projektas

Mantas Abramavičius

Projekto autorius

doc. Lina Čeponienė

Vadovė

Kaunas, 2025

Kauno technologijos universitetas

Informatikos fakultetas

ETL procedūrų vizualizavimas naudojant UML veiklos ir klasių diagramas

Baigiamasis magistro projektas

Veiklos skaitmeninimas ir sistemų architektūros (6211BX009)

Mantas Abramavičius

Projekto autorius

doc. Lina Čeponienė

Vadovė

asist. Algirdas Šukys

Recenzentas

Kaunas, 2025

Kauno technologijos universitetas

Informatikos fakultetas

Mantas Abramavičius

ETL procedūrų vizualizavimas naudojant UML veiklos ir klasių diagramas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs (-usi);

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Mantas Abramavičius

Patvirtinta elektroniniu būdu

Mantas Abramavičius. ETL procedūrų vizualizavimas naudojant UML veiklos ir klasių diagramas. Magistro projektas / vadovė doc. dr. Lina Čeponienė; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Informacijos sistemos, Informatikos mokslai.

Reikšminiai žodžiai: ETL procesai, UML diagramos, SQL analizė, atvirkštinė inžinerija, automatizuota dokumentacija.

Kaunas, 2025. 96 p.

Santrauka

ETL (angl. *Extract, Transform, Load*) procedūros yra sudėtingi duomenų transformavimo procesai, kurių dokumentavimas dažnai atliekamas rankiniu būdu, o tai lemia didelį darbo sąnaudų kiekį ir informacijos neapibrėžtumą. Dėl dažnų procedūrų pokyčių ir ETL procedūrose egzistuojančio SQL programinio kodo sudėtingumo kyla iššūkių užtikrinant jų atsekamumą. Šio darbo tikslas – sukurti priemonę, leidžiančią automatizuoti ETL procedūrų dokumentavimą SQL kodą transformuojant į UML veiklos ir klasių diagramas.

Darbo metu išanalizuoti egzistuojantys dokumentavimo metodai, UML taikymo galimybės bei suformuluota nauja dokumentavimo metodika, kuri remiasi ETL procedūros elementų ir UML modelio komponentų atitikmenimis. Sukurtas įrankis, automatiškai generuojantis UML diagramas pagal šias procedūras, buvo realizuotas naudojant XMI failų generavimą.

Metodika ir įrankis buvo eksperimentiškai ištirti, analizuojant realias ETL procedūras ir vertinant sugeneruotų diagramų atitikimą. Taip pat atlikti interviu su IT specialistais, kurių įžvalgos padėjo įvertinti priemonės pritaikomumą praktiniuose scenarijuose. Tyrimas parodė, kad daugeliu atvejų UML modeliai teisingai perteikia procedūrų logiką, tačiau sudėtingesnės SQL konstrukcijos vis dar kelia iššūkių. Sukurtas sprendimas gali būti naudingas dokumentavimo, analizės ir žinių perdavimo kontekstuose, ypač duomenų migracijos ir sistemų priežiūros projektuose.

Mantas Abramavičius. Visualization of ETL Procedures Using UML Activity and Class Diagrams. Master's Final Degree Project / supervisor assoc. prof. dr. Lina Čeponienė; Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): Information Systems, Computing.

Keywords: ETL processes, SQL, UML diagrams, reverse engineering, automated documentation.

Kaunas, 2025. 96 pages.

Summary

ETL (Extract, Transform, Load) procedures are complex data transformation processes, and their documentation is often performed manually, resulting in high labor costs and informational ambiguity. Frequent changes to procedures and the complexity of the SQL code within them present challenges in ensuring traceability. The aim of this thesis is to develop a solution that enables automated documentation of ETL procedures by transforming SQL code into UML activity and class diagrams.

The study includes an analysis of existing documentation methods, an evaluation of the applicability of UML, and the formulation of a new documentation methodology based on the mapping between ETL procedure elements and UML model components. A tool was developed to automatically generate UML diagrams from SQL procedures using XMI file generation.

The methodology and tool were evaluated experimentally by analyzing real ETL procedures and assessing the accuracy of the generated diagrams. In addition, interviews with IT professionals were conducted to assess the practical applicability of the solution. The study showed that, in most cases, the UML models accurately reflect the logic of the procedures, though more complex SQL constructs still pose challenges. The proposed solution can be useful in contexts such as documentation, analysis, and knowledge transfer, particularly in data migration and system maintenance projects.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

Santrumpos:

- ETL – *Extract, Transform, Load*;
- BPMN – *Business Process Model and Notation*;
- UML – *Unified Modeling Language*;
- XML – *eXtensible Markup Language*;
- DW – *Data Warehouse*;
- BI – *Business Intelligence*;
- XMI – *XML Metadata Interchange*;
- SQL – *Structured Query Language*;
- CASE – *Computer aided software engineering*.

Terminai:

- *Extract, Transform, Load* – duomenų perkėlimo procesas, apimantis duomenų išgavimą iš šaltinio, jų transformavimą į analizei ar ataskaitų teikimui tinkamą formatą ir įkėlimą į duomenų bazę ar duomenų saugyklą (*Data warehouse*);
- *Data Warehouse* – centralizuota saugykla, kurioje saugomi struktūrizuoti ir organizuoti duomenys iš įvairių šaltinių;
- *Legacy system* – sena informacinė sistema arba programinė įranga, kuri naudojama organizacijos veiklai vykdyti, tačiau nebeatspindinti šiuolaikinių standartų ar reikalavimų;
- *Business intelligence* – technologijų, procesų ir įrankių naudojimas veiklos duomenims rinkti, analizuoti ir pateikti. Ši informacija naudojama sprendimams priimti organizacijose;
- *Data Lineage* – duomenų kilmės sekimo principas, leidžiantis identifikuoti, kaip ir iš kur konkreti reikšmė buvo gauta.

Įvadas

Šis darbas priklauso „Veiklos skaitmeninimo ir sistemų architektūrų“ programai, „Informacijos sistemų“ krypčiai.

Darbo problematika ir aktualumas

Duomenų ištraukimo, transformavimo ir įkėlimo (angl. *Extract, Transform, Load,* toliau ETL) procesai yra esminė duomenų inžinerijos dalis taikoma duomenų migracijos projektuose, verslo analitikos sprendimų kūrime, duomenų integracijos, sinchronizacijos ir standartizavimo iniciatyvose. ETL procedūros dažnai įgyvendinamos kaip SQL pagrindu rašytas programinis kodas, kuris apdoroja duomenis iš skirtingų šaltinių ir paruošia juos analizės ar saugojimo sistemoms.

Vienas iš reikšmingų iššūkių – egzistuojančių ETL procedūrų dokumentacijos trūkumas. Kadangi šios procedūros kuriamos laipsniškai, prisitaikant prie nuolat kintančių verslo reikalavimų, jų turinys dažnai keičiasi, o modifikacijos nėra sistemingai fiksuojamos ar vizualizuojamos. Tai apsunkina procedūrų palaikymą, verslo logikos perpratimą, žinių perdavimą naujiems komandos nariams ir audito procesus. Daugelyje organizacijų dokumentacijos parengimas laikomas antraeiliu uždaviniu, todėl, laikui bėgant, sprendimų logika tampa neaiški ar net prarandama.

Ši problema tampa ypač aktuali, kai organizacija kaupia šimtus ar tūkstančius ETL scenarijų, o poreikis juos dokumentuoti atsiranda tik vėliau – kai reikia atlikti peržiūras, sistemų integracijas arba sprendimų priežiūrą. Tokiu atveju procesų analizė tampa rankiniu ir daug laiko reikalaujančiu darbu. Norint efektyviai dokumentuoti tokias procedūras, būtinas sprendimas, galintis automatizuotai generuoti vizualias diagramos formas, perteikiančias pagrindinius duomenų srautus ir verslo logiką.

Darbo tikslas ir uždaviniai

Tikslas palengvinti ETL procedūrų dokumentavimo procesą, pasiūlant UML diagramų generavimo iš ETL procedūrų kodo metodiką. Išskirti šie uždaviniai:

1. išanalizuoti ETL procedūrų dokumentavimo eigą;
2. išanalizuoti dabartines priemones ETL procedūrų dokumentavimui;
3. išanalizuoti UML pritaikymo ETL procedūrų dokumentavimui galimybes;
4. sudaryti ETL procedūrų dokumentavimo metodiką;
5. suprojektuoti bei sukurti ETL procedūrų dokumentavimo įrankį;
6. eksperimentiškai įvertinti sukurtos metodikos ir įrankio veikimą taikant konkrečių ETL procedūrų rinkinį.
7. surinkti naudotojų grįžtamąjį ryšį ir apibendrinti rezultatus, remiantis kokybine analize.

Darbo rezultatai ir jų svarba

Siūlomas metodas padėtų ETL projektų komandoms lengvai dokumentuoti procedūras ir jų pakitimus. Dėl to, informaciniai mainai tarp projekto suinteresuotųjų pusių, vadovybės ir kitų šalių patobulėtų – būtų sumažinta nesusipratimo rizika tarp visų projekto dalyvių. Diagramų failus taip pat būtų galima panaudoti norint įvesti naujus darbuotojus į projekto komandą, taip pateikiant svarbią informaciją apie procedūrų infrastruktūrą, loginius ryšius ir kitas operacijas.

Darbo struktūra

Darbas susideda iš penkių pagrindinių skyrių. Pirmajame skyriuje pateikiama probleminės srities analizė. Apžvelgiami ETL procesų dokumentavimo iššūkiai, analizuojamos esamos priemonės, vertinamas UML taikymo potencialas bei suformuluojami darbo tikslai ir uždaviniai. Antrajame skyriuje aprašoma siūlomos dokumentavimo metodikos struktūra, pagrįsta ETL procedūrų SQL kodo elementų ir UML modelio komponentų atitikmenimis. Trečiajame skyriuje pateikiamas metodo realizacijos projektas ir įgyvendinimo detalės. Ketvirtajame skyriuje aprašoma metodo realizacija bei testavimo išvados. Penktojo skyriaus metu pateikiami eksperimento rezultatai – tiek kiekybinė analizė, taikant metodiką realioms SQL procedūroms, tiek kokybinis vertinimas remiantis interviu su IT specialistais. Galiausiai pateikiamos apibendrinančios išvados, siūlomos tobulinimo kryptys ir įvertinama darbo praktinė vertė.

1. Probleminės srities analizė

1.1. Analizės tikslas

Darbo analizėje siekiama išanalizuoti ETL procedūrų dokumentavimo eigą ir su ja susijusias problemas. Taip pat norima apžvelgti ETL procesą, kaip šias procedūras būtų galimą išversti į diagramas ir kokia galima nauda kyla iš to.

1.2. Tyrimo objektas, sritis ir problema

Duomenų saugyklos yra centralizuotos struktūros, skirtos duomenų saugojimui ir analizei, kuriose saugomi dideli duomenų kiekiai iš įvairių organizacijos šaltinių. Duomenų saugyklos paskirtis yra pateikti vieningą ir integruotą visos įmonės duomenų vaizdą. Pagal šį aspektą, organizacijos vadovybė bei kiti asmenys šiuos lengvai prieinamus duomenis gali analizuoti ir gauti svarbių įžvalgų. Jos skirtos verslo žvalgybos (angl. *business intelligence* – BI) veiklai, įvairių ataskaitų pateikimui ir minėtai duomenų analizei palaikyti. Ši duomenų integracija į vieną centralizuotą vietą yra pasiekiama per ETL procesus.

ETL procesai turi ypatingai didelę svarbą duomenų integracijos, migravimo projektuose, bet taip pat turi ir nemažą iššūkių rinkinį. Neišsprendus šių problemų, projektas geriausiu atveju gali vėluoti arba kilti jo kaina, o blogiausiu – žlugti. Esminės ETL proceso problemos yra silpna duomenų kokybė, integracijų kompleksiškumas bei dokumentacija. Darbas siekia sumažinti pastarosios problemos poveikį projektams.

Ši analogiška veikla dėl savo opios prigimties yra aprašyta keliuose moksliniuose straipsniuose, tačiau visi iš jų turi skirtingas atvaizdavimo metodologijas, pvz. „Modeling ETL Activities as Graphs“ [1], „Representing ETL Flows with BPMN 2.0“ [2], tačiau nepavyko rasti straipsnių kuriuose ETL procedūros būtų generuojamos į UML modelių failus – visi iš jų šį procesą tik atvaizduoja grafiškai. Tyrime siekiama sukurti įrankį, kurio išvestis taptu redaguojama diagrama.

Šio tyrimo objektas – ETL procedūrų dokumentavimas, apimantis jų struktūros, turinio ir dokumentavimo poreikio analizę. Tyrimas orientuotas į ETL procedūrų automatizuoto dokumentavimo metodus ir įrankius, siekiant nustatyti efektyviausius būdus jų formalizavimui. Siūlomas problemos sprendimas – įrankio sukūrimas, leidžiantis automatizuotai analizuoti ETL procedūras ir transformuoti jas į UML veiklos bei klasių diagramas dokumentavimo tikslais.

1.3. ETL procedūrų dokumentavimo analizė

Dokumentavimo eiga

ETL yra duomenų integracijos procesas, kurio pagalba duomenys yra perkeliami iš vienos sistemos ar aplinkos į kitą, dažniausiai dedikuotą duomenų saugyklą, naudojant duomenų transformacijas ir valymą. Šios pirminės sistemos gali varijuoti nuo paprastų *Excel* failų iki stambių, *legacy system* tipo programų. Sudėtingos transformacijos gali būti atliekamos sukuriant vienkartinio panaudojimo programas naudojant *C#, Java* ir kitas programavimo kalbas, arba pačiame duomenų bazės įrankyje naudojant SQL tipo kalbas [3].

Pats ETL pobūdis taip pat gali skirtis tarp projekto dydžio. Duomenų perkėlimo užduotį gali paskirti stambi įmonė su sudėtingais veiklos procesais ir dideliu kiekiu naudojamų sistemų. Tokio tipo projektuose yra ypatingai svarbi ETL procesų kontrolė ir jos dokumentavimas – darbai gali užtrukti

ne vienerius metus, o tai reiškia didelę žmonių kaitą. Dėl šio fakto, procesai kurie nėra dokumentuojami, gali būti interpretuoti skirtingai tarp naujai paskirtų žmonių, kas iššaukia dar didesnę projekto trukmę [3][4].

Priešingai, duomenų perkėlimo užduotis gali būti vykdoma ir vidiniams įmonės interesams. Priklausomai nuo organizacijos, ši procesų kontrolė būna laisvesnė, tačiau dokumentacijos užduotis vis tiek išlieka svarbi. Kaip pavyzdys, organizacija ETL projekte gali perkelti kelių individualių sistemų duomenis į vieną centralizuotą programinę įrangą. Šią dokumentaciją būtų galima dar kartą panaudoti iškilus būtinybei papildomai perkelti duomenis ir iš kitų sistemų [3][4].

Nepaisant projekto dydžio, ETL procesų dizainas susideda iš kelių žingsnių [3]:

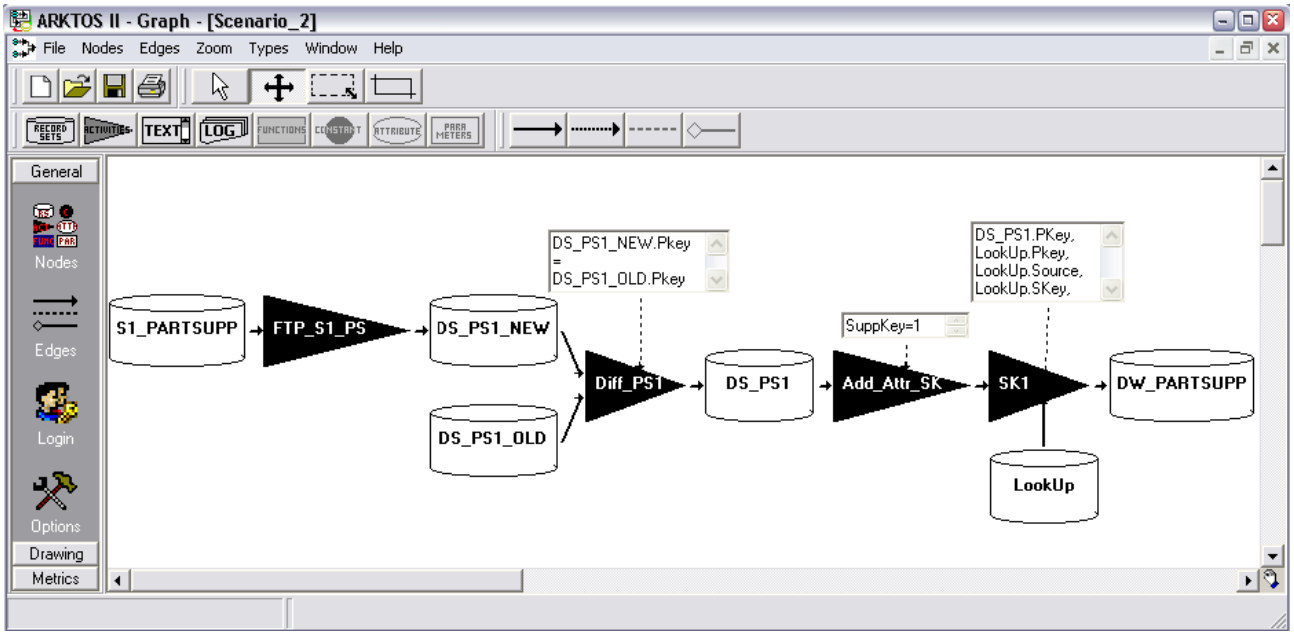
- pasirenkami duomenų išgavimo (dažniausiai keli skirtingi heterogeniniai) šaltiniai, kurie bus naudojami ETL procese;
- iš šaltinių gaunami duomenys transformuojami arba išvedami nauji. Kai kurie esminiai šio etapo žingsniai būtų duomenų filtravimas, reikšmių skaičiavimas, duomenų formatų transformavimas, automatinis identifikatorių generavimas ir kt.;
- duomenys sujungiami, norint kelis šaltinius sujungti į vieną;
- pasirenkama numatomų duomenų įkėlimo aplinka;
- šaltinio duomenų atributai yra susiejami su numatomų duomenų atributais;
- transformuoti duomenys yra įkeliami į numatomą aplinką.

ETL proceso transformacijos žingsnyje taip pat galima valyti duomenis, tačiau šio proceso įrankiai paprastai turi mažai integruotų duomenų valymo galimybių. Duomenų valymas yra susijęs su klaidų ir duomenų neatitikimų aptikimu ir jų pašalinimu, siekiant pagerinti duomenų kokybę. Rankinis ETL procesų kūrimas, priežiūra bei dokumentavimas padidina duomenų saugyklų kūrimo, diegimo, veikimo ir priežiūros kainą [3].

Dabartinės priemonės ETL procedūrų dokumentavimui

Daugelis duomenų bazių valdymo sistemų kūrėjų jau yra įdiegę ETL ir jo dokumentavimo įrankius [5]. Kai kurie iš jų yra *Oracle Warehouse Builder* [6], *IBM InfoSphere DataStage* [7] ir *Microsoft SQL Server Management Studio* [8]. Taip pat yra atliktas nemažas kiekis tyrimų apie ETL įrankius, iš kurių vienas svarbesnių – *Arktos II* [9]. Šis, skirtingai nei didžiųjų informacinių technologijų kompanijų įrankių, yra nemokamai pasiekiamas ir licencijuotas pagal GNU bendrąją viešąją licenciją (angl. *GNU General Public Licence*).

Arktos II (1 pav.) gali apibrėžti šaltinio ir numatomų duomenų saugyklas, transformacijos veiklas bei duomenų srautus. Visa informacija, apibrėžianti ETL procesą, gali būti fiksuojama naudojant formas arba paprastas paspaudimo ir nuvilkimo (angl. *drag-and-drop*) operacijas. Naudotojas gali tyrinėti scenarijuje jau apibrėžtus duomenų šaltinius ir jų veiklą. Šiam grafiškam atvaizdavimui nėra naudojama jokia konkreti modeliavimo kalba – kiekvienas elementas turi savo numatytą reikšmę. Pavyzdžiui, cilindras gali būti duomenų šaltinis, tarpinis duomenų rinkinys arba numatomų duomenų saugykla [9]. *Arktos II* įrankio kūrėjų mokslinis straipsnis [1] taip pat yra apžvelgiamas 1.5.1 skyriuje.



1 pav. *Arktos II* įrankio ETL proceso iliustracija [9].

Tačiau iš praktinės pusės visi minėti stambių organizacijų (*Oracle, IBM, Microsoft*) kurti įrankiai susiduria su išvardintomis problemomis [5]:

- ETL įrankis visuomet yra priklausomas nuo duomenų bazės produkto, todėl jis neturi įvairiapusiškumo;
- ETL įrankiai, sukurti skirtingų įmonių, neturi vieno unifikuoto standarto, ir taip pat nesilaiko viešojo duomenų saugyklų metaduomenų standarto (angl. *Common Warehouse Metamodel*);
- šių įrankių kaina yra didelė, todėl mažo ir vidutinio dydžio organizacijos negali jų įpirkti.

UML pritaikymo ETL procedūrų dokumentavimui galimybės

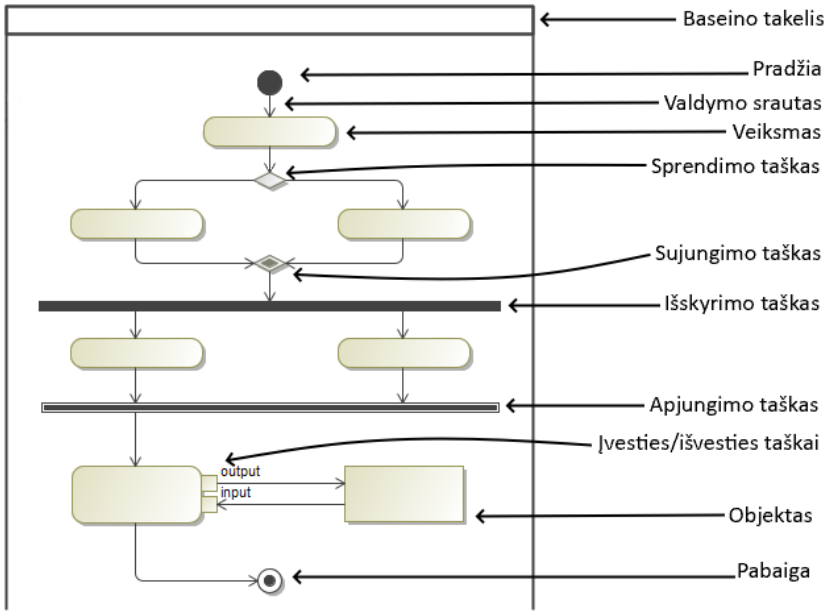
ETL procedūrų pavertimas į UML modeliavimo kalbos diagramas nėra naujas dalykas. Vienas iš pirmųjų bandymų buvo pateiktas dar 2003 metais konferencijos *Conceptual Modeling -- ER 2003* [10] straipsnyje *A UML Based Approach for Modeling ETL Processes in Data Warehouses* [3]. Minėtame ir kituose analogiškuose metoduose esminė problema nurodoma tokia, kad procedūras be naudingo atvaizdavimo būdo yra sudėtinga palaikyti ir prižiūrėti. Šias procedūras galima nesudėtingai perteikti UML veiklos (angl. *activity*) tipo diagramose [11]. Tai diagramos, kurios atvaizduoja dinaminius sistemos aspektus, modeliuojant valdymo srautus iš vienos veiklos į kitą pasirinktame procese. UML veiklos diagramos yra taikomos įvairiose srityse, pvz. programinės įrangos kūrime, veiklos procesų valdyme, sistemų analizėje. Tai universali priemonė norint sumodeliuoti dinamiškus, ir dažniausiai sudėtingus, sistemų aspektus.

Pagrindiniai UML veiklos diagramos elementai (2 pav.) apima veiksmus (angl. *actions*), valdymo srautus (angl. *control flows*), sprendimo ir sujungimo taškus (angl. *decision* ir *merge nodes*), taip pat išskyrimo ir apjungimo taškus (angl. *fork* ir *join nodes*). Veiksmai reprezentuoja konkrečias operacijas ar užduotis, sudarančias modeliuojamą veiklą. Šie veiksmai sujungiami valdymo srautais, kurie nurodo veiksmų seką ir tarpusavio priklausomybę.

Sprendimo taškai žymi vietas, kur srautas gali pasisukti skirtingomis kryptimis, priklausomai nuo nustatytų sąlygų. Sujungimo taškai leidžia apjungti alternatyvius srautus į vieną. Išskaidymo taškai nurodo lygiagrečius veiksmus, o apjungimo taškai sinchronizuoja šiuos srautus atgal į bendrą eigą.

Be to, pavaizduoti įvesties ir išvesties taškai (angl. *InputPin*, *OutputPin*), kurie naudojami nurodyti duomenų srautą tarp veiksmų. Taip pat matomas objekto mazgas (angl. *Object Node*), skirtas konkretiems duomenų vienetams, perduodamiems tarp veiksmų, vaizduoti.

Visi šie elementai modeliuojami baseino takeliuose (angl. *swimlanes*), kurie naudojami veiklai atskirti pagal atsakingus aktorius – tai gali būti konkretūs naudotojai, sistemos ar jų komponentai. Toks suskirstymas padeda aiškiai vizualizuoti, kurios veiklos dalys priskiriamos kuriam vykdytojui.



2 pav. Pavyzdinė UML veiklos diagrama su elementų paaiškinimais.

Įrodyta, kad UML veiklos diagramos tinkamai atvaizduoja ETL procesus koncepciniame lygmenyje. Veiklos modeliavimas akcentuoja sekas ir sąlygas, norint koordinuoti proceso žemo lygio elgesius, o tai leidžia modeliuoti dinaminius ETL aspektus ir duomenis. UML veiklos diagramos veiksmo elementas, reprezentuojantis vieną žingsnį procese, gali būti paverčiamas į ETL procedūros dalį. Pavyzdžiui, įmonės pardavimų suvestinė, kurią galima atvaizduoti kaip veiklos veiksmą, ETL procese išreiškiamas kaip parduotų prekių kiekio sumavimas [11].

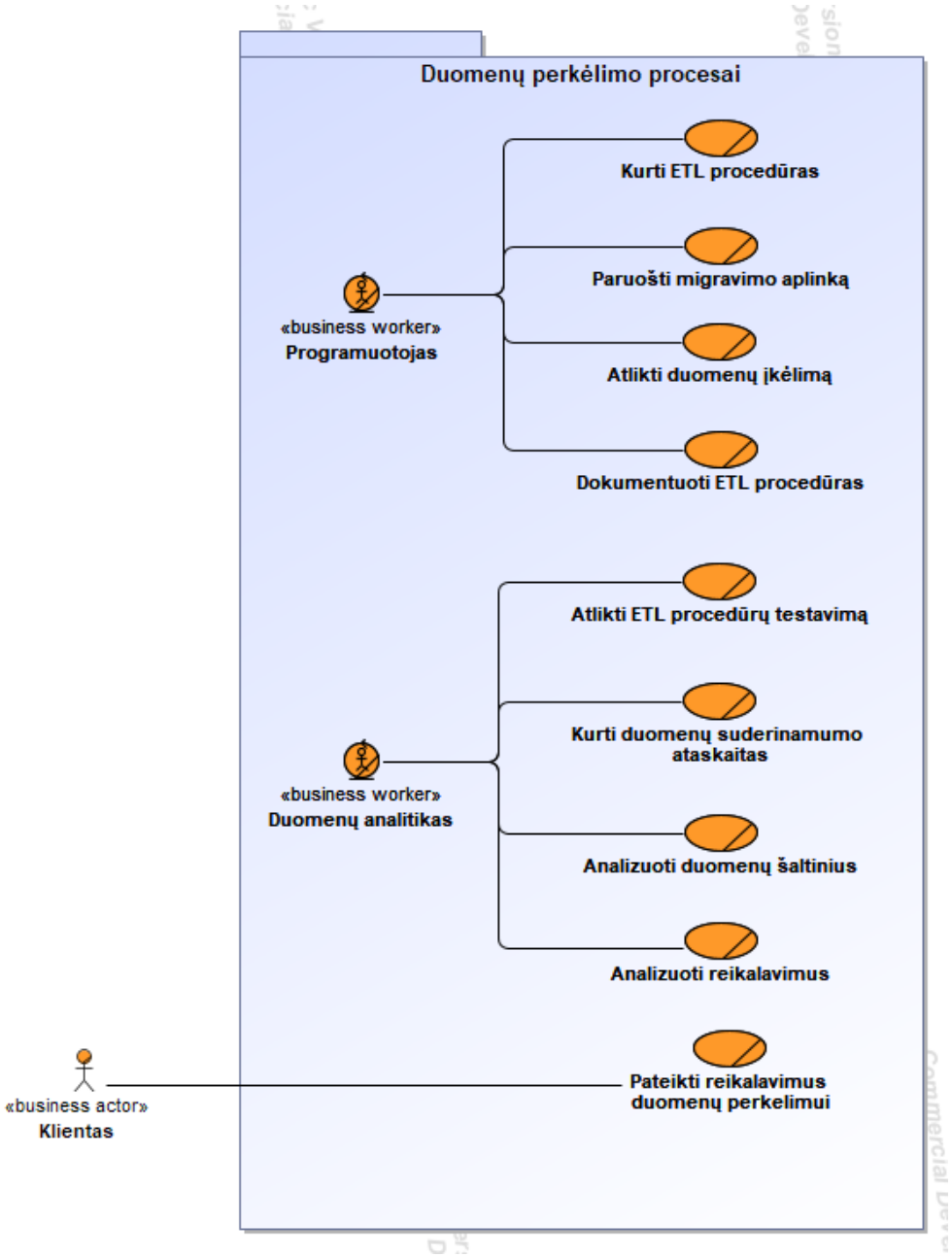
ETL procedūrų dokumentavimo analizė parodė, kad nepriklausomai nuo duomenų perkėlimo projekto apimties, dokumentacijos žingsnis vis tiek išlieka svarbiu aspektu. Dabartiniai egzistuojantys ETL atvaizdavimo įrankiai susiduria su problemomis, kurias yra sunku ištaisyti. Taip pat, bandymai atvaizduoti procedūras į UML modeliavimo kalbos diagramas taip pat nėra naujas dalykas.

1.4. ETL procedūrų dokumentavimo naudotojų analizė

Duomenų perkėlimo procese numatomi naudotojų tipai yra programuotojai, duomenų inžinieriai, veiklos analitikai, arba pastarųjų dviejų pareigybių junginys – duomenų analitikai (3 pav.). Taip pat

aktualus ir klientas, kuris pateikia reikalavimus visam projektui, ir kuriam metodo sukurti diagramos failai taip pat būtų pravartūs.

3 pav. pateikta įprasta veiklos panaudojimo diagrama duomenų migracijos projekte. Svarbu paminėti, kad tai nėra vienintelis duomenų migracijos vykdymo būdas. Kaip buvo minima 1.3 skyriuje, šie projektai gali būti vidiniai – be išorinio kliento, ir/arba mažos apimties, kur tokie aspektai kaip suderinamumo ataskaitų pateikimas nėra svarbūs ar net nėra atliekami.



3 pav. Duomenų migracijos projekto su išoriniu užsakovu veiklos panaudojimo diagrama.

Naudotojų atsakomybės yra išskirstytos į kelias pagrindines:

- programuotojai ir duomenų analitikai gali pasinaudoti šio planuojamo įrankio galimybėmis norint sekti ETL procedūrų progresą ir kaitą. Taip pat norint apmokyti naujus darbuotojus, kurie įsitraukia į projekto eigą vėliau;

- duomenų analitikai diagramas galėtų parodyti klientams ir jų informacinių technologijų specialistams norint aptarti procedūrų korektiškumą ir suderinamumą tarp sutartų projekto reikalavimų;
- duomenų migracijos projektuose dažnai sutinkama praktika yra gauti kliento sutikimą dėl atlikto darbo po kiekvieno ETL ciklo iteracijos. Šios diagramos galėtų tapti kaip priedas ir įrodymas apie tinkamai atliktą darbą suinteresuotosioms pusėms.

Visi šio siūlomo metodo naudotojai turi aukštą patirties lygį informacinėse technologijose. Taip yra dėl to, nes jis numatomas asmenims, kurie tiesiogiai dirba ETL projektuose – plačiajai rinkai šis įrankis nėra pritaikytas, nes tam nėra objektyvaus pagrindimo. Dėl šios siauros specifikos jis nebus plačiai žinomas ne ETL specialistų rate, tačiau dėl įvardintos problematikos galėtų tapti labai svarbia dalimi ETL projektuose.

Numatomos naudotojų kiekis priklausomas nuo vykdomo projekto dydžio. Tai galėtų būti vidinis įmonės projektas, kuriame darbų komanda ir duomenų kiekis nebūtų pernelyg dideli. Iš kitos pusės, galbūt projektas vykdomas tarp dviejų ar daugiau stambių informacinių technologijų įmonių, kuriuose darbų komandos narių kiekis galėtų viršyti net kelis tuzinus. Tiek pirmajam, tiek antrajam variantui dokumentavimo įrankis reikalingas, tačiau pastarajam, dėl didesnės svarbos, šis įrankis yra aktualesnis. Teigiama, kad ir numatomas naudotojų kiekis taptų daugiau nei 30 žmonių viename projekte.

1 lentelė. ETL procedūrų problemų aprašymas.

Problema	**Esama situacija**
Sudėtinga sekti pakitimus ETL procedūrose.	Po kiekvienos ETL ciklo iteracijos procedūros keičiasi ir evoliucionuoja. Ankstesniame cikle dar nebuvę duomenys gali atsirasti vėlesniame, todėl reikia nuolat peržiūrėti ir adaptuoti esamą kodą. Jei nėra naudojami versijų valdymo įrankiai ar aiškūs pakeitimų žymėjimo principai, pokyčių sekimas tampa sunkiai valdomas, ypač žmonėms, kurie procedūrų nekūrė.
ETL procedūros dažniausiai nėra dokumentuojamos dėl laiko stygiaus.	Dauguma projektų, ypač IT srityje, vykdomi griežtais terminais, todėl didžiausias prioritetas skiriamas funkcionalumo įgyvendinimui. Dokumentavimas dažnai laikomas antraeiliu darbu. ETL kūrėjai visą dėmesį skiria kodo rašymui, o dokumentacijai laiko nelieka.
Sudėtinga pateikti procedūrų logiką suinteresuotosioms pusėms, kurios neturi programavimo patirties.	Aukštesnioji vadovybė ar verslo analitikai, siekiantys peržiūrėti ETL procesų logiką ar įvertinti jų atitikimą verslo taisyklėms, dažnai neturi techninių žinių. Kadangi SQL kodas jiems yra neįskaitomas, sprendimų teisingumas ar atsekamumas lieka nepatikrintas.

Siūlomas artefaktas padeda išspręsti šias 1 lentelėje išskirtas problemas:

- palengvinamas ETL procedūrų pokyčių sekimas. Įrankis gali būti naudojamas kiekvieno ETL ciklo pabaigoje siekiant archyvuoti ir vizualizuoti procedūrų versijas. Tokiu būdu sudaromos sąlygos retrospektyviai peržvelgti įvykusius pakeitimus ir geriau suprasti jų atsiradimo kontekstą;
- užtikrinamas ETL procedūrų dokumentavimas. UML veiklos ir klasių diagramos generuojamos automatiškai pagal pateiktą ETL procedūros SQL kodą. Tai leidžia išvengti

rankinio dokumentavimo, sumažina klaidų tikimybę ir padeda efektyviau panaudoti komandos išteklius;
- supaprastinamas procedūrų logikos pateikimas. Grafinis UML atvaizdavimas leidžia geriau suprasti procedūros struktūrą ir logiką net tiems naudotojams, kurie neturi programavimo patirties. Tokios diagramos gali būti naudingos tiek techniniams, tiek netechniniams suinteresuotiesiems asmenims.

Naudotojų analizė rodo, kad šis sprendimas daugiausia aktualus ETL specialistams, tačiau net ir riboto naudotojų rato atveju įrankis gali tapti reikšminga pagalbine priemone ETL projektuose, ypač siekiant efektyviai spręsti dokumentacijos ir palaikymo problemas.

1.5. ETL procesų vizualizavimo metodų apžvalga

Šiame skyriuje pateikiama esamų ETL procedūrų dokumentavimo ir modelių generavimo metodų analizė. Atrinkti metodai apima tiek akademinius, tiek praktinius sprendimus, kurių tikslas – vizualiai perteikti duomenų transformavimo logiką arba automatizuoti jos analizę. Kiekvienas iš pasirinktų metodų išryškina skirtingą požiūrį į ETL procedūrų interpretavimą, pvz. nuo modelių generavimo pagal grafų teoriją iki formalizuotų specifikacijų ar vizualizacijos BPMN ir UML diagramomis. Analizei pasirinkti metodai buvo atrinkti atsižvelgiant į jų taikymo pobūdį, naudojamas technologijas, vizualizacijos lygį ir ryšį su šio metodo kryptimi. Vieni jų pabrėžia struktūrinių komponentų formalią analizę, kiti – praktinį įrankių taikymą ar grafinės informacijos perteikimą.

Be mokslinių metodų, analizėje pristatoma ir kodo analizės biblioteka, kuri, nors ir nėra akademinis šaltinis, šiame darbe atlieka svarbų vaidmenį. Šis įrankis naudojamas praktinėje dalyje ETL procedūrų kodo analizavimui ir jo struktūros interpretavimui.

Analizuojami metodai leidžia išskirti esminius skirtingų problematikų sprendimo bruožus, kuriuos vėliau galima palyginti su šiame darbe siūlomu metodu. Ši analizė padeda įvertinti siūlomo sprendimo išskirtinumą, taikymo ribas ir galimus privalumus kitų kontekste.

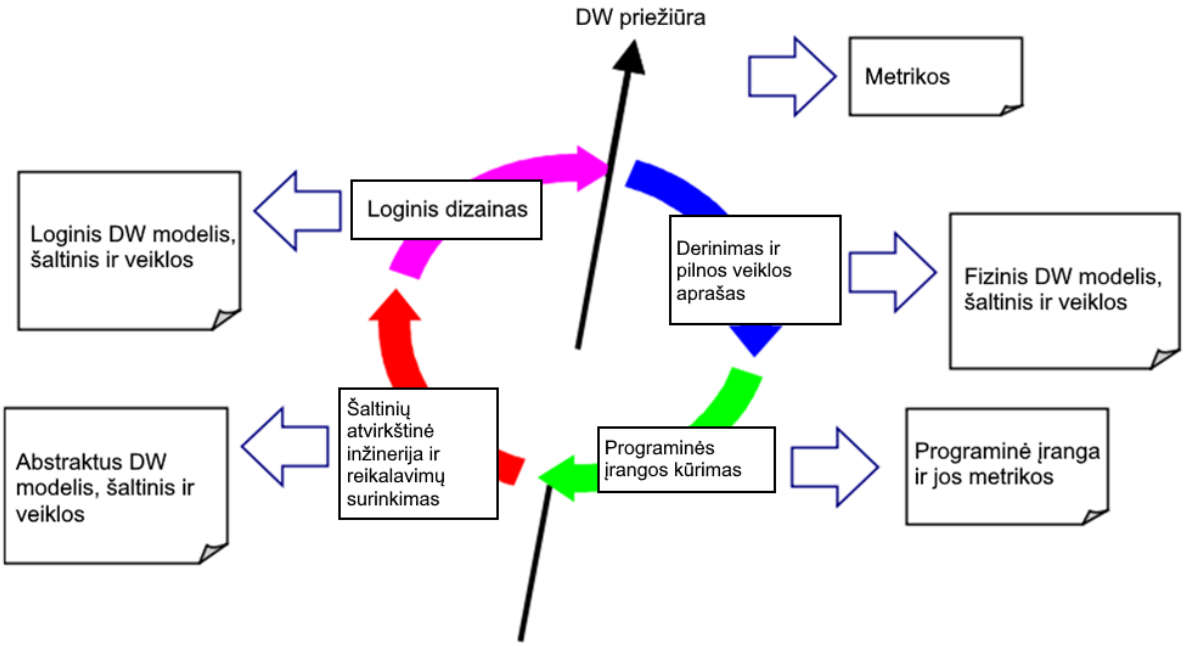
1.5.1. ETL modeliavimo grafais metodas

Vienas iš teorinių pagrindų, susijusių su ETL procesų analizės ir modelio kūrimo metodikomis, yra ETL modeliavimo grafais metodas [1]. Šiame tyrime siūloma sisteminga ETL procesų formalizavimo metodika, grindžiama grafų struktūros naudojimu, kurioje apibrėžiami du pagrindiniai elementai – veiklos (angl. *activities*) ir duomenų saugyklos (angl. *data stores*).

Šio metodo esmė – ETL proceso struktūrizavimas per formalų grafų modelį, kuriame veiklos atitinka duomenų transformacijos operacijas (pavyzdžiui, duomenų ištraukimas, transformavimas ar įkėlimas), o duomenų saugyklos žymi šaltinius, tarpinius rezultatus ar galutinius įrašus. Taip sukuriamas vadinamasis architektūros grafas (angl. *architecture graph*), kuriame nurodomi skirtingų komponentų ryšiai ir priklausomybės. Ryšių tipai, tokie kaip *Provider, Part-of, Regulator* ir *Instance-of* leidžia detaliai aprašyti, kaip duomenys keliauja tarp veiklų ir saugyklų bei kokios yra jų sąveikos.

Modeliavimas grindžiamas ir papildomais aspektais, tokiais kaip duomenų kilmės (angl. *data lineage*) stebėjimas bei procesų efektyvumo analizė naudojant specialias metrikas – atsakomybės laipsnį (angl. *responsibility*) ir priklausomybės laipsnį (angl. *dependence*). Šios metrikos leidžia vertinti, kiek tam tikros veiklos yra svarbios bendrame ETL proceso kontekste bei kaip galimi pokyčiai paveiktų visą sistemą.

Tiriant ETL procesų modeliavimą, metode taip pat pateikiamas viso duomenų sandėlio (DW) projektavimo ir valdymo ciklas, kuriame ETL veiklos užima itin svarbią vietą (4 pav.). Ši schema apima duomenų šaltinių analizę, loginį ir fizinį modeliavimą, programinės įrangos kūrimą, metrikų taikymą bei duomenų sandėlio priežiūrą. Tokiu būdu nurodoma, kad ETL procesai yra neatsiejama duomenų sandėlio gyvavimo ciklo dalis, o jų kokybė ir struktūra tiesiogiai veikia sistemos efektyvumą.



4 pav. DW gyvavimo ciklas ir jo ETL proceso iliustracija [1].

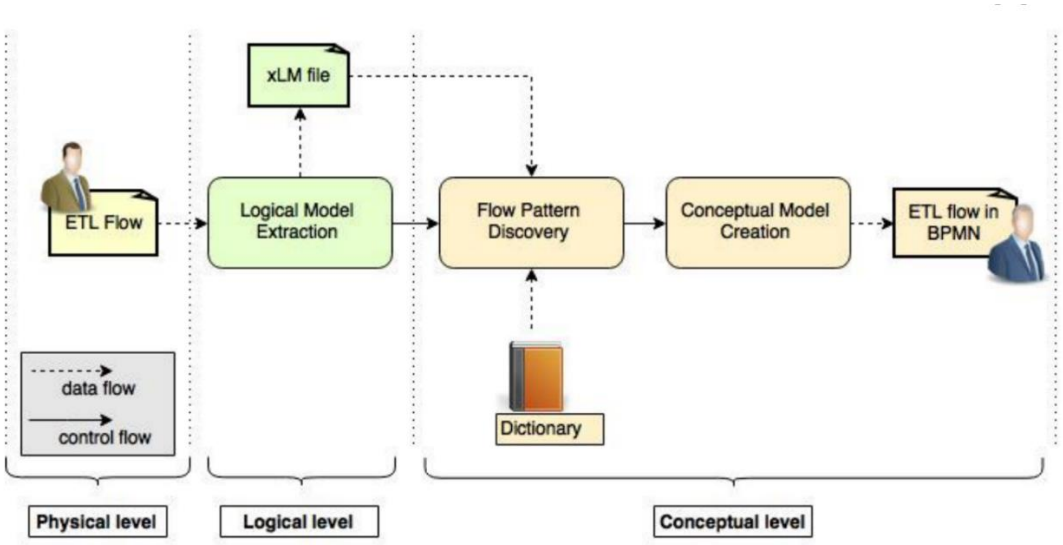
Šio darbo kontekste, ETL modeliavimo grafais metode pateiktos idėjos yra svarbios dėl kelių priežasčių. Visų pirma, jos paaiškina būtinybę ETL procesus vaizduoti ne kaip pavienius užklausų fragmentus, bet kaip nuoseklias, logiškai susijusias veiklų sekas. Antra, architektūros grafų struktūra atitinka pagrindinį šio darbo tikslą – kurti formalų ETL procedūrų vaizdavimą naudojant UML veiklos ir klasių diagramas. Nors šio darbo metodika nėra tiesioginis grafų konstravimas, pagrindinė idėja – sukurti analizuojamą, vizualizuojamą ETL procesų modelį visiškai atitinka grafų metode pasiūlytus principus.

1.5.2. ETL procedūrų atvaizdavimas BPMN 2.0 modeliavimo kalboje

Vienas iš labiausiai išplėtotų metodų, siūlančių konceptualizuoti ETL procesus, pateikiamas ETL procedūrų atvaizdavimo BPMN 2.0 modeliavimo kalboje metode [2]. Tyrimo tikslas – automatizuoti loginio ETL srauto transformaciją į aukštesnio lygmens konceptualų modelį, pasitelkiant BPMN 2.0 notaciją. Tokia konversija padeda ne tik atsisakyti techninių detalių pertekliaus, bet ir pateikti procesų logiką naudotojams suprantama forma.

Pateiktas sprendimas remiasi trijų abstrakcijos lygių (fizinio, loginio ir konceptualaus) atskyrimu. Nors klasikinėje duomenų bazių inžinerijoje modeliavimas vyksta nuo konceptualaus iki fizinio lygmens, čia pasitelkiama atvirkštinė inžinerija – transformacijos vykdomos kryptimi nuo fizinio prie konceptualaus lygmens.

Loginio modelio išgavimas iš ETL srauto (5 pav.) atliekamas konvertuojant jį į XML formato loginį aprašą, vadinamą xLM. Šis aprašas vaizduoja ETL procesą kaip orientuotą grafą, kuriame kiekvienas mazgas (angl. *node*) turi informaciją apie operacijos tipą, įgyvendinimo tipą, įvesties/išvesties schemas ir papildomus požymius. Tokia struktūra leidžia tiksliai apibrėžti duomenų srautus ir operacijų charakteristikas.



5 pav. ETL procedūrų atvaizdavimo BPMN 2.0 modeliavimo kalboje architektūros diagrama [2]

Vėlesnis etapas – tai srauto šablonų (angl. *flow pattern*) atpažinimas, kuris vykdomas remiantis specialiu žodynu, aprašančiu galimus paprastus ir sudėtinius ETL šablonus. Žodyne naudojama formalizuota gramatika, leidžianti aiškiai apibrėžti, kokie operacijų tipai ir jų seka sudaro konkretų šabloną, ir kaip jie turėtų būti atvaizduoti BPMN modelyje. Šiame žodyne numatyta galimybė plėsti šablonų aibę.

Kai srauto struktūra atitinka tam tikrą žodyno aprašą, vykdoma konversija į BPMN elementus: užduotis, procesų srautus, vartus ir kt. Toks modelis leidžia ne tik vizualiai suprantamai pateikti duomenų apdorojimo logiką, bet ir sukurti modelius, kurie artimi kitų suinteresuotųjų pusių suvokimui. Svarbu paminėti, kad nors nagrinėjamo darbo sprendimas orientuojasi į bendrą modelį, jis taip pat leidžia detaliau įsigilinti į specifinių transformacijų interpretavimą ir jų tinkamą pateikimą.

Pateikta architektūra (5 pav.) aiškiai atskleidžia transformacijos žingsnius: nuo fizinio ETL srauto iki BPMN modelio, įskaitant loginį xLM generavimą, šablonų identifikavimą ir konceptualaus modelio konstravimą. Tai parodo, kaip apdorojamas originalus duomenų transformacijos srautas, galiausiai sukuriant galutiniam diagramos naudotojui pritaikytą grafinę išraišką.

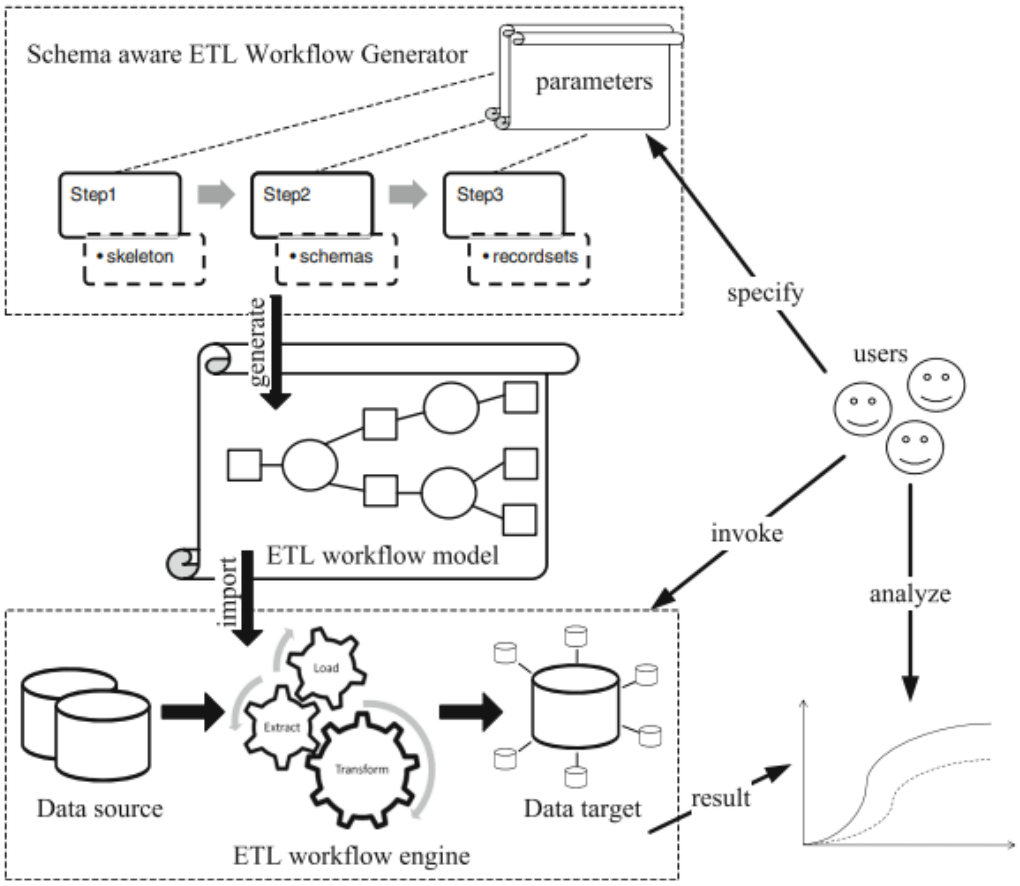
Pažymėtina, kad nors teorinis modelis palaiko įvairių SQL sistemų išraiškas, šiame darbe ir kuriamame sprendime naudojama tik SQL Server dialektui skirta xLM išraiška, dėl šios technologijos dominavimo nagrinėjamoje aplinkoje.

1.5.3. ETL darbo eigos generatorius

Vienas iš pažangesnių metodų ETL procesų analizės srityje yra ETL darbo eigos generatoriaus pasiūlymas [12]. Šio metodo esmė – ETL atvejų generavimas, kurie atspindi tiek kontrolės srautą,

tiek duomenų srautą. Tai leidžia imituoti realius ETL darbo srautus, kartu išlaikant prasmingą schemų ir įrašų struktūrą.

Šio sprendimo architektūra (6 pav.) remiasi trimis pagrindiniais žingsniais. Karkaso kūrimas, kuriame apibrėžiamas veiklų skaičius, jų tipai ir tarpusavio sąveikos taisyklės, sukuriant grafą, kuris atspindi kontrolės srautą. Schemos generavime nustatomi įėjimo/išėjimo atributų kiekiai bei jų tarpusavio priklausomybės, kurios kyla iš veiklų semantikos. Įrašų generavimas remiasi semantiniais reikalavimais, kuriuose suformuojami duomenų rinkiniai, atitinkantys apibrėžtas schemas bei veiklos logiką (pvz., SELECT operacijos filtravimo sąlygas).



6 pav. ETL darbo eigos generatoriaus architektūros diagrama [12]

Skiriamasis bruožas – ETL generatoriaus gebėjimas kurti tiek kontrolės srautą, tiek duomenų srautą, pritaikytą pagal naudotojo nurodytus parametrus, pavyzdžiui, veiklų tipus, schemų atributų kiekius. Dėl to, generuojami atvejai yra tinkami tiek funkcinių galimybių testavimui, tiek ETL našumo analizei. Naudojamos veiklų klasifikacijos remiasi ETL veiklų taksonomija. Kiekviena veikla turi griežtai apibrėžtus įėjimo/išėjimo ryšius, atributų tipus bei jų priklausomybes. Tai leidžia minėtus atvejus adaptuoti prie realių scenarijų.

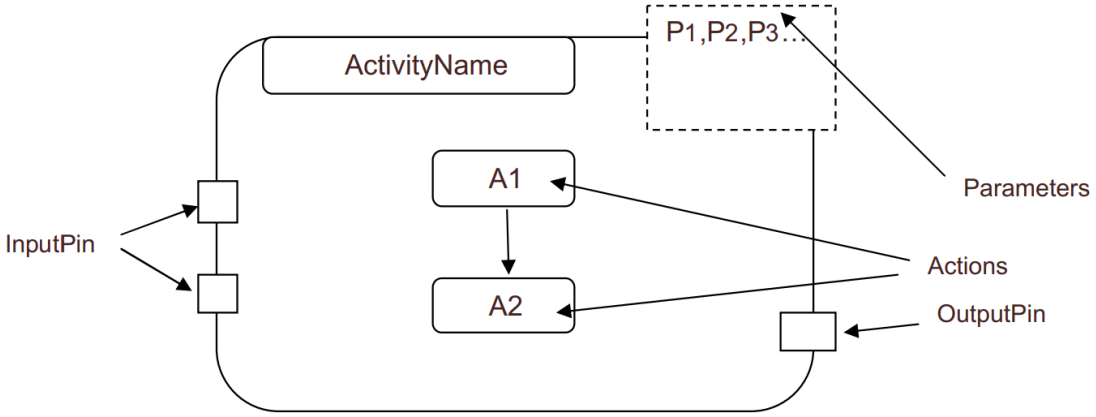
Svarbu pažymėti, kad generuoti darbo srautai gali būti suformuoti įvairiais šablonais keičiant veiklos parametrus. Tokiu būdu galima imituoti specifinius ETL projektavimo atvejus.

Apibendrinant, šis metodas pasižymi aukštu lankstumo lygiu, leidžiančiu naudotojams ne tik modeliuoti, bet ir automatiškai generuoti korektiškus, įvairaus sudėtingumo ETL srautus testavimo tikslams.

1.5.4. Eksperimentų rinkinys, skirtas duomenų saugyklose esančių ETL procesų UML veiklos diagramoms patvirtinti

Metodas [11] buvo pasirinktas dėl panašumų su šio darbo siekiamos diagramos išvesties modeliavimo kalbos – UML. Pateikiamas priemonių rinkinys ETL procesų struktūriniam sudėtingumui vertinti, taip pat siūlomas formalių taisyklių rinkinys procedūrų transformavimui į UML veiklos diagramos elementus. Šis metodas pasirinktas dėl glaudžių sąsajų su šio darbo kuriamos dokumentavimo priemonės išvesties formatu – UML veiklos diagrama. Tyrime nagrinėjami sprendiniai leidžia ETL veiklas struktūrizuoti, įvertinti jų sudėtingumą bei pateikti formalų vizualizavimo pagrindą. Šie aspektai yra atitinkamai įvertinami empiriškai bei teoriškai. Eksperimentų rinkinys nebus analizuojamas, nes jis nėra susijęs su šiuo tyrimu.

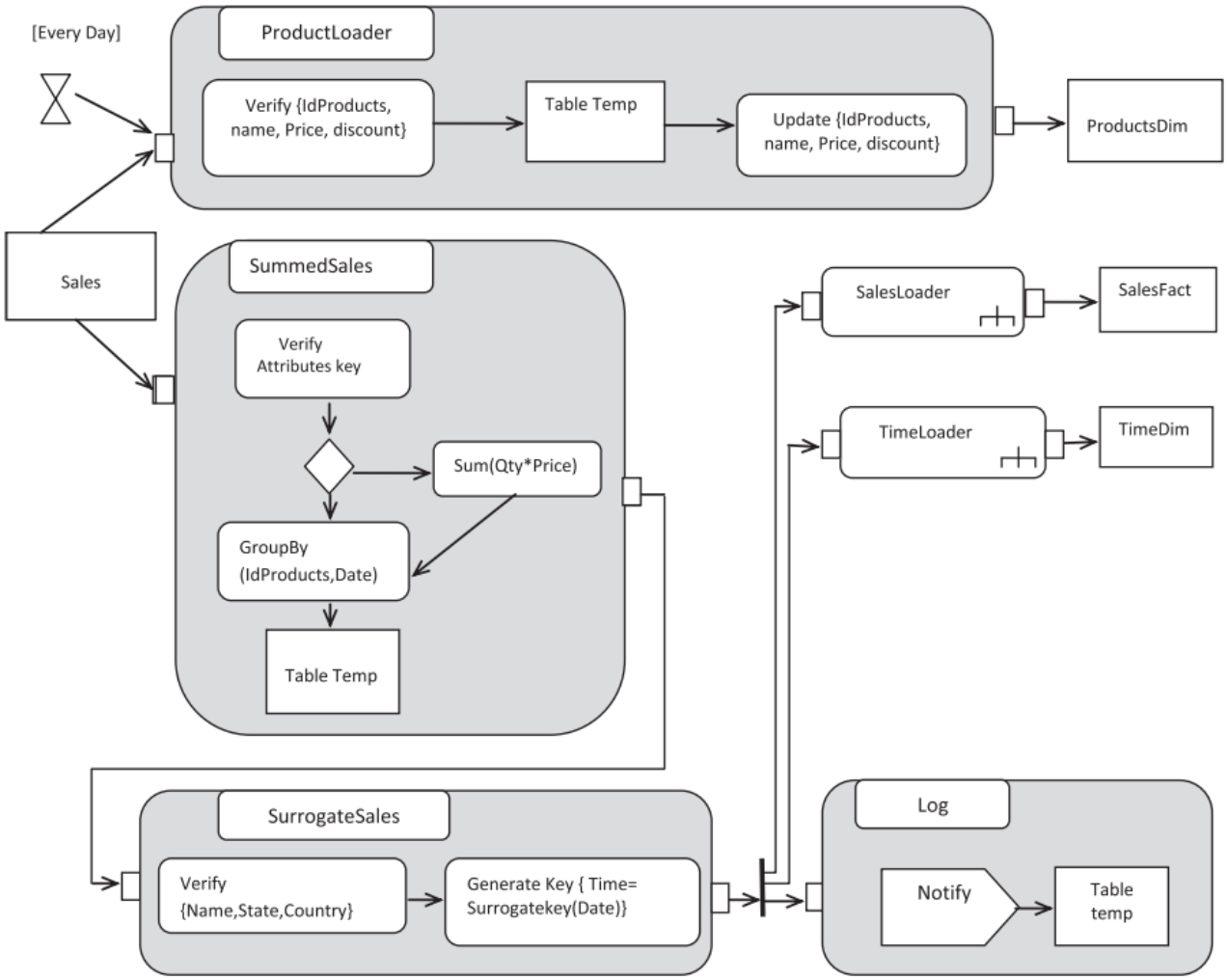
Nagrinėjamas metodas siūlo transformuoti ETL procedūrų operacijas, tokias kaip agregavimas (angl. *Aggregation*), filtravimas (angl. *Filter*), sujungimas (angl. *Join*) ir kitas, į atitinkamus UML veiklos diagramos elementus, pasitelkiant UML metamodelio klasifikacijas (7 pav.). Šie elementai grindžiami UML veiklos diagramos metamodeliu, o konkretūs veiksmai atspindi ETL proceso vykdymo seką. Diagramose naudojami įvesties smeigtukai (angl. *InputPin*) yra skirti duomenų gavimui iš duomenų šaltinių – duomenų bazių ar kitų saugyklų – o išvesties smeigtukai (angl. *OutputPin*) perduoda transformuotus duomenis tolimesniems veiksmams.



7 pav. Šablonas, skirtas nurodyti ETL procesus naudojant UML veiklos diagramų elementus [11]

Naudojant šiuos veiksmus, metodas pateikia galimą ETL procedūros pavertimą į UML veiklos diagramą (8 pav.). ETL procedūros yra išskirstytos į veiksmus, kurios turi sąryšius su kitomis procedūromis arba duomenų saugyklomis. Šie veiksmai, UML vadinami struktūrizuotais veiksmo

mazgais (angl. *structured activity node*), savo viduje turi smulkesnius veiksmus, tokius kaip duomenų patikrinimas, duomenų atnaujinimas ir kt.



8 pav. ETL proceso pavyzdys, pavaizduotas UML veiklos diagrama [11]

Nors analizuojamas metodas yra orientuotas į eksperimentinį metrikų taikymą ETL veiklos diagramoms, jo struktūrinis modelis bei transformacijos taisyklės gali būti laikomos tinkamu pagrindu ETL procedūrų pavertimui į UML veiklos diagramas ir šiame siūlomame metode. Ši metodika leidžia sistemingai identifikuoti, kokios ETL procedūrų dalys gali būti konvertuojamos į UML elementus, tokiu būdu užtikrinant formalų ir nuoseklų dokumentavimo procesą.

1.5.5. ETL procedūrų modeliavimas naudojant BPMN ir reliacinę algebrą

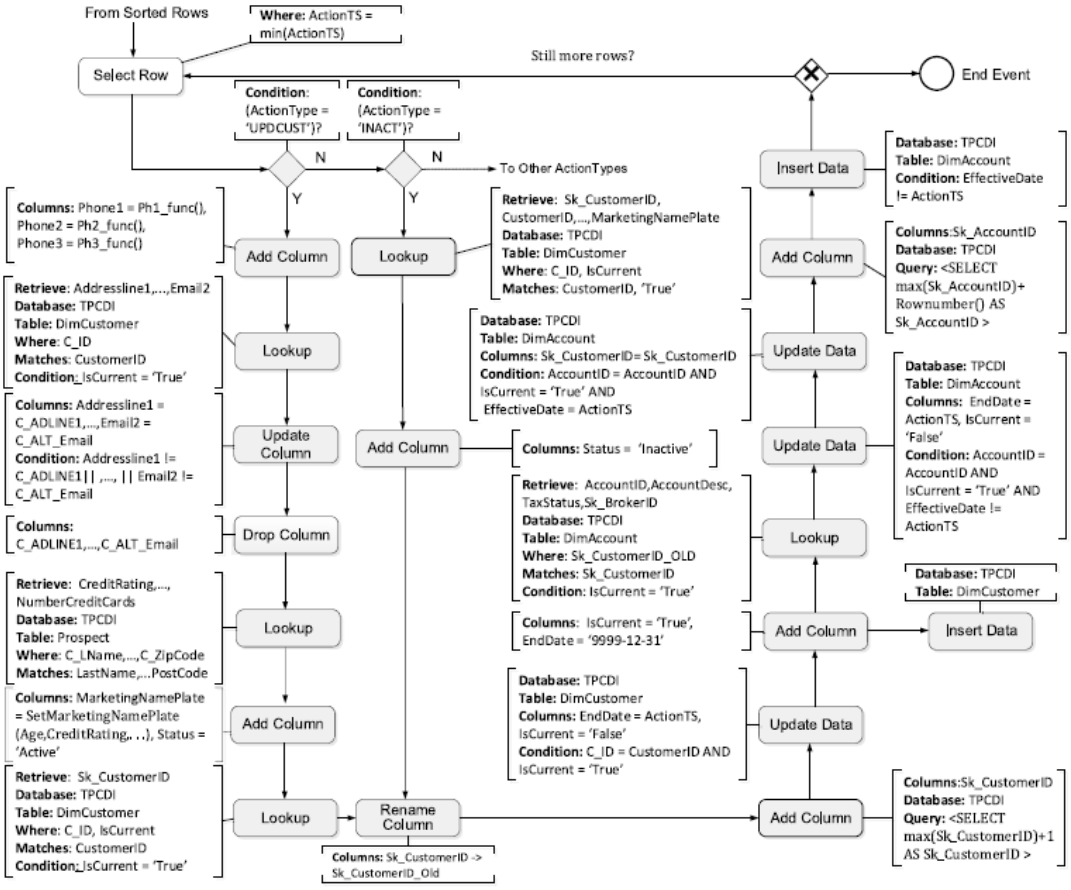
Viena iš nagrinėtų metodikų [20] siūlo ETL procesus vaizduoti pasitelkiant verslo procesų modeliavimui skirtą BPMN kalba, kuri yra papildoma formalia duomenų transformacijų specifikacija, pagrįsta reliacinės algebros principais. Tokia struktūra sujungia procesų atvaizdavimą su tiksliu duomenų apdorojimo veiksmų apibrėžimu. Kiekvienas modelio elementas atspindi konkretų ETL proceso žingsnį – nuo duomenų paruošimo iki jų įkėlimo į tikslinę sistemą.

Vizualioji dalis, vadinama „BPMN4ETL“, skirta modeliuoti duomenų apdorojimo žingsnius grafiškai, kiekvieną veiksmą atvaizduojant atskiru simboliu, išlaikant bendrą loginę seką ir

priklausomybes tarp veiklų. Kiekvienas tokio modelio elementas atspindi konkretų duomenų transformacijos aspektą – duomenų paiešką, papildymą, filtravimą ar įkėlimą į galutinę sistemą.

Norint sumažinti dviprasmybių riziką ir užtikrinti tikslų operacijų apibrėžimą, grafinis modelis papildomas formalia duomenų transformacijos specifikacija, išreikštą išplėstais reliacinės algebros operatoriais. Tokia struktūra leidžia aprašyti ETL logiką taip, kad ji būtų aiški ne tik žmogiškam vartotojui, bet ir vykdytina automatizuotomis priemonėmis, nepriklausomai nuo konkrečios SQL kalbos ar platformos.

Pavyzdinėje diagramoje (9 pav.) pateikiamas kliento duomenų atnaujinimo scenarijus, kuriame perteikiami pagrindiniai duomenų apdorojimo žingsniai. Modelis leidžia lengviau suprasti procesų eigą tiek techniniams, tiek verslo vartotojams.



9 pav. BPMN4ETL diagrama, skirta kliento atnaujinimui [20]

Ši metodika iš dalies sutampa su šiame darbe siūlomu sprendimu, nes abi jos siekia pavaizduoti ETL logiką konceptualiame lygyje, atskiriant ją nuo konkretaus SQL įgyvendinimo. Tačiau esminis skirtumas yra tai, kad siūlomo metodo atveju orientuojamasi į atvirkštinį procesą – ne nuo modelio link kodo, o nuo esamo kodo link UML diagramų, kurios galėtų būti naudojamos dokumentavimo tikslais.

1.5.6. Esamų sprendimų palyginimas

Norint įvertinti egzistuojančių ETL procesų dokumentavimo metodų ir priemonių tinkamumą, buvo atlikta penkių metodų analizė pagal konkrečius, praktinio taikymo ir techninio įgyvendinimo

kriterijus. Kiekvienas metodas buvo vertinamas atsižvelgiant į tai, ar jis pateikia realų įrankį, ar leidžia procedūras vizualizuoti, ar galima automatizuotai generuoti diagramas bei kokia modeliavimo kalba yra naudojama. Taip pat buvo nagrinėta, ar metodai palaiko atvirkštinę inžineriją iš ETL procedūros, bei kitus aspektus.

Be to, buvo įtraukti kriterijai, susiję su praktiniu pritaikomumu: ar įrankis gali apdoroti sudėtingas procedūras, ar leidžia išsaugoti diagramas failo formatu, ar gali būti integruojamas su kitais modelių kūrimo įrankiais, ir ar metodas leidžia sekti duomenų kilmę.

Metodų pavadinimai 2 lentelėje pateikti originaliu pavadinimu.

2 lentelė. Esamų sprendimų palyginimas

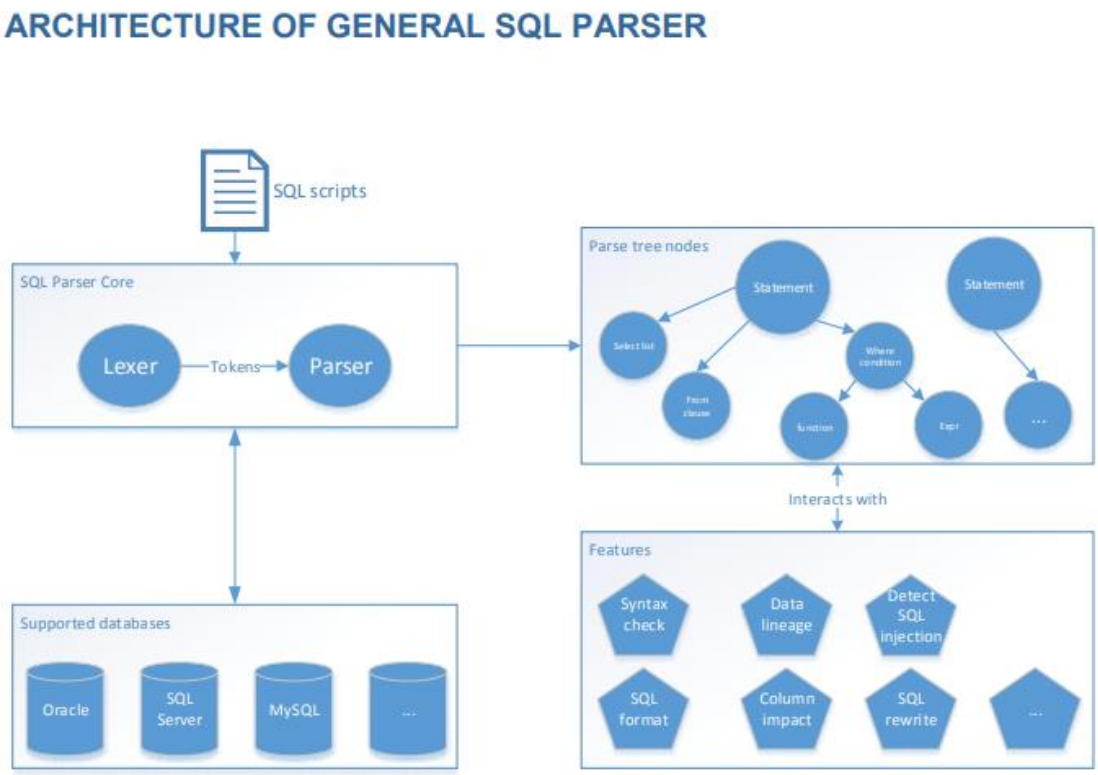
Palyginimo kriterijus	***Modeling ETL Activities as Graphs***	***Representing ETL Flows with BPMN 2.0***	***A schema aware ETL workflow generator***	***A family of experiments to validate measures for UML activity diagrams of ETL processes in data warehouses***	***Design and Implementation of ETL Processes Using BPMN and Relational Algebra***
Pateiktas realus įrankis	Ne	Taip	Taip	Ne	Ne
Įrankis laisvai prieinamas	Ne	Ne	Ne	Ne	Ne
Naudotojui nereikia sukurti pirminių procedūrų taisyklių	Ne	Ne	Taip	Ne	Taip
Yra galimybė procedūras pateikti vizualiai	Taip	Ne	Taip	Taip	Taip
Yra galimybė procedūras pateikti kaip diagramos failą	Ne	Taip	Ne	Ne	Taip
Diagramos yra pateikiamos pačiame įrankyje	Ne	Ne	Taip	Ne	Ne
Įrankis gali sugeneruoti sudėtingų procedūrų diagramas	Ne	Ne	Ne	Ne	Ne
Diagramos modeliavimo kalba	Grafai	BPMN	Grafai	UML	BPMN4ETL
Metode pateiktos diagramos laikosi visų modeliavimo kalbos taisyklių	Taip	Taip	Taip	Ne	Taip
Leidžia atlikti atvirkštinę inžineriją iš ETL procedūros	Ne	Taip	Ne	Ne	Ne
Gali būti integruotas su UML modeliavimo įrankiais	Ne	Ne	Ne	Ne	Ne

Pateikti testavimo ar validavimo rezultatai	Ne	Taip	Taip	Taip	Taip
Metodas leidžia sekti duomenų kilmę	Taip	Ne	Ne	Ne	Me

Apibendrinant, dauguma nagrinėtų metodų pateikia teorinius sprendimus arba riboto funkcionalumo įrankius, dažnai orientuotus į specifines modeliavimo kalbas ar platformas (pvz., BPMN, grafai). Tik keli metodai leidžia generuoti diagramas. Visgi, iš analizuotų sprendimų retas pilnai orientuotas į automatizuotą dokumentavimą iš SQL pagrindu sukurtų ETL procedūrų, kas rodo egzistuojančią nišą praktiniam sprendimui šioje srityje.

1.6. ETL procedūrų programinio kodo analizės įrankis

Vienas iš esminių šio metodo techninių komponentų yra kodo analizės biblioteka [14] (angl. *General SQL Parser*), kuri atlieka SQL užklausų analizę ir struktūrizavimą. Ši programinė priemonė yra specializuota tam, kad konvertuotų neapdorotą SQL tekstą į formalizuotą, hierarchinę struktūrą, vadinamą abstrakčiuoju sintaksės medžiu (angl. *Abstract syntax tree*). Tokia struktūra leidžia detaliai analizuoti SQL operacijas, atskiriant jų pagrindinius sintaksinius komponentus – lenteles, stulpelius, sąlygas, funkcijas, ryšius tarp duomenų objektų ir kitus elementus (10 pav.).



10 pav. *General SQL Parser* bibliotekos architektūra [18].

Kodo analizės biblioteka palaiko įvairias SQL programinių kalbų versijas, įskaitant *Microsoft SQL Server*, *Oracle*, *MySQL*, *PostgreSQL*, *DB2* ir kitas populiarias duomenų bazių sistemas. Ši savybė yra svarbi atsižvelgiant į praktikoje dažnai pasitaikančią situaciją, kuomet organizacijos naudoja skirtingų gamintojų duomenų bazių technologijas. Tačiau siūlomame metode, siekiant užtikrinti

metodikos aiškumą ir nuoseklumą, analizei ir duomenų transformacijų dokumentavimui naudojama tik *Microsoft SQL Server* technologija.

Naudojant šią biblioteką, SQL tekstas nėra analizuojamas kaip paprasta eilutė, bet išskaidomas į struktūrinius vienetus. Pavyzdžiui, „SELECT“ komanda yra interpretuojama kaip objektas su vaikais – pasirenkamais stulpeliais, duomenų šaltiniais (lentelėmis), filtravimo sąlygomis („WHERE“ sąlygomis) bei galimais agregavimo ar grupavimo komponentais. Kiekvienas šis komponentas yra atvaizduojamas kaip atskiras sintaksės medžio mazgas, turintis aiškius hierarchinius ryšius.

Kaip pavaizduota 10 pav., biblioteka veikia suskirstant analizės procesą į kelis etapus: pradinė teksto analizė atliekama leksiniu (angl. *Lexer*) lygmeniu, kur SQL tekstas išskaidomas į atskirus žodinius vienetus (angl. *tokens*). Vėliau šie vienetai perduodami „Parser“ moduliui, kuris kuria abstraktų sintaksės medį. Ši struktūra leidžia vystyti tokias funkcijas kaip sintaksės klaidų aptikimą, duomenų kilmės analizė (angl. *Data Lineage*), stulpelių poveikio analizė, SQL injekcijų aptikimas ir kt.

Biblioteka leidžia programiškai pasiekti šiuos mazgus, todėl galima realizuoti kompleksinius užklausų analizės ir transformacijos scenarijus. Šio siūlomo metodo kontekste tai leidžia išgauti reikšmingą informaciją apie ETL procedūrose naudojamus duomenų šaltinius, duomenų transformacijas, filtravimo logiką, jungimo („JOIN“) sąlygas, laikinų lentelių kūrimą ir kitus svarbius aspektus, kurie vėliau naudojami UML diagramų formavimui.

Dar vienas svarbus aspektas – biblioteka palaiko tiek standartines SQL funkcijas, tiek ir naudotojo aprašytas konstrukcijas, leidžiančias lanksčiau prisitaikyti prie realių praktinių poreikių. Be to, biblioteka turi galimybę aptikti sintaksės klaidas bei nurodyti jų vietą programiniame kode, kas leidžia anksti identifikuoti problematiškas vietas dar iki tolimesnio duomenų transformacijų dokumentavimo proceso.

Apibendrinant, kodo analizės biblioteka šiame metode atliktų svarbią užduotį – iš neformalios SQL užklausų struktūros sukuria tvarkingą bei formalizuotą, kuri tampa pagrindu tolimesnei sistemingai ETL procedūrų analizės ir dokumentavimo eigai.

1.7. Siekiamo sprendimo apibrėžimas

Analizuojant susijusius tyrimus ir metodikas, paaiškėjo dvi pagrindinės kryptys, kaip gali būti sprendžiama ETL procedūrų dokumentavimo problema. Daugumoje darbų ETL procedūrų elementai yra siejami su analogiškais modeliavimo kalbos (pvz., UML) artefaktais. Vienas iš taikomų būdų yra naudoti iš anksto apibrėžtą žodyną, kuris leidžia automatiškai generuoti diagramas, remiantis procedūrų atpažintais šablonais. Tokiu atveju, kiekvienam naujam procedūrų struktūros ar semantikos atvejui žodynas turi būti papildomas rankiniu būdu.

Kadangi ETL projektai pasižymi dideliu dinamiškumu, toks žodyno pildymo procesas greitai tampa neefektyvus ir sunkiai išlaikomas. Vis dėlto, šio metodo privalumas – galimybė sugeneruoti standartizuotą diagramos failą, kurį galima naudoti kitose modeliavimo priemonėse.

Kita analizuota metodų grupė pagrįsta matematinėmis formulėmis ir išraiškomis, kurios aprašo procedūros loginę struktūrą. Tokie metodai dažniausiai pateikia rezultatus grafais arba specializuotais vizualizavimo formatais, nepriklausančiais UML ar kitoms standartizuotoms modelių kalboms. Nors šis variantas leidžia atlikti tikslią procedūrų analizę bei greitaveikos testus, jis neapima galutinės

vizualizacijos UML diagramų pagrindu ir neturi praktinio įrankio, kuris konvertuotų matematinę išraišką į modelį.

Atsižvelgus į šias dvi alternatyvas, pasirinktas metodas grindžiamas pirmuoju požiūriu – procedūrų elementų susiejimu su UML veiklos diagramos komponentais, vadovaujantis apibrėžtomis transformavimo taisyklėmis. Tam tikras bazinių šablonų žodynas ar jų atitikmuo įrankyje gali būti naudojamas, tačiau jo plėtra nepaliekama naudotojui – transformavimo logika apibrėžiama pačiame sprendime. Tokiu būdu išvengiama rankinio įsikišimo kiekvienam atvejui, o rezultatas – generuojamos UML diagramos, kurios gali būti ne tik peržiūrimos įrankyje, bet ir eksportuojamos į kitas modeliavimo priemones, tokias kaip *Magic Systems of Systems Architect* [13]. Šis pasirinkimas leidžia sprendimą integruoti į realius darbo procesus, išlaikant tiek formalumą, tiek praktinį pritaikomumą.

1.8. Analizės išvados

1. Atlikta analizė parodė, kad ETL projektų komandos dažnai susiduria su reikšmingais iššūkiais dokumentuojant procedūras. Dokumentacijos trūkumas arba jos nepakankamas detalumas gali lemti projekto vėlavimus ir žinių praradimą. Rankinis dokumentavimo procesas padidina duomenų saugyklų kūrimo, diegimo, palaikymo ir priežiūros kaštus, kas tiesiogiai veikia projekto biudžetą bei resursų planavimą.
2. Išanalizuoti esami ETL procedūrų dokumentavimo sprendimai parodė, kad dauguma jų yra specializuoti konkrečioms technologinėms aplinkoms ir neturi vieningos standartizacijos. Dėl to jų taikymas ribotas įvairiuose projektuose, ypač kai reikalingas lankstumas ar dokumentavimas nepriklausomai nuo platformos.
3. Tyrimas atskleidė aiškų ryšį tarp ETL procedūrų struktūros ir UML veiklos diagramų. Šių diagramų semantika leidžia pateikti ETL veiksmus vizualiai, padedant geriau suprasti procesų logiką ir jų eigą. UML diagramos gali būti veiksminga priemonė dokumentacijai, ypač kai siekiama pateikti informaciją netechninėms suinteresuotosioms šalims.

2. ETL procedūrų vizualizavimo naudojant UML veiklos ir klasių diagramas skirtos metodikos reikalavimų specifikacija ir projektas

2.1. Reikalavimų specifikacija

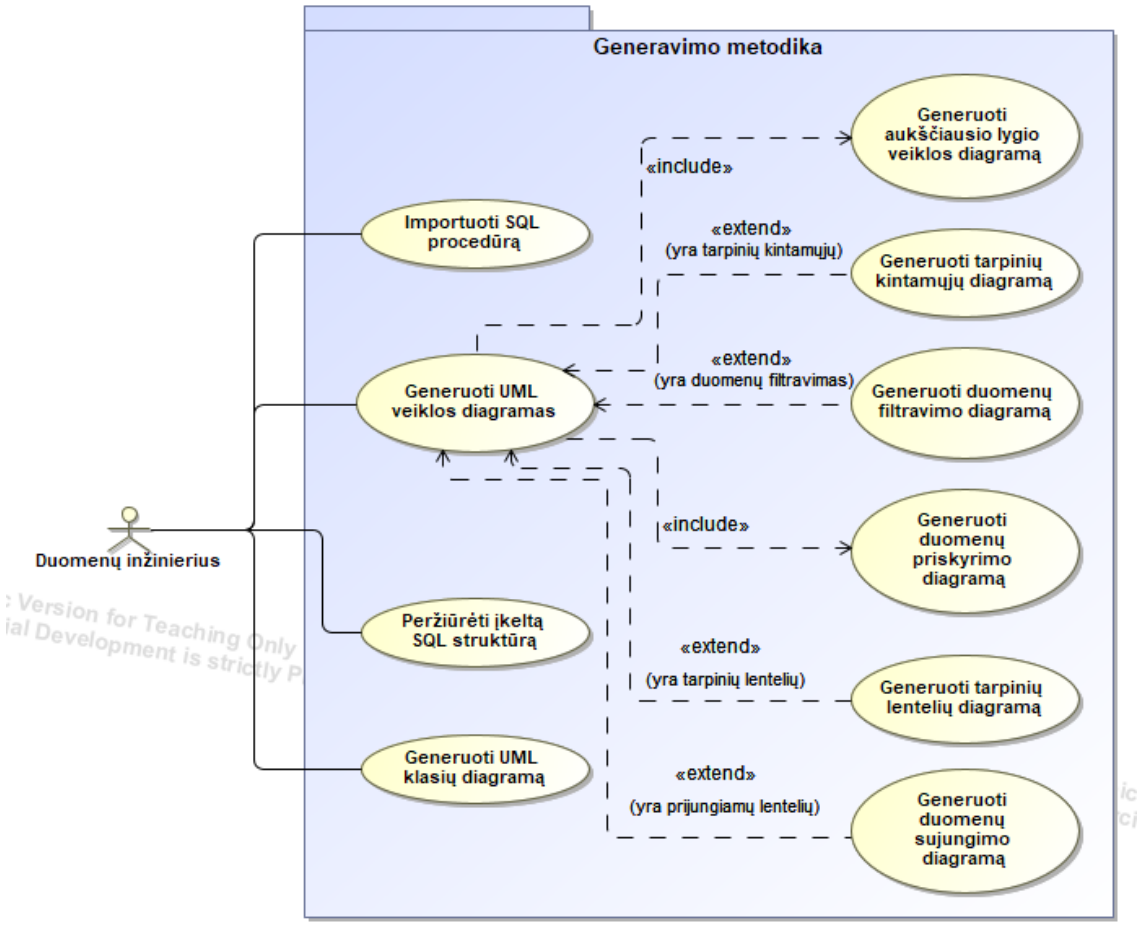
Norint efektyviai sugeneruoti UML diagramas iš ETL procedūrų kodo, būtina apibrėžti naudotojo veiksmus su įrankiu ir struktūruoti pagrindines įrankio funkcijas pagal jų vykdymo tvarką. 11 pav. pateikiama panaudojimo atvejų diagrama, atspindinti pagrindinius įrankio naudojimo scenarijus.

Pirmasis naudotojo žingsnis – ETL procedūros importavimas į sistemą. Ši procedūra analizuojama naudojant kodo analizės biblioteką, kuri pateikia kodu paremtą vaizdinį atvaizdą. Nors konceptualiai šis struktūrizavimas gali būti siejamas su SQL metamodeliu, praktikoje procedūra nėra transformuojama į formalų modelio objektų rinkinį. Gauta informacija tarnauja kaip pagrindas tolimesniam UML diagramų generavimui.

Naudotojui suteikiama galimybė peržiūrėti analizės metu įkeltą SQL struktūrą – lenteles, stulpelius, naudotus filtrus, laikinas lenteles bei tarpinius kintamuosius. Remiantis šiais duomenimis, gali būti pradedamas UML veiklos bei klasių diagramų generavimas.

UML veiklos diagramos generuojamos kaip atskiri vienetai: aukščiausio lygio procedūros vaizdas, tarpinių kintamųjų, filtravimo, duomenų sujungimo, laikinosios lentelės bei galutinių reikšmių priskyrimo diagramos. Kai kurios diagramos (pvz., filtravimo ar sujungimo) generuojamos tik tuomet, kai tokie elementai aptinkami procedūros struktūroje. UML klasių diagrama generuojama remiantis analizės metu nustatytomis duomenų lentelėmis ir jose naudotais stulpeliais. Diagramos neišplečia visos duomenų bazės schemos – atvaizduojami tik tie lentelės atributai, kurie iš tikrųjų yra paminėti procedūros kode.

Sugeneruotas diagramas naudotojas gali eksportuoti į standartizuotą XMI formatą. Jis leidžia diagramas įkelti į CASE įrankius, palaikančius šį standartą, ir kuriose galima atlikti tolimesnę analizę ar įtraukti jas į dokumentavimo procesus.



11 pav. ETL procedūrų transformavimo į UML diagramas metodikos panaudojimo atvejų diagrama

2.2. Dalykinės srities modelis

Norint aiškiai apibrėžti UML diagramų generavimo įrankio veikimą, būtina pateikti esybių klasių modelį, kuris atspindėtų svarbiausius objektus, jų tarpusavio ryšius bei informaciją, naudojamą generavimo metu.

Pagrindinis elementas modelyje yra SQL procedūra, kuri pateikiama naudotojo ir yra analizės bei vizualizacijos proceso pradžia. Kiekviena procedūra gali apdoroti kelias duomenų lenteles, kurios pagal paskirtį suskirstytos į šaltinio, tikslines bei laikinąsias. Lentelėms priskiriami konkretūs stulpeliai, kurie atlieka svarbią funkciją duomenų sujungimo, filtravimo bei rezultatų priskyrimo veiksmuose.

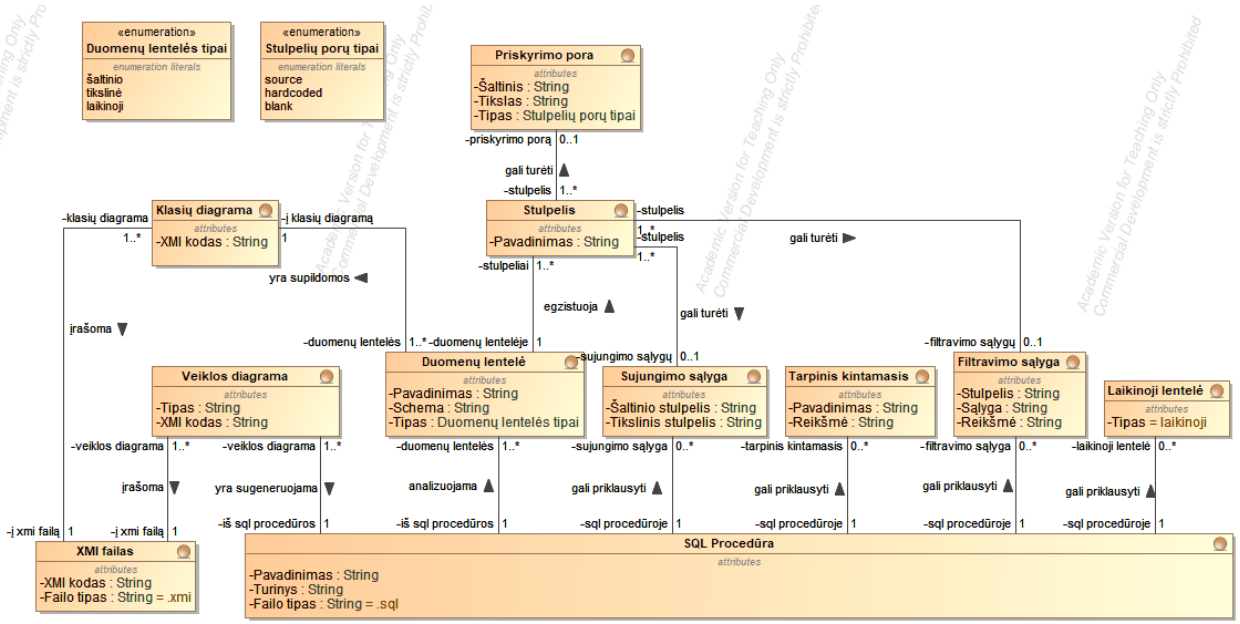
Modelyje numatytas tarpinių kintamųjų panaudojimas, kurie gali būti apibrėžiami SQL procedūroje kaip laikinos reikšmės, naudojamos tarpiniams skaičiavimams ar reikšmių saugojimui. Filtravimo operacijos reprezentuojamos per filtravimo sąlygas, kurios aprašo, kuriam stulpeliui yra taikoma sąlyga bei kokia lyginamoji reikšmė naudojama. Tuo tarpu sujungimo sąlygos leidžia nurodyti, kaip šaltinio ir prijungiamosios lentelės yra susietos pagal rakto stulpelius.

Sugeneruoti duomenų priskyrimai tarp šaltinio ir tikslinio stulpelių modelyje pateikiami per priskyrimo poras, kuriose nurodomi stulpelių pavadinimai bei priskyrimo tipas (pvz., „source“,

„hardcoded“, „blank“). Šis priskyrimas būtinas siekiant pavaizduoti, kokiu būdu duomenys yra įrašomi į tikslinę lentelę.

Analizės pagrindu sugeneruojamos dvi pagrindinės diagramos – veiklos diagrama ir klasių diagrama. Veiklos diagrama vaizduoja duomenų transformacijos procesus, o klasių diagrama – duomenų struktūras, t. y., duomenų lenteles bei jų stulpelius. Abi šios diagramos yra eksportuojamos kaip XMI failai, kuriuos galima naudoti išorinėse UML modeliavimo aplinkose.

Esybių klasių diagrama yra vaizduojama 12 pav. Modelis padeda atskleisti, kokie informacijos objektai dalyvauja UML diagramų generavimo procese bei kaip jie yra susiję tarpusavyje.



12 pav. ETL procedūrų generavimo į UML diagramas esybių klasių diagrama.

2.3. Formalizuotas ETL procedūrų vizualizavimo naudojant UML veiklos ir klasių diagramas aprašas

2.3.1. Konvertavimas iš SQL programinio kodo

Siekiant generuoti UML diagramas iš SQL programinio kodo, pirmasis žingsnis yra procedūros suskaidymas į teorinius SQL metamodelio elementus. Šiame etape numatoma, kad pateiktas SQL scenarijus bus interpretuojamas kaip struktūrizuotų vienetų rinkinys, kuriame kiekviena reikšminga dalis – duomenų šaltiniai, jų tarpusavio ryšiai, filtravimo sąlygos, duomenų laukų transformacijos bei laikini rezultatai bus identifikuoti kaip atskiri elementai. Toks požiūris leidžia pagrįsti tolesnę šių elementų susiejimo su UML veiklos diagramos metamodelio eiga. Teoriniu lygmeniu, šis suskaidymas remiasi prielaida, kad kiekvienas SQL kalbos fragmentas atlieka aiškią funkciją procedūros logikoje. Tokiu būdu, loginiai blokai yra traktuojami kaip savarankiški metamodelio komponentai. Jie sudaro pagrindą sistemingam procedūros analizės procesui, leidžiančiam priskirti SQL semantiką UML struktūroms.

Šiam suskaidymo žingsniui įgyvendinti pasitelkiama kodo analizės biblioteka [14] (angl. *General SQL Parser*), leidžianti pateiktą procedūros kodą interpretuoti ne kaip tekstinę informaciją, o kaip logiškai suskirstytą struktūrą. Bibliotekos paskirtis šiame metodikos etape yra perimti užduotį

identifikuoti procedūros sudėtines dalis – duomenų lenteles, laukus, ryšius tarp lentelių, sąlyginius filtrus, priskyrimo operacijas bei kitus elementus, ir pateikti juos standartizuota, tolesnei analizei tinkama forma. Taikant šį sprendimą, nėra būtina rankiniu būdu išskirti atskirų SQL elementų, nes biblioteka leidžia automatiškai gauti semantinius ryšius tarp skirtingų kodo dalių.

Bibliotekos pagalba galima identifikuoti duomenų šaltinius, analizuoti, kokios lentelės naudojamos užklausose, kokie stulpeliai yra susieti su konkrečiais veiksmais, kokios sąlygos taikomos filtravimui, kokios operacijos vykdomos duomenų jungimo metu ir kokios reikšmės priskiriamos naujiems ar esamiems laukams. Kiekvienas aptiktas semantinis vienetas yra suvokiamas kaip loginė procedūros dalis, kuri vėliau susiejama su atitinkamu UML veiklos diagramos elementu. Šis susiejimas grindžiamas tuo, kad kiekvienas išskirtas procedūros komponentas turi savo atitikmenį UML metamodelio struktūroje. Pavyzdžiui, stulpelių transformacijos žymimos veiklos veiksmais, o filtravimo sąlygos – sprendimų taškais.

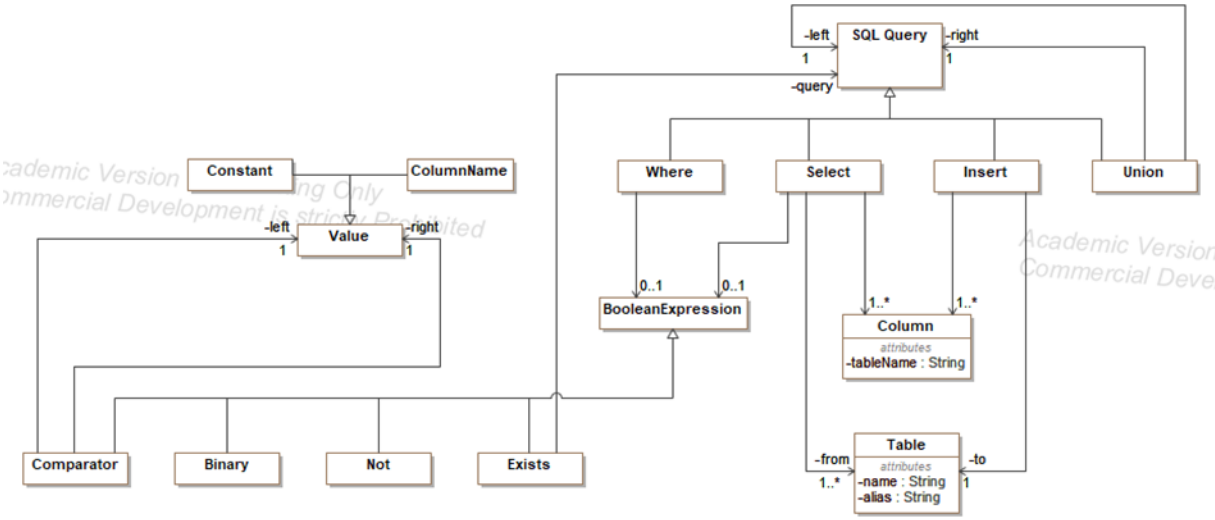
Svarbu pabrėžti, kad šiame metodikos žingsnyje SQL metamodelis išlieka teoriniu pagrindu – procedūros suskaidymas nėra realizuojamas kuriant atskirą, formalizuotą SQL metamodelį. Vietoje to, siekiama pasinaudoti analizės įrankio galimybėmis struktūrizuoti procedūros informaciją pakankamu lygmeniu, kad būtų galima įvykdyti numatytą susiejimo su UML diagrama procesą. Šis sprendimas leidžia išvengti perteklinių tarpinių žingsnių, sumažina sistemos sudėtingumą ir užtikrina sklandų analizės rezultatų perėjimą į UML diagramų generavimo etapą.

Tokio metodo taikymas reikalauja aiškiai apibrėžto loginio ryšio tarp procedūros elementų ir jų funkcijų. Aptiktų lentelių, stulpelių, jungčių ir filtravimo sąlygų semantika turi būti išlaikoma analizuojant ir interpretuojant procedūrą, nes tik tokiu būdu galima tiksliai atkurti procesų logiką diagramose. Šioje metodikoje laikomasi nuostatos, kad kiekvienas procedūros fragmentas yra ne tiesiog tekstas, o veikla, turinti aiškų tikslą ir funkciją duomenų sraute.

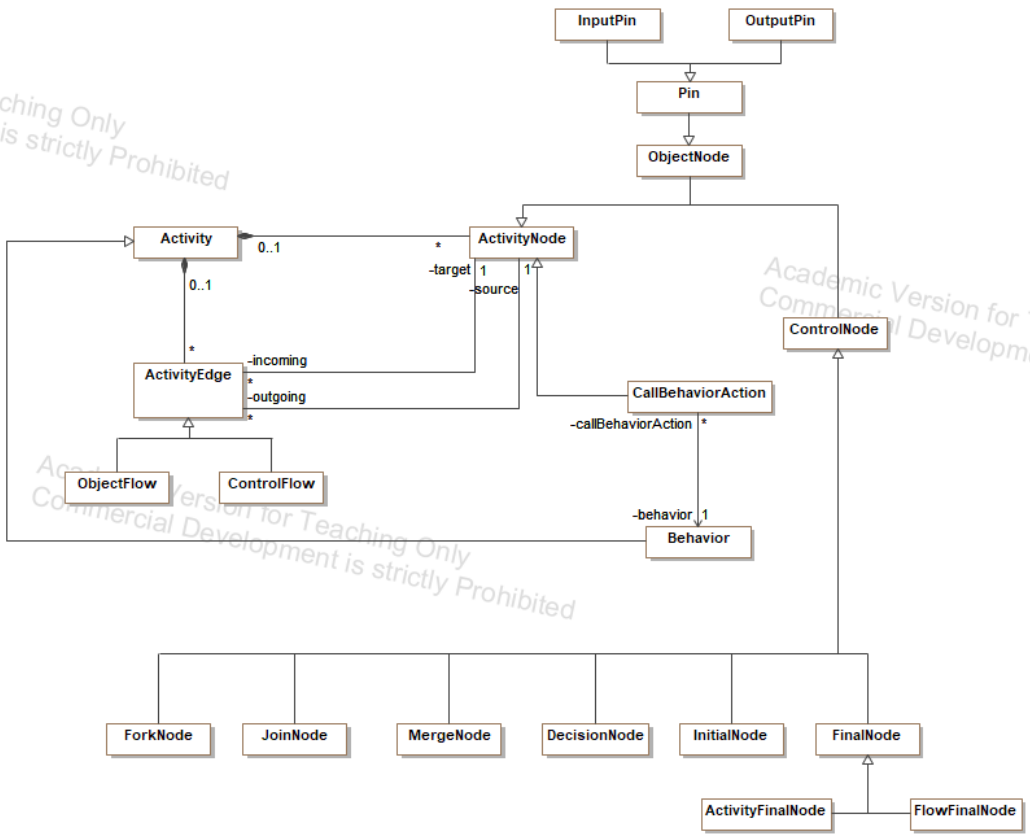
Apibendrinant, SQL procedūros suskaidymas į teorinius metamodelio elementus yra pagrindinis žingsnis, leidžiantis pereiti nuo tekstinio aprašymo prie struktūrizuoto modelio. Taikant kodo analizės biblioteką, procedūros logika išskaidoma į atskirus komponentus, kurie vėliau panaudojami generuojant UML veiklos diagramas, išlaikant originalios procedūros struktūrą ir loginę seką.

2.3.2. Diagramų generavimas

Siekiant perkelti SQL procedūros semantinę logiką į UML veiklos diagramų struktūrą, būtina apibrėžti ryšį tarp naudojamų metamodelių. Jų teorinė struktūra padeda sistemingai susieti procedūros komponentus su UML veiklos diagramos elementais. Šiuos ryšius iliustruoja 13 pav. pateikiamas SQL metamodelio diagramos fragmentas bei 14 pav. vaizduojama UML veiklos diagramos metamodelio struktūra.



13 pav. SQL metamodelis [15]



14 pav. UML veiklos diagramos metamodelio fragmentas [17]

Kuomet SQL procedūra yra suskaidoma į atskirus komponentus, tokius kaip duomenų šaltiniai, jų stulpeliai, filtravimo sąlygos, sujungimo veiksmai, jie metodikoje yra susiejami su atitinkamais UML veiklos diagramos metamodelio elementais. Šio žingsnio tikslas – išlaikyti ryšį tarp procedūros logikos ir jos vizualios reprezentacijos, leidžiant naudotojui ne tik suprasti, bet ir analizuoti duomenų srautą bei procesų struktūrą UML formatu.

Kiekvienas iš kodo analizės bibliotekos gaunamų komponentų atlieka specifinę funkciją SQL procedūroje, todėl gali būti logiškai priskirtas tam tikram UML veiklos diagramos elementui, kuris perteikia to komponento paskirtį ir vietą bendrame procese. SQL metamodelis grindžiamas supaprastinta struktūra, kurioje tokios esybės kaip „Select“, „Insert“, „Where“ yra susietos loginiais ryšiais, atspindinčiais dažniausiai naudojamas SQL operacijas, kaip siūloma [15] metode. UML veiklos metamodelio fragmentas remiasi oficialia UML specifikacija [17] ir yra papildytas procesų modeliavimui skirtomis struktūrinėmis praktikomis, aptartomis šiuolaikiniuose tyrimuose [16].

Siekiant nuosekliai susieti SQL procedūrų komponentus su UML veiklos diagramos elementais, metodikoje numatomas atitikmenų rinkinys, pateiktas **3** lentelėje. Ši lentelė padeda apibrėžti, kaip konkretūs SQL vienetai turėtų būti interpretuojami generuojant UML diagramas, užtikrinant vieningą struktūrą nepriklausomai nuo procedūros turinio ar sudėtingumo.

3 lentelė. Metamodelių susiejimo ryšių taisyklės

Metamodelių ryšys		**Paaiškinimas**
SQL metamodelio elementas	**UML metamodelio elementas**	
Insert	*Activity*	Reikalingas norint atvaizduoti duomenų įkėlimo operaciją į pasirinktą duomenų saugyklą.
Insert	*CallBehaviorAction*	Reikalingas norint atvaizduoti duomenų įkėlimo operaciją į pasirinktą duomenų saugyklą aukščiausiame diagramos hierarchijos lygyje
Select	*Activity*	Reikalingas norint atvaizduoti duomenų pasirinkimo operaciją iš vienos ar kelių duomenų šaltinių.
Select	*CallBehaviorAction*	Reikalingas norint atvaizduoti duomenų pasirinkimo operaciją iš vienos ar kelių duomenų šaltinių aukščiausiame diagramos hierarchijos lygyje
Where	*Activity*	Reikalingas norint atvaizduoti duomenų filtravimo (transformacijos) etapą.
Select	*ObjectNode*	Nurodo duomenų šaltinius, kurie naudojami ETL procedūroje duomenų įkėlimo etape.
Insert	*ObjectNode*	Nurodo lenteles, į kurias yra įkeliama transformuota informacija duomenų saugykloje.
Select	*InputPin*	Pateikia informaciją, kokiam „Activity“ veiksmui yra naudojamos duomenų lentelės jų paėmimui iš duomenų šaltinių.
Insert	*OutputPin*	Pateikia informaciją, kokiam „Activity“ veiksmui yra naudojamos duomenų lentelės jų įkėlimui į duomenų saugyklą.
Table	*ObjectNode*	Skirtas pateikti informacijai apie veiksme reikalingą duomenų lentelę

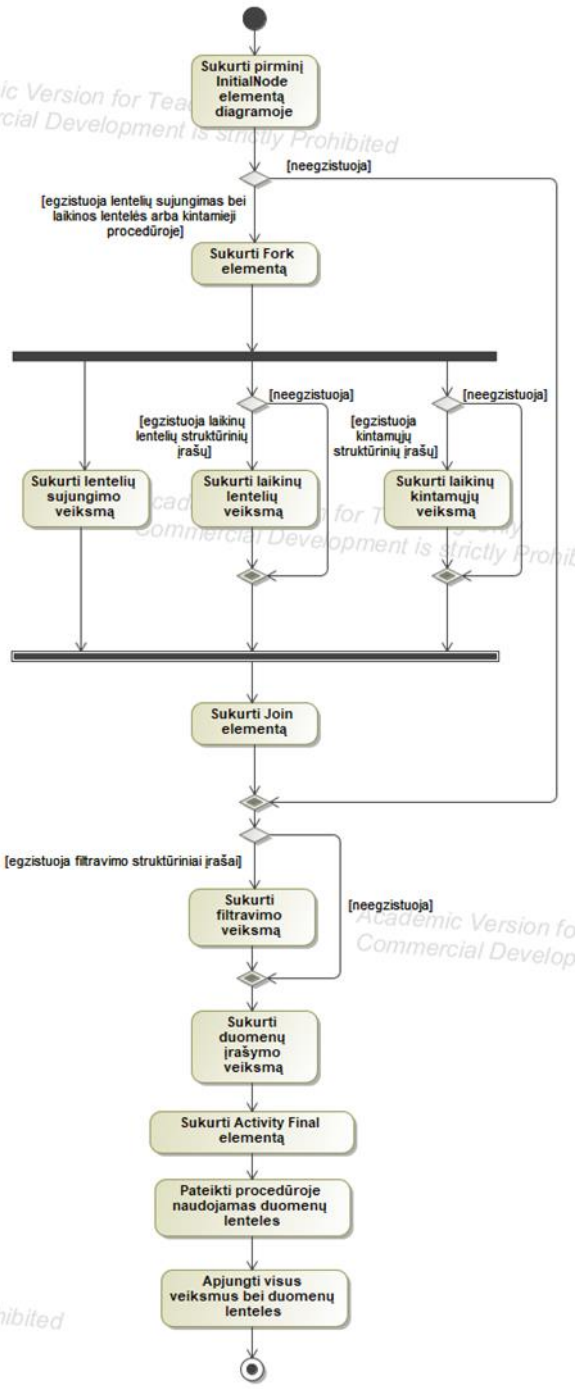
Value	*Activity*	Skirtas norint pavaizduoti sujungiamų (pagal „LEFT JOIN“, „INNER JOIN“ ir kt.) duomenų lentelių raktinių stulpelių pavadinimus.
Where ir *ColumnName*	*Activity*	Filtravimo operacijose skirtas pateikti kokiam duomenų lentelės stulpeliui yra vykdomas filtravimas

Norint tiksliai suprasti šiame poskyryje pateiktas sąvokas, būtina susipažinti su tam tikrais specializuotais terminais, kurie yra naudojami toliau skyriuje. **4** lentelė apibrėžia šias sąvokas bei paaiškina jų reikšmes.

4 lentelė. Naudojamų sąvokų paaiškinimas

Sąvoka	**Paaiškinimas**
Kintamasis	SQL kodo elementas, kurio reikšmę galima suteikti dinamiškai. Procedūroje žymimi „@” simboliu.
Struktūrinis įrašas (arba įrašas)	Programinio kodo objektas, kuriame kaupiama programuotojo nurodoma informacija
Sąrašas	Šio skyriaus kontekste naudojamas kartu su struktūriniais įrašais. Tai programinio kodo objektas, kuris kaupia nurodytus įrašus kaip vienetus sąraše.
Šaltinio ir prijungiamoji lentelės	Naudojama dviejų lentelių apjungimo kontekste. Šių tipų lentelės yra sujungiamos stulpelių raktų pora.
Veiksmas	UML veiklos diagramos „Action“ tipo elementas.
Mazgas	Naudojamas kartu su įvesties arba išvesties sąvokomis, UML veiklos diagramos elementas.
Įvestis ir išvestis	UML veiklos diagramos „InputPin“ ir „OutputPin“ elementai
Trumpinys	SQL programiniame kode plačiai naudojamas lentelių nurodymo būdas, kuomet pilnam lentelės pavadinimui yra suteikiamas kitoks, kodo kontekste naudojamas, pavadinimas. Pavyzdžiui, „lenteles_pavadinimas AS lent“, kur „lent“ būtų trumpinys.

Veiklos diagramai sugeneruoti pirmiausia būtina sukurti loginę struktūrą, kuri apibrėžia ETL proceso eigą nuo pradžios iki pabaigos. Tokia struktūra išreiškiama UML veiklos diagramos forma, kurioje kiekvienas veiksmas atitinka tam tikrą duomenų transformavimo ar paruošimo etapą. Aukščiausio lygio diagrama – tai pagrindinė veiklos iliustracija, kurioje atvaizduojami pagrindiniai duomenų apdorojimo etapai bei ryšiai tarp jų. Ji taip pat nurodo, kokie konkretūs veiksmai turi būti toliau detalizuojami atskiruose subprocesuose.



15 pav. Aukščiausio hierarchijos lygio veiklos diagramos kūrimo logika

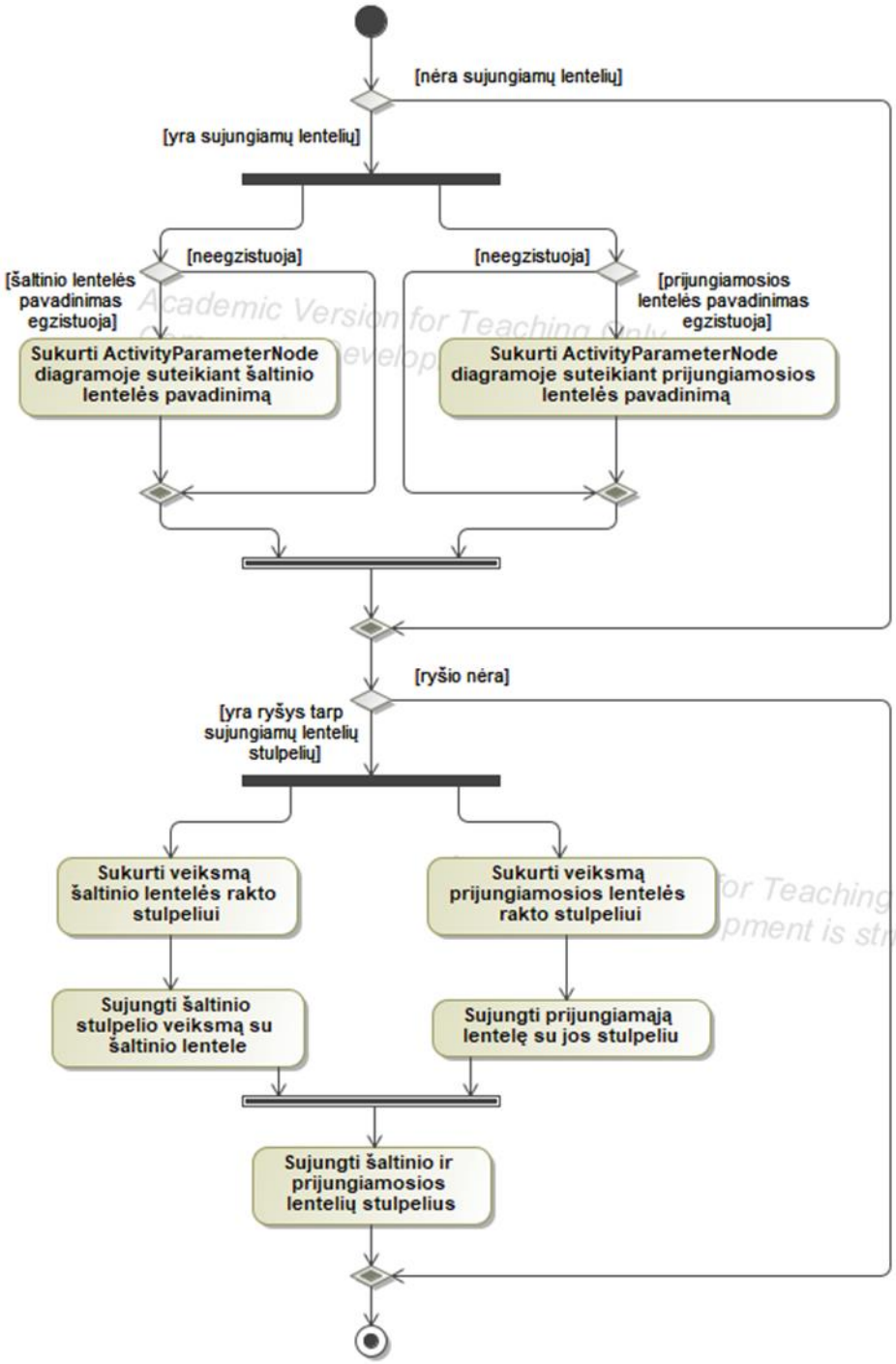
Diagramos kūrimas (15 pav.) prasideda nuo pradinio mazgo (*InitialNode*) sukūrimo, kuris žymi proceso pradžią. Po jo, naudojamas šakos mazgas (*ForkNode*), kuris leidžia atskirti kelių veiksmų srautus lygiagrečiam vykdymui. Tolimesni veiksmai skirstomi į tris pagrindines grupes: laikinų lentelių kūrimą, laikinų kintamųjų išgavimą, bei šaltinių lentelių apdorojimą (sujungimų taikymą). Šie trys veiksmai yra generuojami tik tuomet, jei jų poreikis identifikuojamas pagal scenarijų – pavyzdžiui, jeigu egzistuoja laikinos lentelės ar kintamieji. Visi trys veiksmai yra prijungiami prie šakos mazgo ir vykdomi lygiagrečiai.

Užbaigus šiuos veiksmus, visi trys (jei egzistuoja) srautai yra suvedami į sujungimo mazgą (*JoinNode*). Šioje vietoje duomenys apjungiami, kad būtų galima pereiti prie sekančio etapo – filtravimo veiksmo, jei egzistuoja atitinkamos sąlygos (pvz., „WHERE“ sakiniai).

Kitas etapas – tai rezultatų užpildymas (*Fill Results*), kuris visada egzistuoja ir žymi paskutinį duomenų transformacijos žingsnį – t. y. įrašymą į tikslinę lentelę. Šis veiksmas turi ryšį su visais šaltiniais, kintamaisiais bei laikinais struktūriniais įrašais, kurių duomenys yra naudojami įrašymui. Duomenų srautai į šį veiksmą yra atvaizduojami naudojant UML „ObjectFlow“ elementus tarp duomenų mazgų (*CentralBufferNode*) ir veiksmų įvesties/išvesties elementų (*InputPin, OutputPin*).

Diagramos pabaigoje sukuriamas galutinis mazgas (*ActivityFinalNode*), žymintis proceso pabaigą. Visi veiksmai ir mazgai tarpusavyje yra sujungiami aiškiai apibrėžtais srautais (*ControlFlow* ir *ObjectFlow*), kurie generuojami dinamiškai – atsižvelgiant į aptiktus SQL modelio elementus. Nors kai kurie veiksmai (pvz., filtravimas ar laikinų lentelių kūrimas) gali egzistuoti tik tam tikrais atvejais, pati diagramos struktūra ir logika visada išlieka vienoda.

Šaltinių lentelių apdorojimo ir tarpusavio sujungimo proceso veikla UML diagramoje sugeneruojama (16 pav.) tik tuo atveju, jei ETL procedūroje identifikuojami lentelių jungimo ryšiai, t. y. kai duomenys iš skirtingų lentelių yra susiejami pagal tam tikrus stulpelius. Ši veikla yra esminė norint pavaizduoti, kaip procedūros metu duomenys integruojami iš kelių šaltinių į vieną bendrą struktūrą.



16 pav. Sujungiamų lentelių veiklos diagramos kūrimo logika

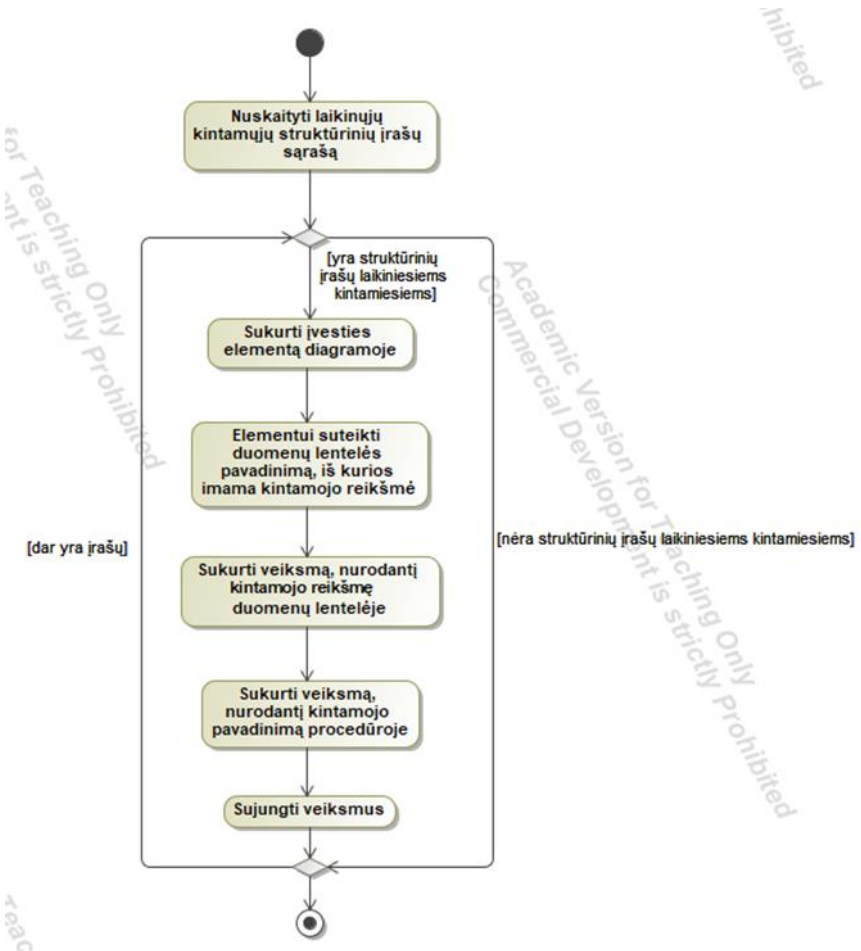
Veiklos pradžioje sukuriami vienas ar keli „ActivityParameterNode“ elementai. Jie atspindi kiekvieną lentelę, kuri dalyvauja jungimo procese. Kiekvienam lentelės pavadinimui suteikiamas unikalus mazgo identifikatorius, o jų kiekis priklauso nuo to, kiek skirtingų lentelių yra susieta jungimo operacijose. Šie įvesties mazgai naudojami kaip duomenų šaltiniai visiems tolimesniems jungimo veiksmams.

Toliau, kiekvienam sujungimo ryšiui, identifikuotam analizės metu, generuojami du „CallBehaviorAction“ mazgai, kurie atvaizduoja jungime dalyvaujančius stulpelius – vienas iš

pirminės lentelės, kitas iš prijungiamos lentelės. Šie stulpeliai dažniausiai yra raktų poros arba kitos loginės sąsajos, leidžiančios susieti duomenis. Kiekvienas iš šių veiksmų mazgų yra sujungiamas su atitinkamu lentelės įėjimo mazgu per „ObjectFlow“ jungtį, kas nurodo, kad konkretus stulpelis priklauso tam tikrai lentelei. Tokiu būdu išlaikoma aiški ir logiška lentelės bei stulpelio priklausomybė UML struktūroje. Kai du stulpeliai yra paruošti jungimui, tarp jų sukuriamas „ControlFlow“ ryšys, kuris papildomai pažymimas jungimo tipu (pvz., „INNER JOIN“, „LEFT JOIN“ ir pan.). Tai leidžia vizualiai identifikuoti, kokio tipo ryšys yra naudojamas tarp dviejų šaltinių.

Šios veiklos pabaigoje gaunama struktūra, kuri iš UML perspektyvos atspindi kiek šaltinių naudojamą, kaip jie susieti, kokie stulpeliai naudojami sujungime bei koks jungimo tipas pritaikytas. Visa ši logika atvaizduojama „Activity“ bloke su pavadinimu „Apply joins to source tables“, kuris vėliau UML diagramoje yra prijungiamas prie aukščiausio lygio „CallBehaviorAction“ mazgo, atspindinčio šį konkretų veiksmą. Jei SQL scenarijuje JOIN sąlygų nėra – ši veikla nėra generuojama.

Laikinų kintamųjų veikla (17 pav.) yra viena iš trijų galimų lygiagrečių veiklų aukščiausio lygio veiklos diagramoje. Jos paskirtis – išgauti kintamuosius iš laikino tipo duomenų, dažniausiai naudojamų papildomų reikšmių saugojimui ETL proceso metu. Veiklos pradžioje sukuriamas „ActivityParameterNode“, kuris žymi įėjimo duomenis – tai yra lentelės, iš kurių ištraukiami kintamieji. Dažniausiai šios lentelės yra konfigūracinės (angl. *configuration*) struktūros, kuriose saugomi raktiniai stulpeliai, leidžiantys priskirti reikšmes konkretiems kintamiesiems.



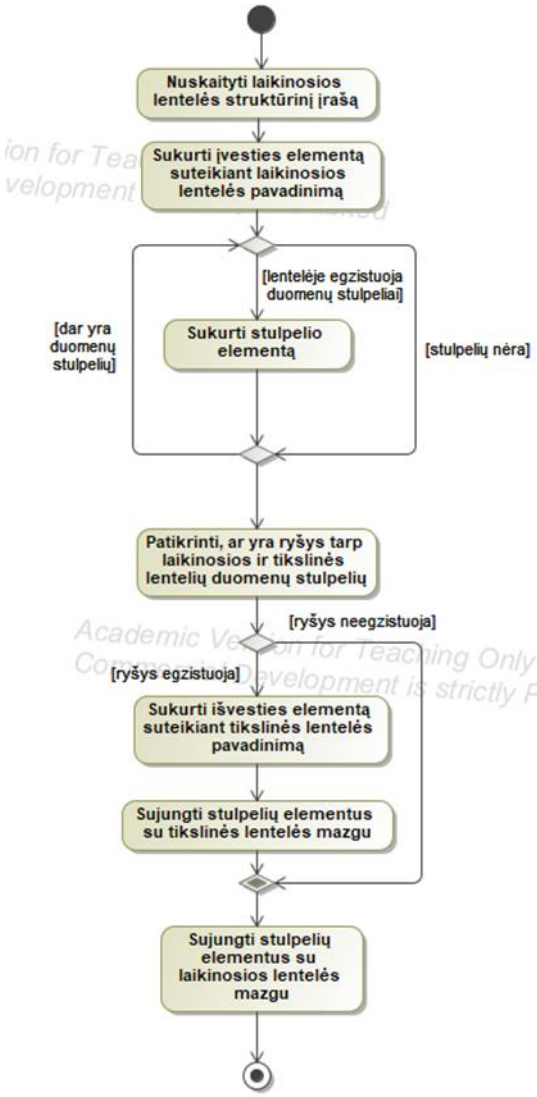
17 pav. Laikinųjų kintamųjų UML veiklos diagramos kūrimo logika

Toliau kiekvienam identifikuotam kintamajam yra generuojami du mazgai: pirmasis – tai „CallBehaviorAction“, kuris atspindi kintamojo reikšmės paėmimą pagal filtravimo sąlygą o antrasis – kitas „CallBehaviorAction“, kuris apibrėžia paties kintamojo reikšmę. Abu šie mazgai yra sujungti UML „ControlFlow“ jungtimi, kuri loginėje struktūroje reiškia, kad iš lentelės išgavus reikiamą reikšmę, ji priskiriama kintamajam.

Be to, tarp įėjimo mazgo ir pirmojo veiksmo sukuriamas „ObjectFlow“ ryšys, kuris nusako, kad kintamojo reikšmė yra paimama iš tam tikros duomenų struktūros. Kiekvienas toks dviejų mazgų rinkinys veikia kaip savitas ETL žingsnis – jis nurodo, kad tam tikru pagrindu (paprastai pagal nurodomą procedūroje duomenų stulpelį) yra paimama reikšmė ir priskiriama specifiniam laikinam kintamajam.

Pabaigoje visi šie elementai integruojami į vieną „Activity“ bloką, kuris vėliau UML diagramoje yra prijungiamas prie aukščiausio lygio „CallBehaviorAction“ mazgo, žyminčio šią veiklą. Jeigu kintamųjų nėra, visa ši diagramos dalis nėra generuojama.

Paskutinis veiklos diagramos lygiagrečių veiksmų apima laikinų lentelių struktūros sukūrimą (18 pav.). Šios veiklos tikslas – vizualiai atvaizduoti duomenų stulpelių susiejimą tarp laikinosios bei tikslinės lentelių.



18 pav. Laikinųjų lentelių UML veiklos diagramos kūrimo logika

Veikla pradedama nuo įvesties duomenų nustatymo – tai yra „TemporaryMapping“ struktūriniai įrašai, kurie buvo sugeneruoti analizės metu. Kiekvienas toks įrašas saugo informaciją apie laikiną lentelę – jos pavadinimą bei stulpelių sąrašą. Šie duomenys naudojami kuriant mazgus, kurie žymi tiek lentelę, tiek jos duomenų struktūrą.

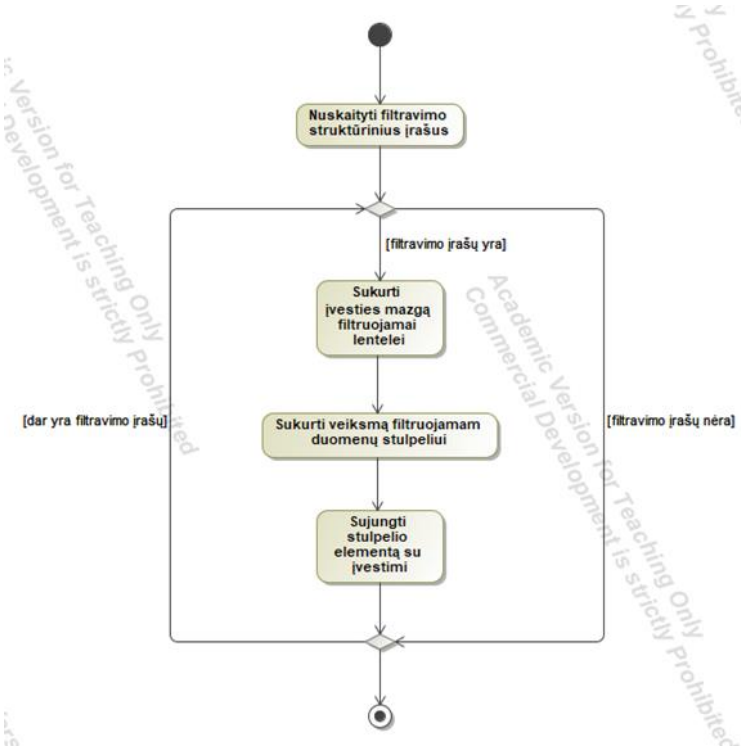
Kiekvienai laikinai lentelei diagramos pradžioje sukuriamas įvesties mazgas, kuris žymi jos egzistavimą procedūroje. Taip pat, sukuriamas ir tikslinės lentelės išvesties mazgas. Toliau kiekvienam lentelės stulpeliui sukuriamas atskiras „CallBehaviorAction“ mazgas, kuris žymi reikšmės įrašymo veiksmą.

Šie veiksmai sujungiami su lentelės įvesties mazgu. Kiekvienas veiksmo mazgas turi pavadinimą, atitinkantį stulpelio vardą, ir stereotipą, kuris nurodo jo paskirtį kaip tarpinio elemento.

Esant stulpelių ryšiui tarp laikinosios bei tikslinės lentelių, jos yra sujungiamos su atitinkamu tikslinės lentelės išvesties mazgu. Tokiu būdu, duomenų stulpelis, kuris yra naudojamas tiek laikinojoje, tiek tikslinėje lentelėse, yra identifikuojamas.

Ši veikla UML modelyje leidžia atskirai parodyti duomenų srautus, kurie per ETL procedūrą keliauja per laikinąsias struktūras, nenaudojant jų tiesiogiai kaip galutinės paskirties elementų.

Filtravimo veikla (19 pav.) UML veiklos diagramoje atspindi žingsnį, kuomet pagal nurodytas sąlygas yra įkeliamas tik tam tikrus kriterijus atitinkančios reikšmės iš šaltinių lentelių. Ši veikla generuojama tik tuo atveju, jei ETL procedūroje egzistuoja bent viena duomenų filtravimo sąlyga.



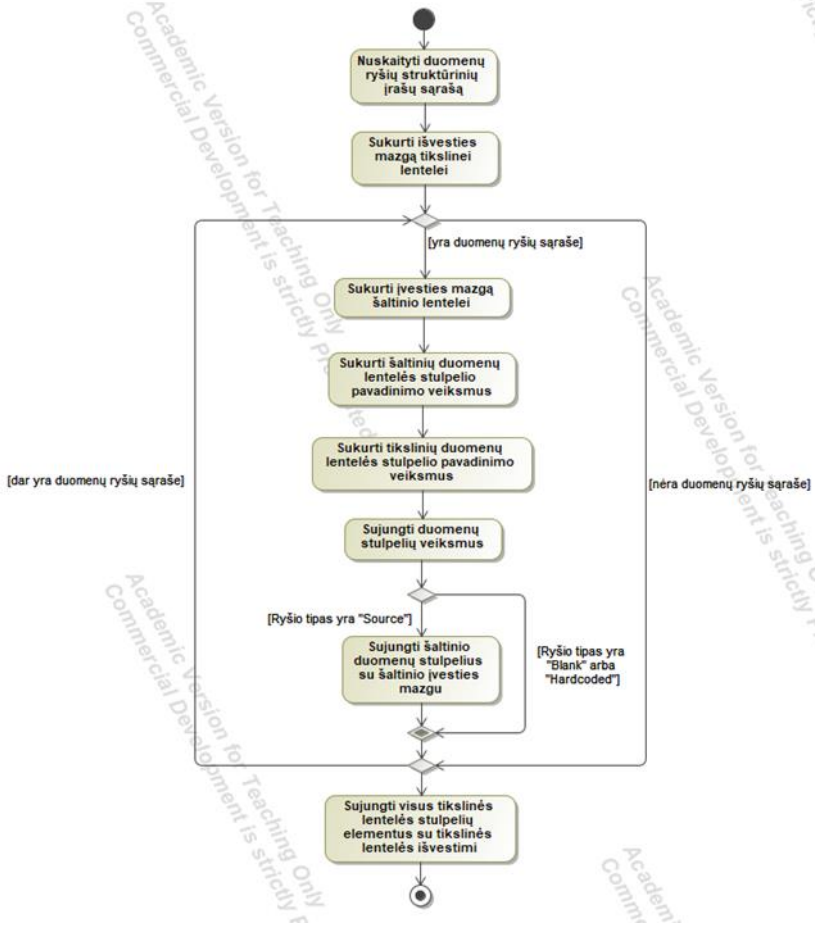
19 pav. Filtravimo UML veiklos diagramos kūrimo logika

Veikla pradedama sukuriant „ActivityParameterNode“ mazgus kiekvienai lentelei, kuriai taikomos filtravimo sąlygos. Šie mazgai žymi įėjimo taškus į veiklą ir parodo, iš kokių lentelių yra imami duomenys prieš filtravimą. Kiekvienam tokio tipo mazgui suteikiamas pavadinimas, susietas su konkrečia lentele. Tolesniame etape kiekvienai filtravimo sąlygai sukuriamas „CallBehaviorAction“ mazgas. Kiekvienas toks mazgas atspindi konkretų stulpelį, kuriam taikomas loginis palyginimas. Jie yra tiesioginė UML atitiktis filtravimo veiksmui. Kiekvienam jų suteikiamas unikalus identifikatorius, sudarytas iš stulpelio pavadinimo ir palyginimo operatoriaus, konvertuoto į naudotojui lengviau suprantamą tekstą (pvz. operatorius „=“ išreiškiamas angliškų žodžiu *Equals*).

Kiekvienas „CallBehaviorAction“ mazgas yra sujungiamas su savo atitinkamu įėjimo mazgu per „ObjectFlow“ jungtį. Ši jungtis nurodo, kad filtravimo sąlyga naudoja konkretų lentelės duomenų srautą kaip šaltinį. Taip išlaikoma loginė priklausomybė tarp lentelės ir jos stulpelyje taikomo filtro. Papildomai, kiekvienam filtravimo veiksmui yra priskiriamas stereotipas, atitinkantis loginę operaciją, kuris vėliau UML diagramoje naudojamas vizualiai ryšių atskirčiai. Baigus visų sąlygų generavimą, šie mazgai yra sudedami į vieną „Activity“ bloką pavadinimu „Apply filtering to the data“, kuris vėliau prijungiamas prie aukščiausio lygio diagramos per „CallBehaviorAction“ mazgą. Tokiu būdu ši veikla tampa vientisos ETL eigos dalimi.

Jeigu SQL scenarijuje „WHERE“ sąlygų nėra – visas filtravimo veiklos blokas nėra generuojamas, o duomenų srautas aukščiausio lygio diagramoje eina tiesiai iš sujungimo žingsnio į rezultatų užpildymą.

Paskutinis etapas ETL proceso veiklos diagramoje yra duomenų įrašymas į tikslinės lentelės struktūrą (20 pav.). Šis žingsnis vizualiai atvaizduojamas kaip atskira UML veiklos diagrama. Jos tikslas – parodyti, kaip konkretūs duomenų stulpeliai (šaltiniai) yra susiejami su tikslinės lentelės stulpeliais, remiantis anksčiau suformuotais ryšiais.



20 pav. Duomenų įrašymo į tikslinę duomenų lentelę UML veiklos diagramos kūrimo logika

Veikla pradedama nuo įvesties mazgų (angl. *ActivityParameterNode*) sukūrimo kiekvienai duomenų struktūrai, iš kurios yra gaunami duomenys. Tai gali būti šaltinių lentelės arba laikinos lentelės, priklausomai nuo to, kas buvo identifikuota struktūrinio įrašo kūrimo etape. Taip pat, sukuriamas vienas tikslinės duomenų lentelės išvesties mazgas. Siūlomas metodas gali apdoroti tik tokias procedūras, kuriose duomenys užpildomi tik į vieną lentelę.

Kiekvienam lentelių susiejimo ryšiui yra kuriamas atskiras veiksmo mazgas. Šis mazgas reprezentuoja konkretaus stulpelio užpildymo operaciją tikslinėje lentelėje. Mazgo pavadinimas atitinka paskirties stulpelio vardą.

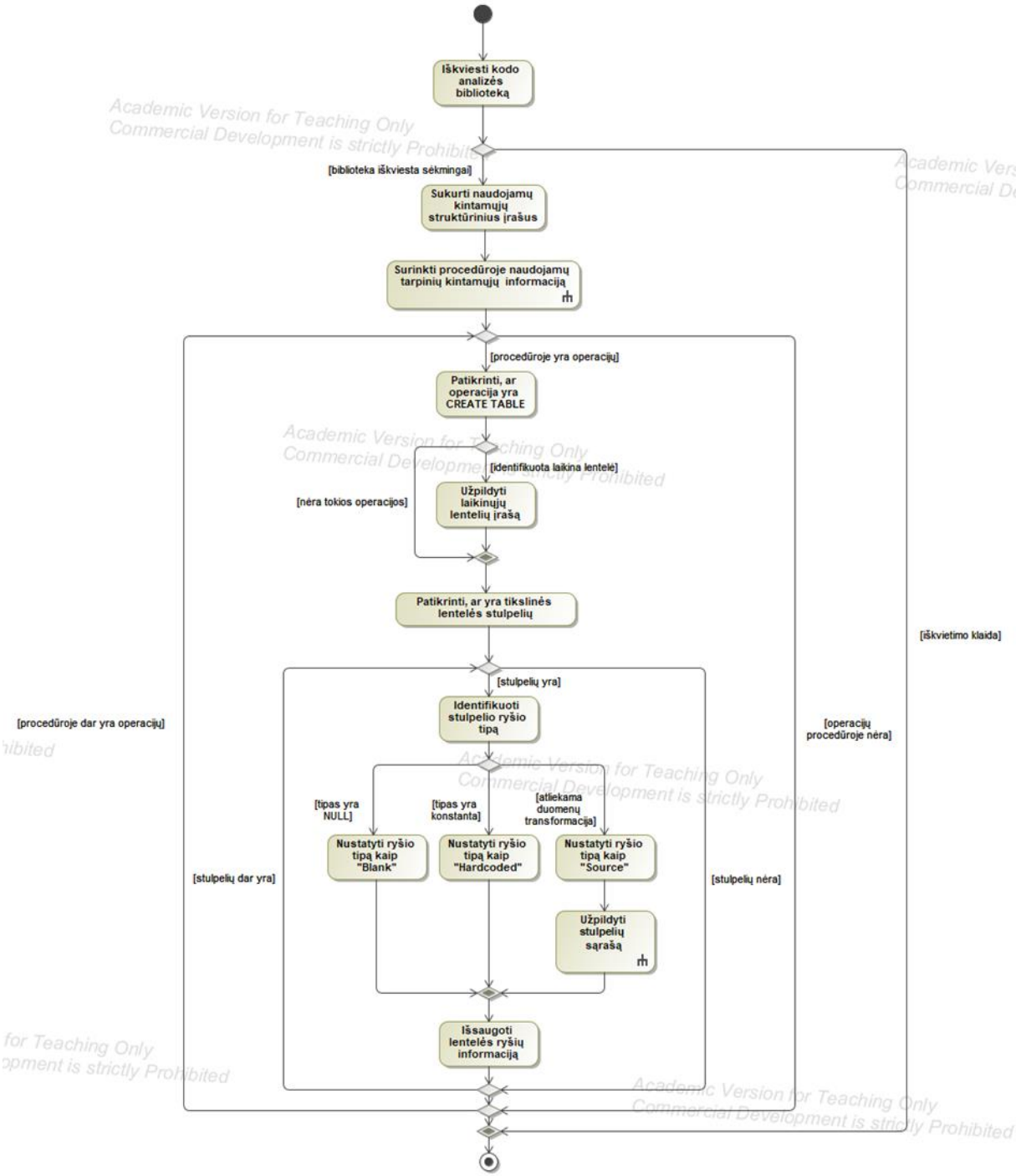
Tuo atveju, kai reikšmės tipas yra „Source“, šis elementas yra sujungiamas su duomenų lentelės mazgu iš kurios jis kilęs, taip nurodant, kuriai duomenų lentelei vaizduojamas stulpelio elementas

priklauso. Tuo tarpu reikšmėms, kurios yra „Hardcoded“ arba „Blank“, jungtys su šaltinio lentelėmis nėra būtinos. Tokiu atveju atvaizduojamas tik pats veiksmas, apibrėžiančiu priskyrimo tipą.

Kiekvienai procedūroje paminėtų tikslinės duomenų lentelės stulpelių yra sukuriami atitinkami mazgai. Šie sujungiami kartu su šaltinio lentelės stulpeliais. Priklausomai nuo ryšio tipo, šie du elementai gali būti sujungiami vienu-su-daug, daug-su-vienu arba vienu-su-vienu ryšiais.

Siekiant nuosekliai atskleisti kuriamos UML diagramų generavimo metodikos veikimo logiką, būtina neapsiriboti vien tik galutinių diagramų kūrimo veiksmų aprašymu. Kadangi dauguma veiklų tiesiogiai remiasi anksčiau paruoštais programiniais struktūriniais įrašais, tokiais kaip laikinieji kintamieji, duomenų šaltinių stulpeliai ir filtravimo sąlygos, pateikiami šių įrašų paruošimo procesų teoriniai aprašymai. Parodoma, kaip pirminė SQL užklausų informacija yra transformuojama į struktūrizuotus duomenis, vėliau naudojamus UML veiklos ir klasių diagramų formavimui.

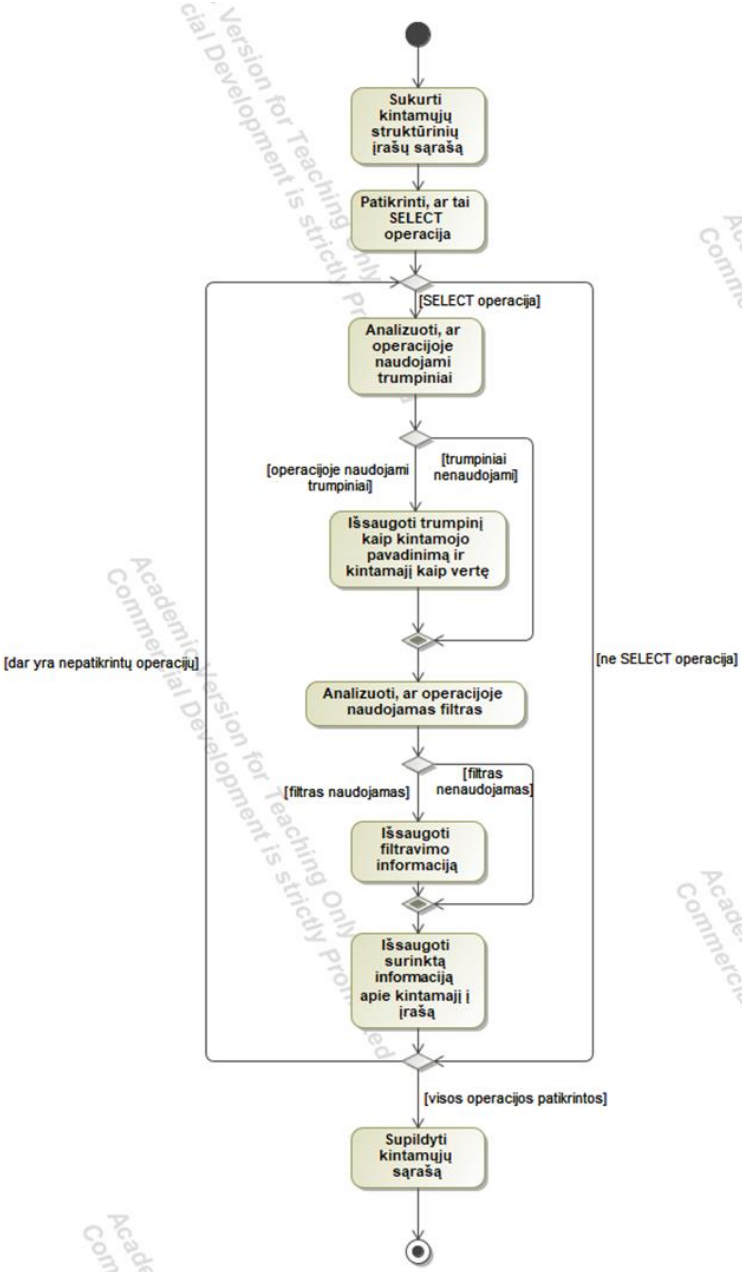
Duomenų ryšių (angl. *mapping*) sukūrimo logika (21 pav.) yra esminis žingsnis paruošiant reikalingą informaciją apie transformuojamą ETL procedūrą į UML veiklos diagramos struktūrą. Jos tikslas – nustatyti, kokie tai šaltiniai ir kokiu būdu jie susiję su duomenų paskirties struktūra. Tam naudojamas metodas, remiantis kodo analizės biblioteka, laikinų kintamųjų identifikavimui ir naudojamų operacijų analizei.



21 pav. Duomenų ryšių struktūrinių įrašų kūrimo logika

Procesas pradedamas nuo visos SQL užklausos paruošimo. Sujungiamos atskiros teksto eilutės į vieną vientisą užklausą, kuri perduodama bibliotekai. Jeigu analizė sėkminga, inicijuojamas kintamųjų, laikinų lentelių ir duomenų ryšių sąrašų išvalymas bei paruošimas naujai informacijai. Tolesniame etape išrenkami visi laikinieji kintamieji (22 pav.), dažniausiai iš „SELECT“ operacijų. Šie kintamieji kaupiami, nes jų reikšmės gali būti naudojamos kaip duomenų šaltiniai arba sąlygų dalys. Vėliau vykdoma pagrindinių SQL operacijų analizė. Kiekvienas sakinys (angl. *statement*) analizuojamas, siekiant nustatyti, ar tai yra laikinos lentelės kūrimo („CREATE TABLE“) operacija.

Jeigu lentelės pavadinime aptinkamas atitinkamas ženklas (pvz., „#“ *SQL Server* sintaksėje), iš tos operacijos išrenkami stulpelių pavadinimai ir saugomi kaip laikinos duomenų lentelės.



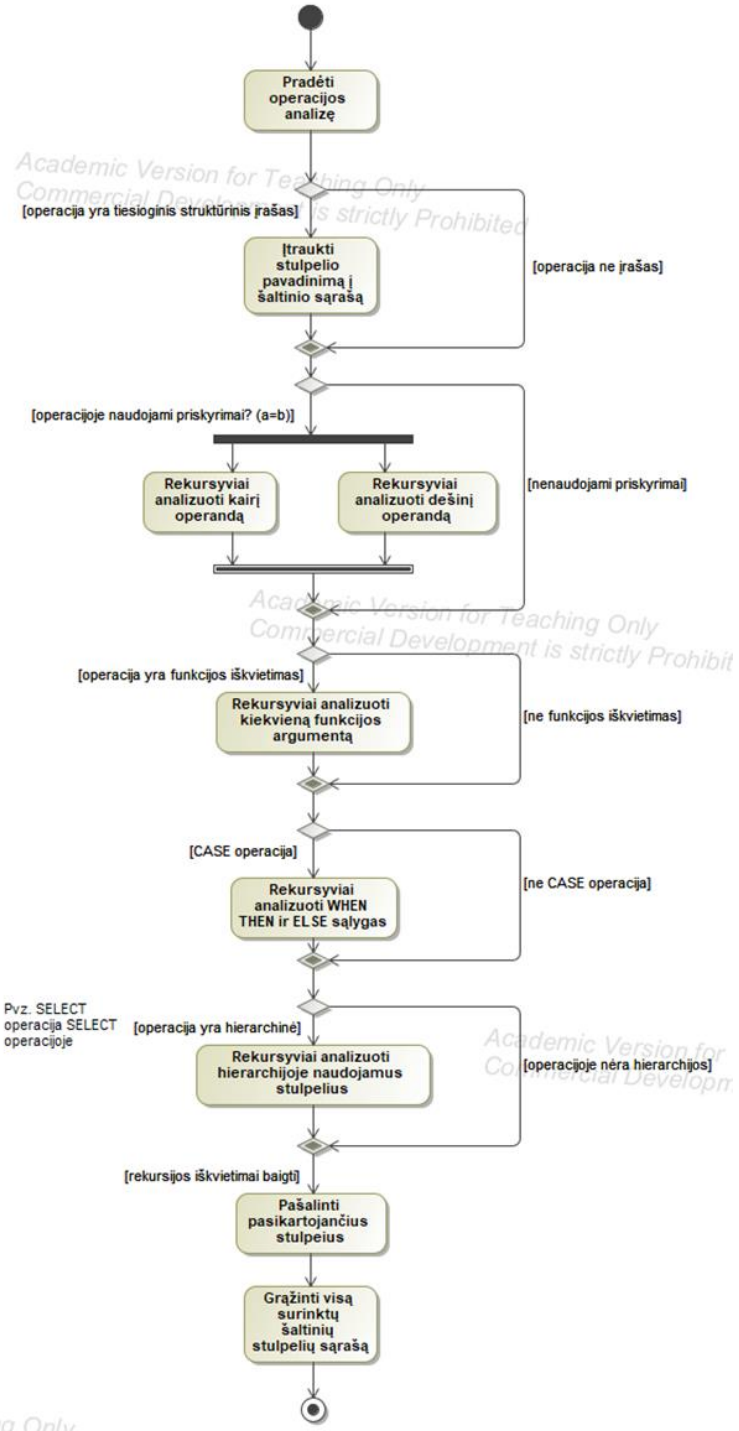
22 pav. Surinkti procedūroje naudojamų tarpinių kintamųjų informacija veiklos diagrama

Jeigu procedūroje naudojama lentelė nėra laikina, jos stulpeliai yra saugomi kaip galutinės duomenų struktūros atributai. Kiekvienas tikslinės duomenų lentelės stulpelis yra siejamas su atitinkama reikšme „SELECT“ užklausoje, esančioje „INSERT“ viduje. Ši reikšmė yra papildomai klasifikuojama pagal jos prigimtį:

- jeigu reikšmė yra „NULL“, duomenų ryšio tipas žymimas kaip „Blank“ (tuščias);
- jeigu reikšmė yra konstanta (pvz., skaičius arba tekstas), tai laikoma „Hardcoded“ reikšme;
- jeigu reikšmė yra duomenų transformacijos operacija, kurią galima susieti su konkrečiu šaltiniu ar funkcija, tai klasifikuojama kaip „Source“.

Tuo atveju, kai reikšmė priskiriama kaip „Source“, atliekamas papildomas veiksmas – jos analizė, siekiant identifikuoti, kokie šaltiniai (lentelės ir stulpeliai) yra naudojami (23 pav.). Tam pasitelkiama rekursija, kuri detaliai analizuoja kiekvieną operaciją:

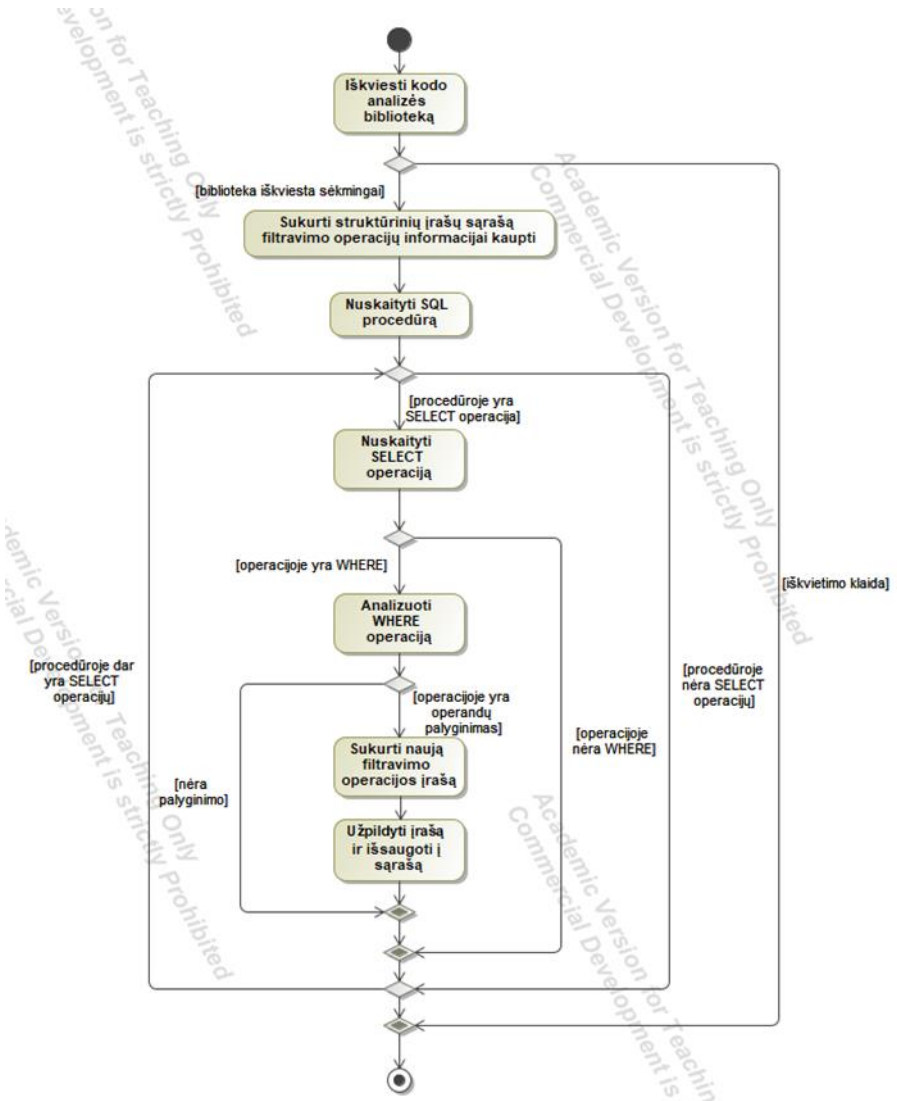
- jeigu tai paprastas duomenų priskyrimas iš šaltinio stulpelio į tikslinę, jis įtraukiamas į šaltinių sąrašą;
- jeigu tai kelių šaltinio stulpelių sujungimo į vieną transformacija, yra analizuojami abu operandai;
- jeigu tai funkcijos kvietimas, išanalizuojami visi naudojami kintamieji;
- jeigu tai sąlyginė „CASE“ operacija, rekursyviai analizuojamos „WHEN“, „THEN“ ir „ELSE“ šakos;
- jeigu operacija yra hierarchinė (angl. *subquery*), kiekviena grąžinama reikšmė nagrinėjama atskirai minėtos rekursijos būdu.



23 pav. Užpildyti stulpelių sąrašą veiklos diagrama

Po visų analizės etapų surinktas stulpelių sąrašas išvalomas nuo pasikartojimų ir naudojamas kaip pagrindas duomenų ryšio struktūriniam įrašui sukurti. Šis įrašas apima tikslinio stulpelio pavadinimą, šaltinio stulpelius (pvz. Jeigu ryšys „Source“, užpildomi stulpelių pavadinimai), reikšmės tipą bei kitą reikalingą informaciją. Galiausiai visi surinkti duomenų ryšiai kaupiami sąraše, kuris bus naudojamas tolesniems diagramų generavimo žingsniams.

Siekiant teisingai modeliuoti ETL procesų eigą, būtina taip pat nustatyti ir išsaugoti filtravimo logiką (24 pav.). Šiam tikslui atliekama filtravimo sąlygų identifikacija ir jų struktūrizavimas į struktūrinius įrašus.



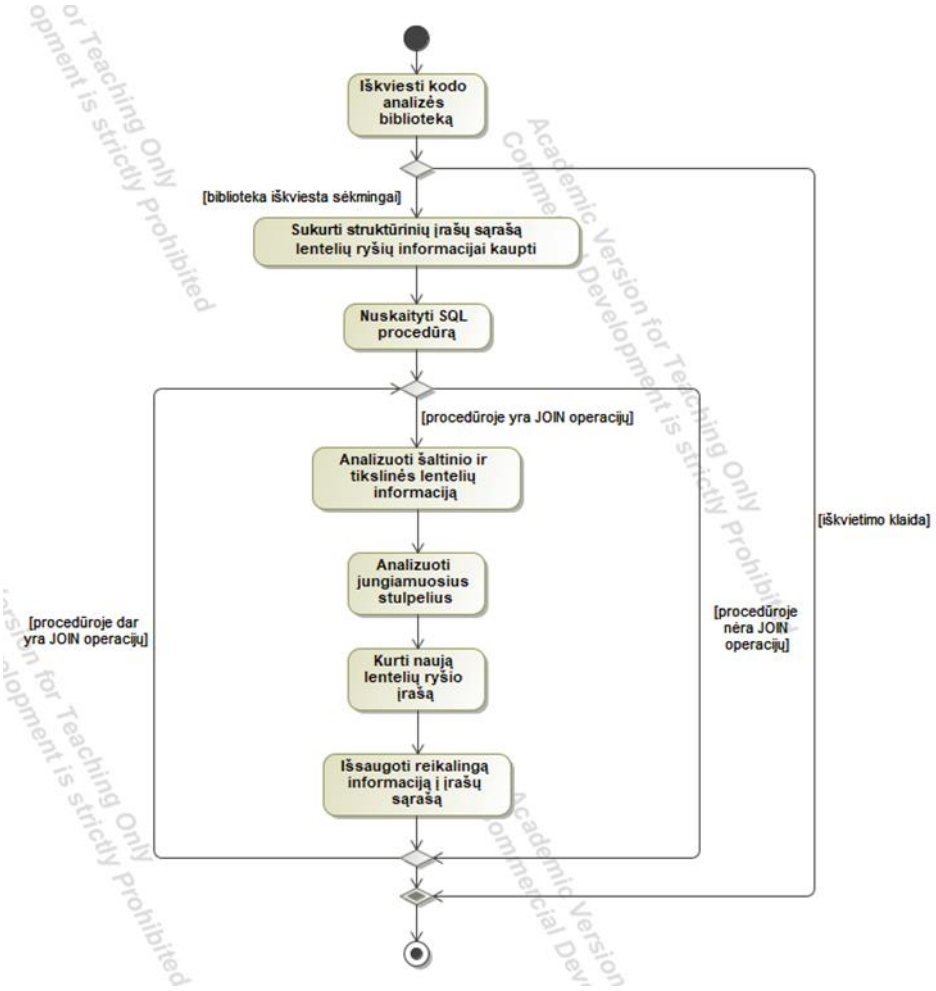
24 pav. Filtravimo struktūrizuotų įrašų kūrimo logikos UML veiklos diagrama

Šio proceso pradžioje analizuojamos SQL operacijų struktūros, dėmesį skiriant filtravimo sąlygoms, apibrėžiamoms naudojant įvairius palyginimo operatorius (pvz., lygybę, nelygybę, didesnės arba mažesnės reikšmės palyginimus). Kiekviena nustatyta filtravimo sąlyga yra transformuojama į atskirą struktūrinį įrašą, kuriame kaupiama informacija apie filtravimo stulpelį, duomenų lentelės pavadinimą ir taikomą palyginimo operaciją.

Operatoriai, aptikti SQL išraiškose, papildomai interpretuojami taip, kad būtų suteikiama aiški ir naudotojui suprantama reikšmė. Pavyzdžiui, simbolis „=“ yra konvertuojamas į reikšmę „Equals“, o „!=“ – į „Not Equals“. Ši papildoma informacija leidžia vėliau UML diagramose tiksliau perteikti duomenų filtravimo logiką.

Sukaupti filtravimo sąlygų struktūriniai įrašai vėliau naudojami kuriant atitinkamus veiksmus veiklos diagramose, taip užtikrinant, kad duomenų srautų modelyje būtų tinkamai atspindėtas filtravimo veiksmų vaidmuo ir jų taikymo specifika.

Norint modeliuoti duomenų lentelių sujungimus ir užtikrinti teisingą duomenų transformacijų vizualizaciją, būtina identifikuoti, kokios lentelės yra tarpusavyje susietos ir kokiomis sąlygomis vykdomi jų sujungimai (25 pav.).



25 pav. Lentelių ryšių struktūrinių įrašų kūrimo logikos UML veiklos diagrama

Analizuojamos SQL užklausos, dėmesį skiriant duomenų sujungimo operacijoms, tokioms kaip vidiniai sujungimai („INNER JOIN“), kairiniai sujungimai („LEFT JOIN“) ir kiti standartiniai jungimo būdai. Kiekvienam nustatytam sujungimui struktūrizuotai fiksuojama pirminė lentelė, prijungiamoji lentelė, naudojamas sujungimo tipas bei konkretūs stulpeliai, pagal kuriuos vykdoma sąsaja.

Analizės metu identifikuojami sujungimo sąlygų komponentai – kuris lentelės stulpelis yra susiejamas su kitos lentelės stulpeliu. Tokiu būdu sukuriami duomenų sąsajų struktūriniai įrašai, kuriuose saugoma visa būtina informacija apie duomenų šaltinių tarpusavio ryšius ir jų loginę priklausomybę.

Šie duomenys vėliau naudojami kuriant veiklos diagramų veiksmus, kurie atvaizduoja duomenų šaltinių tarpusavio sąveiką, nurodant tiek sujungimo tipą, tiek stulpelių sąsajas.

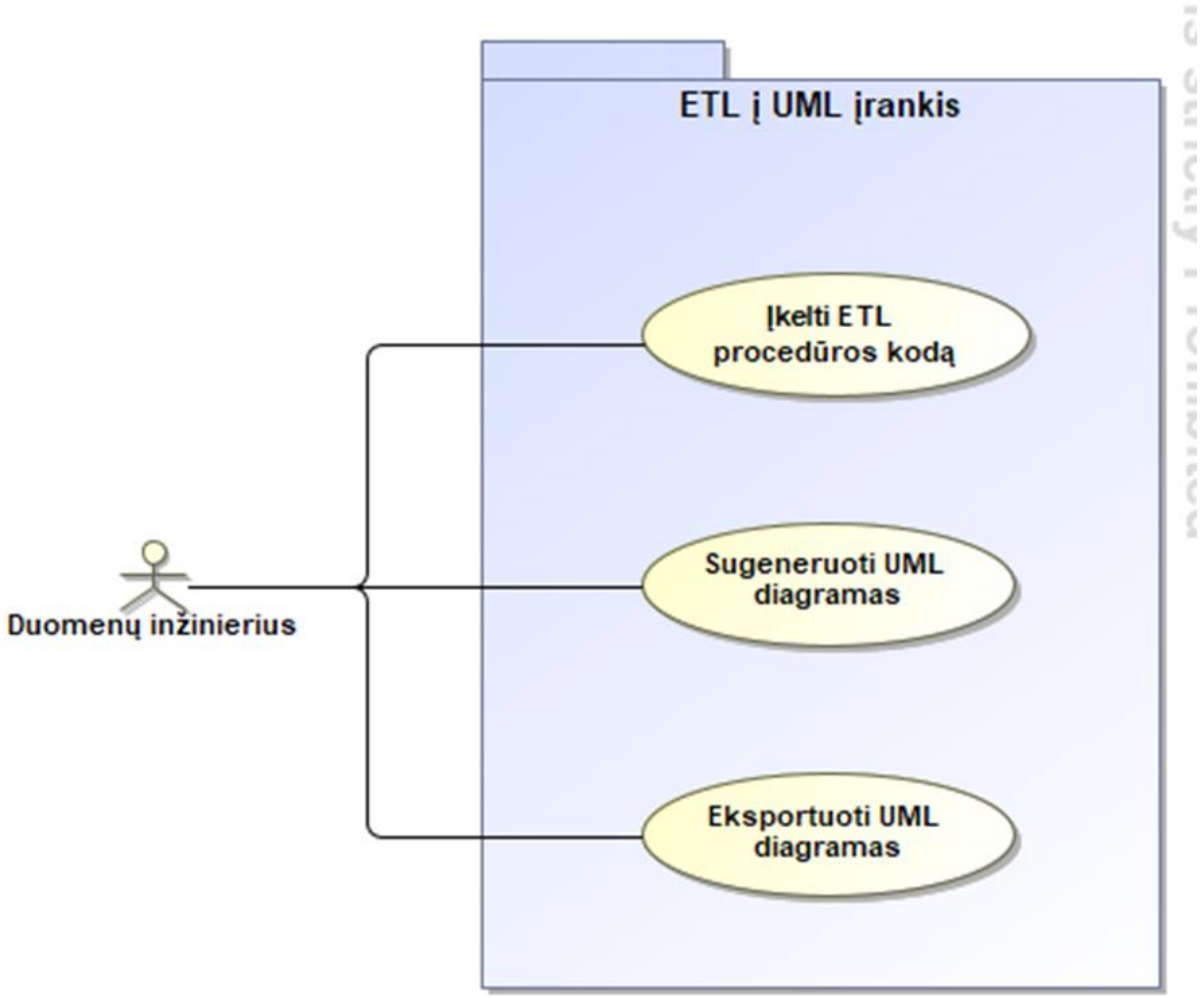
2.4. Reikalavimų apibendrinimas

Antrame skyriuje apžvelgti ETL procedūrų generavimo į UML diagramas reikalavimai. Buvo nustatyta, kad norint atlikti šių dviejų sričių konvertavimą, privalu procedūrą struktūrizuotai paversti į SQL metamodelio elementus. Šiai užduočiai atlikti bus pasinaudota kodo analizės biblioteka, kuri ne tik gali konvertuoti programinio kodo dalis į metamodelio elementus, bet taip pat ir pateikti informaciją apie naudojamas duomenų lenteles klasių diagramoje. Be šios diagramos, taip pat bus pateikiama ir veiklos diagrama, kurioje nurodomos ETL proceso dalys kaip veiklos diagramos veiksmai. Šias veiklas bus galima peržiūrėti smulkiau – kokie duomenų šaltiniai naudojami, kokios filtravimo operacijos, duomenų priskyrimai bei kiti aspektai naudojami kiekvienam etape. Norint sėkmingai sugeneruoti veiklos diagramą, SQL metamodelio elementai bus paverčiami į atitinkamus UML veiklos diagramos metamodelio elementus. Šių metamodelių vertimo procesai taip pat iliustruoti veiklos diagramomis.

3. ETL procedūrų vizualizavimo naudojant UML veiklos ir klasių diagramas skirto įrankio realizacijos projektas

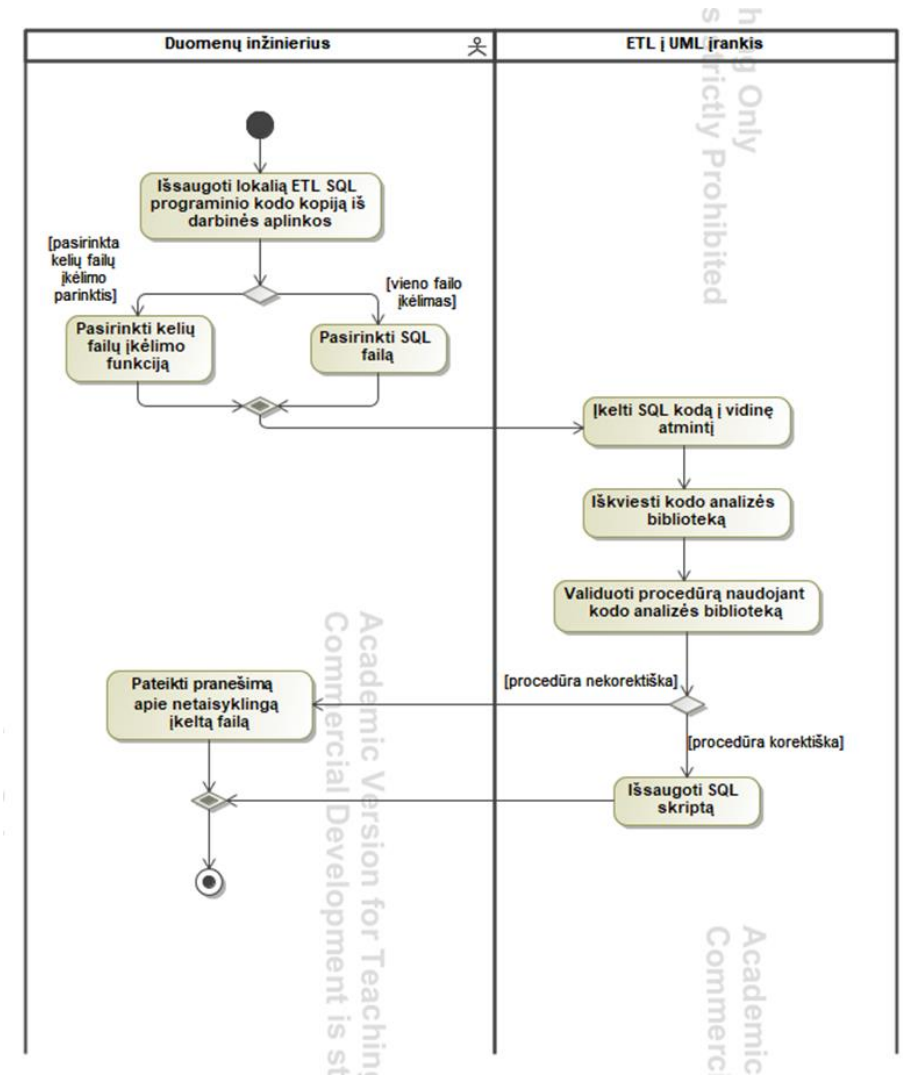
3.1. Vizualizavimo įrankio panaudojimo atvejai

Įrankis, skirtas ETL procedūrų dokumentavimui, turi tris pagrindinius panaudojimo atvejus, kuriuose realizuojami failo įkėlimo, UML diagramų generavimo ir jų eksporto panaudojimo atvejai (26 pav.). Šie veiksmai sudaro įprastą naudotojo sąveiką su įrankiu, leidžiantį transformuoti ETL procedūrų SQL programinį kodą į UML veiklos ir klasių diagramas. Visi veiksmai atliekami naudojantis paprasta grafine naudotojo sąsaja, o svarbiausia įrankio funkcionalumo dalis – automatinis generavimas be papildomų konfigūracijos veiksmų.



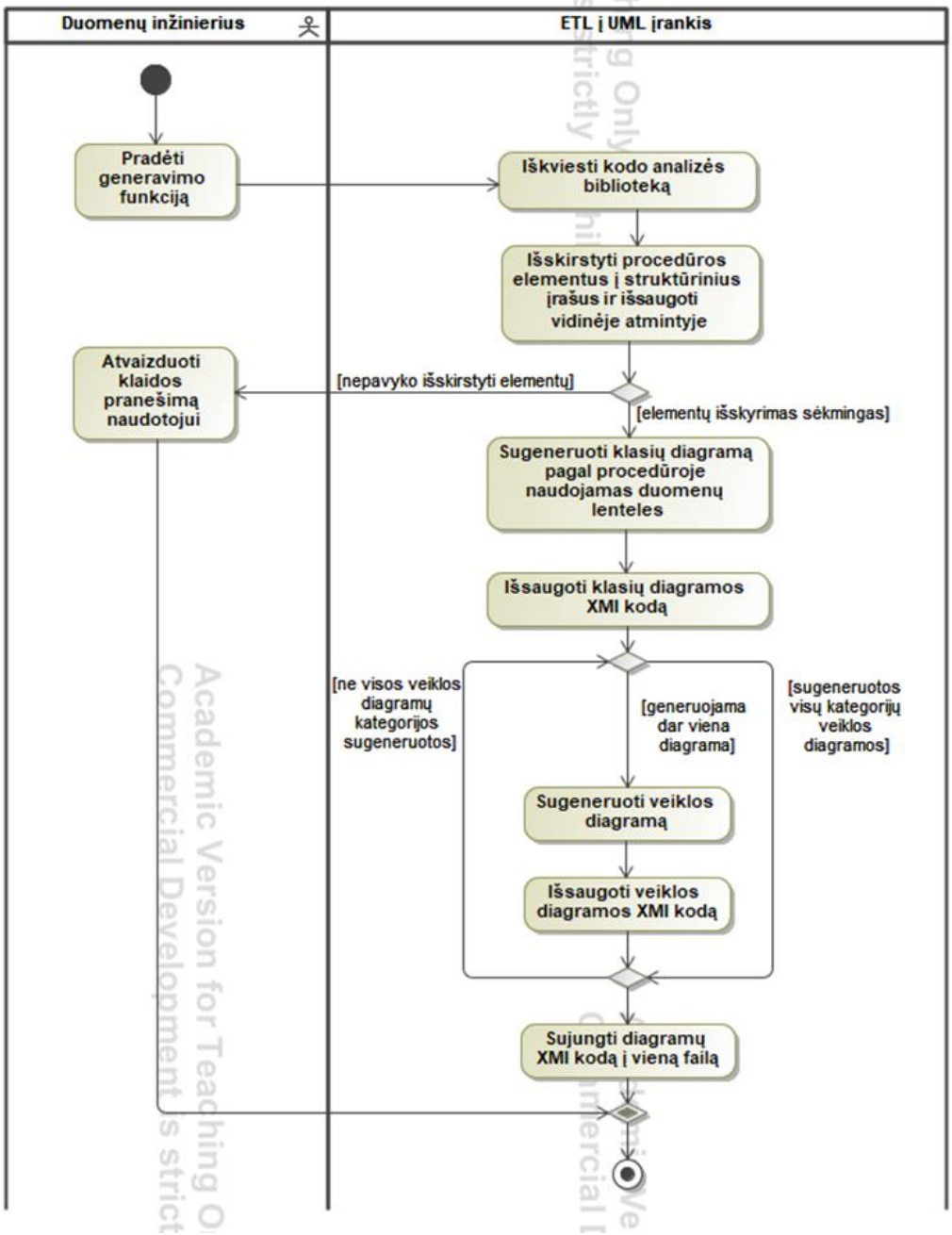
26 pav. ETL procedūrų generavimui į UML diagramas įrankio panaudojimo atveju diagrama.

ETL kodo įkėlimas (27 pav.) įrankyje pradedamas pasirinkus SQL failą iš naudotojo kompiuterio. Šiame žingsnyje įrankis pasinaudoja kodo analizės biblioteką, konvertuojančią pateiktą SQL kodą į struktūrizuotą duomenų medį. Ši analizė vykdoma automatiškai, o naudotojas informuojamas apie sintaksės klaidas, jeigu tokių aptinkama. Jei analizė sėkminga, failo turinys toliau laikomas vidinėje įrankio atmintyje.



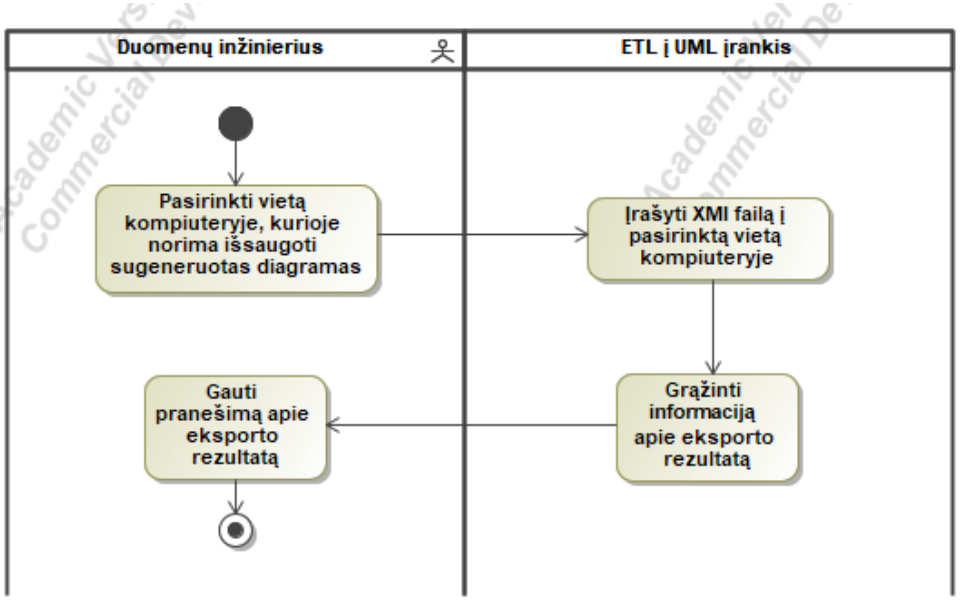
27 pav. Įkelti ETL procedūros kodą panaudojimo atvejo veiklos diagrama.

UML diagramos generavimas (28 pav.) remiasi SQL kodo analize. Gautas sintaksės medis toliau apdorojamas – klasifikuojami pagrindiniai elementai, tokie kaip filtravimai, sujungimai, lentelių tipai ir kt. Šie elementai įrankio viduje atitinka UML veiklos arba klasių diagramos komponentus. Diagramos sugeneruojamos standartiniu XMI formatu, kuris leidžia diagramas importuoti į kitus įrankius. Kiekviena veiklos diagramos generavimo iteracija išsaugo jos informaciją vidinėje atmintyje. Klasių bei skirtingų veiklos diagramų XMI kodo dalys yra sujungiamos į vieną failą.



28 pav. Sugeneruoti UML diagramą panaudojimo atvejo veiklos diagrama.

Diagramos eksportas (29 pav.) leidžia naudotojui gauti sugeneruotą modelį XMI faile. Įrankio realizacijoje šis failas išsaugomas naudotojo pasirinktoje sistemos vietoje pavadinimu „output.xmi“,. Galimybė peržiūrėti diagramą pačiame įrankyje nėra numatyta – šis aspektas paliktas kitiems modelių redagavimo įrankiams.



29 pav. Eksportuoti UML diagramą panaudojimo atvejo veiklos diagrama.

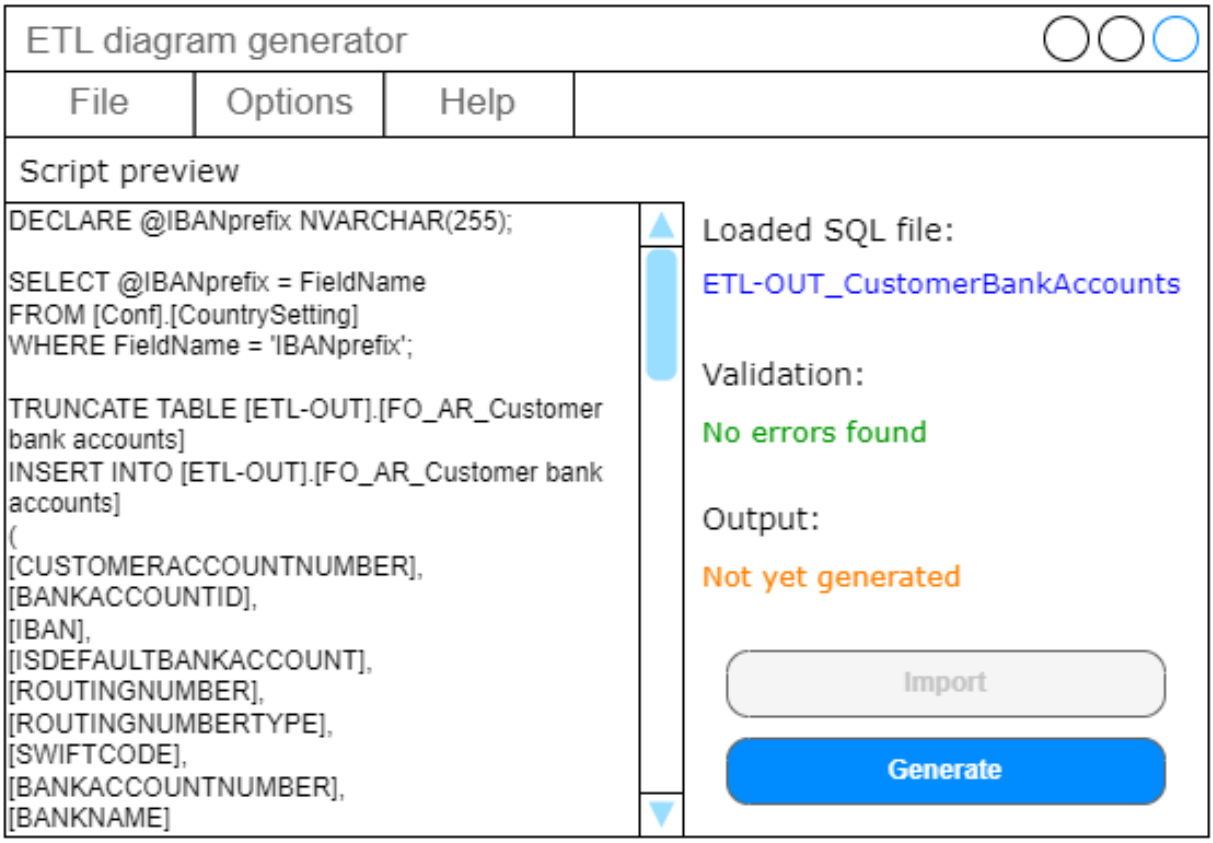
Šie trys panaudojimo atvejai sudaro visą ETL procedūros pavertimo į UML diagramas procesą. Nors naudotojo sąveika yra minimali, analizės ir konvertavimo procesas užtikrina, kad dokumentavimo rezultatai būtų pateikti nuosekliai bei tiksliai.

3.2. Vizualizavimo įrankio naudotojų sąsajos modelis

Įrankio naudotojo sąsaja realizuota kaip vieno lango programa, kurioje naudotojui pateikiamos visos pagrindinės funkcijos (30 pav.). Sąsaja išdėstyta intuityviai – kairėje pusėje pateikiamas įkeltos ETL procedūros kodo peržiūros langas, dešinėje – informacija apie įkeltą failą, validacijos rezultatus bei diagramos generavimo būseną. Įrankis neskirstomas į atskirus modulius ar ekranus, nes funkcionalumas išskaidytas žingsniais, atitinkančiais visą procedūros dokumentavimo ciklą.

Įrankio analizės logika paleidžiama automatiškai – SQL failas įkeltas pasitelkiant failų įkėlimo funkciją, kuris palaiko tiek vieno, tiek kelių failų importo galimybę. Įkėlus failą, automatiškai iškviečiama kodo analizės biblioteka, kuri tikrina jo sintaksę. Analizės rezultatai pateikiami dešinėje sąsajos pusėje – tai leidžia greitai identifikuoti, ar procedūra tinkama tolimesniam apdorojimui. Jei nustatoma klaidų, jos pateikiamos informacinėje srityje. Informacinė sritis yra pateikiama naudotojui atskirai ją iškvietus. Klaidos ar nenumatytos situacijos fiksuojamos įrankio vidinėje logikoje, o naudotojui pateikiama aiški išvestis kaip iššokantis langas su reikalinga informacija, arba tekstas informacinėje srityje.

Paspaudus „Generate“ mygtuką, įrankis automatiškai sugeneruoja UML veiklos ir klasių diagramų XMI failą. Failas įrašomas į naudotojo pasirinktą vietą kompiuteryje iš anksto nustatytu pavadinimu. Tokiu sprendimu siekiama sumažinti sąsajos sudėtingumą ir užtikrinti greitą naudojimąsi be perteklinių žingsnių.



30 pav. ETL procedūrų generavimui į UML diagramas įrankio naudotojo sąsajos eskizas

3.3. Vizualizavimo įrankio komponentų diagrama

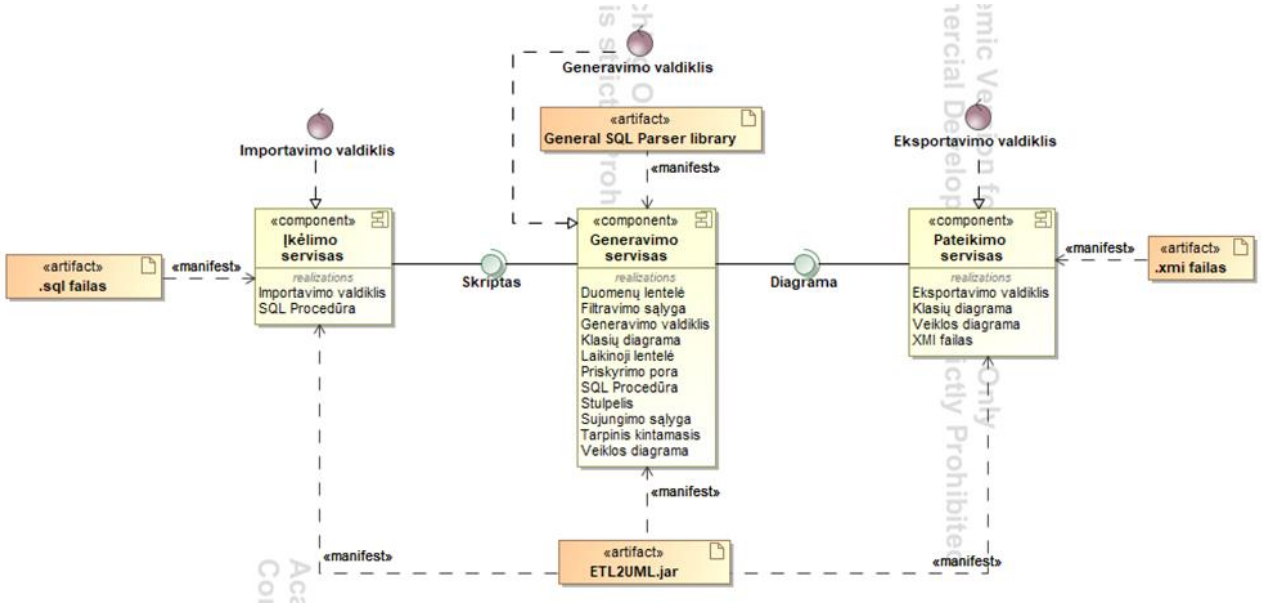
Komponentų diagrama (31 pav.) vaizduoja ETL procedūrų konvertavimo į UML diagramas įrankio vidinę architektūrą, suskirstytą į tris pagrindinius servisus pagal funkcijas – įkėlimą, generavimą ir rezultatų pateikimą. Tokia struktūra nurodo loginį įrankio veikimo suskirstymą į modulius, kurių kiekvienas valdo atskirą generavimo proceso etapą.

Įkėlimo servisas apdoroja pirminę naudotojo pateiktą įvestį – SQL failą. Šis komponentas realizuoja esybę „SQL Procedūra“, kurioje saugomas visas įkeltas kodas bei jo atributai, tokie kaip pavadinimas ar turinys. Komponente taip pat egzistuoja esybė „Importavimo valdiklis“, kuris atsakingas už failų įkėlimą bei korektiškumo patikrą. Kadangi šiame etape dirbama tiesiogiai su fiziniu failu, servisas susietas su „.sql“ failo artefaktu. Artefaktas šiuo atveju žymi ne tik failo egzistavimą sistemoje, bet ir nurodo, kad komponentas dirba su įvedamais duomenimis, kurių struktūra dar nėra analizuota.

Generavimo servisas sudaro didžiausią dalį sistemos logikos ir realizuoja esybes, tiesiogiai susijusias su SQL kodo analize bei UML diagramų kūrimu. Pvz., esybės „Filtravimo sąlyga“, „Sujungimo sąlyga“ bei kt. atitinka SQL kodo elementus, kurie sugeneruojami po kodo analizės ir toliau naudojami diagramų konstravimui. Šiam komponentui taip pat priskirtas Generavimo valdiklis, kuris koordinuoja veiksmų seką nuo kodo analizės bibliotekos kvietimo iki UML diagramos sudarymo. Kadangi visa struktūrizavimo veikla vykdoma pasitelkiant kodo analizės biblioteką, ši pateikta kaip atskiras artefaktas jos originaliu pavadinimu. Tokio sprendimo tikslas – atskirti išorinių įrankių priklausomybę nuo kuriamo įrankio, taip suteikiant daugiau lankstumo jį palaikant ar keičiant.

Pateikimo servisas yra paskutinis logiškai atskirtas vienetas, skirtas sugeneruotų duomenų pavertimui į XMI formato failą. Esybės „Klasių diagrama“ ir „Veiklos diagrama“ reprezentuoja sugeneruotų modelių struktūrą, o XMI failas – jau paruoštą eksportuoti išvestį. Eksportavimo valdiklis šiame kontekste atsakingas už sąsają su naudotoju. Jis pateikia galutinį failą bei valdo informacinius pranešimus. Panašiai kaip ir įkėlimo servise, šis komponentas dirba su fiziniu failu – todėl susietas su „.xmi“ failo artefaktu.

Galiausiai, visi komponentai susieti su bendru „.jar“ tipo artefaktu „ETL2UML.jar“. Šis artefaktas žymi viso įrankio vykdomąjį vienetą (angl. *executable*), kuriame integruoti visi trys serviso blokai.



31 pav. ETL procedūrų generavimui į UML diagramas įrankio komponentų diagrama

3.4. Vizualizavimo įrankio diegimo diagrama

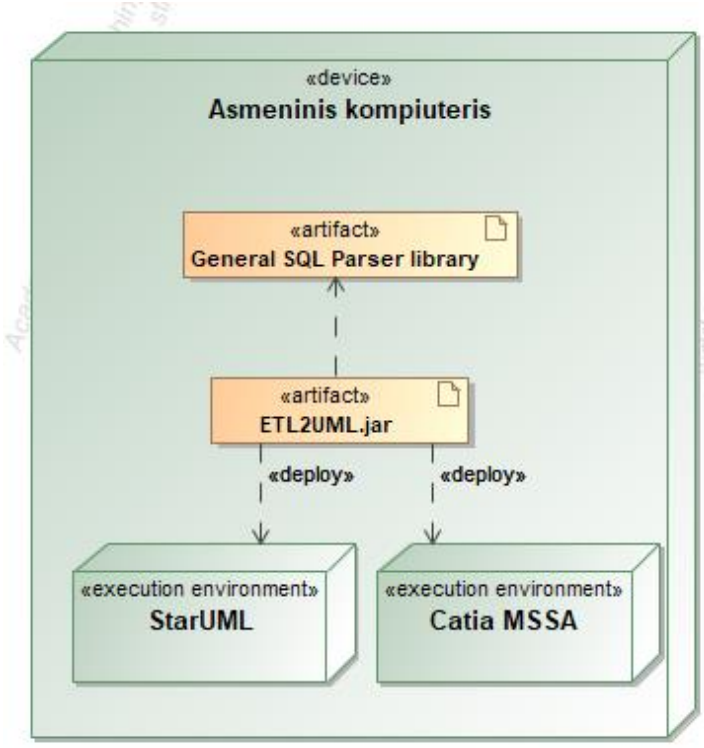
Diegimo diagrama (32 pav.) vaizduoja programinės įrangos komponentų fizinį išdėstymą naudotojo darbo aplinkoje. Įrankis, skirtas ETL procedūrų konvertavimui į UML diagramas, diegiamas kaip *Java* programinės kalbos vykdomasis failas („ETL2UML.jar“), kurį naudotojas paleidžia savo kompiuteryje.

Sėkmingam veikimui būtina kodo analizės biblioteka (diagramoje žymima originaliu pavadinimu), atsakinga už ETL procedūros analizę. Ši priklausomybė į programos vykdomąjį failą įtraukta jau diegimo metu, todėl naudotojui papildomų veiksmų atlikti nereikia. Tokia integracija leidžia išvengti rankinio konfigūravimo ir sumažina diegimo klaidų tikimybę.

Programa neturi integruoto diagramų peržiūros funkcionalumo, todėl sugeneruotų rezultatų atvaizdavimui reikalinga papildoma modelių analizės programinė įranga. Diagramoje pavaizduotas ryšys su „StarUML“ ir „Catia Magic Systems of Systems Architect“ įrankiais žymi galimas XMI failų peržiūros priemones. Šie konkretūs įrankiai pasirinkti iliustraciniams tikslams – realiame naudojime naudotojas gali rinktis ir kitus UML diagramas palaikančius įrankius, jei jie suderinami su sugeneruotu XMI formatu.

Nors XMI yra standartizuotas formatas (ISO/IEC 19503:2005 [**Error! Reference source not found.**]), praktikoje įvairios modeliavimo programos jį interpretuoja skirtingai. Dėl šios priežasties,

pagrindinis išvesties failas buvo pritaikytas darbui su „Magic Systems of Systems Architect“ programa. Bandant tą patį failą importuoti į kitus įrankius, gali iškilti suderinamumo problemų, susijusių su žymėjimo niuansais ar kitais aspektais.



32 pav. ETL procedūrų generavimui į UML diagramas įrankio diegimo diagrama

4. ETL procedūrų vizualizavimo naudojant UML veiklos ir klasių diagramas įrankio realizacija ir testavimas

4.1. Vizualizavimo įrankio realizacijos ir veikimo aprašas

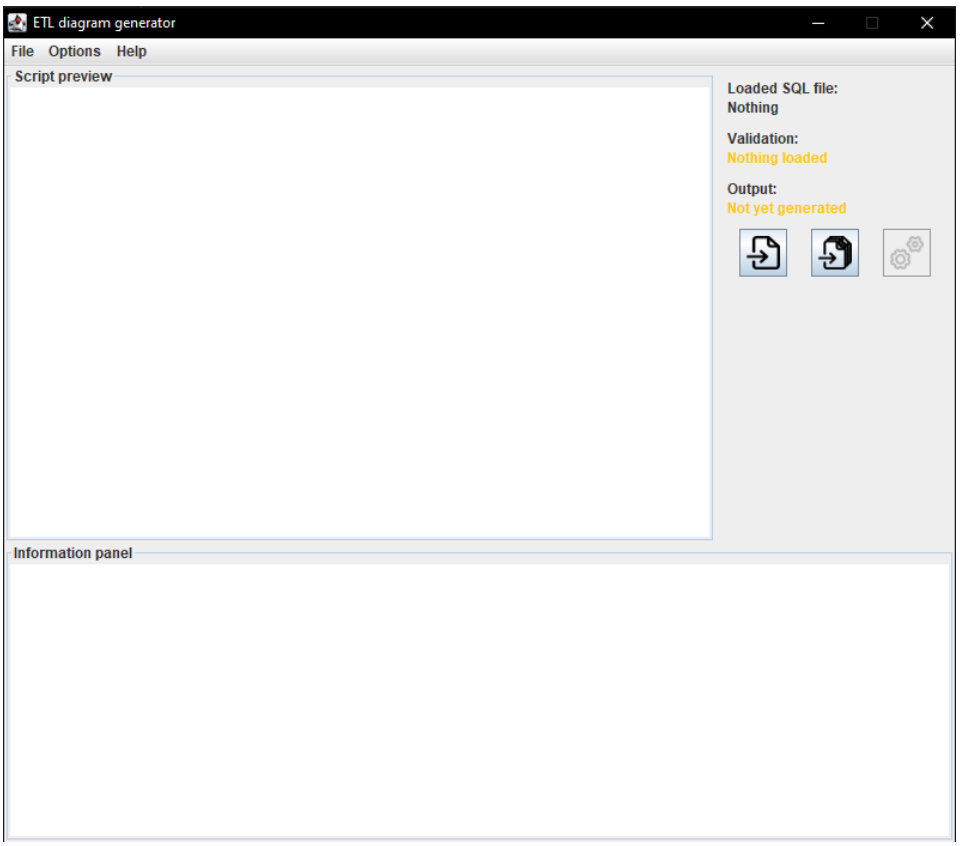
ETL procedūrom generuoti į UML diagramas skirtam įrankiui realizuoti naudojama *Java* programinė kalba (17.0.1 versija). Į šį įrankį taip pat integruota ir kodo analizės biblioteka, kurioje pateikiami įvairūs algoritmai bei komponentai, skirti analizuoti, formatuoti ar kitaip keisti SQL užklausas neprisijungiant prie duomenų bazių. Šiame įrankyje formatavimo bei redagavimo galimybės nebus naudojamos – reikalinga tik patogi sąsaja procedūrų analizei.

3.1 skyriuje minėti panaudojimo atvejai yra atitinkamai realizuoti įrankyje – procedūra yra įkeliama į vidinę atmintį, naudojant biblioteką skirtingos kodo dalys priskiriamos objektams, kuriai galima manipuliuoti bei nurodyta metodika sugeneruojamas XMI failas, kuris pateikiamas naudotojui. Šiai užduočiai atlikti realizuoti du pagrindiniai programiniai metodai – „loadProcedure“ ir „generateDiagrams“ (33 pav.).

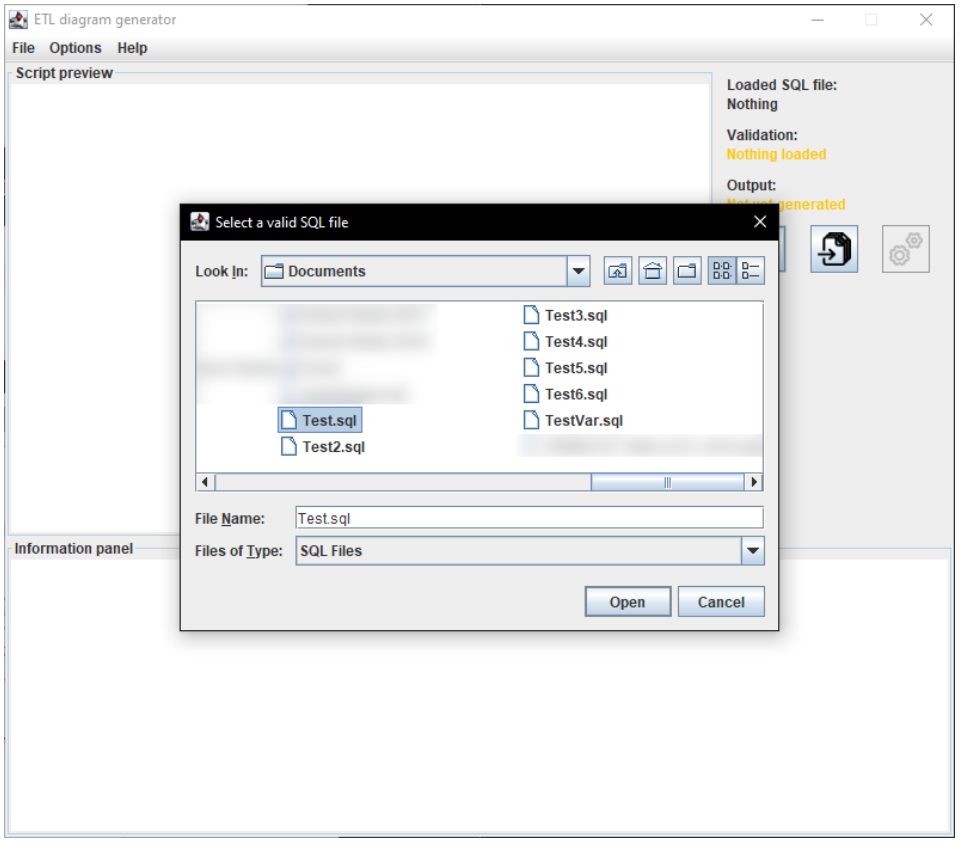
```
public void loadProcedure(String location){
    readFile(location);

    if(ScriptFile.exists()) {
        getInformation(ScriptFile);
        removeComments(ScriptLines);
        showText(ScriptLines);
        tryParser(ScriptLines);
    }
}

public void generateDiagrams(String savePath){
    tryParser(ScriptLines);
    createClass(Entities);
    createXMI(Class, savePath);
}
```

33 pav. Pagrindiniai įrankio metodai.

Naudotojui, įsijungus įrankio programą, yra pateikiamas langas (34 pav.), kuris sukurtas pagal naudotojo sąsajos modelį (30 pav.). Pasirinkus pirmą mygtuką eilėje, pateikiamas naujas iššokantis langas (35 pav.), kuriame nurodoma reikalingo procedūros failo vieta kompiuteryje. Pasirinkus failą, sužadinamas „loadProcedure“ metodas, kuris pateiktą failą nuskaito, pašalina komentarus, pateikia informaciją derinimo (angl. *debugging*) režimu bei iškviečia biblioteką. Objektas, kuris pateikiamas jai, yra procedūros failo eilučių sąrašas. Taip yra todėl, nes vieno failo biblioteka nepriima – jis turi būti išskaidytas į pavienes „String“ objekto tipo eilutes. Bibliotekos iškvietimas naudojamas abiejuose metoduose. Pirmą kartą biblioteka patikrina, ar šis pateiktas failas yra sintaksiškai teisingas SQL kodas. Ne SQL programinio kodo failai jau yra limituojami failo pasirinkimo lange.



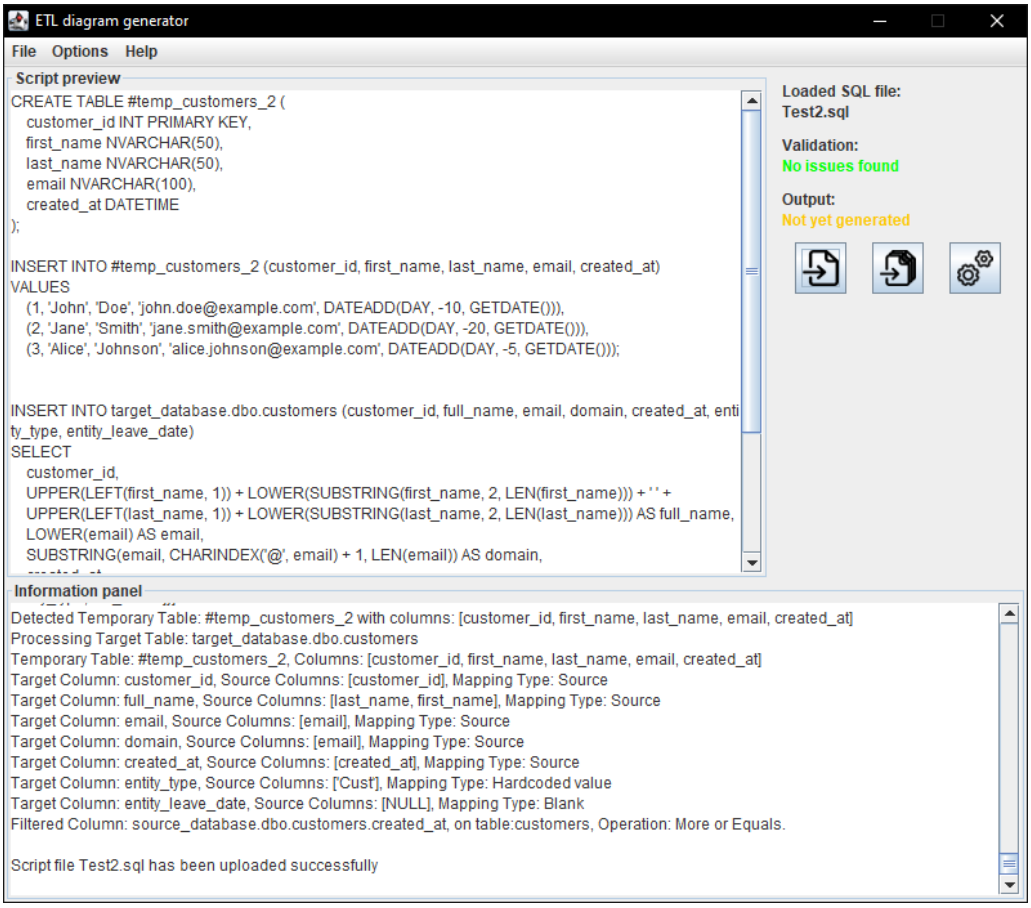
34 pav. Pirminis įrankio langas.



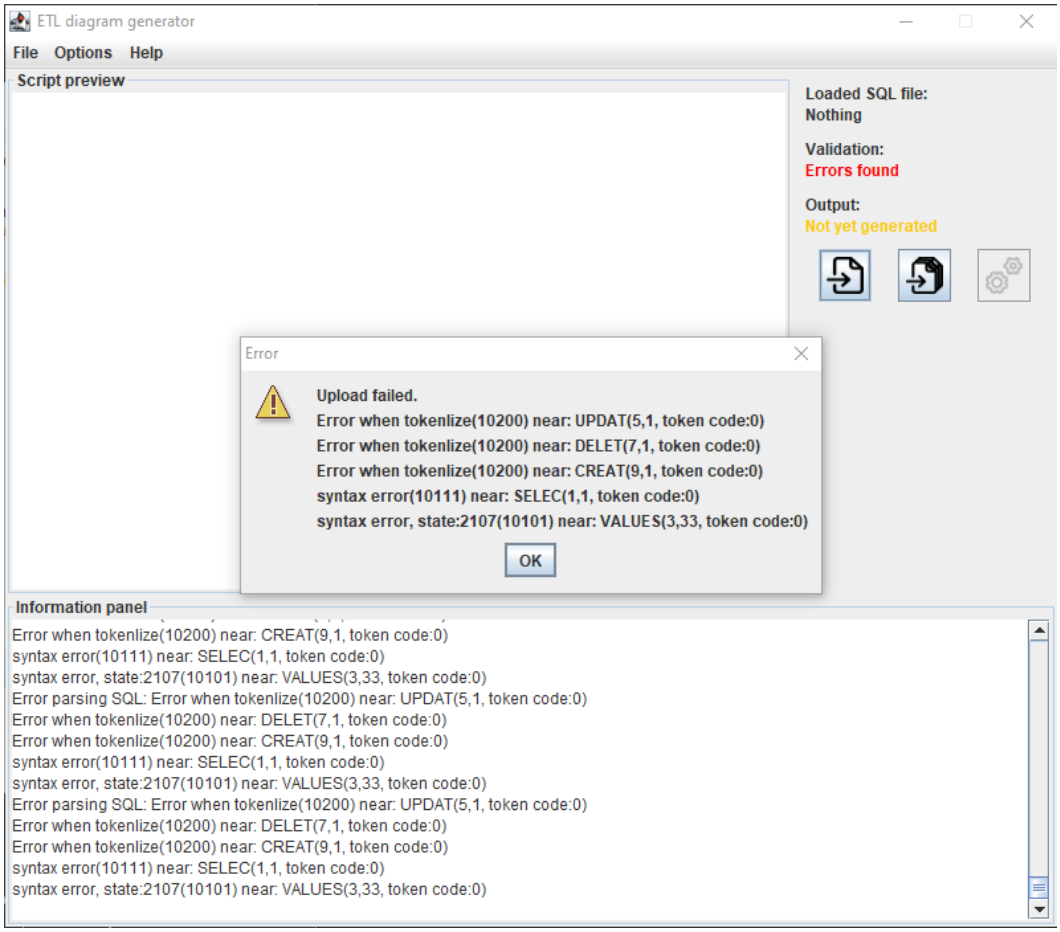
35 pav. Procedūros **f**ailo pasirinkimo langas.

Įkėlus pasirinktą failą, jo turinys yra pateikiamas kairėje pusėje esančiame teksto lauke (36 pav.). Taip galima įsitikinti, ar naudotojo pasirinktas failas visgi yra tas, kurį norima apdoroti. Taip pat, papildomai dešinėje lango pusėje pateikiami failo patikrinimo rezultatai, pasirinktos procedūros pavadinimas. Atlikus failo įkėlimą, naudotojui iškyla galimybė pasirinkti trečiąjį mygtuką eilėje, kuris pradeda diagramos generavimo procesą. Jeigu patikra sėkminga, pateikiamas atitinkamas pranešimas. Tačiau jeigu programinis kodas nėra korektiškas, naudotojas informuojamas iššokančiu langu (37 pav.). Pateikiami klaidų pranešimai yra sugeneruojami kodo analizės bibliotekos pagalba.

Papildomai naudotojui yra suteikiama galimybė įkelti bei sugeneruoti daugiau nei vieną procedūrą. Norint atlikti šį veiksmą reikia pasirinkti kelių failų įkėlimo ypatybę – antrąjį mygtuką eilėje. Skirtumas tarp vienos ir kelių procedūrų įkėlimo yra tas, kad failo peržiūros lange pastaruoju atveju nėra pateikiamas procedūros kodas.

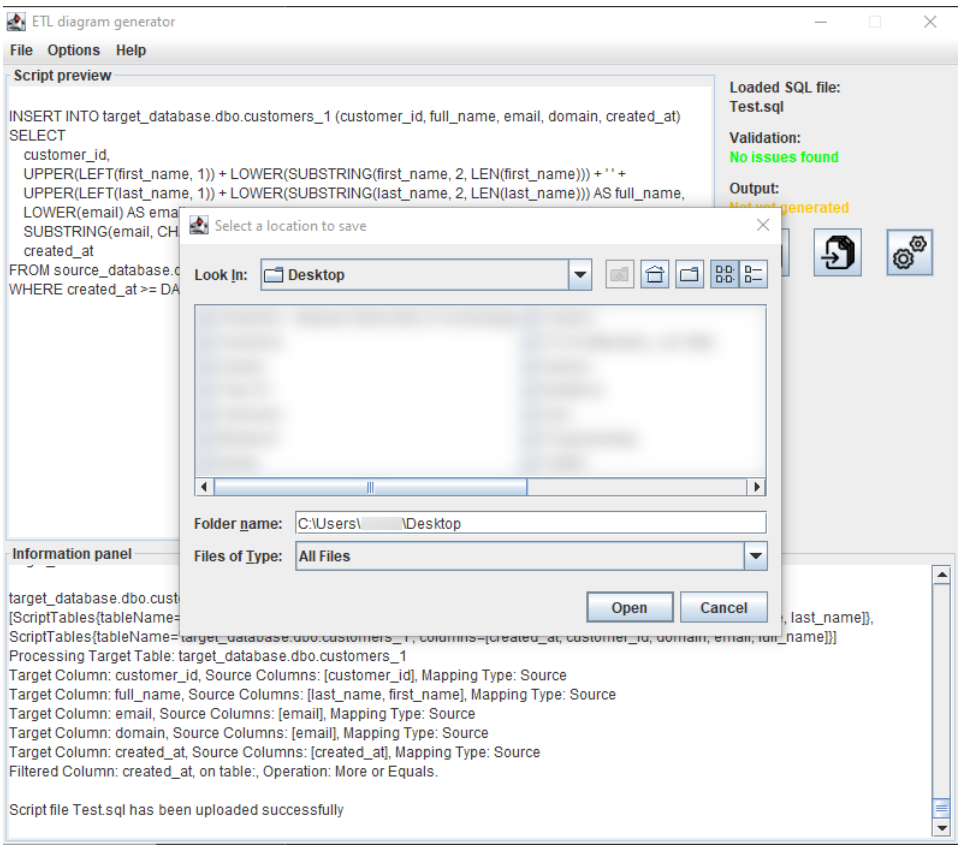


36 pav. Įrankio langas su įkeltos procedūros atvaizdavimu.

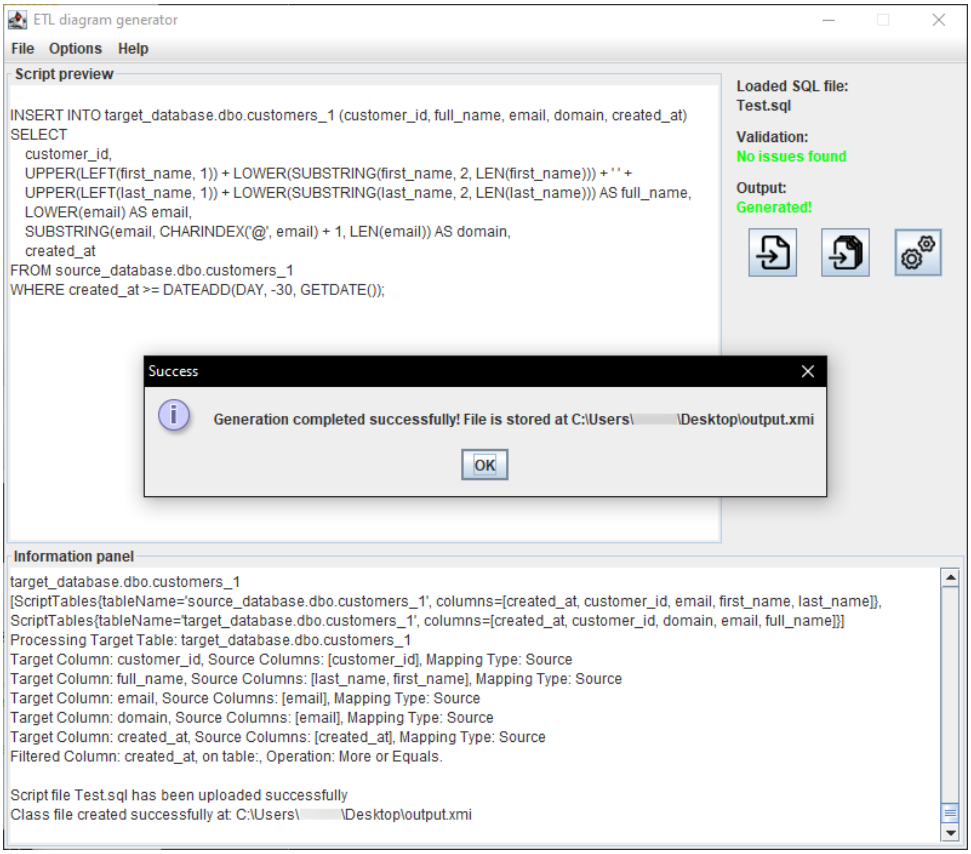


37 pav. Klaidos pranešimas, įkėlus nekorektišką procedūrą.

Naudotojui pasirinkus diagramos generavimo ypatybę (trečiasis mygtukas eilėje), pateikiamas failo išsaugojimo langas, kuriame galima pasirinkti tik aplankų direktorijas (38 pav.). Pasirinkus vietą, naudotojas informuojamas pranešimu apie sėkmingą išsaugojimą (39 pav.). Šio įrankio sugeneruotos diagramos pavadinimas yra „output.xmi“.



38 pav. Diagramos išsaugojimo kompiuteryje langas.



39 pav. Diagramos sėkmingo išsaugojimo pranešimas.

Klasių diagrama

Klasių diagramai sugeneruoti naudojamas metodas pasinaudoja bibliotekos lentelių išskirstymo algoritmu. Pateikus tinkamą failą, biblioteka pateikia procedūroje naudojamus duomenų lentelių bei šių lentelių stulpelių sąrašus. Šie sąrašai yra surūšiuojami pagal eiliškumą, kur pirmoji aptikta lentelė bus pirmoji lentelių sąraše, o jai priklausantys duomenų stulpeliai bus pirmieji stulpelių sąraše. Jų pavadinimų struktūra taip pat yra suvienodinama – lentelių pavadinimo struktūra yra:

$$duomenų\_bazės\_pavadinimas.schema.lentelė$$

ir stulpelių:

$$duomenų\_bazės\_pavadinimas.schema.lentelė.stulpelis$$

Šis formatavimas bei duomenų suskirstymas į du sąrašus leidžia nesudėtingai sukurti XMI formato failą diagramai. Šio failo struktūra (40 pav.) tarp įvairių pateikiamų procedūrų skirsis tik jos „packagedElement“ bei „ownedAttribute“ turiniu – pradžia ir pabaiga visada bus tokia pati.

```
<xmi:XMI xmlns:uml="http://www.omg.org/spec/UML/20110701"
  xmlns:xmi=" http://www.omg.org/spec/UML/20110701">
  <uml:Package xmi:id="ppp" xmi:label="p1">
   <packagedElement xmi:type="uml:Class" xmi:id="ccc" name="c1">
    <ownedAttribute xmi:type="uml:Property" name="a1"/>
    <ownedAttribute xmi:type="uml:Property " name ="a2"/>
   </packagedElement >
  </uml:Package>
</xmi:XMI>
```

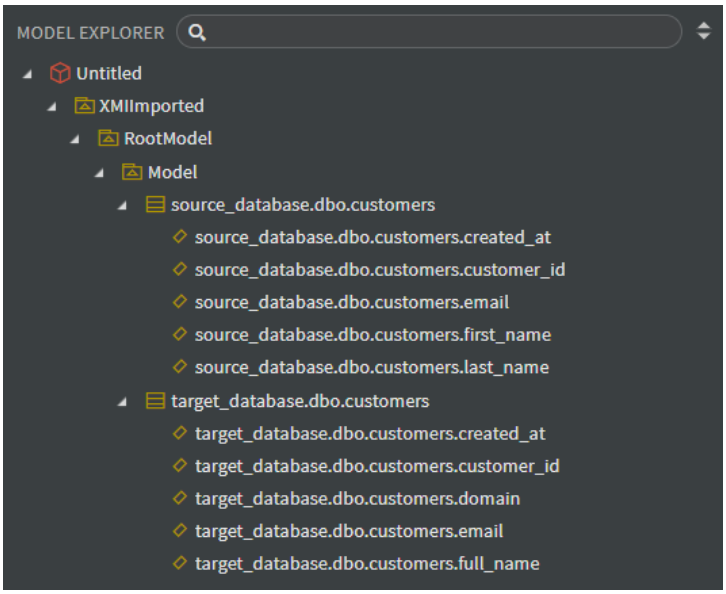
40 pav. Pavyzdinė XMI failo struktūra [19].

Visa šio failo informacija laikoma „String“ tipo objekte, kurio turinys, naudojant *Java* „Files.copy()“ metodą, yra įrašomas į išvesties failą. Tačiau šiam objektui gauti reikia sudėti tris kintamuosius – failo pradžią, kuri niekada nesikeičia, klasių ir jos atributų turinį, išreikštą XMI formatu bei pabaigą, kuri irgi taip pat nesikeičia. Antram kintamajam gauti pasinaudojama dvigubu ciklu (angl. *nested loop*), kur einama per lentelių bei stulpelių sąrašus. Lentelių informacija yra įrašoma kaip „packagedElement“ tipas, kuris yra aukštesnis hierarchijos lygis nei „ownedAttribute“. Lentelės stulpeliai yra įrašomi į pastarąjį tipą.

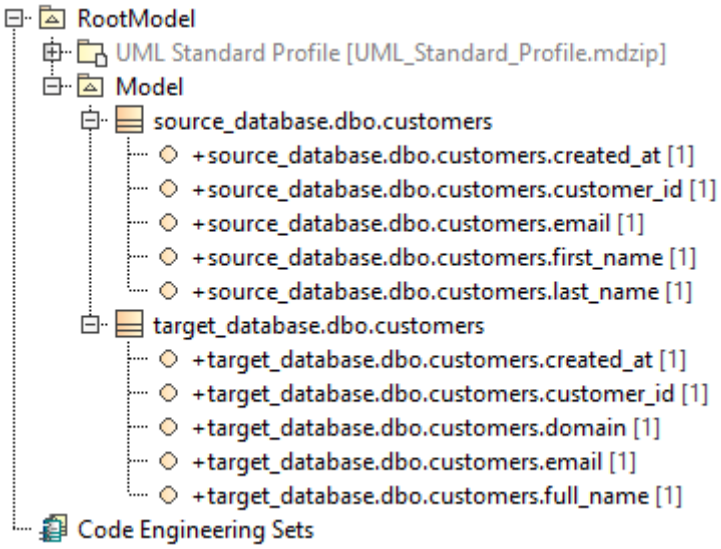
Šiuose cikluose taip pat egzistuoja patikrinimas – jeigu stulpelio pilno pavadinimo pradžia panaši į tikrinamosios lentelės pradžią, tai reiškia, kad stulpelis priklauso tai lentelei. Priešingu atveju, vidinis ciklas yra sustabdomas, o išoriniame cikle einama prie kitos lentelės palyginimui bei „packagedElement“ tipas yra uždaromas. Šiuos veiksmus atlikus, sugeneruojamas klasių diagramos failas (41 pav.), kurį galima įkelti į UML modeliavimo programas, tokias kaip *StarUML* (42 pav.) arba *Magic Systems of Systems Architect* (43 pav.).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:uml="http://schema.omg.org/spec/UML/2.0" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1">
    <xmi:Documentation exporter="UML Generator" exporterVersion="0.1"/>
    <uml:Model xmi:id="root" xmi:type="uml:Model" name="RootModel">
        <packagedElement xmi:id="Model" name="Model" visibility="public" xmi:type="uml:Model">
            <packagedElement xmi:id="source_database.dbo.customers" name="source_database.dbo.customers" visibility="public" i
                <ownedAttribute xmi:id="source_database.dbo.customers.created_at" name="source_database.dbo.customers.created_
                <ownedAttribute xmi:id="source_database.dbo.customers.customer_id" name="source_database.dbo.customers.custome
                <ownedAttribute xmi:id="source_database.dbo.customers.email" name="source_database.dbo.customers.email" visibi
                <ownedAttribute xmi:id="source_database.dbo.customers.first_name" name="source_database.dbo.customers.first_na
                <ownedAttribute xmi:id="source_database.dbo.customers.last_name" name="source_database.dbo.customers.last_name
            </packagedElement>
            <packagedElement xmi:id="target_database.dbo.customers" name="target_database.dbo.customers" visibility="public" i
                <ownedAttribute xmi:id="target_database.dbo.customers.created_at" name="target_database.dbo.customers.created_
                <ownedAttribute xmi:id="target_database.dbo.customers.customer_id" name="target_database.dbo.customers.custome
                <ownedAttribute xmi:id="target_database.dbo.customers.domain" name="target_database.dbo.customers.domain" visi
                <ownedAttribute xmi:id="target_database.dbo.customers.email" name="target_database.dbo.customers.email" visibi
                <ownedAttribute xmi:id="target_database.dbo.customers.full_name" name="target_database.dbo.customers.full_name
            </packagedElement>
        </packagedElement>
    </uml:Model>
</xmi:XMI>
```

41 pav. Įrankio sugeneruoto XMI klasių diagramos failo turinys.



42 pav. Įrankio sugeneruotos klasių diagramos peržiūra per *StarUML* programą.



43 pav. Įrankio sugeneruotos klasių diagramos peržiūra per *Magic Systems of Systems Architect* programą.

Pirmojo hierarchijos lygio veiklos diagrama

Veiklos diagramai sugeneruoti taip pat yra pasinaudojamos bibliotekos funkcijos. Norint pateikti procedūroje naudojamas duomenų lenteles veiklos diagramoje, taikoma tokia pati logika kaip ir klasių diagramoje. Toliau, procedūra yra išskirstoma į individualias SQL komandas ir paverčiama į hierarchinę medžio struktūrą. Tokiu būdu tampa lengviau paversti šiuos elementus į UML metamodelio atitikmenis ir perteikti metodo logiką diagramoje.

Kadangi aukščiausio hierarchijos lygio veiklos diagrama nežymiai skiriasi pagal naudotojo pateikiama procedūrą, kai kurie jos elementai gali būti realizuoti nedinamiškai. Veiklos diagramoje pradžios ir pabaigos mazgai, duomenų užpildymo į tikslinę lentelę bei duomenų užkrovimo transformacijos darbams veiksmai visuomet egzistuos, nes be jų procedūra būtų netiksli. Būtent šie elementai realizacijos programiniame kode yra nekintami (angl. *hard-coded*).

Veiklos diagramos failui sukurti reikia kelių metodų (44 pav.) Kadangi naudotojui pateikiamas failas yra XMI formato, įrankis konstruoja teksto eilutes į logišką, kitų modeliavimo įrankių suprantamą, tekstinį failą. Kai kurie veiksmai diagramoje, pvz., duomenų įkėlimas, gali turėti keletą duomenų šaltinių, kurie privalo turėti ryšį su šiuo veiksmu. Tokiais atvejais, teksto failo konstravimas viename metode tampa labai nepatogus, nes „<node>“ atributas (kuris yra UML veiksmo išraiška XMI formate) gali turėti vieną arba daugiau jam priskirtų „<input>“ atributų. Šiai ir analogiškoms problemoms išspręsti realizacijoje naudojami atskiri „create<...>Action“ metodai kiekvienam dinamiškam veiksmui (45 pav.) Jų išvestys yra „String“ tipo objektai, kurie yra sudedami pagrindiniame „createActivityDiagram“ metode, o šis pateikia galutinį XMI failą. „SourceAndDestinationNodes“ bei „sourceAndDestinationConnections“ metodai veiklos diagramoje pateikia procedūroje naudojamas duomenų lenteles, priskiria dinamiškai kintančius įvesties bei išvesties smeigtukus (*InputPin, OutputPin*) ir juos sujungia tarpusavyje.

44 pav. Pagrindiniai veiklos diagramos generavimo metodai.

```
public String createLoadSourcesAction(ArrayList<String> tables){
    StringBuilder output = new StringBuilder();

    output.append("\t\t\t<node xmi:id=\"Sources\" name=\"Sources\" visibility=\"public\" " +
                "xmi:type=\"uml:CallBehaviorAction\" isLocallyReentrant=\"false\" isSynchronous=\"true\" behavior=\"sources_0\">\n");

    for (String table : tables) {
        if (!table.equals(destinationTable)) {
            output.append("\t\t\t\t<input xmi:id=\"input").append(inputCounter).append("\" visibility=\"public\" " +
                    "isStatic=\"false\" isLeaf=\"false\" isReadOnly=\"false\" isOrdered=\"false\" isUnique=\"false\"" +
                    " xmi:type=\"uml:InputPin\"/>\n");
            inputCounter++;
        }
    }

    output.append("\t\t\t</node>\n");

    return output.toString();
}
```

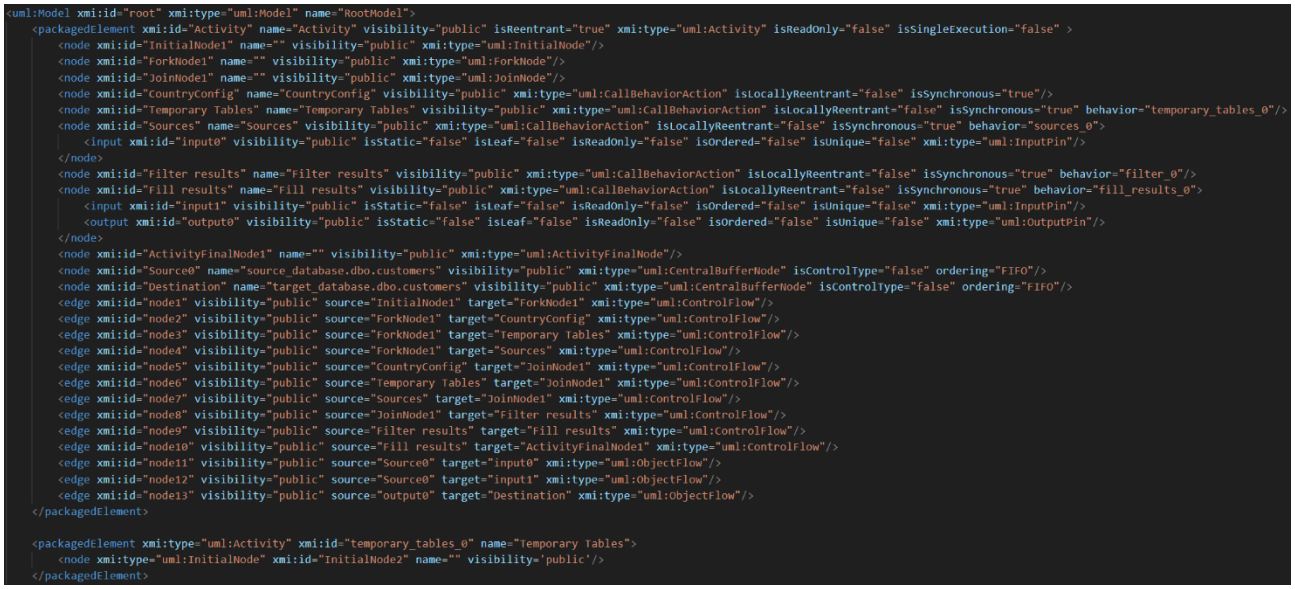
45 pav. Duomenų užkrovimo veiksmo generavimo programinis kodas.

Kadangi veiklos diagrama pateikiama dviem hierarchijos lygiais, kiekvienas veiksmas veiklos diagramoje savyje turi atitinkamą papildomą veiklos diagramą, kurioje pateikiama išsamesnę procedūros logiką. Tam realizuoti pasinaudojama „create<...>Diagram“ metodais. Jie, kaip ir prieš tai minėti metodai, pateikia tekstinius objektus, kurie sudedami pagrindiniame „createActivityDiagram“. Skirtumas tarp jų yra tai, kad jų nereikia įterpti į aukščiausio hierarchijos lygio XMI „<packagedElement>“ atributą. Taip yra dėl to, nes kiekviena diagrama yra kaip atskiras veiksmas, tik šie veiksmai yra tarpusavyje susiję – be pagrindinės diagramos šios papildomos

nepateiktų nieko informatyvaus. Visa šios veiklos diagramos struktūrą, atsižvelgiant į XMI hierarchijos reikalavimus, pateikta 46 pav. ir 5 lentelėje:

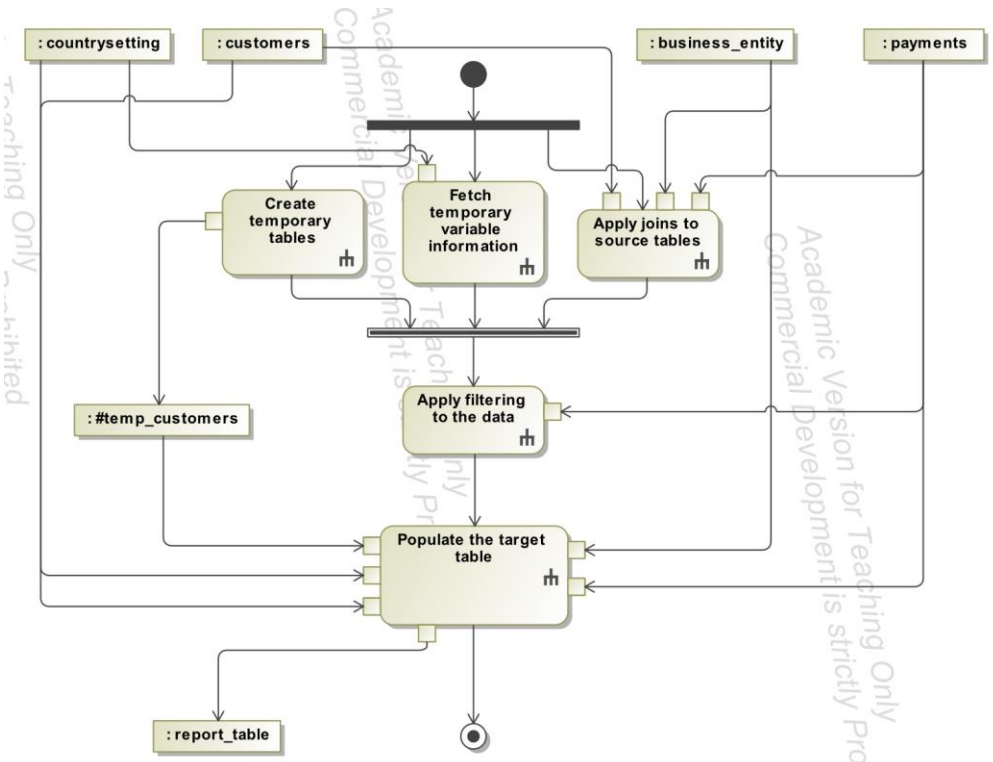
5 lentelė. Veiklos diagramos XMI failo struktūra.

1. Veiklos diagramos modelis (<Model>)
1.1. Aukščiausio lygio veiklos diagrama (<PackagedElement>)
1.1.1. Veiklos diagramos elementai (<Node>)
1.1.1.1. Įvesties bei išvesties smeigtukai (<Input> arba <Output>)
1.1.2. Veiklos diagramos elementų sujungimai (<Node>)
1.2. Duomenų užkrovimo veiklos diagrama (<PackagedElement>)
1.2.1. Veiklos diagramos elementai (<Node>)
1.2.1.1. Įvesties bei išvesties smeigtukai (<Input> arba <Output>)
1.2.2. Veiklos diagramos elementų sujungimai (<Node>)
1.3. ...

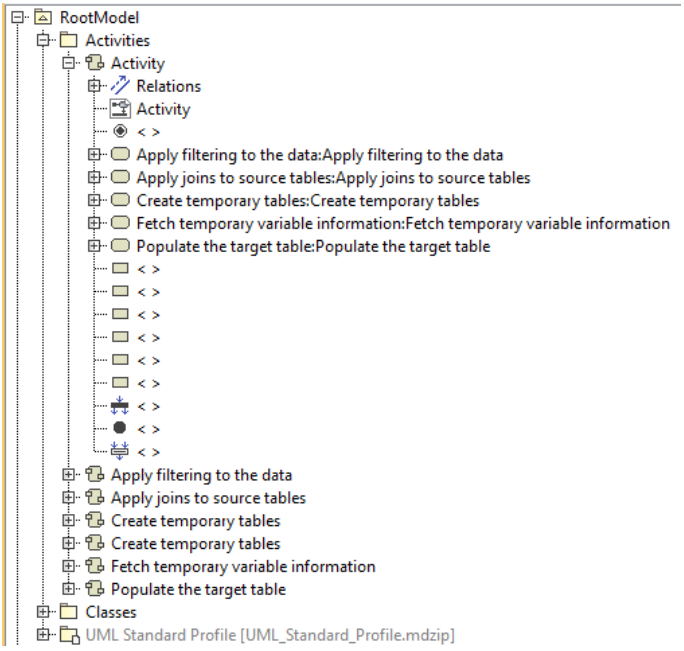


46 pav. Įrankio sugeneruoto XMI veiklos diagramos failo turinys.

Pagal šią hierarchiją, įrankis sugeneruoja atitinkamą XMI failą ir pateikia naudotojui, kurį galimą atidaryti per norimą UML modeliavimo įrankį – peržiūrėti diagramą (47 pav.) bei jos struktūrą (48 pav.)



47 pav. Įrankio sugeneruota veiklos diagramos peržiūra per *Magic Systems of Systems Architect* programą.



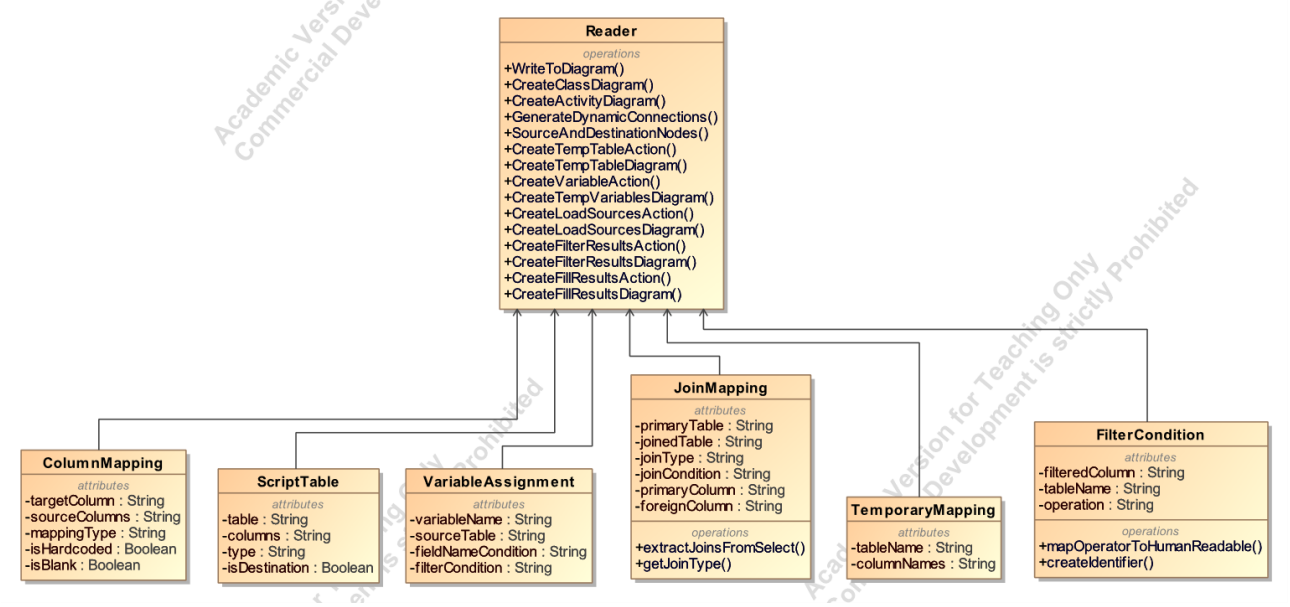
48 pav. Įrankio sugeneruotos veiklos diagramos elementų peržiūra per *Magic Systems of Systems Architect* programą.

Antrojo hierarchijos lygio veiklos diagramos

Antrojo hierarchijos lygio diagramos pateikia daugiau tikslios informacijos įrankio naudotojui apie dokumentuojamą ETL procesą. Pirmojo lygio veiklos diagramoje (47 pav.) šios diagramos yra

indikuojamos su „grėblio“ simboliu, kaip pvz. „Populate the target table“, „Apply filtering to the data“ ir t.t.

Programinė metodo realizacija egzistuoja įrankio „Reader“ klasėje (49 pav.). Tai yra pagrindinis komponentas, valdantis visą procesą – nuo SQL kodo analizės iki galutinio UML diagramos sugeneravimo ir eksportavimo. Klasė naudoja pagalbines klases, tokias kaip „JoinMapping“, „TemporaryMapping“ ar „VariableAssignment“, kurios saugo informaciją apie lentelių sujungimus, laikinas lenteles, kintamuosius ir kitus ETL proceso elementus.



49 pav. Įrankio klasių struktūra su nurodytais metodais bei kintamaisiais

Generuoti duomenų priskyrimo diagramą

„Populate the target table“ diagramos generavimui reikia, kad įrankis galėtų automatiškai identifikuoti, iš kokių šaltinio duomenų stulpelių gaunami duomenys kiekvienam tiksliniam stulpeliui. Pavyzdžiui, jeigu duomenų transformacijos etape naudojamas kelių šaltinio stulpelių sujungimas į vieną, tai turi būti matoma ir diagramoje. Metodai „createFillResultsDiagram“ (44 pav.), „createMappings“ bei „extractSourceColumns“ (50 pav.) programiškai realizuoja visą reikalingą logiką šiai diagramai sugeneruoti.

```
// Used in the createFillResultsDiagram() method to display and connect the column objects
public void createMappings(ArrayList<String> lines){...}

// Helper function for createMappings()
public static void extractSourceColumns(TExpression expr, List<String> sourceColumns) {...}
```

50 pav. Programiniai metodai, skirti duomenų stulpelių bei jų ryšių realizacijos logikai

Vienas pagrindinių metodų – „extractSourceColumns“ yra rekursyvus. Tai reiškia, kad jis nuosekliai keliauja per kiekvieną „SELECT“ užklausą ir bando rasti šaltinio stulpelius. Jei procedūroje yra pateikiamas paprastas stulpelio pavadinimas, jis iš karto įtraukiamas į šaltinių sąrašą. Jei tai sudėtingesnė transformacija, pavyzdžiui, funkcija ar sąlyga, metodas keliauja giliau į vidinę struktūrą.

Tuomet išskiriamas kiekvienas funkcijos argumentas, „CASE“ sąlygos „WHEN/THEN“ arba „ELSE“ reikšmes, ir ieškomi stulpeliai, naudojami šiose dalyse. Tokiu būdu aptinkami visi galimi šaltinio duomenų stulpeliai.

Kuomet visi šaltinio stulpeliai yra surinkti, jie įkraunami rinkinio duomenų objektą (angl. *Set*), kad būtų pašalinti pasikartojimai. Pasikartojimai susidaro tuomet, kuomet į du tikslinių duomenų stulpelius yra surašoma informacija, kilusi iš to paties šaltinio. Jų nepašalinus, galutinėje diagramoje šis šaltinio stulpelis kaip „ObjectNode“ pasikartotų du kartus. Atlikus tai, unikalus šaltinių sąrašas ir jų susiejimas su tiksliniu stulpeliu yra išsaugomi kaip „ColumnMapping“ (51 pav.) klasių objektai. Kiekvienas objektas turi tris svarbius elementus: tikslinių duomenų stulpelį „TargetColumn“, šaltinių stulpelių sąrašą „SourceColumns“ bei sujungimo tipą „MappingType“. Šių tipų priskyrimo logika stipriai remiasi ant bibliotekos pateikiamų privalumų – lengvai galima priskirti ar sujungimas priklauso „Source“, „Hardcoded value“ bei „Blank“ tipams.

```
public class ColumnMapping {

    private String targetColumn;
    private List<String> sourceColumns;
    private String mappingType;
    private boolean isHardcoded;
    private boolean isBlank;

    public ColumnMapping(String targetColumn, List<String> sourceColumns,
                         String mappingType, boolean isHardcoded,
                         boolean isBlank) {
        this.targetColumn = targetColumn;
        this.sourceColumns = sourceColumns;
        this.mappingType = mappingType;
        this.isHardcoded = isHardcoded;
        this.isBlank = isBlank;
    }

    public List<String> getSourceColumns() {
        return sourceColumns;
    }

    public String getMappingType() {
        return mappingType;
    }

    public String getTargetColumn() {
        return targetColumn;
    }

    public boolean isHardcoded(){
        return isHardcoded;
    }

    public boolean isBlank(){
        return isBlank;
    }
```

51 pav. Duomenų stulpelių bei jų ryšių programinio objekto struktūra

„CreateFillResults“ metodas sukuria atitinkamą XMI struktūros failo dalį ir ją prideda prie pagrindinio išeigos failo. Jame naudojamas „ColumnMapping“ objektų sąrašas yra panaudojamas aprašant duomenų srautus tarp šaltinių ir tikslinių lentelių. Programinės realizacijos kodas cikliškai atlieka instrukcijas per objektų sąrašą ir pagal kiekvieną objektą sukuria XMI struktūros dalį, kuri atspindi tikslinius bei šaltinių stulpelius kaip „ObjectNode“ tipus bei ryšius tarp jų, naudojant „ObjectFlow“ tipus. Visa logika metode padalinama į tris dalis. Pirmiausia suformuojami tiksliniai

duomenų stulpeliai ir jų ryšiai su „Output“ mazgu, kur mazgas yra tikslinių duomenų lentelės pavadinimas. Toliau, panaši logika naudojama ir šaltinio duomenims, tik šie susiejami su „Input“ mazgu. Taip pat šiame žingsnyje atsižvelgiama į tai, ar analizuojamas šaltinis atvyko iš kokios nors duomenų lentelės ar ne. Kitaip tariant, ar tai buvo tuščios ar ranka įrašytos vertės priskyrimas tiksliniam duomenų stulpeliui. Tokiais atvejais, „Input“ mazgas nėra sujungiamas su atitinkamu „ObjectNode“. Trečioje dalyje programinis metodas sukuria UML ryšius tarp šaltinių ir tikslinių duomenų pridedant atitinkamus „<edge>“ XMI tipus.

Generuoti duomenų filtravimo diagramą

„Apply filtering to the data“ diagramos generavimo logika analizuoja SQL užklausas ir nustato, kokios filtravimo sąlygos yra naudojamos. Pavyzdžiui, jei SQL užklausa turi sąlygą kurioje naudojamas daugiau arba lygu operatorius, įrankis atpažįsta, kad filtravimas atliekamas stulpeliui, kuris minimas prie „WHERE“ veiksmo, o pati operacija yra „Daugiau arba lygu“. Šiame metode taip pat yra naudojamas rekursyviai naudojamas metodas – „extractConditionsFromExpression“ (52 pav.), kuris analizuoja bibliotekos jau sugeneruotą medžio struktūrą. Metodas „extractFilterConditions“ (53 pav.) yra naudojamas tik prieiti iki WHERE veiksmo procedūroje. Naudojant bibliotekos „TExpression“ metodą, yra išskiriami filtravimo stulpeliai bei jų sąlygos. Gauti rezultatai kaupiami „FilterCondition“ (54 pav.) objektų sąraše. Jie naudojami norint išsaugoti filtruojamo stulpelio pavadinimą bei operacijos tipą. Taip pat objektas turi ir kelias papildomas operacijas. „MapOperatorToHumanReadable“ priskiria operatorių programinę išraišką į anglų kalbos tekstą, kurį galima pritaikyti kaip stereotipą „ObjectNode“ tipui, taip suteikiant papildomą informaciją naudotojui apie atliktą filtravimą. Metodas „createIdentifier“ yra naudojamas XMI tipų identifikavimo priskyrimui.

```
public void extractFilterConditions(ArrayList<String> lines) {
    String sqlQuery = String.join( delimiter: "\n", lines);
    sqlParser.sqltext = sqlQuery;

    int parseResult = sqlParser.parse();
    if (parseResult != 0) {
        System.out.println("Error parsing SQL: " + sqlParser.getErrormessage());
        return;
    }

    filters = new ArrayList<>();

    for (int i = 0; i < sqlParser.sqlstatements.size(); i++) {
        TCustomSqlStatement statement = sqlParser.sqlstatements.get(i);

        if (statement instanceof TInsertSqlStatement) {
            TInsertSqlStatement insertStatement = (TInsertSqlStatement) statement;

            // Get the subquery for the INSERT statement
            TSelectSqlStatement selectStatement = insertStatement.getSubQuery();

            if (selectStatement != null && selectStatement.getWhereClause() != null) {
                // Extract the WHERE clause
                TExpression whereExpr = selectStatement.getWhereClause().getCondition();

                // Recursively analyze the WHERE clause
                extractConditionsFromExpression(whereExpr, filters);
            }
        }
    }
```

52 pav. Programinis metodas, skirtas atpažinti filtravimo veiksmą procedūroje

```
private void extractConditionsFromExpression(TExpression expr, List<FilterCondition> filterConditions) {
    if (expr == null) {
        return;
    }

    // Check if the expression is a comparison
    if (expr.getExpressionType() == EExpressionType.simple_comparison_t) {
        String filteredColumn = expr.getLeftOperand().toString();
        String comparisonOperator = expr.getComparisonOperator().toString();
        filters.add(new FilterCondition(filteredColumn, comparisonOperator));
    }

    // Check if the expression has logical operators (e.g., AND, OR)
    if (expr.getExpressionType() == EExpressionType.logical_and_t || expr.getExpressionType() == EExpressionType.logical_or_t) {
        extractConditionsFromExpression(expr.getLeftOperand(), filterConditions);
        extractConditionsFromExpression(expr.getRightOperand(), filterConditions);
    }
}
```

53 pav. Programinis metodas, skirtas suteikti tipą filtravimo veiksmui

```
public class FilterCondition {

    private String filteredColumn;
    private String operation;


    public FilterCondition(String filteredColumn, String comparisonOperator) {
        this.filteredColumn = filteredColumn;
        this.operation = mapOperatorToHumanReadable(comparisonOperator);
    }


    private String mapOperatorToHumanReadable(String comparisonOperator) {
        switch (comparisonOperator) {
            case "=":
                return "Equals";
            case "!=":
                return "Not Equals";
            case ">":
                return "Greater Than";
            case ">=":
                return "More or Equals";
            case "<":
                return "Less Than";
            case "<=":
                return "Less or Equals";
            default:
                return "Unknown";
        }
    }

    public String getFilteredColumn() {
        return filteredColumn;
    }

    public String getOperation(){
        return operation;
    }

    public String createIdentifier(){
        String operator = operation;
        operator = operator.replaceAll( regex: " ", replacement: "_");
        return filteredColumn + "_" + operator;
    }
```

54 pav. Duomenų filtravimo programinio objekto struktūra

Metodas „CreateFilterResultsDiagram“ (55 pav.) naudojamas XMI failo dalies generavimui. Šiuo atveju, analogiškai kaip ir „CreateFillResultsDiagram“ , yra cikliškai einama per „FilterCondition“ objektų rinkinį. Kiekvienai filtravimo sąlygai sukuriamas „ObjectNode“ bei atitinkamas „ObjectFlow“ elementai. Sukurtas identifikatorius yra priskiriamas duomenų srautui, ne duomenų mazgui. Nors šie identifikatoriai naudotojui nėra matomi, be jų būtų neišvengiamas konfliktas tarp minėtų dviejų UML elementų. Duomenų srautas susiejamas su filtruojamu duomenų stulpeliu bei šaltinio „Input“ mazgu. Metodas, atlikęs visus veiksmus, prideda XMI kodą prie bendros failo struktūros.

```
public String createFilterResultsDiagram(){
    StringBuilder output = new StringBuilder();

    output.append("" +
            "\t\t\t<packagedElement xmi:type=\"uml:Activity\" xmi:id=\"filter_0\" name=\"Filter results\">\n" +
            "\t\t\t\t<node xmi:type=\"uml:ActivityParameterNode\" xmi:id=\"Filter_input0\" name=\"input\" visibility='public'/>\n"
    );

    for(int i = 0; i < filters.size(); i++){
        output.append("\t\t\t\t<node xmi:id=\"filter_").append(filters.get(i).createIdentifier())
                .append("\" name=\"").append(filters.get(i).getFilteredColumn())
                .append("\" visibility=\"public\" xmi:type=\"uml:CentralBufferNode\" isControlType=\"false\" ordering=\"FIFO\"/>\n");

        output.append("\t\t\t\t<edge xmi:id=\"node").append(nodeCounter)
                .append("\" visibility=\"public\" source=\"Filter_input0\" target=\"filter_")
                .append(filters.get(i).createIdentifier()).append("\" xmi:type=\"uml:ObjectFlow\"/>\n");

        nodeCounter++;
    }

    output.append("\t\t\t</packagedElement>\n");

    return output.toString();
}
```

55 pav. Filtravimo veiksmo XMI failo dalies generavimo metodas

Generuoti duomenų sujungimo diagramą

Metodas „createLoadSourcesDiagram“ generuoja „Apply joins to source tables“ veiklos diagramą, kuri vizualizuoja ETL proceso metu atliekamus duomenų lentelių sujungimus. Jis remiasi „JoinMapping“ objekto informacija, kurioje yra saugomi lentelių ryšiai, įskaitant pagrindines (angl. *primary*) lenteles, sujungiamas (angl. *foreign*) lenteles, jų stulpelius bei sujungimo sąlygas. Pirmasis metodo žingsnis sukuria UML veiklos elementą, kuris nurodo visas duomenų lenteles bei jų sujungimus. Toliau, jis tikrina „JoinMapping“ objektus, kuriame saugomi lentelių sujungimai. Kiekvienai pagrindinei ir prijungiamajai lentelei metodas diagramoje generuoja įvesties taškus, taip nurodant lentelės pavadinimą. Toliau, sugeneruojami elementai, kurie nurodo lentelių stulpelių informaciją, taip atvaizduojant, kurie stulpeliai yra jungimo raktai. Galiausiai, įvesties taškai bei stulpelių elementai yra sujungiami tarpusavyje, taip nurodant sąsajas tarp jų.

„JoinMapping“ klasė apibrėžia sujungimo informaciją, įskaitant pirminės ir papildomos lentelės pavadinimus, sujungimo tipą (pvz., „INNER JOIN“, „LEFT JOIN“) bei sujungimo sąlygą. Ši informacija išgaunama analizuojant SQL užklausas, naudojant metodą „extractJoinsFromSelect“ (56 pav.), kuris aprašytas pačioje klasėje. Jis tikrina „SELECT“ užklausų sujungimus, lentelių ryšius ir iš jų sudaro „JoinMapping“ objektų sąrašą. Taip pat, sujungimai yra toliau detalizuojami, išskiriant pirminės lentelės prijungimo bei prijungiamosios lentelės stulpelių raktų informaciją.

```java
public static List<JoinMapping> extractJoinsFromSelect(TSelectSqlStatement selectStatement) {
    List<JoinMapping> joinMappings = new ArrayList<>();

    if (selectStatement.getJoins() != null) {
        for (int i = 0; i < selectStatement.getJoins().size(); i++) {
            TJoin join = selectStatement.getJoins().getJoin(i);

            if (join.getJoinItems() != null) {
                for (int j = 0; j < join.getJoinItems().size(); j++) {
                    TJoinItem joinItem = join.getJoinItems().getJoinItem(j);
                    String joinType = getJoinType(joinItem);
                    String primaryTable = selectStatement.tables.getTable( position: 0).getFullName();
                    String joinedTable = joinItem.getTable().getFullName();

                    String joinCondition = joinItem.getOnCondition() != null ? joinItem.getOnCondition().toString() : "Unknown Condition";

                    String primaryColumn = "";
                    String foreignColumn = "";
                    if (joinItem.getOnCondition() != null) {
                        TExpression onCondition = joinItem.getOnCondition();
                        String[] columns = onCondition.toString().split( regex: "=");
                        if (columns.length == 2) {
                            primaryColumn = columns[0].trim();
                            foreignColumn = columns[1].trim();
                        }
                    }

                    joinMappings.add(new JoinMapping(primaryTable, joinedTable, joinType, joinCondition, primaryColumn, foreignColumn));
                }
            }
        }
    }
    return joinMappings;
}
```

56 pav. Programinis kodas, skirtas analizuoti lentelių sujungimus

Metodas „extractJoinMappings” (57 pav.), kuris aprašytas pagrindinėje įrankio klasėje tikrina SQL užklausas, naudodamasis kodo analizės bibliotekos funkcionalumu ir identifikuoja visus lentelių sujungimus. Metodas egzistuoja tik dėl to, kad būtų galima aptikti reikalingą užklausą, kurioje yra naudojami lentelių sujungimai. Ši užklausa yra perduodama minėtam „extractJoinsFromSelect“ metodui. Taip pat, „extractJoinMappings“ užtikrina, kad lentelių sujungimai yra tiksliai identifikuoti pasinaudojant „getJoinType“ (58 pav.) metodu. Jeigu jungimo operacija yra neatpažįstama arba įrankio nepalaikoma, būtent tokia informacija ir yra pateikiama.

```java
public void extractJoinMappings(ArrayList<String> lines) {
    String sqlQuery = String.join( delimiter: "\n", lines);
    sqlParser.sqltext = sqlQuery;

    int parseResult = sqlParser.parse();
    if (parseResult != 0) {
        System.out.println("Error parsing SQL: " + sqlParser.getErrormessage());
        return;
    }

    joins = new ArrayList<>();

    for (int i = 0; i < sqlParser.sqlstatements.size(); i++) {
        TCustomSqlStatement statement = sqlParser.sqlstatements.get(i);

        if (statement instanceof TInsertSqlStatement) {
            TInsertSqlStatement insertStatement = (TInsertSqlStatement) statement;

            // Extract subquery (SELECT statement inside INSERT)
            TSelectSqlStatement selectStatement = insertStatement.getSubQuery();
            if (selectStatement != null) {
                joins.addAll(JoinMapping.extractJoinsFromSelect(selectStatement));
            }
        } else if (statement instanceof TSelectSqlStatement) {
            // Standalone SELECT statements (if any)
            joins.addAll(JoinMapping.extractJoinsFromSelect((TSelectSqlStatement) statement));
        }
    }
}
```

57 pav. Programinis kodas, skirtas supildyti lentelių sujungimo objekto informaciją

```
private static String getJoinType(TJoinItem joinItem) {
    EJoinType type = joinItem.getJoinType();

    if (type == EJoinType.inner) return "INNER JOIN";
    if (type == EJoinType.left) return "LEFT JOIN";
    if (type == EJoinType.right) return "RIGHT JOIN";
    if (type == EJoinType.full) return "FULL JOIN";
    if (type == EJoinType.cross) return "CROSS JOIN";

    return "UNKNOWN JOIN (" + type + ")";

}
```

58 pav. Pagalbinis programinis kodas, skirtas identifikuoti lentelių sujungimo tipą

Generuoti tarpinių lentelių diagramą

Metodas „createTempTableDiagram“ generuoja „Create temporary tables“ veiklos diagramos fragmentą, kuris vizualizuoja laikinuosius duomenų lentelių sukūrimo veiksmus ETL proceso metu. Šis metodas remiasi „TemporaryMapping“ (59 pav.) objektais, kuriuose saugoma laikinosios lentelės pavadinimo ir jos stulpelių informacija. Laikinosios lentelės yra tarpinės lentelės, kurios naudojamos apdorojimo metu ir nėra galutinės duomenų saugojimo lentelės.

Pirmasis metodo žingsnis sukuria veiklos diagramą, kuris reprezentuoja laikinuosius duomenis bei susijusius veiksmus. Šis kūrimo žingsnis turi patikrą, kur jeigu laikinosios lentelės nėra naudojamos, diagrama taip pat nėra sukuriama. Toliau, kiekvienai laikinajai lentelei bei jos stulpeliams sukuriami įvesties taškai kurie nurodo lentelės pavadinimą įvesties elemente bei jai priklausančių stulpelių elementai. Toliau sukuriami elementai, kurie reprezentuoja kiekvieną laikinosios lentelės stulpelį ir jo duomenų srautą. Kadangi šioje diagramoje yra pateikiamas ryšys tarp galutinės duomenų lentelės stulpelių bei laikinosios stulpelių analogų, taip pat yra vykdoma patikra tarp šių stulpelių. Jeigu metode yra randamas ryšys tarp šių dviejų tipų lentelių, vienodas stulpelio elementas yra sujungiamas tiek su tikslinės lentelės įvesties taškų, tiek su laikinosios išvesties tašku. Taip pat, tikslinės lentelės įvesties taškas yra generuojamas tik tada, kai yra aptiktas bent vienas bendrinis stulpelis.

„TemporaryMapping“ klasė yra pagrindinis duomenų objektas, skirtas laikinosioms lentelėms aprašyti. Ji saugo lentelės pavadinimą bei stulpelių sąrašą. Šie duomenys užpildomi naudojant pagrindinio įrankio metodo „createMappings“ logiką, kuris kaupia visų naudojamų lentelių informaciją. Metodas analizuoja SQL užklausas ir identifikuoja tiek įprastas, tiek laikinąsias lenteles bei jų stulpelius, kurie yra reikalingi skaičiavimams ar filtravimams ETL proceso metu.

```
public class TemporaryMapping {

    private String tableName;
    private List<String> columnNames;

    public TemporaryMapping(String tableName, List<String> columnNames) {
        this.tableName = tableName;
        this.columnNames = columnNames;
    }

    public String getTableName() { return tableName; }

    public List<String> getColumnNames() { return columnNames; }

    @Override
    public String toString() { return "Temporary Table: " + tableName + ", Columns: " + columnNames; }
}
```

59 pav. Laikinosios duomenų lentelės programinio objekto struktūra

Generuoti tarpinių kintamųjų diagramą

Metodas „createTempVariablesDiagram“ (60 pav.) generuoja „Fetch temporary variable information“ veiklos diagramą, kurioje atvaizduojami laikini kintamieji. Metodo veikimas remiasi „VariableAssignment“ objektais, kurie aprašo, kaip kintamieji priskiriami ir kokiomis sąlygomis jie naudojami. Objektai naudojami metode, kuris diagramoje sukuria reikalingus elementus. Pirmasis elementas nurodo, kokia buvo taikoma filtravimo sąlyga norint išsaugoti vertę iš kintamųjų lentelės bei stulpelio pavadinimą, kuriame ši vertė buvo laikoma. Antrasis elementas nurodo kintamojo pavadinimą, kuris naudojamas procedūroje bei iš kokio stulpelio šį vertė buvo išsaugota. Ryšiai tarp kintamųjų ir kintamųjų lentelių yra atvaizduojami duomenų srautais, nurodančiais duomenų priskyrimo eigą. Papildomai į diagramą įtraukta filtravimo sąlyga, kad jeigu kintamieji nėra naudojami, nėra sugeneruojama ir diagrama.

```
public String createTempVariablesDiagram(List<VariableAssignment> variables) {

    if(variables.isEmpty())
        return "";

    StringBuilder output = new StringBuilder();

    output.append("\t\t\t<packagedElement xmi:type=\"uml:Activity\" xmi:id=\"temporary_variables_0\" name=\"Fetch temporary variable information\">\n");


    for (int i = 0; i < variables.size(); i++) {
        VariableAssignment variable = variables.get(i);

        output.append("\t\t\t\t<node xmi:type=\"uml:ActivityParameterNode\" xmi:id=\"temp_variable_input0\" name=\"")
              .append(variables.get(i).getSourceTable()).append("\" visibility=\"public\"/>\n");

        // Create the load node
        output.append("\t\t\t\t<node xmi:id=\"temp_variable_load_").append(i).append("\" ")
                .append("name=\"").append(variable.getFilterCondition()).append("\" ")
                .append("visibility=\"public\" xmi:type=\"uml:CentralBufferNode\" isControlType=\"false\" ordering=\"FIFO\"/>\n");

        // Create a conn with the input node to the load node
        output.append("\t\t\t\t<edge xmi:id=\"temp_variable_edge_").append(i).append("_1\" ")
                .append("visibility=\"public\" source=\"temp_variable_input0\" ")
                .append("target=\"temp_variable_load_").append(i).append("\" ")
                .append("xmi:type=\"uml:ObjectFlow\"/>\n");

        // Create the node for the variable name
        output.append("\t\t\t\t<node xmi:id=\"temp_variable_value_").append(i).append("\" ")
                .append("name=\"").append(variable.getVariableName()).append("\" ")
                .append("visibility=\"public\" xmi:type=\"uml:CentralBufferNode\" isControlType=\"false\" ordering=\"FIFO\"/>\n");

        // Create an edge connecting the load node to the variable node
        output.append("\t\t\t\t<edge xmi:id=\"temp_variable_edge_").append(i).append("_2\" ")
                .append("visibility=\"public\" source=\"temp_variable_load_").append(i).append("\" ")
                .append("target=\"temp_variable_value_").append(i).append("\" ")
                .append("xmi:type=\"uml:ObjectFlow\"/>\n");
    }

    output.append("\t\t\t</packagedElement>\n");

    return output.toString();
}
```

60 pav. Laikinųjų kintamųjų veiklos diagramos generavimo metodas

Metodas pradedamas nuo „VariableAssignment“ objektų analizės, kurie jau būna užpildyti informacija anksčiau vykdant „extractVariableAssignments“ (61 pav.) funkciją. Šis metodas naudoja SQL užklausas, kad identifikuotų laikinojo kintamojo priskyrimus. Tikrinami kiekvieno kintamojo priskyrimo ryšiai su šaltinio lentelėmis, taip pat priskyrimo ir filtravimo sąlygos.

„ExtractVariableAssignments“ naudoja „extractFromSelect“ kaip pagalbinį metodą, kuris analizuoja SELECT užklausas ir iš jų išskiria kintamųjų priskyrimus. „extractFromSelect“ išskiria priskyrimo kintamąjį, pradinę lentelę, laukų sąlygas ir galimus filtrus. Ši informacija yra apdorojama ir išsaugoma „VariableAssignment“ objektuose.

```java
public List<VariableAssignment> extractVariableAssignments(ArrayList<String> lines) {
    String sqlQuery = String.join( delimiter: "\n", lines);
    sqlParser.sqltext = sqlQuery;

    int parseResult = sqlParser.parse();
    if (parseResult != 0) {
        System.out.println("Error parsing SQL: " + sqlParser.getErrormessage());
        return Collections.emptyList();
    }

    List<VariableAssignment> extractedAssignments = new ArrayList<>();

    for (int i = 0; i < sqlParser.sqlstatements.size(); i++) {
        TCustomSqlStatement statement = sqlParser.sqlstatements.get(i);

        // Handle select variable
        if (statement instanceof TSelectSqlStatement) {
            extractFromSelect((TSelectSqlStatement) statement, extractedAssignments,  variableName: null);
        }

        // Fallback manually using regex
        String sqlText = statement.toString().toUpperCase();
        if (sqlText.matches( regex: "SET\\s+@\\w+\\s*=\\s*\\(SELECT.*")) {
            String variableName = sqlText.split( regex: "=")[0].replace( target: "SET",  replacement: "").trim();
            if (statement instanceof TSelectSqlStatement) {
                extractFromSelect((TSelectSqlStatement) statement, extractedAssignments, variableName);
            }
        }
    }

    variableAssignments.clear();
    variableAssignments.addAll(extractedAssignments);

    return extractedAssignments;
}
```

61 pav. Programinis kodas, skirtas analizuoti laikinojo kintamojo pasirinkimo dalį procedūroje

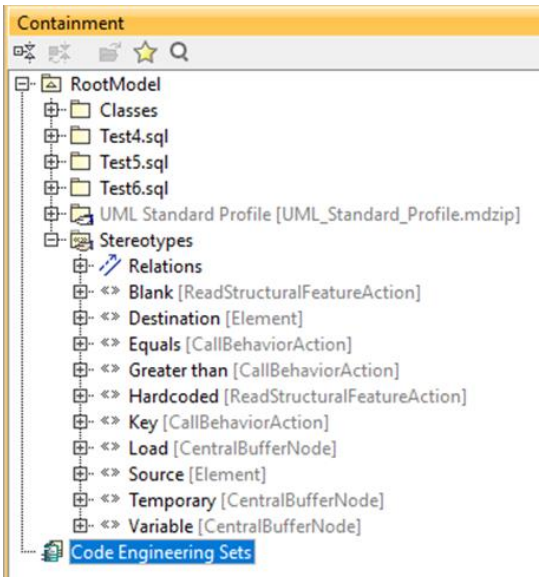
„VariableAssignment“ (62 pav.) klasė yra pagrindinė duomenų struktūra, aprašanti laikinojo kintamojo informaciją. Ji saugo kintamojo pavadinimą, šaltinio lentelę (šiuo atveju lentelę, kurioje saugojami kintamieji), reikalingo kintamojo sąlygą bei filtravimo sąlygą. Ši struktūra leidžia apibrėžti kintamojo naudojimo kontekstą.

```java
public class VariableAssignment {
    private String variableName;
    private String sourceTable;
    private String fieldNameCondition;
    private String filterCondition;

    public VariableAssignment(String variableName, String sourceTable, String fieldNameCondition, String filterCondition) {
        this.variableName = variableName;
        this.sourceTable = sourceTable;
        this.fieldNameCondition = fieldNameCondition;
        this.filterCondition = filterCondition;
    }

    public String getFilterCondition() {
        return filterCondition;
    }

    public String getSourceTable() {
        return sourceTable;
    }

    public String getVariableName() {
        return variableName;
    }

    @Override
    public String toString() {
        return "VariableAssignment{" +
                "variableName='" + variableName + '\'' +
                ", sourceTable='" + sourceTable + '\'' +
                ", fieldNameCondition='" + fieldNameCondition + '\'' +
                ", filterCondition='" + filterCondition + '\'' +
                '}';
    }
}
```

62 pav. Laikinojo kintamojo programinio objekto struktūra

Stereotipai

Kartu su sugeneruotomis diagramomis, įrankis taip pat sukuria nestandartinių (angl. *custom*) stereotipų profilį (63 pav.), kurio paskirtis – papildyti UML veiklos modelius semantine informacija apie ETL logiką. Šie stereotipai atvaizduoja įvairias operacijas, ryšius ir transformacijas, kurios yra būdingos ETL procedūroms, tačiau dažnai lieka neaiškios vizualiniuose modeliuose be papildomo konteksto.



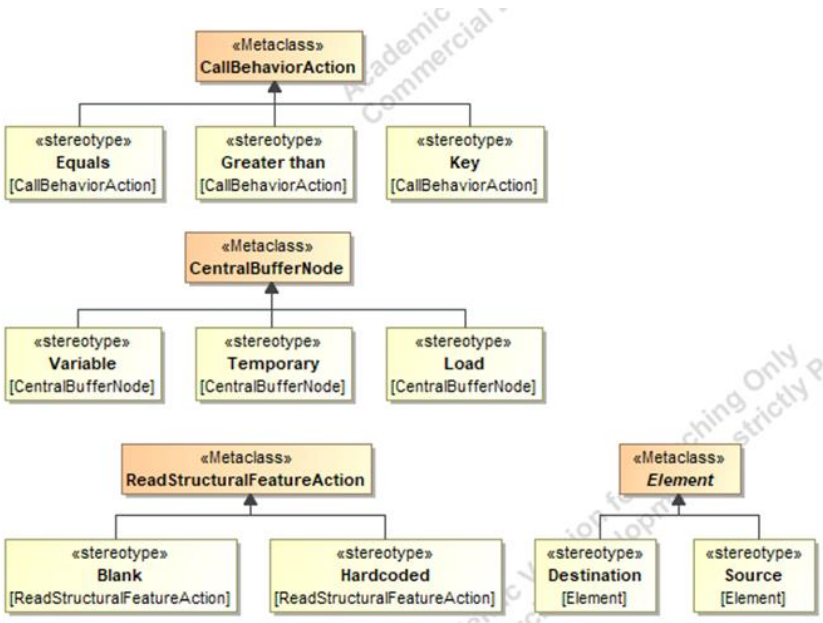
63 pav. Įrankio sugeneruotų nestandartinių stereotipų peržiūra

Stereotipai turi tiesioginį ryšį su UML metamodelio klasėmis. Pavyzdžiui, „Destination“ stereotipas taikomas tik „AddStructuralFeatureValueAction“ tipo veiklos elementams (64 pav.). Šis ryšys užtikrina, kad papildoma semantinė informacija būtų integruota į UML veiklos diagramas.

Įrankyje realizuoti stereotipai klasifikuoti pagal dažniausiai pasitaikančius ETL scenarijus:

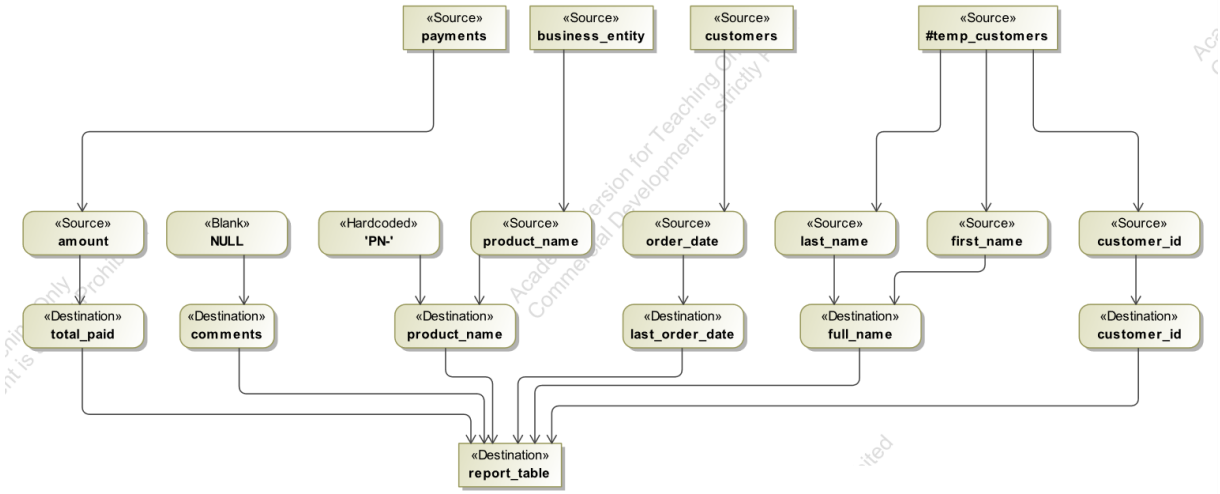
- „CallBehaviorAction“ elementams priskiriami loginių sąlygų stereotipai („Equals“, „Greater than“, „Key“);
- „ReadStructuralFeatureAction“ – duomenų nuskaitymo kilmės šaltinio ar tipo žymėjimui („Source“, „Hardcoded“, „Blank“);
- „CentralBufferNode“ – tarpiniams kintamiesiems ar laikinoms lentelėms („Temporary“, „Load“, „Variable“);
- „Element“ – objektų, tokių kaip šaltinių ir tikslinių lentelių, papildomam apibrėžimui („Source“, „Destination“).

Svarbu paminėti, kad šis stereotipų profilis yra statinis, t. y. nėra generuojamas dinamiškai atsižvelgiant į kiekvienos procedūros specifinę logiką. Įrankyje apibrėžti tik dažniausiai pasitaikantys stereotipų taikymo atvejai. Tai leidžia sumažinti perteklinių elementų rizika, tačiau taip pat reiškia, kad sudėtingesniuose scenarijuose, pavyzdžiui, vykdant filtravimo operacijas su keliomis loginėmis sąlygomis, tam tikri veiklos elementai gali likti be papildomos informacijos.



64 pav. Įrankyje naudojamų stereotipų sąryšis su UML metaklasėmis diagrama

65 pav. pateikiama UML veiklos diagrama su integruotais nestandartiniais stereotipais, kuriuose vizualiai pateikiama duomenų įrašymo į tikslinę lentelę logika. Šioje diagramoje stereotipas „Destination“ pažymi galutinio įrašymo tašką tiek tikslinės lentelės, tiek jos stulpelių kontekste. „Source“, „Blank“ ir „Hardcoded“ nurodo stulpelių reikšmių kilmę bei veiksmų seką, susijusią su duomenų paruošimu. Tokiu būdu, stereotipai leidžia naudotojui lengviau suprasti duomenų srautų paskirtį.



65 pav. *Populate the target table* diagrama su nestandartiniais stereotipais

4.2. Sukurto įrankio testavimo modelis, duomenys, rezultatai

Įrankio testavimas buvo atliktas rankiniu būdu, taip siekiant patikrinti jo tikslumą ir funkcionalumą. Testavimo metu buvo analizuojami įrankio sugeneruoti UML diagramos elementai ir lyginami su projektavimo etape apibrėžta ETL procedūrų logika. Pagrindinis dėmesys buvo skiriamas tam, ar įrankis tiksliai atspindi užklausų struktūrą, duomenų srautus ir operacijų seką UML diagramose.

Testavimas apėmė įvairius ETL scenarijus, įskaitant lentelių sujungimus, laikinas lenteles, filtravimo sąlygas ir laikinus kintamuosius.

Naudotos procedūros buvo išskirtos į kelis sudėtingumo lygius – be laikinųjų kintamųjų ir lentelių, be laikinųjų kintamųjų arba kintamųjų lentelių ir galiausiai, su šiais dviem aspektais. Taip pat buvo tikrinamos ir procedūros, kuriose duomenų filtravimo nebuvo aprašyta ir atvirkščiai. Visa testavimo procedūra buvo atlikta rankiniu būdu įkeliant ETL procedūras į įrankį ir lyginant rezultatus su tikėtinomis diagramomis pagal įrankio projektavimą.

Testavimo rezultatai parodė, kad nors pats diagramų kūrimo principas veikia tik su minimaliomis klaidomis, taip pat trūksta dalies aspektu kurie buvo nurodyti projektavimo dalyje. Pvz. pateikiami stereotipai nėra dinamiški veiklos elementuose. Tai reiškia, kad įrankyje yra realizuoti tik procedūrose dažniausiai pasitaikantys atvejai, kuriose nurodomi stereotipai kaip papildoma informacija. Tai geriausiai pastebima filtravimo veiklos diagramoje, kai procedūroje naudojama sudėtinga filtravimo logika. Tokiais atvejais, prie filtruojamo stulpelio elemento nėra suteikiamas pagalbinis stereotipas. Šiai problemai išspręsti tektų papildyti stereotipų profilį visiems įmanomiems atvejams. Tačiau, išbandžius minėtus procedūrų tipus (su filtravimu, be jo ir t.t.) pastebima, kad įrankis teisingai sugeneruoja visus kitus dinaminius elementus. Sugeneruotas XMI formato failas taip pat yra sąlyginai lengvai skaitomas – naudojami atitraukimai tarp atributų, o taip perteikiami hierarchijos ryšiai. Galiausiai, įrankio greitaveika yra labai sparti – sugeneruotos diagramos failas yra pateikiamas akimirksniu.

Įrankio testavimas, atliktas rankiniu būdu, parodė, kad jis tiksliai generuoja UML diagramas, atspindinčias ETL procedūrų logiką. Tačiau trūksta kai kurių projektavimo dalyje numatytų funkcijų, tokių kaip stereotipai. Nepaisant šių trūkumų, įrankis demonstruoja aukštą tikslumą dinaminių elementų generavime, greitą veikimą ir aiškų sugeneruotų XMI failų formatą.

5. Eksperimentinis ETL procedūrų vizualizavimo naudojant UML veiklos ir klasių diagramas sprendimo tyrimas

5.1. Eksperimento planas

Šiame etape siekiama įvertinti sukurto ETL procedūrų dokumentavimo įrankio pritaikymo galimybes praktinėje aplinkoje. Eksperimentinis vertinimas susideda iš dviejų dalių.

Pirmojoje dalyje atliekamas įrankio testavimas su realiomis ETL procedūromis, imituojančiomis tipinius proceso veiklą. Procedūros buvo atrinktos taip, kad apimtų įvairaus sudėtingumo loginius elementus ir struktūrinius skirtumus, leidžiančius įvertinti įrankio lankstumą bei jo generuojamų diagramų atitikimą tikrosioms duomenų transformacijoms. Šios dalies metu vertinamas įkelto programinio kodo interpretacijos tikslumas, sugeneruotų UML veiklos ir klasių diagramų struktūrinis atitikimas bei atvejų, kuriuose analizė buvo neįmanoma ar klaidinga, dažnumas. Vertinimo pagrindui suformuota loginė ETL ir UML elementų atitikmenų struktūra, kurioje aprašyti tipiniai SQL programinio kodo konstrukcijų ir UML veiklos modelio elementų ryšiai. Ši lentelė pasitelkiama kaip metodinis pagrindas vertinant, ar generuojami UML elementai tinkamai perteikia SQL struktūrą.

Interviu metu dalyviams buvo pateikiami atviri klausimai apie jų darbo su SQL užklausomis patirtį, iššūkius analizuojant sudėtingus duomenų procesus bei vertinimus apie demonstruotą sprendimą. Pokalbio metu taip pat buvo pademonstruotas sukurtas įrankis: parodytas SQL užklausų įkėlimo, XMI failų generavimo ir UML diagramų vizualizavimo procesas. Dalyviai galėjo teikti pastabas ir įžvalgas apie įrankio funkcionalumą ir pritaikomumą.

Interviu buvo atlikti su trimis IT specialistais, turinčiais skirtingą patirtį dirbant su ETL procedūromis, SQL užklausomis ir duomenų apdorojimo procesais. Pirmasis dalyvis – IT analitikas, dirbantis vienoje iš didžiųjų Lietuvos įmonių, atsakingas už IT infrastruktūros priežiūrą ir sisteminius sprendimus. Nors tiesiogiai nedirba su ETL procesais, jo kasdienė veikla apima SQL užklausų analizę, todėl jis galėjo įvertinti įrankio aktualumą iš naudotojo perspektyvos. Antrasis dalyvis – duomenų inžinierius, aktyviai dirbantis su duomenų migracijos projektais. Jo profesinis kontekstas ir užduotys yra panašios į tas, kurias apima šiame darbe nagrinėjama tematika, todėl jo įžvalgos atspindi praktinę patirtį tokio tipo projektuose. Trečiasis dalyvis – duomenų mokslininkų komandos vadovas, kuris nedalyvavo visame interviu, tačiau turi platų supratimą apie duomenų inžinerijos iššūkius ir sprendimus. Jis taip pat yra susipažinęs su duomenų migracijos projektais, kadangi prižiūri komandą, kurioje dirba ir antrasis interviu dalyvis. Šių ekspertų įvairovė užtikrino vertingą grįžtamąjį ryšį apie sukurtos priemonės praktinį pritaikomumą.

Surinkti interviu duomenys buvo apdorojami taikant kokybinę turinio analizę. Iš pradžių atsakymai buvo surinkti ir perrašomi formalia kalba. Vėliau atlikta teminė analizė, kurios metu buvo išskirtos pagrindinės temos, tokios kaip įrankio naudingumas, pritaikymo sritys, numatomi iššūkiai bei pasiūlymai tobulinimui. Svarbiausios įžvalgos buvo pagrindžiamos tiesioginėmis citatomis iš respondentų atsakymų. Tokiu būdu siekiama užtikrinti visapusišką tiek kiekybinį, tiek kokybinį sukurto įrankio vertinimą.

Interviu metu pateikti klausimai buvo šie:

- Ar galėtumėte paaiškinti, kokias užduotis atliekate, kuriose reikia rašyti arba peržiūrėti SQL kodą?
- Ar dažniau dirbate su kitų parašytu SQL kodu, ar su savo?

- Ar esate kada nors turėjęs suprasti arba paaiškinti, ką daro didelė SQL operacija? Kaip paprastai tą darote?
- Kas, Jūsų nuomone, yra sudėtingiausia skaitant arba palaikant didelius SQL užklausų rinkinius?
- Ar esate naudoję diagramas siekdami geriau suprasti SQL procesus?
- Ar matytumėte vertę procese, kuris automatiškai paverstų SQL užklausas į diagramas?
- Kokia buvo Jūsų pirmoji reakcija pamačius šį įrankį? Ar, Jūsų manymu, jis būtų naudingas?
- Ar savo darbo aplinkoje matytumėte atvejų, kuriuose tokia vizualizacija būtų naudinga?
- Ar įžvelgėte kokių nors galimų problemų, susijusių su tokiu įrankiu?
- Kokie aspektai, Jūsų nuomone, būtų svarbūs, kad įrankis tinkamai atliktų savo funkcijas?
- Ar naudotumėte tokį įrankį, jei jis būtų prieinamas?
- Ar turite papildomų pastabų?

5.2. Eksperimento rezultatai

5.2.1. SQL programinio kodo testavimo rezultatai

Šioje dalyje atliekama kiekybinė penkių realių ETL procedūrų analizė, siekiant įvertinti, kaip tiksliai sukurtas įrankis transformuoja procedūras į UML veiklos diagramas pagal anksčiau aprašytą metodiką. Vertinimo tikslas – patikrinti, ar kiekvienas SQL procedūros elementas turi savo atitikmenį UML metamodelyje ir ar sugeneruotos diagramos atspindi realią duomenų srauto logiką.

Analizei pasirinktos penkios ETL procedūros:

- pirmoji procedūra buvo pasirinkta kaip bazinis testavimo atvejis, pasižymintis sąlyginai paprasta struktūra. Ji apima tipinę duomenų transformavimo ir įrašymo logiką, leidžiančią įvertinti, kaip įrankis generuoja diagramas esant nedidelei procedūros apimčiai;
- antroji procedūra pasižymi kiek didesniu laukų kiekiu ir sąlyginėmis reikšmėmis. Jos tikslas – patikrinti, kaip įrankis interpretuoja skirtingų tipų duomenų transformacijas bei sprendžia paprastas loginio pasirinkimo situacijas;
- trečioji procedūra buvo įtraukta dėl jos struktūrinio sudėtingumo – joje naudojami keli informacijos šaltiniai bei loginių sąlygų deriniai. Tai leidžia įvertinti priemonės gebėjimą modeliuoti išplėstinius kontrolės ir duomenų srautus;
- ketvirtoji procedūra pasirinkta kaip paprasto formato pavyzdys su mažesniu veiksmų skaičiumi. Ji tarnauja kaip kontrolinis atvejis, skirtas įvertinti, ar įrankis korektiškai apdoroja minimalios struktūros procedūras;
- penktoji procedūra įtraukta dėl tam tikrų netipinių konstrukcijų, kurios dažnai pasitaiko realiuose projektuose. Ji naudinga vertinant, kaip įrankis interpretuoja dinamiškesnes SQL išraiškas.

Visos jos pasižymi skirtingu sudėtingumo lygiu, duomenų šaltinių skaičiumi bei transformacijų logika, todėl sudaro pakankamą reprezentatyvią imtį testavimui. Kiekviena procedūra buvo įkelta į sukurtą įrankį, kuris automatiškai sugeneravo atitinkamas UML veiklos diagramas. Tuomet buvo lyginamas teorinis tikėtinų UML elementų skaičius (pagal analizuotą SQL struktūrą) su realiai sugeneruotais elementais.

6 lentelėje pateikiama loginė ETL ir UML elementų atitikmenų struktūra, kuria remiantis buvo atliktas kiekybinis įvertinimas. Išskirtos pagrindinės SQL elementų kategorijos – veiksmų tipai, duomenų šaltiniai, transformacijos, ir jų UML atitikmenys veiklos ar klasių diagramose.

6 lentelė. Loginė ETL ir UML elementų atitikmenų struktūra

ETL elementai	**UML elementai**
Bendras eilučių skaičius procedūroje	Elementų skaičiaus kiekvienoje UML diagramoje suma
Komentarų skaičius	Nėra
Tuščių eilučių skaičius	Nėra
„SELECT“ operacijų skaičius	Veiklos diagramų veiksmų skaičius Sugeneruotų klasių skaičius
„INSERT“ operacijų skaičius	Veiklos diagramų veiksmų skaičius Sugeneruotų klasių skaičius
„JOIN“ operacijų skaičius	*ForkNode* ir *JoinNode* skaičius Sugeneruotų klasių skaičius
„WHERE“ operacijų skaičius	Veiklos diagramų veiksmų skaičius Atributų skaičius klasėse
„CASE“ operacijų skaičius	Bendras valdymo srautų skaičius
Naudojamų lentelių skaičius	Sugeneruotų klasių skaičius
Pasirinktų stulpelių skaičius	Atributų skaičius klasėse
Naudojamų kintamųjų skaičius	Įvesties ir išvesties mazgų skaičius *ForkNode* ir *JoinNode* skaičius
Sukurtų laikinų lentelių skaičius	Sugeneruotų klasių skaičius *ForkNode* ir *JoinNode* skaičius

7 lentelėje pateikiami kiekybiniai duomenys apie kiekvieną išanalizuotų ETL procedūrų. Nurodyti eilučių, komentarų, stulpelių, lentelių ir transformacijų skaičiai – visa tai naudota kaip pagrindas numatomiems UML elementų kiekiams įvertinti.

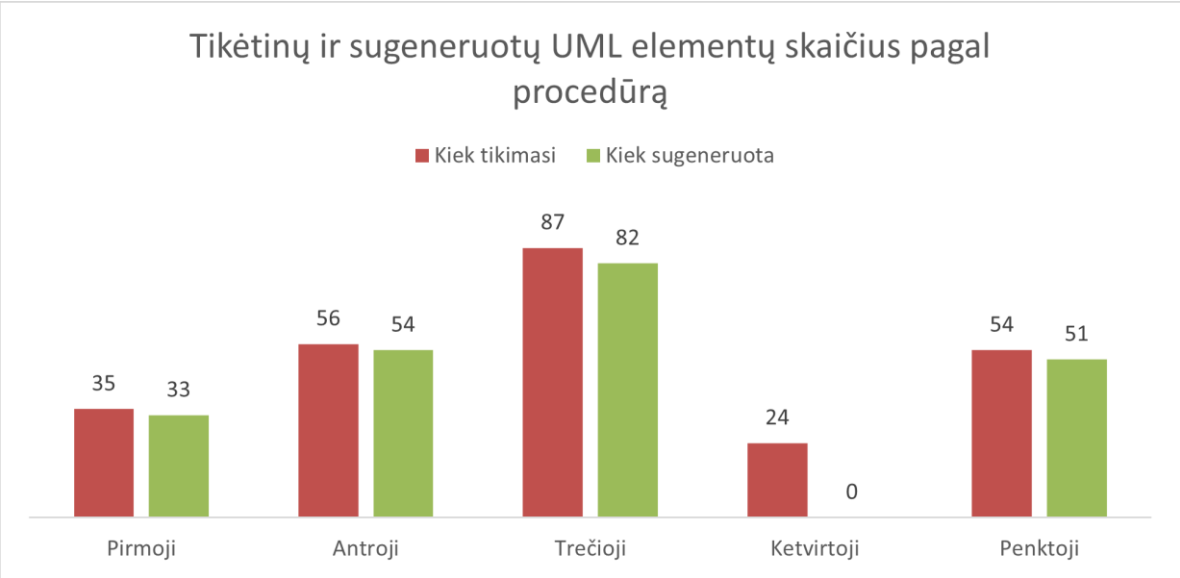
7 lentelė. Kiekybiniai duomenys apie ETL procedūras

ETL elementai	**Pirmoji procedūra**	**Antroji procedūra**	**Trečioji procedūra**	**Ketvirtoji procedūra**	**Penktoji procedūra**
Bendras eilučių skaičius procedūroje	59	102	152	97	48
Komentarų skaičius	5	2	2	5	6
Tuščių eilučių skaičius	6	6	12	11	7
„SELECT“ operacijų skaičius	1	1	1	2	1
„INSERT“ operacijų skaičius	1	1	2	1	1
„JOIN“ operacijų skaičius	1	1	5	1	3
„WHERE“ operacijų skaičius	1	0	0	1	1
„CASE“ operacijų skaičius	2	5	4	0	0
Naudojamų lentelių skaičius	1	3	5	4	0
Pasirinktų stulpelių skaičius	10	28	39	7	34
Naudojamų kintamųjų skaičius	0	0	0	0	0
Bendras transformacijų skaičius	4	10	13	3	8
Sukurtų laikinų lentelių skaičius	0	0	1	0	0

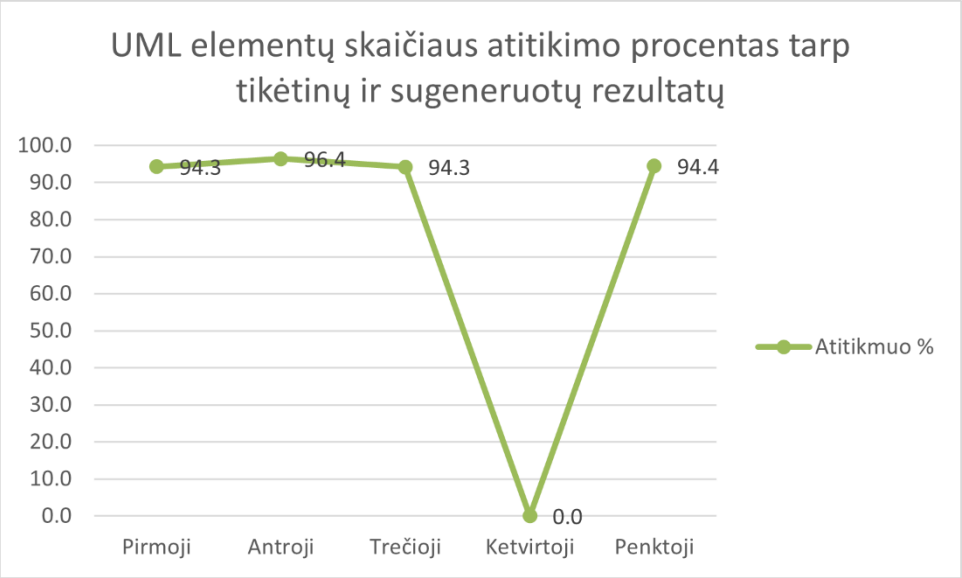
8 lentelė lygina sugeneruotų UML elementų skaičių su teoriškai tikėtinu skaičiumi. Vertinamos tiek veiksmų, srautų, tiek lentelių ir įvesties/išvesties komponentų atitiktys. Lentelėje atskirai išskiriami atvejai, kai tam tikri elementai buvo praleisti arba nepilnai identifikuoti.

8 lentelė. UML diagramų generavimo analizės rezultatai

UML elementai	**Sugeneruotų UML elementų skaičius**					**Kiek UML elementų tikimasi**				
	1-oji proc.	**2-oji proc.**	**3-oji proc.**	**4-oji proc.**	**5-oji proc.**	**1-oji proc.**	**2-oji proc.**	**3-oji proc.**	**4-oji proc.**	**5-oji proc.**
Elementų skaičiaus kiekvienoje UML diagramoje suma	33	54	82	-	51	35	56	87	24	54
Veiklos diagramų veiksmų skaičius Sugeneruotų klasių skaičius	24 3	47 3	61 7	-	36 5	24 3	47 3	61 7	16 2	36 5
Veiklos diagramų veiksmų skaičius Sugeneruotų klasių skaičius	24 3	47 3	61 7	-	36 5	24 3	47 3	61 7	16 2	36 5
ForkNode ir *JoinNode* skaičius Sugeneruotų klasių skaičius	0 3	0 3	2 7	-	0 5	0 3	0 3	2 7	0 2	0 5
Veiklos diagramų veiksmų skaičius Atributų skaičius klasėse	24 15	47 36	61 48	-	36 26	24 15	47 36	61 48	16 8	36 24
Bendras valdymo srautų skaičius	21	34	52	-	31	21	34	52	11	31
Sugeneruotų klasių skaičius	3	3	7	-	5	3	3	7	2	5
Atributų skaičius klasėse	15	36	48	-	26	15	36	36	8	24
Įvesties ir išvesties mazgų skaičius *ForkNode* ir *JoinNode* skaičius	6 0	3 0	10 2	-	8 0	4 0	5 0	15 2	4 0	12 0
Sugeneruotų klasių skaičius *ForkNode* ir *JoinNode* skaičius	3 0	3 0	7 2	-	5 0	3 0	3 0	7 2	2 0	5 0



66 pav. Sugeneruotų ir tikėtinų UML elementų skaitinės analizės juostinė diagrama



67 pav. Sugeneruotų ir tikėtinų UML elementų skaitinės analizės procentinė linijinė diagrama

Rezultatai tiek lentelėje (**8** lentelė), tiek grafikuose (66 pav., 67 pav.) parodė, kad dauguma pagrindinių UML elementų buvo sugeneruoti teisingai ir atitiko lūkesčius. Pavyzdžiui, trečioje procedūroje buvo tikėtasi 61 UML veiksmų elemento ir visi jie buvo sėkmingai sugeneruoti, o veiksmų ir valdymo srautų logika išlaikė numatytą struktūrą. Tiek pirmoje, tiek antroje procedūrose taip pat užfiksuota beveik visiška veiksmų atitiktis tarp tikėtinos ir gautos diagramų struktūros.

Nepaisant to, analizės metu buvo identifikuota keletas reikšmingų išimčių. Ketvirtosios procedūros nepavyko apdoroti – įrankis pateikė klaidos pranešimą, nurodydamas, kad pateiktas SQL programinio kodo failas yra nekorektiškas. Išanalizavus šį failą, pastebėta, kad klaidą sukėlė „CONVERT“ operacijos apdorojimas, kuris nebuvo tinkamai interpretuotas analizės metu. Šis atvejis parodė esminį kodo analizės bibliotekos apribojimą – tam tikros SQL funkcijos, ypač kai jos yra kompleksinės ar netipiškos, nėra tinkamai palaikomos.

Penktoji procedūra iš esmės buvo teisingai sugeneruota, tačiau įrankis pridėjo dvi papildomas UML klasių atributų reikšmes, kurios realiai nebuvo būtinos. Tai įvyko dėl „CROSS APPLY“ funkcijos naudojimo, kurioje dalis stulpelių buvo apskaičiuojami dinamiškai. Šios reikšmės buvo interpretuotos kaip atskiri atributai, nors semantiškai jos galėjo būti laikomos kaip tarpiniai skaičiavimai, o ne kaip pastovūs duomenų struktūros komponentai.

Be to, yra tam tikrų neatitikimų dėl duomenų šaltinių identifikavimo. Kai kurios lentelės, naudojamos kaip įvesties šaltiniai, nebuvo atpažintos ir pridėtos į UML diagramas. Tai dažniausiai įvyksta tuomet, kai lentelės pasiekiamos netiesiogiai arba per sudėtingas „SELECT“ išraiškas. Šis trūkumas paveikė bendrą sugeneruotų UML elementų skaičių, tačiau neturėjo reikšmingos įtakos veiksmų sekos ar srautų logikai.

Apibendrinant, įrankio funkcionalumas didžiąja dalimi atitiko metodikoje keliamus tikslus. „INSERT/SELECT“ operacijos, „JOIN“ sąlygos, „CASE“ logika bei elementarios transformacijos buvo interpretuojamos teisingai ir korektiškai pateikiamos UML veiklos ir klasių diagramose. Pastebėti neatitikimai atspindi tam tikrus dabartinius įrankio logikos ribojimus, tačiau jie netrukdo vertinti sprendimo kaip tinkamo realių procedūrų dokumentavimui, ypač kai yra naudojamas standartizuotas SQL programinio kodo rašymo stilius.

5.2.2. Interviu rezultatai

Siekiant įvertinti sukurto įrankio praktinį pritaikomumą bei naudotojų lūkesčius, buvo atlikti pusiau struktūrizuoti interviu su trimis skirtingą patirtį turinčiais IT specialistais. Visi jie turi praktikos dirbant su SQL programiniu kodu ir duomenų apdorojimo scenarijais, tačiau jų profesinės rolės ir patirtis su ETL procesais skyrėsi. Atsakymų analizė atlikta taikant teminę analizę, kurios metu buvo identifikuotos šios pagrindinės temos iš respondentų atsakymų (1 priedas):

- priemonės naudingumo įvertinimas. Visi dalyviai sutiko, kad įrankis gali turėti reikšmingą praktinę vertę sudėtingų procedūrų atveju arba situacijose, kur reikia greitai suprasti struktūrą. Duomenų inžinierius akcentavo naudą duomenų migracijos projektuose, kuriuose būtina greitai perprasti kitų sukurtas procedūras. IT analitikas pažymėjo, kad toks įrankis galėtų padėti ne tik ETL kontekste, bet ir bendresniuose SQL scenarijuose, kur svarbu suvokti užklausų struktūrą. Tuo tarpu trečiasis dalyvis pabrėžė, kad priemonė galėtų būti naudinga aukštesnio lygmens ataskaitoms arba kai reikia apžvelgti procesus organizacijos mastu;
- pastebėtos rizikos ir iššūkiai. Respondentai įvardijo keletą galimų iššūkių, susijusių su priemonės taikymu. Vienas iš jų – kaip tiksliai įrankis gali interpretuoti sudėtingas ar nestandartines SQL struktūras. Buvo pastebėta, kad generuojamos diagramos kai kuriais atvejais gali tapti per daug detalios arba sunkiai interpretuojamos be papildomos konteksto informacijos. Taip pat buvo atkreiptas dėmesys į tai, kad automatizuotas vizualizavimas negali visiškai pakeisti žmogaus interpretacijos, ypač kai procedūros logika yra neformali arba prastai struktūruota;
- potencialios taikymo sritys. Visi dalyviai nurodė skirtingus scenarijus, kuriuose priemonė galėtų būti naudinga. Duomenų inžinierius pabrėžė jos aktualumą procedūrų dokumentavimui ir greitesniam jų peržiūrėjimui vykdant migracijos projektus. IT analitikas išskyrė galimybę naudoti įrankį kaip komunikacijos priemonę tarp skirtingo techninio pasirengimo naudotojų. Trečiasis dalyvis įžvelgė potencialą naudoti tokias diagramas komandinėse peržiūrose bei planavimo procesuose, kuriuose būtina turėti apžvalginę informaciją apie duomenų transformacijas;

– naudojimo pastabos ir tobulinimo kryptys. Buvo išreikštas poreikis standartizuoti SQL kodą tam, kad įrankis veiktų kuo tiksliau. Kai kurie dalyviai pasiūlė, kad norint padidinti tikslumą, būtų naudinga naudoti tam tikrus šablonus arba laikytis tam tikrų programavimo praktikų. Taip pat buvo paminėta, kad įrankio vizualizacijos galėtų būti pritaikytos skirtingiems naudotojų profiliams, t.y. sukuriant atskirus vaizdus tiek techniniams specialistams, tiek vadovams ar verslo analitikams;

Apibendrinant, interviu rezultatai rodo, kad sukurtas sprendimas turi potencialo įvairiuose praktiniuose kontekstuose, tačiau jo sėkmingas pritaikymas priklausytų nuo naudojimo scenarijų, ELT procedūros programinio kodo kokybės bei naudotojų lūkesčių dėl vizualizacijos išsamumo ir aiškumo.

5.3. ETL procedūrų vizualizavimo naudojant UML veiklos ir klasių diagramas įrankio veikimo ir savybių analizė, kokybės kriterijų įvertinimas

Atlikti eksperimentai bei pusiau struktūruoti interviu su specialistais leidžia pateikti sukurtos priemonės veikimo vertinimą pagal kokybės kriterijus, tokius kaip generavimo tikslumas, naudojimo paprastumas, interpretacijos aiškumas bei praktinio pritaikomumo galimybės. Surinkti duomenys rodo, kad siūlomas sprendimas iš esmės atitinka keliamus funkcionalumo ir naudingumo tikslus, tačiau taip pat atskleidžia ir keletą svarbių apribojimų, į kuriuos reikėtų atsižvelgti.

Įrankis sėkmingai atlieka pagrindinę numatytą funkciją – generuoja UML veiklos ir klasių diagramas pagal pateiktą ETL procedūros SQL programinį kodą. Keturiose iš penkių testuotų procedūrų dauguma veiklos ir klasių diagramų komponentų buvo tiksliai sugeneruoti. Tačiau taip pat buvo identifikuoti keli reikšmingi ribojimai. Vienos procedūros (ketvirtosios) nepavyko apdoroti dėl kodo analizės bibliotekos nesugebėjimo tinkamai interpretuoti „CONVERT“ funkcijos, kas sukėlė sintaksės klaidos pranešimą. Kitu atveju (penktoji procedūra), dėl „CROSS APPLY“ operacijos, klasės diagrama buvo papildyta stulpeliais, kurie semantiškai neatitiko tikėtino tikslo. Šie atvejai rodo, kad sudėtingesnės arba netipinės SQL konstrukcijos gali turėti tiesioginę įtaką automatinio generavimo tikslumui.

Interviu rezultatai papildė šiuos techninius pastebėjimus kokybiniu požiūriu. Visi trys dalyviai pabrėžė, kad UML vizualizacijos būtų naudingos siekiant greičiau suprasti procedūrų logiką, ypač dirbant su kompleksinėmis ar paveldėtomis procedūromis. Minėta, kad tokios priemonės padėtų projektų audito, naujų darbuotojų įvedimo, dokumentacijos ir komunikacijos kontekstuose. Tai leidžia teigti, kad sprendimas atitinka naudojimo paprastumo ir praktinio pritaikomumo kriterijus, net jei kai kuriais atvejais gali prireikti papildomo naudotojo įsitraukimo. Taip pat buvo išreikštos pastabos dėl galimų įrankio tobulinimo krypčių. Respondentai pažymėjo, kad įrankio tikslumas galėtų būti padidintas taikant struktūrizuotą SQL programavimo stilių ar iš anksto apibrėžtus programinio kodo šablonus.

Apibendrinant galima teigti, kad sukurta priemonė yra funkcionali, konceptualiai pagrįsta ir turinti praktinį potencialą realiuose ETL projektuose. Nors šiuo metu ji nėra pilnai atspari visoms sudėtingoms SQL struktūroms, esami rezultatai rodo aiškią naudą ir pateikia tolesnio tobulinimo kryptis.

5.4. ETL procedūrų vizualizavimo naudojant UML veiklos ir klasių diagramas įrankio taikymo rekomendacijos

Atlikto tyrimo rezultatai rodo, kad sukurtas sprendimas gali būti praktiškai taikomas įvairiose, su ETL procedūrų dokumentavimu susijusiose, srityse. Pirmiausia, įrankis galėtų būti naudojamas esamų SQL programinio kodo ir duomenų srautų dokumentavimui, ypač organizacijose, kuriose šiuo metu trūksta nuoseklios procesų vizualizacijos. Tokia dokumentacija būtų naudinga tiek sistemų vystymo, tiek palaikymo etapuose.

Be to, įrankis gali būti pritaikomas žinių perdavimo ir naujų darbuotojų mokymo procesuose. Automatiškai sugeneruojamos duomenų srautų diagramos leistų greičiau ir aiškiau perteikti sudėtingų duomenų struktūrų logiką tiems, kurie tik pradeda dirbti su sistemomis. Taip pat, įrankis galėtų būti naudojamas duomenų srautų analizei sistemų priežiūros ar audito metu, siekiant greičiau identifikuoti pagrindinius duomenų transformacijų kelius. Dėl šių praktinių taikymo galimybių viena informacinių technologijų įmonė priėmė sprendimą integruoti šį įrankį į savo infrastruktūrą (2 priedas).

Svarbu paminėti, kad siekiant užtikrinti aukštesnį automatinės analizės tikslumą, būtų rekomenduotina taikyti tam tikrus SQL programavimo standartus. Visų pirma, rekomenduojama naudoti aiškią ir vieningą užklausų struktūrą – nuoseklų „JOIN“ sąlygų išdėstymą, aiškų trumpinių naudojimą bei vengti komplikuotų hierarchinių išraiškų. Taip pat pravartu riboti dinamines konstrukcijas, tokias kaip „CROSS APPLY“, kurios gali būti interpretuojamos dviprasmiškai arba sukelti perteklinių elementų generavimą. Galiausiai, naudinga laikytis stulpelių aiškumo principo – aiškiai nurodyti laukų pavadinimus vietoje „*“ simbolio, vengti nereikalingų „CONVERT“ ar „CAST“ operacijų be būtinybės. Tokių standartų taikymas padeda pagerinti kodo analizės bibliotekos darbo tikslumą ir sumažina interpretavimo klaidų riziką.

Ateityje įrankio tobulinimas turėtų būti orientuotas į sudėtingesnių SQL struktūrų interpretacijos gerinimą bei platesnės ETL procesų logikos palaikymo plėtrą.

Išvados

1. Išanalizuota ETL procedūrų dokumentavimo eiga parodė, kad šie procesai yra sudėtingi dėl didelės duomenų transformacijų ir lentelių ryšių apimties. Rankinis dokumentavimas reikalauja daug laiko ir dažnai pasižymi klaidomis, todėl automatizuoti sprendimai yra būtini darbo sąnaudoms sumažinti.
2. Dabartinių ETL dokumentavimo priemonių analizė atskleidė, kad daugelis jų yra nepakankamai universalios arba orientuotos į konkrečias platformas. Šios priemonės dažnai riboja lankstumą arba neatitinka skirtingų procesų specifikos, todėl reikalingas įrankis, galintis automatizuoti šių procedūrų dokumentavimą nepriklausomai nuo duomenų šaltinio.
3. UML diagramų pritaikymas ETL procesams yra tinkamas dėl galimybės vizualiai modeliuoti duomenų srautus, procesų eigą ir ryšius tarp elementų. UML veiklos ir klasių diagramos leidžia išsamiai atspindėti ETL procedūrų logiką, duomenų lentelių sąsajas, transformacijas bei kintamųjų naudojimą.
4. Siūloma ETL procedūrų dokumentavimo metodika grindžiama SQL programinio kodo analize, siekiant automatiškai generuoti UML veiklos ir klasių diagramų modelius. Ji remiasi iš anksto apibrėžtomis transformavimo taisyklėmis, leidžiančiomis automatiškai generuoti diagramas iš ETL procedūrų.
5. Sukurtas ETL procedūrų dokumentavimo įrankio projektas aprėpia SQL sintaksės analizę, duomenų lentelių ir jų atributų identifikavimą, lentelių ryšių analizavimą bei UML diagramų generavimą. Projektas suteikia galimybę automatizuoti procedūrų dokumentavimo procesą. Atlikta realizacija sėkmingai įgyvendino lentelių sujungimų, laikinųjų kintamųjų ir lentelių, filtravimo sąlygų diagramų generavimą. Generuojamos UML diagramos tiksliai atspindi SQL kodo logiką, todėl įrankis gali būti pritaikytas įvairiuose duomenų migravimo ir transformavimo projektuose.
6. Eksperimentinis priemonės testavimas su realiomis ETL procedūromis parodė, kad didžioji dalis jų veiksmų buvo interpretuojami tiksliai, o UML diagramose atvaizduota logika iš esmės atitiko originalų programinį kodą. Tačiau nustatyta, kad sudėtingesni atvejai gali sukelti interpretavimo netikslumų, ypač jei kodas nestruktūruotas ar parašytas nestandartiškai.
7. Interviu su IT srities specialistais parodė, kad siūloma priemonė yra naudinga dokumentavimo, analizės ir žinių perdavimo procesuose. Dalyviai akcentavo vizualizacijos pranašumus suprantant sudėtingas procedūras ir įvardijo galimas taikymo sritis, tačiau taip pat pažymėjo, kad įrankio tikslumas gali būti padidintas taikant struktūruotą programavimo praktiką.

Literatūros sąrašas

1. VASSILIADIS, P.; SIMITSIS, A.; SKIADOPOULOS, S. Modeling ETL Activities as Graphs. Iš: *Design and Management of Data Warehouses*. Torontas, 2002. Prieiga per internetą: https://www.researchgate.net/publication/220841971_Modeling_ETL_activities_as_graphs. [žiūrėta 2025-05-26].
2. SAMOTA, E. Representing ETL Flows with BPMN 2.0. 2015. Prieiga per internetą: https://upcommons.upc.edu/bitstream/handle/2117/77830/108936.pdf. [žiūrėta 2025-05-26].
3. TRUJILLO, J.; LUJÁN-MORA, S. A UML Based Approach for Modeling ETL Processes in Data Warehouses. Iš: *International Conference on Conceptual Modeling*, pp. 307-320. Čikaga, 2003. Prieiga per internetą: https://link.springer.com/chapter/10.1007/978-3-540-39648-2_25. [žiūrėta 2025-05-26].
4. DEUFEMIA, V.; GIORDANO, M.; POLESE, G.; TORTORA, G. A visual language-based system for extraction–transformation–loading development. *Software – practice and experience*. t. 44 (2013), nr. 12, pp. 1417-1440. ISSN: 1097-024X.
5. SONG, X.; YAN, X.; YANG, L. Design ETL Metamodel based on UML Profile. Iš: *Second International Symposium on Knowledge Acquisition and Modeling*. Wuhan, 2009. Prieiga per internetą: https://ieeexplore.ieee.org/abstract/document/5362446. [žiūrėta 2025-05-26].
6. Oracle. Oracle Warehouse Builder. Interneto puslapis. Prieiga per internetą: https://www.oracle.com/application-development/technologies/warehouse/warehouse-builder.html. [žiūrėta 2023-12-21].
7. IBM. IBM DataStage. Interneto puslapis. Prieiga per internetą: https://www.ibm.com/products/datastage. [žiūrėta 2023-12-21].
8. Microsoft. What is SQL Server Management Studio (SSMS)? Interneto puslapis. Prieiga per internetą: https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16. [žiūrėta 2023-12-21].
9. VASSILIADIS, P.; SIMITSIS, A.; GEORGANTAS, P.; SKIADOPOULOS, S.; TERROVITIS, M. Arktos II: A Design tool for ETL scenarios. Interneto puslapis. Prieiga per internetą: https://www.cs.uoi.gr/~pvassil/projects/arktos_II/arktos_2005_pgeor/index.html. [žiūrėta 2023-12-21].
10. SONG, I.-Y.; LIDDLE, S. W.; LING, T.-W.; SHEUERMANN, P. 22nd International Conference on Conceptual Modeling. Iš: *Conceptual Modeling -- ER 2003*. Čikaga, 2003. Prieiga per internetą: https://link.springer.com/book/10.1007/b13244. [žiūrėta 2025-05-26]
11. MUÑOZ, L.; MAZÓN, J.-N.; TRUJILLO, J. A family of experiments to validate measures for UML activity diagrams of ETL processes in data warehouses. *Information and Software Technology*, t. 52 (2010), nr. 11, pp. 1188-1203. ISSN: 0950-5849.
12. DU, N.; YE, X.; WANG, J. A schema aware ETL workflow generator. Iš: *Information Systems Frontiers*, t. 16 (2014), p. 453-471. Prieiga per internetą: https://link.springer.com/article/10.1007/s10796-012-9352-2. [žiūrėta 2025-05-26].
13. DASSAULT SYSTEMES. Magic Systems of Systems Architect. Programa. Version 2024x. 2023-11-10. Prieiga per internetą: https://www.3ds.com/products/catia/catia-magic. [žiūrėta 2025-05-26].
14. Gudu Software Limited. General SQL Parser. Interneto puslapis. Prieiga per internetą: https://www.sqlparser.com/index.php. [žiūrėta 2024-06-18].

15. CLEENEWERCK, T.; KURTEV, I. Separation of concerns in translational semantics for DSLs in model engineering. Iš: *2007 ACM Symposium on Applied Computing (SAC)* p. 985-992. Seulas, 2007. Prieiga per internetą: https://dl.acm.org/doi/10.1145/1244002.1244218. [žiūrėta 2025-05-26].
16. ZDUN, U.; HENTRICH, C.; DUSTDAR, S. Modeling Process-Driven and Service-Oriented Architectures Using Patterns and Pattern Primitives. Iš: *ACM Transactions on the Web*, 2007, 1(3), 14–(iki pabaigos). Prieiga per internetą: https://dl.acm.org/doi/10.1145/1281480.1281484. [žiūrėta 2025-05-26]
17. Object Management Group. Unified Modeling Language: Infrastructure. Interneto puslapis. Prieiga per internetą: https://www.omg.org/spec/UML/2.1/Beta1/Infrastructure/PDF. [žiūrėta 2024-06-18]
18. Gudu Software Limited. General SQL Parser for Java Developer Guide. Dokumentacija. Prieiga per internetą: https://sqlparser.com/kb/gsp_for_java_developer_guide.pdf. [žiūrėta 2025-01-21].
19. Object Management Group. XML Metadata Interchange (XMI). Versija 2.5.1. Interneto puslapis. Prieiga per internetą: https://www.omg.org/spec/XMI/2.5.1/PDF. [žiūrėta 2025-01-21].
20. AWITI, J.; VAISMAN, A. A.; ZIMÁNYI, E. Design and implementation of ETL processes using BPMN and relational algebra. *Data & Knowledge Engineering*, t. 129 (2020). ISSN: 0169-023X. Prieiga per internetą: https://doi.org/10.1016/j.datak.2020.101837. [žiūrėta 2024-05-15].
21. INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO/IEC 19503:2005, *Information technology – XML Metadata Interchange (XMI)*. Geneva: ISO, 2005.

Priedai

1 Priedas. Atsakymai į interviu klausimus

– Ar galėtumėte paaiškinti, kokias užduotis atliekate, kuriose reikia rašyti arba peržiūrėti SQL kodą?

  1 respondentas: Dažniausiai dirbu su SQL užduotimis, skirtomis duomenų analitikai ir apdorojimui, taip pat formuojant įvairias ataskaitas.

  2 respondentas: Tikrinu duomenų kokybę, rašau užklausas, kai reikia atsakyti į klausimus, susijusius su duomenų patikimumu ar tam tikrais tikslais. Dažnai tenka aiškintis, koks yra duomenų šaltinis, iš kur jis ateina, kokios lentelės naudojamos, kokie tarp jų raktai ir ryšiai. Taip pat svarbu įvertinti taikomą verslo logiką – ar laukai yra naudojami tikslingai, ar logika yra teisinga ir atitinka sprendžiamą problemą.

  3 respondentas: –.

– Ar dažniau dirbate su kitų parašytu SQL kodu, ar su savo?

  1 respondentas: Dažniausiai tenka dirbti su kitų žmonių parašytomis procedūromis, tačiau kartais pats rašau užklausas, kai reikia pasiruošti ataskaitas.

  2 respondentas: Šiuo metu dažniau dirbu su kitų parašytu kodu – tai priklauso nuo projekto. Kai dirbu su duomenų migracija, dažniausiai tenka analizuoti klientų kodą. Jei projektą kuriu nuo nulio, tada rašau savo kodą.

  3 respondentas: –.

– Ar esate kada nors turėjęs suprasti arba paaiškinti, ką daro didelė SQL operacija? Kaip paprastai tą darote?

  1 respondentas: Taip, nuolat tenka suprasti ir aiškinti didesnes SQL operacijas. Mano metodas – eiti žingsnis po žingsnio, piešti procesą schematiškai, rodyti duomenų pavyzdžius. Kartais pasitelkiu BPMN diagramas, kad žmonėms būtų aiškiau.

  2 respondentas: Taip, tikrai esu susidūręs su tokiais atvejais. Paprastai tokią operaciją reikia išskaidyti į mažesnes dalis. Žiūriu pagal duomenų šaltinius, iš kur ateina informacija, taikau „data lineage“ principus. Įvertinu, ar yra sudėtingų dalių, kurias dar reikia papildomai skaidyti.

  3 respondentas: –.

– Kas, Jūsų nuomone, yra sudėtingiausia skaitant arba palaikant didelius SQL užklausų rinkinius?

  1 respondentas: Sudėtingiausia, kai nėra bendro vaizdo – tuomet tenka nagrinėti kiekvieną detalę atskirai, einant per duomenų pavyzdžius nuo pradžios iki galo.

  2 respondentas: Didžiausias iššūkis atsiranda tada, kai SQL kodas parašytas neaiškiai. Pavyzdžiui, pavadinimai neatitinka esamų laukų duomenų šaltiniuose, arba kai kodas buvo keistas ir liko chaotiškas – tai primena „spaghetti code“.

  3 respondentas: –.

– Ar esate naudoję diagramas siekdami geriau suprasti SQL procesus?

1 respondentas: Taip. Naudoju, kai reikia nuo nulio vizualizuoti procesus. Kai sistemą reikia vystyt ar tobulinti, tai užtrunka ilgiau, nes reikia nuolat atnaujinti diagramas.

2 respondentas: Taip, esu. Diagramos padeda geriau suprasti SQL procesus ir leidžia matyti bendrą vaizdą. Tam naudoju „data lineage“ įrankius arba savarankiškai braižau schemas, jei reikia greito vaizdinio supratimo.

3 respondentas: –.

– Ar matytumėte vertę procese, kuris automatiškai paverstų SQL užklausas į diagramas?

1 respondentas: Taip, tikrai. Toks įrankis sutaupytų daug laiko ir palengvintų paaiškinimą žmonėms, kurie neturi stiprių techninių žinių.

2 respondentas: Tikrai taip, matau tokiame procese vertę. Automatinės diagramos padėtų aiškintis, kas vyksta kode, padėtų suprasti logiką ir geriau komunikuoti su kolegomis.

3 respondentas: –.

– Kokia buvo Jūsų pirmoji reakcija pamačius šį įrankį? Ar, Jūsų manymu, jis būtų naudingas?

1 respondentas: Įrankis padėtų sisteminti esamas procedūras ir duomenų srautus.

2 respondentas: Sprendimas iš karto sudomino – iškart kilo minčių, kur būtų galima jį pritaikyti.

3 respondentas: Manau, toks įrankis turi potencialo. Gal būtų galima kažką panašaus sukurti ir dirbtinio intelekto pagrindu, tačiau AI dažnai pateikia nepatikimus ar nelogiškus rezultatus. Tuo tarpu šiuo atveju metodas grįstas sistemiškai – rezultatai pateikiami aiškiai, be nereikalingų klaidų ar „nesąmonių“, kaip kartais nutinka su dirbtinio intelekto įrankiais.

– Ar savo darbo aplinkoje matytumėte atvejų, kuriuose tokia vizualizacija būtų naudinga?

1 respondentas: Taip, vizualizacijos būtų naudingos – jos aiškiai parodytų, kaip duomenys transformuojami tarp skirtingų struktūrų.

2 respondentas: Taip, tikrai. Vizualizacija būtų naudinga verslo logikos taisyklių identifikavimui. Jau dabar gerai, kad parodoma šaltinių ir tikslinių stulpelių struktūra, tačiau būtų dar geriau, jei verslo logika būtų išskirta kaip atskiros taisyklės. Taip pat esu susidūręs su problema, kad įprastiniai įrankiai ryšių analizei dažnai yra sudėtingi naudoti bei nepateikia tokio aiškaus vaizdo.

3 respondentas: Taip, tikrai. Vizualizacija būtų naudinga, ypač kai viskas yra aiškiai išskaidyta. Man ypač patiko „Populate the target table“ diagrama – ji atrodo solidžiai ir atspindi struktūrą aiškiai.

– Ar įžvelgėte kokių nors galimų problemų, susijusių su tokiu įrankiu?

1 respondentas: Gali kilti klausimų, kaip įrankis susitvarkytų su sudėtingesniais atvejais. Automatinės vizualizacijos dažnai būna nevisiškai tikslios – gali išsibarstyti vaizdas.

2 respondentas: Manau, kad naudinga būtų istorijos funkcija, kuri parodytų, kas buvo įkelta ir kokia analizė atlikta. Jei kažkas jau buvo analizuota, nereikėtų visko kartoti. Taip pat vertinga būtų galimybė palyginti procedūras – pavyzdžiui, vieną senesnę ir kitą naujesnę, bei parodyti

jų diagramų skirtumus. Tai būtų panašu į versijų kontrolę, bet labiau orientuota į įrodymų pagrindu pagrįstą tikrinimą.

3 respondentas: Būtų labai naudinga turėti funkciją, kuri leistų parodyti diagramų skirtumus – kaip jos pasikeitė laikui bėgant. Tai būtų kažkas panašaus į versijų kontrolę, tik labiau orientuota į įrodymų pagrindu grįstą (angl. *evidence-based*) analizę ar patikrą.

– Kokie aspektai, Jūsų nuomone, būtų svarbūs, kad įrankis tinkamai atliktų savo funkcijas?

1 respondentas: Reikėtų šiek tiek pritaikyti patį kodą, kad įrankis geriau suprastų struktūrą. Kitaip sakant, gal reikėtų tam tikrų kodavimo standartų laikytis.

2 respondentas: Manau, kad labai naudinga funkcija yra galimybė įkelti daugiau nei vieną užklausą vienu metu – tai ženkliai sumažina rankinio darbo kiekį ir padidina efektyvumą.

3 respondentas: Labai svarbi būtų tiesioginė integracija su konkrečia duomenų baze. Taip pat būtų didelis privalumas, jei įrankis palaikytų daugiau skirtingų duomenų bazių sistemų – tai užtikrintų platesnį pritaikomumą.

– Ar naudotumėte tokį įrankį, jei jis būtų prieinamas?

1 respondentas: Jeigu būtų atsižvelgta į sudėtingus atvejus, taip, tikrai naudotų. Išbandytų ir žiūrėtų, kaip veikia su realiomis situacijomis.

2 respondentas: Tikrai būtų įdomu išbandyti. Jeigu būtų galima patogiai ieškoti modelyje, tai labai palengvintų darbą. Dabartiniuose duomenų ryšių įrankiuose tenka grįžti prie procedūrų, kai norisi pažiūrėti konkretų atributą – tai nepatogu ir labai lėtina analizę.

3 respondentas: Taip, tikrai. Toks įrankis būtų vertingas stambiose įmonėse, kuriose yra griežti reguliaciniai reikalavimai ir būtinybė kaupti dokumentaciją pagal specifines taisykles. Tokiais atvejais toks sprendimas palengvintų darbą ir užtikrintų atsekamumą.

2 Priedas. Įrankio diegimo aktas

Intellerts

UAB INTELLERTS

TVIRTINU
Direktorius
(parašas)
Darius Dilijonas

DIEGIMO AKTAS
2025-05-13
Kaunas

Studentas Mantas Abramavičius perdavė sukurtą ETL2UML programoms prototipą. Atliko detalų pristatymą ir mokymus kaip naudotis sukurta aplikacija. Perdavė programinės įrangos kodą. Nuo 2025-05-14 dienos perduota aplikacija pardėta testuoti įmonės vykdomuose duomenų migracijos projektuose.

Perdavė Praktikos studentas	(parašas)	Mantas Abramavičius
Priėmė Duomenų mokslininkų komandos vadovas	(parašas)	Darius Dilijonas

UAB "Intellerts"
Studentų g. 3A-9, LT-50232 Kaunas
Tel. + 370 659 24412

Įmonės kodas 304980882
PVM kodas LT100012194810

https://intellerts.com
info@intellerts.com

68 pav. Įrankio, sukurto pagal darbo metodiką bei pavadintu „ETL2UML“, diegimo aktas