VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Birutė PLIUSKUVIENĖ

ADAPTYVŪS DUOMENŲ MODELIAI PROJEKTAVIME

Daktaro disertacija

Technologijos mokslai, Informatikos inžinerija (07T)

Vilnius, 2008

Disertacija rengta 2003–2008 metais Vilniaus Gedimino technikos universitete.

Darbo mokslinis konsultantas

prof. habil. dr. Petras Gailutis Adomėnas (Vilniaus Gedimino technikos universitetas, technologijos mokslai, informatikos inžinerija – 07T).

http://leidykla.vgtu.lt
VGTU leidyklos TECHNIKA 1497-M mokslo literatūros knyga

ISBN 978-9955-28-282-2

Reziumė

Disertacijoje nagrinėjamos taikomųjų uždavinių, kurių duomenys išreikšti reliacinėmis aibėmis, sprendimus realizuojančių priemonių adaptyvumo problemos. Pagrindiniai tyrimo objektai yra adaptyvieji duomenų modeliai: duomenų išrinkimo modelis, duomenų agregavimo modelis ir duomenų apdorojimo projektavimo modelis. Darbo tikslas – sukurti adaptyviąją duomenų apdorojimo projektavimo technologiją, kuri leistų išrinkti, agreguoti ir apdoroti duomenis keičiant tik šią technologiją sudarančių adaptyviųjų duomenų modelių formalių išraiškų parametrus. Naudojant sukurtą technologiją skirtingiems uždaviniams spęsti taikomas vienas ir tas pats duomenų apdorojimo principas. Kitaip tariant, visą algoritmų ir juos realizuojančių programinių modulių sistemą galime pritaikyti skirtingiems taikomojo pobūdžio uždaviniams spręsti. Tai leidžia sumažinti naujų programinių priemonių kūrimo apimtis ir sąnaudas.

Darbe sprendžiami keli pagrindiniai uždaviniai: taikomojo pobūdžio uždaviniams spręsti reikalingų pirminių duomenų išrinkimas, agregavimas ir šių duomenų apdorojimo projektavimas. Pirmasis uždavinys suformuluotas, atsižvelgiant į tai, kad konkrečiam taikomajam uždaviniui spręsti iš pirminių duomenų visumos būtina atrinkti tik tam uždaviniui reikalingus duomenis ir pateikti juos apdoroti reikiama seka. Antrasis uždavinys siejamas su atrinktų duomenų transformavimu į vieną reliacinę aibę, skirtą konkrečiam uždaviniui spręsti, t. y. duomenų agregavimu. O trečiasis apima duomenų apdorojimo projektavimą, kuris pagrįstas algoritminėmis priklausomybėmis ir jas realizuojančiais programiniais moduliais.

Disertaciją sudaro šeši skyriai, iš kurių paskutinis – bendrosios išvados.

Pirmajame (įvadiniame) skyriuje nagrinėjamas problemos aktualumas, formuluojamas darbo tikslas ir uždaviniai, aprašomas mokslinis darbo naujumas, pristatomi autorės pranešimai ir publikacijos, disertacijos struktūra.

Antrame skyriuje pateikta adaptyviųjų metodų, taikomų objektinėms programoms praplėsti, analizė. Išanalizuoti adaptyvieji metodai, kur adaptyvumas pasiekiamas kuriant programas kaip laisvai susietus besikooperuojančius fragmentus, kurių visuma leidžia išspręsti taikomojo pobūdžio uždavinius.

Trečiame skyriuje pateikiami kuriamos adaptyviosios duomenų apdorojimo projektavimo technologijos bendriausieji bruožai. Suformuluoti ir pateikti minėtosios technologijos kūrimo ir naudojimo etapai. Pateikti šioje technologijoje naudojami duomenų išrinkimo ir agregavimo modeliai, t. y. duomenų identifikavimas ir transformacijos. Detaliai aprašyti pirminių duomenų išrinkimą garantuojantys duomenų identifikatoriai ir galimos jų kombinacijos. Išskirti taikomojo pobūdžio uždaviniui išspręsti atrinktų duomenų išsamumą užtikrinantys lygmenys. Išsamiai aprašyti galimi duomenų transformacijų tipai. Apibrėžtos transformacijų formalios realizavimo išraiškos ir jų pavyzdžiai.

Ketvirtas skyrius skirtas adaptyviajam duomenų apdorojimo projektavimo modeliui, t. y algoritminėms duomenų priklausomybėms. Jas naudojant taikomojo uždavinio sprendimo programa yra sudaroma kaip visų algoritminių priklausomybių realizacijos modulių aibės poaibis. Tad šiame skyriuje apibrėžta duomenų algoritminių priklausomybių koncepcija ir jų elementai. Pateiktas algoritminių priklausomybių grupavimas. Išsamiai aprašyta kiekviena algoritminių priklausomybių grupė. Apibrėžti ir detalizuoti algoritminių priklausomybių realizavimo algoritmai.

Penktame disertacijos skyriuje pateikti adaptyviosios duomenų apdorojimo projektavimo technologijos taikymo pavyzdžiai, t. y. algoritminių priklausomybių lygmeniu aprašyti trys taikomojo pobūdžio uždavinių sprendimai.

Pagrindiniai darbo rezultatai buvo pristatyti 5 mokslinėse (iš jų dviejose tarptautinėse) konferencijose Lietuvoje ir užsienyje. Disertacijos tematika išspausdinti 7 moksliniai straipsniai: trys – tarptautinėse duomenų bazėse referuojamuose periodiniuose mokslo žurnaluose; du – tarptautinėse duomenų bazėse referuojamoje konferencijų medžiagoje; du – recenzuojamoje Lietuvos konferencijų medžiagoje.

Abstract

The dissertation deals with the adaptivity difficulties of the solutions implemented to solve applied problems whose data is expressed as relational sets. The main objects of research are adaptive data models: a data selection model, a data aggregation model and a model for designing data processing. The aim of the work is to create an adaptive technology for designing data processing that would enable to perform data selection, aggregation and processing by changing only the parameters of formal expressions for the adaptive data models forming the technology. While using the technology created for solving different problems the same data processing principle is used. In other words, the whole system of algorithms and program modules implementing them can be adjusted for solving different problems of applied nature. This allows to decrease the volume and expenses of creating new software.

Several main tasks are solved in the work: the selection, aggregation and designing data processing for the primary data needed for solving applied problems. The first task is formulated taking into account that only the data needed for solving a particular applied problem need to be selected from the total data and supplied for processing in a required sequence. The second task is tied to transforming the selected data to a single relational set for solving a particular problem or data aggregation. And the third task encompasses designing data processing that is based on algorithmic dependencies and program modules implementing them.

The dissertation is composed of six chapters, the last of which – the general conclusions.

In the first (introductory) chapter the topicality of the problem is analysed, the aim and tasks of the work are formulated, the scientific novelty of the work is described, publications and papers by the author as well as the dissertation structure are introduced.

In the second chapter the analysis of adaptive methods for extending object-oriented programs is provided. The adaptive methods that achieve adaptivity by creating programmes as loosely coupled collaborating components whose aggregation enables to solve problems of applied nature.

In the third chapter the most general features of the adaptive technology being created for designing data processing are presented. The stages for creating and use of the said technology are formulated and presented. The models for selecting and aggregating data or data identification and transformations in this technology are presented. Data identifiers and their possible combinations that guarantee the provision of primary data are described in detail. The layers guaranteeing the completeness of the data selected for solving problems of applied nature are indicated. Possible data transformation types are fully

described. Formal implementation expressions for transformations and their examples are defined.

The fourth chapter is dedicated to the adaptive model for designing data processing or algorithmic data dependencies. Using the latter a program for solving an applied problem is formed as a subset of the set of all implementation modules for algorithmic dependencies. Thus, in this chapter the concept and the elements of algorithmic data dependencies are defined. The classification of algorithmic dependencies is presented. Each group of algorithmic dependencies is fully described. Algorithms for implementing algorithmic dependencies are defined and detailed.

In the fifth chapter examples of applying the adaptive technology for designing data processing in practice are presented. Three solutions to applied problems are described at the level of algorithmic dependencies.

The main results of the work have been presented in five scientific (two of them international) conferences in Lithuania and abroad. On the topic of the dissertation seven scientific paper have been printed: three – in scientific periodic journals referred in international databases, two – in conference materials referred in international databases, two – in reviewed materials of Lithuanian conferences.

Žymenys

Simboliai

A	identifikatorių reliacinės aibės atributas, kurio reikšmės yra reliacinių aibių schemų kodai
A_j	duomenų reliacinės aibės atributai, kai $1 \le j \le n$
a_x	reliacinės aibės schemos kodas
a'_x	reliacinės aibės kopijos schemos kodas
B	identifikatorių reliacinės aibės atributas, kurio reikšmės yra subjektų kodai
b_y	subjekto kodas
b'_1	reliacinės aibės kopijos subjekto kodas
C_j	reliacinės aibės atributų kodai, kai $1 \le j \le n$
c_{ij}	atributų reikšmės, čia i nurodo reliacinės aibės kortežo eilės numerį, o j – domeno eilės numerį ($1 \le i \le m$; $1 \le j \le n$)
D	identifikatorių reliacinės aibės atributas, kurio reikšmės yra laiko faktoriaus kodai
d_z	laiko faktoriaus kodas
d'_1	reliacinės aibės kopijos laiko faktoriaus kodas
i / j	koordinatės, į kurias pernešami duomenys
K	raktinis atributas, priskirtas reliacinės aibės tam tikram schemos poaibiui
k / l	koordinatės, iš kurių imami duomenys
L_1	laukas, kuriame duotuoju momentu yra nagrinėjama algoritminė reliacinė aibė
L_2	laukas, kuriame duotuoju momentu yra nagrinėjama reliacinė aibė
L_3	laukas, kuriame saugomi tarpiniai algoritminės reliacinės aibės ir reliacinės aibės apdorojimo rezultatai arba galutinis uždavinio sprendimo rezultatas
m	reliacinės aibės eilė, kai $1 \le i \le m$
m_n	programinių modulių rangas, kai $1 \le j \le n$
n	reliacinės aibės rangas, kai $1 \le j \le n$
P	visų algoritminių priklausomybių kodų reliacinė aibė
P_r	P poaibis, t. y. algoritminių priklausomybių kodų reliacinė aibė, skirta konkrečiam uždaviniui spręsti
p^t	transformacijos algoritminės priklausomybės kodas
p^{si}	skaičiavimo-infologinės algoritminės priklausomybės kodas
p^+	skaičiavimo algoritminės priklausomybės kodas
$p^=$	infologinės algoritminės priklausomybės kodas
$p^\rightarrow$	programinių ėjimų algoritminės priklausomybės kodas
p^i	išorinės algoritminės priklausomybės kodas
p^a	nuosavos atributų reikšmių algoritminės priklausomybės kodas

R duomenų reliacinės aibės schema
r duomenų reliacinė aibė
$r^1, r^2, ..., r^d$ reliacinės aibės, kurios skirtingos savo atributų reikšmėmis
$r_1, r_2, ..., r_t$ reliacinės aibės, kurios skirtingos savo atributų reikšmėmis, bet turi tą pačią atributų schemą
z_t duomenų apdorojimo uždavinio kodas

Santrumpos

AP algoritminė priklausomybė
ARA algoritminė reliacinė aibė
DB duomenų bazė
DBVS duomenų bazių valdymo sistema
NF normalinė forma
RRA rezultatinė reliacinė aibė
RA reliacinė aibė

Turinys

1

Įvadas

1.1. Problemos formulavimas

Sprendžiant taikomojo pobūdžio uždavinius, kyla problemų, susijusių su tuo, kad kintant probleminei sričiai keičiasi pirminių duomenų struktūra ir turinys bei sprendžiamų uždavinių algoritmai. Jei iš anksto nenumatomi būsimi pokyčiai, tenka taikomuosius uždavinius projektuoti, programuoti ir įtraukti į veikiančias sistemas iš naujo. Pastaruoju atveju ne tik kyla daug problemų, kai reikia mažinti uždavinio sprendimo sutrikimus, bet ir padaugėja sąnaudų. Kadangi numatyti naujų uždavinių atsiradimą ar esamų uždavinių pokyčius dažnai labai sunku arba neįmanoma, todėl taikomojo pobūdžio uždavinius apdorojanti programinė įranga turi būti adaptyvi. Tai galima pasiekti sudarant programą iš atomarių (nedalomų) programinių modulių, tam pokyčiai turėtų mažiausią poveikį programos struktūrai.

Taigi disertacijoje nagrinėjama taikomųjų uždavinių sprendimus realizuojančių programinių priemonių, kurių nepastovumą lemia pirminių duomenų turinio, jų struktūrų ir sprendžiamų taikomojo pobūdžio uždavinių algoritmų pokyčiai, adaptavimo problema. Problemos sprendimas yra grindžiamas duomenų, kurie išreikšti reliacinėmis aibėmis, modelių adaptavimo metodika. Taikant šią metodiką skirtingiems uždaviniams spęsti bus taikomas vienas ir tas pats duomenų apdorojimo principas. Kitaip tariant, visa algoritmų ir juos realizuojančių programinių modulių sistema galės būti pritaikyta

skirtingiems taikomojo pobūdžio uždaviniams spręsti, todėl sumažėja naujų programinių priemonių kūrimo apimtis.

1.2. Darbo aktualumas

Informacinių technologijų naudotojai pageidauja vis naujų būdų ir galimybių, kuriais būtų galima keisti, apdoroti ir gauti reikiamus duomenis. Tai verčia nuolat keisti naudojamas programines priemones. Todėl atsiranda poreikis projektuoti adaptyvią, t. y. gebančią prisitaikyti, programinę įrangą. *Adaptavimo* terminas vartojamas plačiai, nors įvairių autorių suvokiamas gana skirtingai [3], [31], [54], [55], [61]. Bendrąja prasme adaptavimas suvokiamas kaip programinės įrangos prisitaikymas kintančiai probleminės srities terpei.

Išanalizavus užsienio mokslininkų literatūrą, galima teigti, kad daugelis autorių [1], [6], [28], [29], [31], [52], [58], [71] savo darbuose adaptyvumo sampratą sieja su išplėstinėmis objektinėmis programomis, kurios yra papildytos adaptyviuoju programavimu [1], [9], [16], [29], [30], [31], [34], [41], [44], [56], [61], [67], [68]. Šioje disertacijoje adaptyvumo samprata nėra tiesiogiai sietina su objektinėmis programomis ar adaptyviuoju programavimu. Šiame darbe adaptavimas traktuojamas kaip duomenų apdorojimo projektavimo technologijos gebėjimas prisitaikyti prie skirtingų taikomųjų uždavinių sprendimo ir tai yra užtikrina adaptyvieji duomenų modeliai. Šie modeliai ir leidžia tą patį duomenų apdorojimo principą taikyti skirtingiems uždaviniams spręsti. Šios technologijos ribojimas yra toks, kad apdorojami duomenys turi būti išreikšti reliacinėmis aibėmis (toliau – RA). Kuriami adaptyvieji duomenų modeliai yra skirti struktūrinėms-reliacinėms sistemoms kurti. Daugelis Lietuvos ir užsienio firmų bei mokslo institucijų, nepaisant objektinių, aspektinių ir kitų technologijų, vis dar taiko struktūrinius [13], [26] sistemų kūrimo metodus, kurie orientuoti į reliacinį duomenų pateikimą. Taip yra todėl, kad naujosios technologijos dar nėra brandžios. Atsižvelgiant į tai, prasminga tobulinti struktūrinę metodiką. Tai ir yra šio darbo aktualumo pagrindas.

Disertacijoje apibrėžta adaptyvumo samprata yra analogiška sampratai, kuri pateikiama ir kai kuriuose moksliniuose darbuose [2], [3], atliktuose Lietuvoje.

1.3. Tyrimų objektas

Darbo tyrimų objektai yra adaptyvieji duomenų, išreikštų reliacinėmis aibėmis, modeliai:

- duomenų išrinkimo modelis, kurio pagrindą sudaro RA identifikatoriai;
- duomenų agregavimo modelis, kuris grindžiamas transformacijomis;

- duomenų apdorojimo projektavimo modelis, kurį sudaro algoritminės duomenų priklausomybės ir jas realizuojantys programiniai moduliai.

1.4. Darbo tikslas

Šio darbo tikslas – sukurti adaptyviąją duomenų apdorojimo projektavimo technologiją, kuri leistų duomenis išrinkti, agreguoti ir apdoroti keičiant tik šią technologiją sudarančių adaptyviųjų duomenų modelių formalių išraiškų parametrus.

1.5. Darbo uždaviniai

Darbo tikslui pasiekti kuriama programų projektavimo metodika, kuri reikalauja išspręsti šiuos uždavinius:

1. Išanalizuoti metodus, kurie leidžia užtikrinti taikomiesiems uždaviniams spręsti naudojamų programinių priemonių adaptyvumą.
2. Sudaryti duomenų modelių adaptavimo taikomųjų uždavinių sprendimams metodiką, kai naudojami duomenys yra išreikšti reliacinėmis aibėmis.
3. Apibrėžti duomenų, kurie yra išreikšti reliacinėmis aibėmis, išrinkimo ir agregavimo modelius, t. y. duomenų identifikavimą ir transformacijas.
4. Sukurti duomenų apdorojimo projektavimo modelį, t. y. algoritminių priklausomybių tarp duomenų aibę ir jai adekvačią programinių modulių aibę taip, kad atrenkant tų modulių aibės poaibius (t. y. projektuojant taikomąjį uždavinį), parenkant jų naudojimo seką ar tvarką, būtų galima išspręsti taikomąjį uždavinį, turintį ribojimus: pirminė duomenų struktūra ir tų duomenų apdorojimo rezultatų struktūra yra reliacinė aibė.

1.6. Tyrimo metodika

Literatūroje aprašytų adaptyviųjų metodų analizei atlikti darbe buvo taikyti aprašomieji ir lyginamosios analizės metodai, išanalizuotai medžiagai apibendrinti – indukcijos metodas. Adaptyviųjų duomenų modeliams kurti panaudoti reliacinės algebros elementai, pritaikyti struktūriniai sistemų kūrimo ir indukcijos metodai.

1.7. Mokslinis darbo naujumas

Rengiant disertaciją buvo gauti informatikos inžinerijos mokslui nauji rezultatai:

1. Išplėtota adaptavimo teorija, taikant ją ne objektiniam modeliui, o reliacinėms aibėms. Tai praplečia adaptyviojo projektavimo galimybes.
2. Sudaryta duomenų modelių adaptavimo metodika, skirta taikomiesiems uždaviniams spręsti, panaudojant transformacijas ir surenkamąjį programavimą kaip bendrą technologiją.
3. Transformacijos suskirstytos į: besąlygines, sąlygines, sąlygines ištisines, sąlygines ciklines ir aritmetines. Tai laikytina naujumu, standartizuojant struktūrinio projektavimo transformacijas.

1.8. Ginamieji disertacijos teiginiai

1. Duomenų modelių, kurie išreikšti reliacinėmis aibėmis, adaptavimo metodika.
2. Tos pačios algoritminių duomenų priklausomybių aibės poaibių naudojimas skirtingiems uždaviniams spręsti, t. y. tos aibės adaptavimas konkrečiam taikomajam uždaviniui spręsti.

1.9. Darbo rezultatų aprobavimas

Disertacijos tematika išspausdinti 7 moksliniai straipsniai: trys – tarptautinėse duomenų bazėse referuojamuose periodiniuose mokslo žurnaluose [76A], [77A], [78A]; du – tarptautinėse duomenų bazėse referuojamoje konferencijų medžiagoje [79A], [80A]; du – recenzuojamoje Lietuvos konferencijų medžiagoje [81A], [82A].

Disertacijoje atliktų tyrimų rezultatai buvo pristatyti 5 mokslinėse (iš jų dviejose tarptautinėse) konferencijose Lietuvoje ir užsienyje:

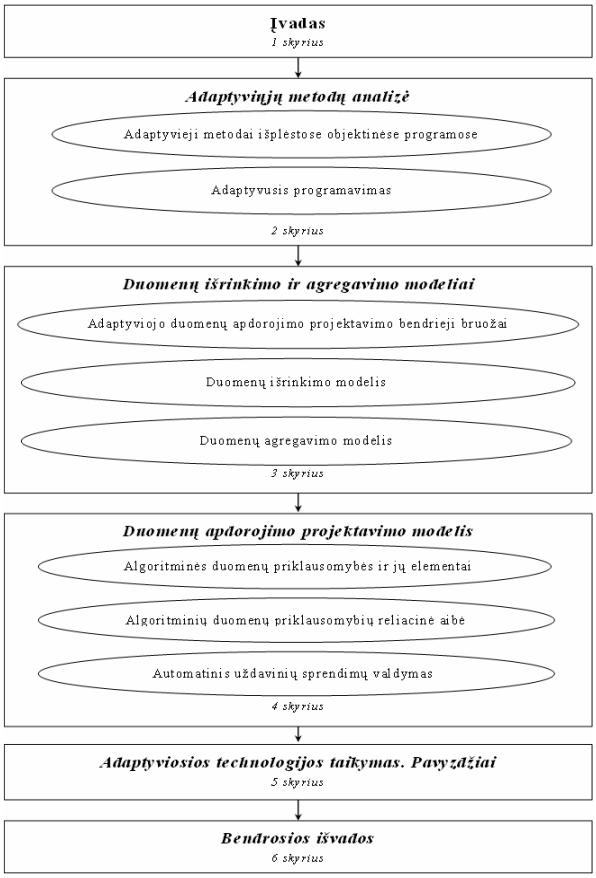
1. Informacinės technologijos: aktualijos ir perspektyvos, 2005 m. Alytuje (Lietuva).
2. Lietuvos matematikų draugijos XLVI konferencija, 2005 m. Vilniuje (Lietuva).
3. 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases, 2006 m. Madride (Ispanija).
4. 6th WSEAS International Conference on Applied Computer Science, 2006 m. Tenerifėje (Ispanija).

5. Lietuvos mokslas ir pramonė: Informacinės technologijos, 2007 m. Kaune (Lietuva).

Atlikti darbai taip pat buvo aprobuoti ir įvertinti tarptautiniame NorFa (*Nordisk Forskerutdanningsakademi*) WIM (*Wireless Information Management*) projekte 2005 metais.

1.10. Disertacijos struktūra

Disertaciją sudaro šeši skyriai, iš kurių pirmasis yra įvadas, o paskutinis – bendrosios išvados. Darbo apimtis – 128 puslapiai, tekste panaudota 181 numeruota formulė, 25 paveikslai ir 20 lentelių. Rašant disertaciją naudotasi 82 literatūros šaltiniais.



1.1 pav. *Bendra darbo struktūra*

2

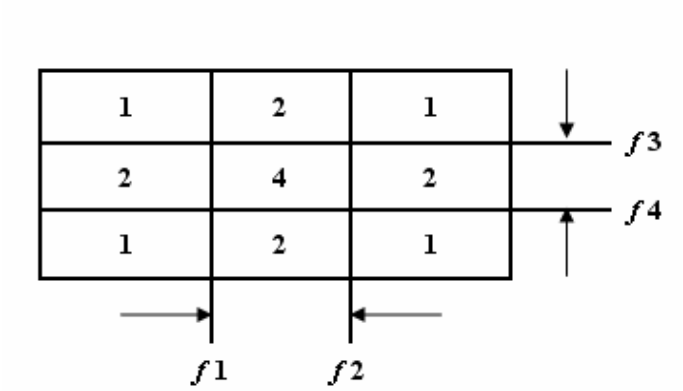
Adaptyviųjų metodų analizė

2.1. Adaptyvieji metodai išplėstose objektinėse programose

Adaptyviojo programavimo pradininkas – Bostono Northeastern universiteto mokslininkas K. J. Lieberherris teigia, kad objektines programas lengviau išplėsti nei programas, kurios sukurtos neobjektiniu stiliumi. Tačiau objektinės programos vis tiek nėra pakankamai lanksčios, kadangi jas sunku pritaikyti įvairiems uždaviniams spręsti [31]. Objektinio programavimo savybė metodus susieti su klasėmis arba su klasių grupėmis [7], [48], [50], [51] yra kartu ir pranašumas, ir trūkumas. Viena vertus, metodų susiejimas su klasėmis leidžia joms gauti ir siųsti pranešimus, bet įvairios objektų klasės skirtingai reaguoja į konkretų pranešimą. Kita vertus, trūkumas tas, kad aiškiai susiedami kiekvieną metodą su konkrečia klase, be reikalo užkoduojame klasių struktūros detales į programą [47]. Todėl programos tampa nepaslankios plėtrai ir naudojimo kaitai. Kitaip tariant, šiuolaikiškos objektinės programos dažnai turi perteklinės, su taikomąja programa susijusios informacijos ir susiaurėja jų pakartotino naudojimo galimybė [31], [40].

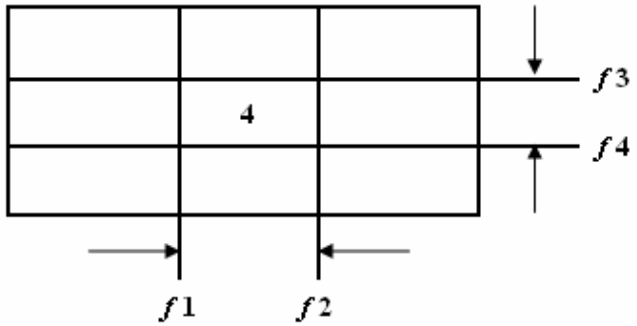
Pasak K. J. Lieberherrio, objektinių programų programuotojai turi nuolat įrašyti klasių struktūros detales į metodus. Taip sukuriamos programos su didele entropija. Tad į programas patenka atsitiktinės klasių struktūros detalės. 2.1 pav. parodyta klasių struktūra (visas kvadratas) ir keturios funkcijos, parašytos

įvairioms klasių struktūros dalims. Pirmoji funkcija ($f1$) naudoja dvi trečiąsias klasių struktūros dalis iš dešinės, antroji funkcija ($f2$) – dvi trečiąsias iš kairės, trečioji funkcija ($f3$) – apatines dvi trečiąsias, o ketvirtoji funkcija ($f4$) – viršutines dvi trečiąsias [31].



2.1 pav. *Objektinio programavimo klasių struktūrų dubliavimas* [31]

Centrinė klasių struktūros dalis yra keturis kartus aprašoma metoduose, realizuojančiuose keturias funkcijas (2.2 pav.). Jei klasių struktūra centre pasikeistų, tai reikėtų atnaujinti net keturias funkcijas [31].



2.2 pav. *Objektinio programavimo centrinės klasių struktūros dalies dubliavimas*

Atsižvelgdamas į tai, K. J. Lieberherris su savo kolegomis adaptyvųjį objektinį programavimą apibrėžė kaip įprastojo objektinio programavimo išplėtimą [29], [31], [57]. Adaptyvusis objektinis programavimas suteikia galimybę išreikšti elementus – klases ir metodus, kurie yra esminiai taikomajai programai, vengiant susiejimo su konkrečios programos klasių struktūra. Adaptyviosios objektinės programos nusako esmines klases ir metodus,

apribodamos klasių struktūros, bandančios pritaikyti adaptyviąją programą, konfigūraciją, nedetalizuojant tokios klasių struktūros. Kitaip tariant, jei objektinių programų programuotojai susieja metodus konkrečiai su klasėmis, tai adaptyviųjų programų programuotojai atideda metodų susiejimą iki tol, kol pateikiamas klasių struktūros adapteris. Šiuo būdu adaptyviųjų programų programuotojai yra skatinami galvoti apie programų šeimas [31], [37].

2.1.1. Adaptyvumas taikant Demetros dėsnį

Taikant Demetros dėsnį [16], [25], [38], [66] susitelkiama į adaptyviąsias programas. Joms mažiau rūpi struktūra, nes apibrėžiant funkcionalumą reikia tik minimalios su klasių struktūros realizacija susijusios informacijos. Tokių programų pranašumas tas, kad jų paskirtis išreiškiama aukštesniu abstrakcijos lygmeniu. Todėl ir šių programų skaitytojams, ir kūrėjams nebereikia analizuoti sudėtingos klasių struktūros. Užtenka dalinių žinių apie klasių struktūrą. Šis aukštesnis abstrakcijos lygmuo, be to, sutrumpina programas ir padaro jas lengviau naudojamas. Adaptyviosios programos kuriamos su dviem laisvai susietais fragmentais: funkcionalumu ir realizacijos klasių struktūra [29], [31].

Adaptyvioji programinė įranga naudojasi dalinėmis klasių struktūros žiniomis, o tai tiesiogiai palaiko iteratyvų programinės įrangos gyvavimo ciklą. Kai pradedami sudėtingi projektai, optimali klasių struktūra nėra žinoma ir yra nustatoma tik projektuojant arba naudojant projektą. Klasių struktūra gali būti lengviau keičiama adaptyviųjų, o ne objektinių programų plotmėje [29], [31], [37]. Klasių bibliotekos gali būti specifikuojamos lanksčiai, apibrėžiant funkcionalumą, laisvai susietą su struktūra. Adaptyviosios programos gali būti sukurtos taip, kad veiktų su įvairiomis klasių bibliotekomis, jei tik adaptyviajai programai reikalingus išteklius parūpina klasių biblioteka [31].

2.1.2. Demetros gyvavimo ciklas

Du baziniai gyvavimo ciklo modelio komponentai yra kūrimo fazių rinkinys (aprašomas veiklomis ir rezultatais) ir vykdymo tvarka, pagal kurią vykdomos veiklos, kad būtų pasiekti rezultatai. Demetros gyvavimo ciklo modelis yra spiralės modelio pritaikymas adaptyviajai programinei įrangai. Yra keturios pagrindinės spiralės modelio veiklos: planavimas, rizikos analizė, inžinerija ir klientų vertinimas. Inžinerijos veikla sukuria kito lygmens produktą ir susideda iš:

- programinės įrangos projektavimo;
- kodo rašymo;
- testavimo.

Inžinerijos veikla kaip įvestis naudoja naudojimo atvejų diagramas ir programos objektus, nustatytus planavimo metu, taip pat rizikos analizės rezultatus, kad būtų galima nuspręsti, ką realizuoti [31].

Projektuojant reikalavimai programinei įrangai paverčiami reprezentacijų rinkiniu. Pirmoji reprezentacija yra adapterių rinkinys, aprašantis klasių struktūrą, architektūrą ir objekto kalbas. Antroji reprezentacija yra adaptyviųjų programų rinkinys (be apvalkalų), kurios aproksimuoja norimą funkcionalumą. Trečioji reprezentacija yra evoliucijos istorijų rinkinys, nusakantis, kokia tvarka kuriami adapteriai ir adaptyviosios programos. Ir adaptyviosios programos, ir evoliucijos istorijos aprašomos glaustomis grafų specifikacijomis [31].

Adapteriams taikomos projektavimo taisyklės leidžia įvertinti adapterių kokybę. Patikrinama, ar adapteriai atitinka taisykles. Jų nepaisymas sukeltų netinkamą adapterių sukurtų programų veikimą. Patikrinus projektavimo taisykles, adapteriai gali būti optimizuojami.

Adaptyviųjų programų projektavimo taisyklės padeda nustatyti adaptyviųjų programų kokybę. Viena iš šių projektavimo taisyklių yra Demetros dėsnis, teigiantis, kad kuriant apvalkalą reikia naudoti tik labai ribotą klasių rinkinį. Adaptyviosios programos kuriamos naudojant dalį klasių struktūrų [32]. Daugelis šablonų, sudarytų šablonų kūrėjų bendruomenės, klasių struktūroms projektuoti gali būti naudojami adaptyviosioms programoms kurti.

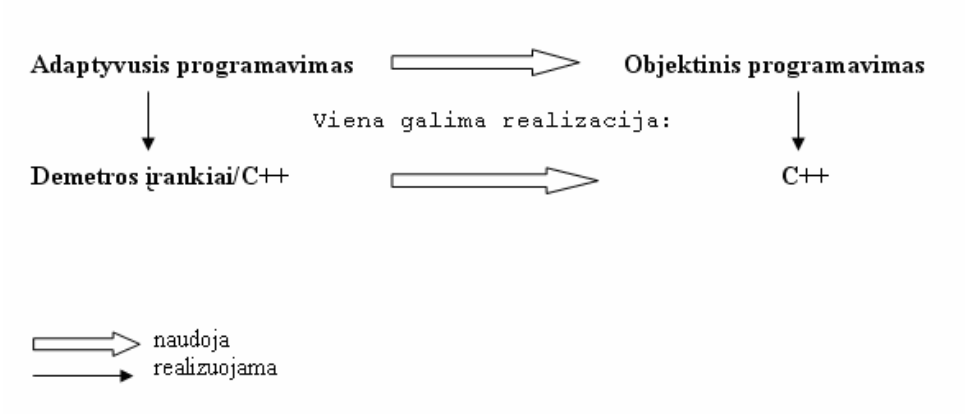
Kodo rašymas apima visų apvalkalų sukūrimą ir Demetros kompiliatoriaus iškvietimą, kad adaptyviosios programos ir adapteriai būtų paversti vykdomosiomis objektinėmis programomis.

Testuojant metu patikrinama, ar visos naudojimo atvejų diagramos buvo realizuotos korektiškai. Testavimo įvestys dažnai nurodomos objekto kalbomis, kurias apibrėžia klientai [31].

Adaptyviosios programos, jei yra naudojamos Demetros metode/C++, remiasi programavimo kalba C++, tačiau adaptyviojo programavimo metodas nepriklauso nuo programavimo kalbų [16]. Tačiau reikalinga konkreti programavimo kalba, kad koncepcijas būtų galima įgyvendinti. Todėl dėl įvairių priežasčių Lieberherris su savo kolegomis kaip pavyzdį pasirinko C++. Taip pat sukurti įrankiai, palaikantys Demetros metodą C++, Borland Pascaliui ir Lispui su objektiniais praplėtimais.

Patobulintos adaptyviosios programos leidžia iki galo pasinaudoti objektinės technologijos pranašumais. Tuo didesnis adaptyvumo pranašumas, kuo didesnės klasių struktūros ir kuo ilgiau programinė įranga naudojama. Adaptyviosios programinės įrangos realizacija yra efektyvi, vykdymo metu sąnaudos neviršija objektinės programinės įrangos sąnaudų. Be to, adaptyviosios programinės įrangos kompiliavimas, pavyzdžiui, į C++, yra spartus [31].

Demetros įrankiai, skirti realizuoti C++ kalba (Demetros įrankiai/C++), yra tik viena iš galimų adaptyviosios programinės įrangos realizacijų (2.3 pav.).



2.3 pav. *Adaptyviojo programavimo realizavimas* [31]

Paprastos programinės įrangos kūrimas su Demetros įrankiais/C++ apima šiuos etapus [31]:

1. Sukuriamas adapteris, nurodantis klasių biblioteką ir objektui skirtą programavimo kalbą.
2. Paleidžiamas projektavimo tikrinimo įrankis, kad būtų patikrintas adapterio neprieštaringumas.
3. Sukuriama adaptyvioji programa, aprašanti norimą taikomosios programos funkcionalumą.
4. Paleidžiamas projektavimo tikrinimo įrankis, kad būtų patikrintas adaptyviosios programos neprieštaringumas ir jos suderinamumas su adapteriu.
5. Generuojamas C++ kodas su Demetros kompiliatoriumi. Kompiliuojamas C++ kodas:

kompiliuoti (ADAPTYVIOSIOS PROGRAMOS, ADAPTERIS) –> vykdomoji C++ programa.

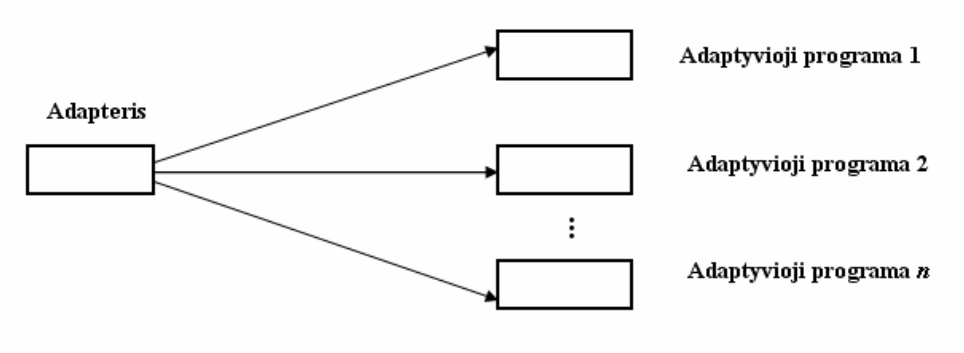
6. Paleidžiama vykdomoji programa, kurios įvestys yra objektų aprašai, tinkami konkrečiam adapteriui.

2.1.3. Adaptyviosios programos ir adapteriai

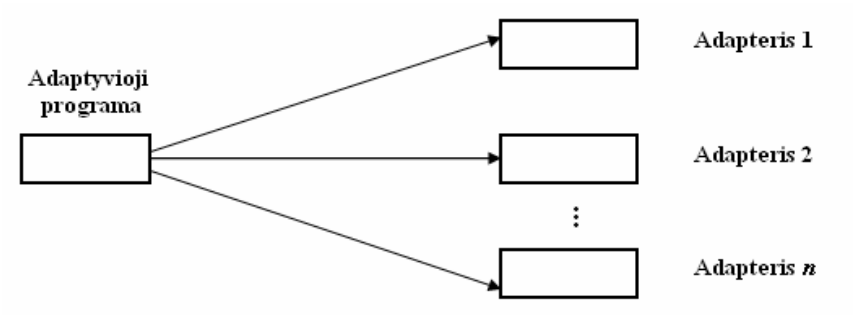
Adaptyviąsias programas sudaro navigacijos specifikacijos, aprašančios, kaip aplankomi objektai, ir apvalkalai, nurodantys vykdomąsias programas (pvz., C++ funkcijų narių sakinius), vykdomas prieš objekto aplankymą ir po jo [31], [44], [58].

Egzistuoja simetriški santykiai tarp adapterių ir adaptyviųjų programų. Šiuo atveju adapteris apibrėžiamas kaip programinis modulis, kuris derina tam tikrų adaptyviųjų programų bendrą naudojimą, kai sprendžiami konkretūs uždaviniai.

Adapteris sukuriamas vieną kartą, kad būtų galima jį naudoti su keliomis adaptyviosiomis programomis, kurios dera prie adapterio (2.4 pav.) [31].



2.4 pav. *Pakartotinis adapterio naudojimas* [31]



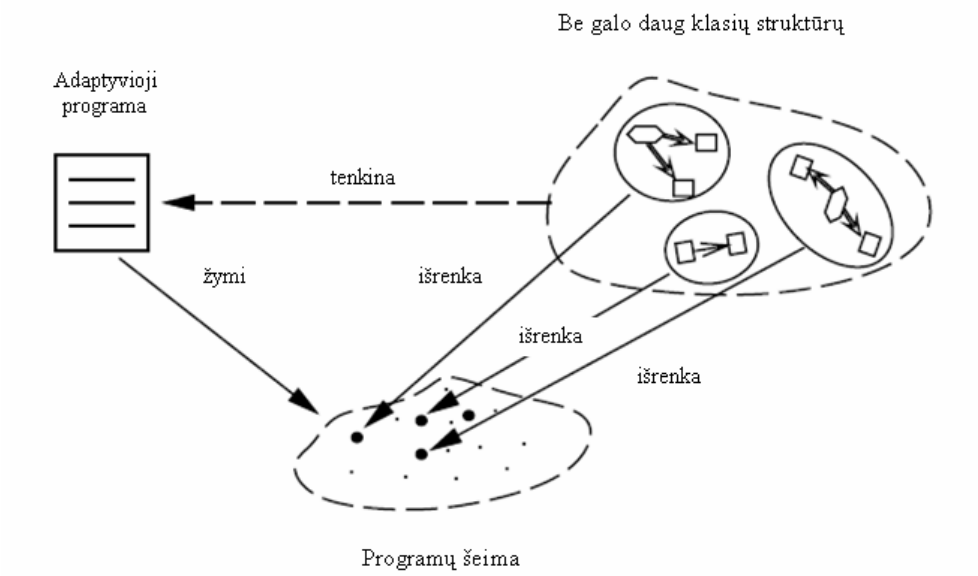
2.5 pav. *Pakartotinis adaptyviosios programos naudojimas* [31]

Adaptyvioji programa gali būti pritaikyta daugeliui skirtingų adapterių, kurie dera su ta programa (2.5 pav.). Taip adaptyvioji programa gali būti pakartotinai naudojama, kai pasikeičia objektų struktūra [31].

2.2. Adaptyvusis programavimas

Įprastines objektines programas sudaro struktūros apibrėžimas, kuris detalizuoja klasių struktūrą, ir funkcionalumo apibrėžimas, kuriuo remiantis realizuojami metodai, susieti su klasių struktūromis. Panašiai adaptyviosios programos apibrėžiamos struktūriškai ir funkcionaliai. Adaptyvioji programa skiriasi tuo, kad klasių struktūros aprašomos tik iš dalies, pasirenkant ribas, kurias turi atitikti pritaikoma klasių struktūra. Be to, funkcionalumas nėra visiškai realizuojamas. Metodai adaptyvioje programoje yra nurodomi tik tada, kai jų reikia, kai jie realizuoja esminę funkcionalumo dalį [31], [40].

Tarkime, yra klasių struktūros, kurios tenkina adaptyviąją programą. Tada žymėdami tokias adaptyviąsias programas ir iš be galo daug klasių struktūrų išrinkdami jas tenkinančias klasių struktūras, galime sudaryti potencialiai begalinę įvairių programų šeimą (2.6 pav.).



2.6 pav. *Begalinė programų šeima, žymima adaptyviosios programos* [31]

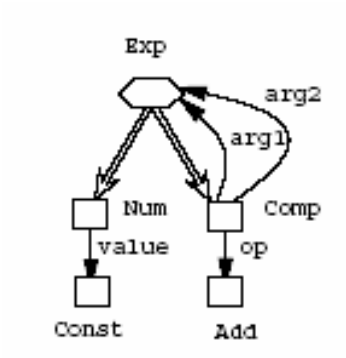
Adaptyviosios programos kūrimo procesas gali būti traktuojamas kaip prielaidų kūrimo procesas. Šios prielaidos išreiškiamos kaip klasių struktūrų, kurių pritaiko adaptyviąją programą, apribojimai. Tokie apribojimai specifikuoja susisiejančių klasių grupes pritaikomų klasių struktūroje. Taigi struktūrinė adaptyviosios programos dalis turėtų nurodyti apribojimus, išreikštus klasių ir sąryšių reikšmių kintamaisiais. Klasių reikšmių kintamieji išvardija klasių egzistavimo prielaidas pritaikomoje klasių struktūroje. Sąryšių reikšmių kintamieji dar labiau apriboja pritaikomų klasių struktūras pašalindami arba įtraukdami sąryšius tarp klasių [31].

K. J. Lieberherris adaptyviosiose objektinėse programose architektūrai aprašyti naudoja klasių grafus. J. Palsbergas [27], [57], [58], [59] savo straipsnyje [57] apibūdindamas adaptyvųjį programavimą, pateikia toliau aprašytus ir šią informaciją detalizuojančius pavyzdžius. Pasak jo, objektinėse programose kiekviena duomenų struktūra yra objektas ir kiekvienas objektas sukuriamas iš klasės. Tokioje programoje objekto formą lemia kintamųjų tipai ir programos poklasių sąryšiai (J. Palsbergas daro prielaidą, kad imami C++ stiliaus atvejų kintamieji, kur kiekvienas tipas yra klasės vardas). Adaptyviosiose programose ši struktūrinė informacija apibendrinama klasių grafu, kurio mazgai yra klasės, o

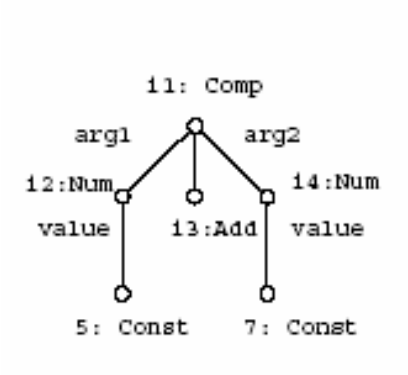
briaunos nurodo arba kintamojo tipą, arba poklasio sąryšį (2.7, 2.8, 2.9. pav.). Analizuojamu atveju klasių grafas turi penkis mazgus, vaizduojančius klases, ir poklasių briaunas (dvigubos strėlės), vaizduojančias, pavyzdžiui, kad *Num* klasė turi *Const* tipo kintamąjį, pavadintą *value* [57].

```
OPERATION print-nested
   TRAVERSE
      [Exp,Comp,Num]
   WRAPPER Num
      (@ print value @)
```

2.7 pav. *Adaptyvioji programa* [57]



2.8 pav. *Klasių grafas* [57]



2.9 pav. *Objektų grafas* [57]

Adaptyvioji programa (2.7 pav.) naudoja navigacijos specifikaciją (užrašytą po *TRAVERSE*) maršrutų rinkiniui tam tikrame klasių grafe (2.8 pav.) pažymėti. Įvairūs klasių grafai pateikia skirtingus maršrutų rinkinius. Navigacijos specifikacija [*Exp, Comp, Num*] nurodo maršrutų iš *Exp* į *Num* per *Comp* rinkinius. Šiuo maršrutų rinkiniu galima eiti per objektus. Pagal nurodytą maršrutų rinkinį bus atliekami šie ėjimai ir navigacija vykdys apvalkalo kodą (parašytą po *WRAPPER* tarp simbolių @). Pavyzdžiui, *Num* apvalkalas vykdomas kiekvieną kartą, kai aptinkamas *Num* klasės objektas. Analizuojama adaptyvioji programa išspausdina (žr. 2.7 pav. pateiktą adaptyviąją programą) visų *Num* objektų, įdėtų į *Comp* objektą, *value* lauką [57].

Taigi K. J. Lieberherrio ir jo kolegų adaptyviojo objektinio programavimo sprendimas – atskirti klasių grafą nuo programos teksto, užuot laikius jį sudėtine teksto dalimi [31], [44]. Turint konkretų klasių grafą, adaptyvioji programa ir klasių grafas gali būti sukompiliuojami į efektyvią objektinę programą programinės įrangos generatoriumi [58], [59]. Akivaizdu, kad adaptyviosios programos gali nekisti pasikeitus klasių grafui, nes tereikia perkompiliuoti pačią programą [57].

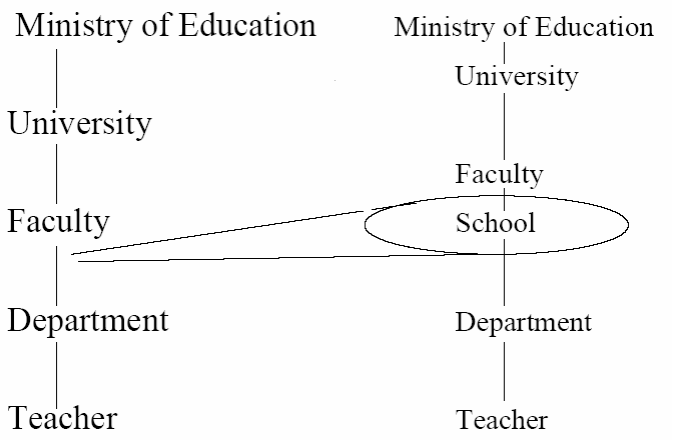
P. Krohas aprašo pavyzdžius, iliustruojančius K. J. Lieberherrio adaptyviojo programavimo pagrįstumą. Savo darbe [29] jis teigia, kad operacijos pranešimų seka priklauso nuo esamos klasių hierarchijos. Taip yra todėl, kad operacija objektinėse programose yra paskleista per įvairių klasių metodus. Kiekvienas iš šių metodų turi „žinoti“ visas klasių hierarchijos detales, kad nusiųstų pranešimą į reikiamą vietą ir kad būtų pratęstas užklausos apdorojimas. Jei dėl sistemos keitimosi klasių hierarchija pasikeistų, paprastai tektų keisti ir pranešimų seką, nes kai kurie navigacijos maršrutai irgi pasikeistų. Todėl operacijos turi būti patikrintos ir perprogramuotos atsižvelgiant į pokyčius. Taip objektinių programų priežiūra pasidaro brangi, nes operacijai atlikti programuotojas turi aprašyti:

- pranešimų seką kaip sekos diagramą;
- metodų aibę, paskleistą per klases, kurios turi priimti ir įvykdyti šiuos pranešimus, kai sujungiama pranešimo srautu.

Taigi objektinės programos yra priklausomos nuo konkrečios klasių struktūros, todėl jas sunku pritaikyti įvykus pokyčiams [29].

Kaip pavyzdį P. Krohas pateikia organizacinės sistemos (2.10 pav.) praplėtimą struktūriniu padaliniu – „school“ (kadangi su paveiksle pateikta informacija susijusios užklausos apibrėžtos anglų klaba, todėl dėl aiškumo ir paveiksle medžiaga pateikiama angliškai).

Po šio struktūrinio pasikeitimo visos užklausos, realizuotos kalbomis, kurios priklauso nuo klasių hierarchijos (pvz., C++, Java), turi būti perprogramuotos. Kaip pavyzdys paimta C++ kalba realizuota ministro užklausa: „Kiek pedagogų (teachers) yra universitetuose?“ (2.11 pav.) [29].



2.10 pav. *Struktūros keitimosi pavyzdys* [29]

```
MinistryOfEducation ← Minister (How many teachers do we have?)
        TotalTeachers : = 0
        P :- MinistryOfEducation . ListOfUniversties
For Each P.University do University ← MinistryOfEducation (How many teachers do you
have?)
    Q :- University.ListOfFaculties
For Each Q.Faculty do Faculty ← University (How many teachers do you have?)
    R :- Facultyl.ListOfDepartments
For Each R.Department do Department ← School (How many teachers do you have?)
        S:-Department.ListOfTeachers
For Each S.Teacher do TotalTeachers:=TotalTeachers + 1
        EndFor
    EndFor
  EndFor
EndFor
Minister ← MinistryOfEducation(TotalTeachers)
```

2.11 pav. *Užklausa prieš keitimąsi* [29]

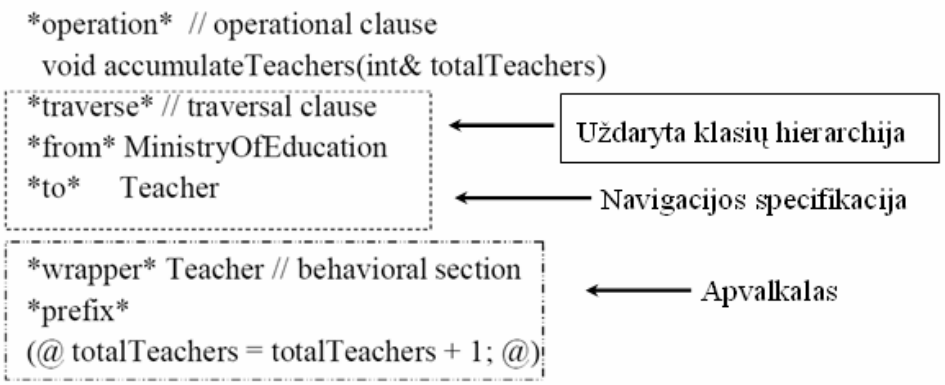
Turimą užklausą perprogramavus ir pritaikius ją prie įvykusių struktūrinių pasikeitimų, ši užklausa turės tam tikrus pakitimus (2.12 pav.) [29].

Remiantis šiuo pavyzdžiu, galima teigti, kad kai kurios objektinės programos priklauso nuo klasių hierarchijos pokyčių. Taip yra todėl, kad objektinė technologija uždaro duomenis ir funkcionalumą klasėse. O adaptyviosios programos apibrėžia funkcionalumą nenurodydamos detalios dalyvaujančių objektų grandininės struktūros. Dėl to šios programos yra mažiau priklausomos nuo konkrečios klasių struktūros ir tampa lengviau pritaikomos prie pasikeitimų ir evoliucijos [29]. Analizuojamu atveju adaptyvioji programa pateikta 2.13 pav.

Ši programa parašyta adaptyvųjį programavimą palaikančia Demeter/Java [25], [42] kalba:

```
MinistryOfEducation ← Minister (How many teachers do we have?)
        TotalTeachers : = 0
        P :- MinistryOfEducation . ListOfUniversties
For Each P.University do University ← MinistryOfEducation (How many teachers do you
have?)
        Q :- University.ListOfFaculties
For Each Q.Faculty do Faculty ←University (How many teachers do you have?)
        R :- Faculty.ListOfSchools
For Each R.School do School ← Faculty (How many teachers do you have?)
        S :- School.ListOfDepartments
For Each S.Department do Department ← School (How many teachers do you have?)
        T:-Department.ListOfTeachers
For Each T.Teacher do TotalTeachers:=TotalTeachers + 1
        EndFor EndFor
    EndFor
EndFor EndFor
Minister ←MinistryOfEducation(TotalTeachers)
```

2.12 pav. *Užklausa po keitimosi* [29]



2.13 pav. *Pavyzdys Demeter/Java kalba* [29]

Šiame pavyzdyje tik paskutinė klasė „Teacher“ turi duomenų, lemiančių rezultatus. Ši klasė turi metodą, kurį reikia konkrečiai aprašyti, ir tai atlikta apvalkale. Visos kitos klasės teikia pranešimą savo poklasiams, o jų atitinkamų metodų blokai turi tik nereikšmingą skleidimo metodiką. Tai gali būti sugeneruojama automatiškai, naudojantis esama klasių hierarchija [29]. Programa, parašyta Demeter/Java, nurodo, kad visi maršrutai iš šakninės klasės „MinistryOfEducation“ į paskutinę „Teacher“ turėtų būti išnaudojami be jokių

apribojimų. Kiekvieną kartą, kai pasiekiamas klasės „Teacher“ objektas, skaitiklis „totalTeacher“ bus padidintas. Tarp simbolių @ pateikta dalis bus įstatyta į klasės „Teacher“ sugeneruotų metodų blokus. Kitų metodų blokuose bus įdedamas tik sklidimo kodas.

Taigi adaptyvusis metodas uždaro operacijos funkcionalumą vienoje vietoje, taip išvengdamas išmėtymo problemos, bet taip pat abstrahuoja klasių struktūros atžvilgiu, taip išvengdamas ir painiavos. Funkcionalumas išreiškiamas lyg aukšto lygio aprašas kaip pasiekti skaičiavimo proceso dalyvius (vadinama navigacijos specifikacija [43], [45]), taip pat ką daryti, kai kiekvienas dalyvis aplankomas (vadinama adaptyviuoju lankytoju [1], [27], [9], [74]). Navigacijos specifikacijos kartu su klasėmis lankytojomis apibrėžia konkretų klasių rinkinių funkcionalumą. Visos taikomosios programos funkcionalumas yra aprašomas navigacijos specifikacijų aibe ir klasių lankytojų aibe. Šio derinio naudojimo pranašumas, palyginti su įprastiniu Java kodu, yra jo adaptyvumas. Navigacijos specifikacijos su lankytojais padeda atskirti klasių struktūrą nuo programos funkcionalumo, nes neįtraukia klasių struktūros detalių į programą. Dėl to programos funkcionalumas mažiau kinta, kai pakeičiama klasių struktūra. Be to, programos, parašytos su navigacijomis ir lankytojais, yra glaustesnės, nes trivialus struktūrų perėjimo kodas neturi būti sukuriamas, o jo generavimas paliekamas automatiniams įrankiams [29].

Pasak P. Kroho, objektinė technologija, uždarydama duomenis ir funkcionalumą klasėse, apsaugo realizaciją nuo kintančių duomenų struktūrų, bet visi pasikeitimai paveikia sąsają [29]. Adaptyvusis programavimas suteikia galimybę programoms turėti klasių hierarchijos sąsają, todėl šios programos nėra jautrios esamos klasių hierarchijos pasikeitimams. Dėl to programos tampa lengviau pritaikomos prie pasikeitimų ir evoliucijos [31].

Apibendrinant galima teigti, kad K. J. Lieberherrio, J. Palsbergo, P. Kroho ir kitų mokslininkų darbuose adaptyviosios programos apibrėžiamos kaip išplėstinės objektinės programos, kuriose adaptyvusis programavimas perkelia navigacijos per daugelio skirtingų klasių objektų susietą struktūros naštą nuo programuotojo kompiliatoriui. Pagrindinis tikslas – nurodyti tik navigacijos orientyrus ir jų vietoje atliekamus veiksmus, o kompiliatoriui palikti perėjimo kodo generavimą, kad būtų surastos „orientyrinės“ klasės ir atliekami veiksmai [71]. Ši abstrakcija skatina pakartotinai panaudoti programas, kadangi ta pati nekeista adaptyvioji programa tinka spręsti daugeliui panašių problemų. Pavyzdžiui, imkime adaptyviąją programą *Vidurkis*, kuri aplanko klasės *Elementas* objektus ir apskaičiuoja juose esančio lauko *kiekis* vidurkį. Ši programa gali būti sukompiliuojama dėl įmonės, paverčiant *Elementas* į *Darbuotojas*, o *kiekis* į *alga*, kad apskaičiuotų darbuotojų atlyginimų vidurkį. Tačiau ši programa taip pat gali būti sukompiliuojama paverčiant *Elementas* į *Prekė*, o *kiekis* į *kaina*. Šis atvejis apskaičiuoja visų sandėlyje esančių prekių vidutinę kainą [71].

Adaptyvioji programa susideda iš dviejų dalių: navigacijos specifikacijos ir apvalkalo (veiksmo) specifikacijos. Navigacijos specifikacija nurodo klases, kurių objektai privalo (arba ne) būti aplankomi tam tikra tvarka, ir objektų kintamuosius, per kuriuos privaloma (arba ne) pereiti. Apvalkalo specifikacija jungia klases su veiksmais, kuriuos reikia atlikti, kai perėjimo metu pirmą kartą prieinamas tos klasės objektas arba kai iš jo išeinama.

Nors navigacijos specifikacija tik nurodo klasių ir objektų kintamųjų, reikiamų programavimo uždaviniui, pavadinimus, faktinė klasių struktūra, kuriai kompiliuojama adaptyvioji programa, gali turėti tarpines klases ir papildomus objektų kintamuosius. Kompiliatorius automatiškai sugeneruoja visą kodą, naudojamą navigacijai, arba ignoruoja šiuos objektus. Panašiu būdu apvalkalo specifikacijų reikia tik toms klasėms, kurių objektus reikia unikaliai apdoroti [71]. Taigi programuotojas rašo svarbias programos dalis, o kompiliatorius užpildo bendrąjį likutį.

2.3. Antrojo skyriaus apibendrinimas

Šioje dalyje išanalizuotų adaptyviųjų metodų kontekste adaptyvumas pasiekiamas kuriant programas kaip laisvai susietus fragmentus, kurių visuma leidžia išspręsti taikomojo pobūdžio uždavinius. Laisvas susiejimas pasiekiamas apibrėžiant objektų navigacijos specifikacijas. Laisvas fragmentų susiejimas lemia adaptyvumą, nes vieno fragmento pokyčiai nepakeičia kitų besikooperuojančių fragmentų paskirties.

Apibendrinant analitinę dalį galima teigti, kad mokslininkų tyrimai ir taip gana lanksčias objektines programas padaryti plačiau pritaikomomis skirtingų taikomojo pobūdžio uždaviniams spręsti, praplečiant jas adaptyviuoju programavimu, įrodo, kad adaptyvumo problema egzistuoja. Kadangi programinių priemonių nepastovumą lemia pirminių duomenų struktūrų, jų turinio ir sprendžiamų taikomojo pobūdžio uždavinių algoritmų pokyčiai, tuo remiantis tikslinga išsikelti šį pagrindinį uždavinį: taikomojo pobūdžio uždavinių sprendimus realizuoti taip, kad uždavinius būtų galima spręsti transformuojant duomenis į pirminius konkretaus uždavinio duomenis ir sudarant tų duomenų apdorojimo programą atrenkant reikiamus modulius iš jau turimos programinių modulių aibės.

3

Duomenų išrinkimo ir agregavimo modeliai

3.1. Adaptyviojo duomenų apdorojimo projektavimo bendrieji bruožai

Šioje disertacijoje, skirtingai negu K. J. Lieberherrio, J. Palsbergo, P. Kroho ir kitų mokslininkų darbuose, adaptyvumo samprata nėra tiesiogiai siejama su objektinėmis programomis. Kuriami adaptyvieji duomenų modeliai skirti apdoroti duomenų struktūroms, kurios išreikštos RA. Kadangi reliacinės aibės, sudarytos iš duomenų, yra žinomas ir plačiai taikomas pateikimo būdas, todėl adaptyvioji duomenų apdorojimo projektavimo technologija, sudaryta iš adaptyviųjų duomenų modelių, gali būti plačiai pritaikoma praktikoje.

Šiame darbe adaptyvieji duomenų, kurie yra išreikšti RA, modeliai yra suskirstyti į:

- duomenų išrinkimo modelį, kurio pagrindą sudaro RA identifikatoriai;
- duomenų agregavimo modelį, kuris grindžiamas transformacijomis;
- duomenų apdorojimo projektavimo modelį, kurį sudaro algoritminės duomenų priklausomybės ir jas realizuojantys programiniai moduliai.

Duomenų agregavimo modelį sudarančios transformacijos yra pagrįstos analogiška idėja kaip ir transformacijos, kurios apibrėžtos klasikinio struktūrinio projektavimo metodikoje [13], [26]. Šioje metodikoje numatytos dvi pagrindinės

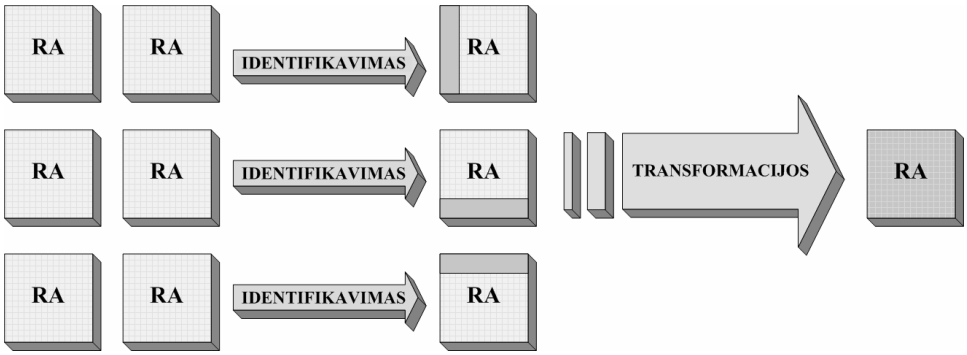
posistemių projektavimo strategijos. Viena iš jų yra transformacijų analizė. Ši strategija naudojama tuomet, kai turime nuoseklius procesus aprašančias duomenų srautų diagramas. Transformacijų analizės strategiją sudaro penkios projektavimo procedūros, kurių viena yra centrinių transformacijų nustatymas. Šios transformacijos atlieka skaičiavimus ir pertvarko pačius abstrakčiausius pradinius duomenis į pačius abstrakčiausius rezultatus [13]. Analogiškus skaičiavimo veiksmus atlieka ir šioje disertacijoje apibrėžtos transformacijos, tačiau jos metodiniu požiūriu nėra taip plačiai išplėtotos kaip tai atlikta struktūrinio projektavimo plotmėje. Nepaisant to, jos atlieka šio tyrimo kontekste apibrėžtas funkcijas. Šioje disertacijoje aprašomos transformacijos yra skirstomos į tipus ir rūšis. Tai galima laikyti transformacijų standartizavimu. Kadangi struktūrinio projektavimo metodikoje tai nėra apibrėžta, vadinasi, tai yra mokslinis šio darbo naujumas.

Apibrėžti duomenų modeliai yra adaptyvūs, kadangi jie gali būti automatizuotai arba automatiškai pritaikomi konkrečiam taikomajam uždaviniui spręsti. Adaptyviųjų duomenų modelių nepastovumą lemia pirminių duomenų turinio, jų struktūrų ir sprendžiamų taikomojo pobūdžio uždavinių algoritmų pokyčiai.

Naudojant adaptyviuosius duomenų modelius projektuojamoms programoms taikomas struktūrizavimo principas [13]. Kitaip tariant, projektuojama programa skaidoma į modulius, kurie vienas nuo kito yra nepriklausomi ir į visumą gali būti jungiami įvairiais deriniais.

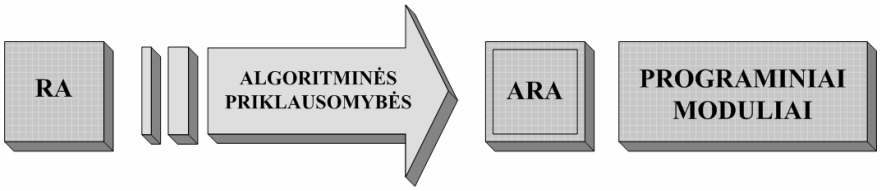
Adaptyvioji duomenų apdorojimo projektavimo technologija taikomojo pobūdžio uždavinių sprendimus leidžia realizuoti trimis etapais. Pirmą etapą sudaro duomenų išrinkimo ir agregavimo modeliai, t. y. duomenų identifikavimas ir transformacijos į taikomojo uždavinio pirminių duomenų reliacinę aibę, antrą – duomenų apdorojimo projektavimo modelis, t. y. algoritminių priklausomybių ir jas realizuojančių programinių modulių atrinkimas, siekiant sudaryti konkretaus taikomojo uždavinio sprendimo programą. Trečiu etapu apdorojami duomenys ir gaunami konkretaus taikomojo uždavinio sprendimo rezultatai.

Pirmu etapu (3.1 pav.) iš visumos pirminių duomenų, kurie pateikti reliacinėmis aibėmis, Taikydami identifikavimo metodus (žr. 3.3 poskyrį) atliekame duomenų atranką, T. y. atrenkame konkrečiam uždaviniui spręsti reikalingus duomenis ir pateikiame juos reikiama tvarka. Toliau, realizuodami duomenų transformacijas, atrinktus pirminius duomenis perkeliame į vieną bendrą RA, kuri gali būti sudaryta ir iš rezultatinių duomenų, nes atlikdami transformacijas galime realizuoti tiek aritmetinius, tiek loginius veiksmus. Taip pat transformacijomis galime keisti pirminių duomenų RA struktūrą ir turinį [76A], [82A]. Taigi identifikavę ir atlikę transformacijas gauname konkretaus taikomojo uždavinio duomenų RA (3.1 pav.).



3.1 pav. *Taikomojo pobūdžio uždavinių sprendimų realizacijos I etapas*

Antru etapu (3.2 pav.) turėdami vieną RA, sudarytą iš duomenų, reikalingų konkrečiam uždaviniui spręsti, šį uždavinį sprendžiame kaip tam tikrą algoritminių priklausomybių tarp duomenų realizaciją, kadangi uždavinio sprendimo algoritmą galime traktuoti, kaip algoritminių priklausomybių seką. Kad būtų galima realizuoti šias algoritmines priklausomybes, kuriami programiniai moduliai, kurių kodai sudaro algoritminių priklausomybių RA. Algoritminės priklausomybės ir jas realizuojantys programiniai moduliai fiziškai suderinami vienas su kitu bet kokiomis kombinacijomis, o logišką jų išdėstymo seką turi nustatyti projektuotojas pagal uždavinio konkretų algoritmą [76A], [82A].



3.2 pav. *Taikomojo pobūdžio uždavinių sprendimų realizacijos II etapas*

Trečiu etapu, realizavus konkrečiam taikomajam uždaviniui spręsti iš programinių modulių sudarytą programą, atliekamas duomenų, kurie yra vienoje bendroje RA, apdorojimas. Apdoroti duomenys pateikiami reliacinėmis aibėmis, kurios vadinamos rezultatinėmis RA (RRA) (3.3 pav.).

3.3 pav. *Taikomojo pobūdžio uždavinių sprendimų realizacijos III etapas*

Taigi algoritminės duomenų priklausomybės realizuojamos programiniais moduliais, kurie vienas kito atžvilgiu gali būti vykdomi neprieštaringai vienas kitam, t. y. nepanaikina vienas kito pirminių duomenų bei rezultatų, ir gali naudotis vienas kito sprendimo rezultatais. Tokiu atveju naudodami šiuos modulius, pagal jų kodų išdėstymo tvarką, galime vykdyti ir spręsti tam tikrą uždavinį. Jeigu reikia spręsti kitą uždavinį, tada iš visų programinių modulių išrenkami iš dalies kitokie programiniai moduliai ir kitokia tvarka, ir tada sprendžiamas kitas uždavinys. Taigi visa algoritmų ir juos realizuojančių programinių modulių sistema gali būti pritaikyta įvairiems taikomojo pobūdžio uždaviniams spręsti ir skirtingiems uždaviniams spręsti nereikia kurti naujų programinių priemonių. Jeigu atsiranda uždavinys, kurio negalima išspręsti dėl algoritmines priklausomybes realizuojančių programinių modulio trūkumo, tai tų modulių sistema papildoma elementariais algoritmines priklausomybes realizuojančiais programiniais moduliais, kad sistema apimtų ir naujojo uždavinio sprendimą [77A].

Sprendžiant skirtingus uždavinius naujų programų neatsiranda, kadangi iš algoritmines priklausomybes realizuojančių programinių modulių aibės yra atrenkami moduliai konkrečiam uždaviniui spręsti. Taikant sukurtą technologiją taikomojo uždavinio programa sudaroma kaip visų algoritminių priklausomybių realizacijos modulių aibės poaibis. Tai leidžia ta pačia algoritminių priklausomybių programinių modulių aibe spręsti įvairius uždavinius nekuriant jiems naujų programų [76A], [80A]. Šis metodas nepainiotinas su vieno, kad ir sudėtingo, uždavinio sprendimu, kai jam gali būti pateikiami skirtingi duomenys savo turiniu ir praktiškai mažai besiskiriantys pirminių duomenų ir sprendimo duomenų struktūra. Buvo kalbama apie skirtingų taikomojo pobūdžio uždavinių sprendimą naudojant skirtingus algoritminių priklausomybių ir jas realizuojančių programinių modulių poaibius.

Itin svarbus ir įsimintinas šios technologijos ribojimas yra tai, kad algoritmines priklausomybes realizuojantys programiniai moduliai adaptyviojoje duomenų apdorojimo projektavimo technologijoje gali būti sukurti tik tuo atveju, kai pirminiai uždavinių duomenys ir sprendimo rezultatai pateikiami 3.1.1 poskyryje apibrėžtomis reliacinėmis aibėmis.

Šios adaptyviosios technologijos gyvybingumui priskirtina tai, kad minėtomis RA galima aprašyti praktiškai visus ūkinius, buhalterinius, valdymo ir kitus duomenis.

3.1.1. Reliacinės aibės ir jų elementai

Adaptyviosios duomenų apdorojimo projektavimo technologijos pagrindinė duomenų struktūra yra duomenų reliacinė aibė *r*:

$$r = \begin{pmatrix} A_1 & A_2 & \dots & A_j & \dots & A_n \\ c_{11} & c_{12} & \dots & c_{1j} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2j} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ c_{i1} & c_{i2} & \dots & c_{ij} & \dots & c_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mj} & \dots & c_{mn} \end{pmatrix}. \quad (3.1)$$

Aibės r elementai A_j (čia $1 \leq j \leq n$) vadinami šios aibės atributais. Aibių elementai c_{ij} vadinami atributų reikšmės. Visi šie elementai yra tarp savęs susiję, nes yra tų pačių atributų reikšmėmis. Taigi reliacine aibe r vadinama duomenų aibė, kuri išreiškia atitiktį tarp atributų, pateiktų viename ir tame pačiame domene

A_1	A_2	...	A_j	...	A_n

,

ir jų reikšmių

c_{11}	c_{12}	...	c_{1j}	...	c_{1n}
c_{21}	c_{22}	...	c_{2j}	...	c_{2n}
...	...	...	...	...	...
c_{i1}	c_{i2}	...	c_{ik}	...	c_{in}
...	...	...	...	...	...
c_{m1}	c_{m2}	...	c_{mj}	...	c_{mn}

.

RA r schema yra jos atributai:

$$R = R(A_j) = R(A_1, A_2, ..., A_j, ... A_n). \quad (3.2)$$

RA r atributų reikšmių aibė $< c_{ij} >$ vadinama RA r domenu, kai j fiksuotas:

$$D = \text{dom}(r) = < c_{ij} >. \quad (3.3)$$

RA domeno A_b poaibis $< c_{ij} >$, čia $j = b$, vadinamas atributo A_b domenu:

$$\text{Dom}(A_b) = c_{1b}, c_{2b}, ..., c_{ib}, ..., c_{mb}. \quad (3.4)$$

RA domeno $< c_{ij} >$ poaibis, sudarytas iš atributų reikšmių, čia i fiksuotas ir $i = a$, vadinamas atributų reikšmių a kortežu:

$$t_a = t(A_a) = c_{a1}, c_{a2}, ..., c_{aj}, ..., c_{an}. \quad (3.5)$$

RA r tam tikras schemos R poaibis vadinamas raktiniu atributu, čia $K \in R$:

$$K = \{A_y\}, y \in j. \quad (3.6)$$

Tada atributų K reikšmės $\{c_{ij}\} \in \{c_{ij}\}$ yra kortežų raktai, t. y. jų identifikatoriai.

Skirtingos RA žymimos

$$r^1, r^2, \ldots, r^d.$$

Skirtingos RA savo atributų reikšmėmis $\{c_{ij}\}$, bet turinčios tą pačią atributų schemą, žymimos

$$r_1, r_2, \ldots, r_t .$$

Apdorojant RA į duomenis galima kreiptis įvairiai:

- į atskirą kortežą;
- į domeno atributo reikšmes;
- į atskirą reikšmę, naudojant jos koordinates aibės lentelėje i/j.

3.1.2. Adaptyviosios technologijos kūrimo ir naudojimo etapai

Kuriant disertacijoje aprašytą adaptyviąją duomenų apdorojimo projektavimo technologiją, apibrėžti technologijos kūrimo ir naudojimo etapai, kurie pateikiami šiame poskyryje.

I. *Reliacinės aibės*

Duomenys, kurie apdorojami taikant adaptyviąją duomenų apdorojimo technologiją, turi būti išreikšti reliacinėmis aibėmis. Tai plačiai naudojama duomenų raiškos forma, nes RA gali būti lengvai pateikta naudotojui kaip paprasta ir akivaizdi duomenų lentelė. Taip pat tai pačiai reliacinei aibei analizuoti ir apdoroti tinka dalis gerai išplėtoto aibių teorijos mechanizmo.

Duomenų reliacinė aibė yra

$$r = \left[\frac{A_j}{c_{ij}} \right], \tag{3.7}$$

čia: A_j – RA atributai, sudarantys jos schemą; c_{ij} – atributų reikšmės; n – RA rangas, o m – RA eilė, kai $1 \le i \le m$, $1 \le j \le n$. Visos atributų reikšmės, turinčios fiksuotą (tą patį) i, vadinamos kortežu, o turinčios tą patį j – atributo A_j domenu [17]. Reliacinių aibių kortežams galima priskirti raktus. Raktu vadinamas atributų aibės A_j poaibis K, kuris užtikrina vienareikšmišką identifikaciją. Tai leidžia lengviau rasti reikiamus kortežus.

Naudojamos duomenų reliacinės aibės turi būti antros normalinės formos (toliau – NF) [14], [22], [72], [75]. Suprantama, kad gali būti ir trečios NF, bet sukurtai technologijai pakanka duomenų, kurie apibrėžiami antrąja normaline forma.

Toliau pateiktuose punktuose aprašyti duomenų identifikavimo ir transformacijų metodai yra lankstūs. Lankstumas reiškia galimybę performuoti duomenis iš bet kokios žinomos ir praktikoje naudojamos duomenų bazės (toliau – DB) schemos į RA ir atvirkščiai – reliacine aibe išreikštus duomenis į konkrečios DB naudojamą duomenų schemą. Kitaip tariant, adaptyviosios

duomenų apdorojimo projektavimo technologijos apibrėžtoje algoritminių priklausomybių aibėje sukūrus tam tikrus algoritminių priklausomybių modulius, atsiranda galimybė sukurti reliacines aibes iš bet kokios struktūros duomenų. Plačiau apie tai yra aprašyta ketvirtame šios disertacijos skyriuje, kuris skirtas algoritminėms duomenų priklausomybėms.

II. *Duomenų išrinkimo modelis*

Šį modelį sudaro RA identifikavimo metodai ir mechanizmas, kuris padeda iškviesti reikiamas RA ir pateikti duomenis tam tikra tvarka ir tam tikrą jų skaičių.

Kad būtų galima iš pirminių duomenų šaltinio išrinkti reliacines duomenų aibes, reikalingas konkrečiam taikomajam uždaviniui spręsti, yra apibrėžti duomenų identifikavimo metodai, kurie detaliai aprašyti 3.3 poskyryje. Duomenų struktūrų identifikatoriai turi analogišką schemą, kaip ir pačios duomenų reliacinės aibės. Daugeliu atvejų, trijų identifikatorių pakanka duomenims RA identifikuoti, nors iš identifikatorių analizės bus lengva pastebėti, kad ir kitoks identifikatorių skaičius jų formalių išraiškų esmės nepakeičia.

Identifikuojant RA, atributas A reiškia konkrečios RA identifikatorių a_x , čia x – konkrečių schemų kodai, atributo B reikšmės yra b_y , čia y – subjekto, kuriam priklauso duomenys, kodas ir D – laiko faktoriaus atributo d_z kodas z. Taigi bet kokia duomenų RA yra identifikuojama identifikatorių reliacine aibe, kur $a_x\, b_y$ ir d_z , yra atributų A, B ir D reikšmės.

Pateikiant glaustą identifikatorių formą, galima a_x b_y ir d_z įvairiai keisti vietomis:

$$a_x\, b_y\, d_z\,,\ a_x\, d_z\, b_y\,,\ b_y\, a_x\, d_z\,,\ b_y\, d_z\, a_x\,,\ d_z\, a_x\, b_y,\ d_z\, b_y\, a_x.$$

Taip pat galima pateikti indeksų identifikatorius dalimis ir nebūtinai iš eilės, pvz.:

$$a_{1\text{-}100} = a_{1\text{-}20}\,,\ a_{21\text{-}56}\,,\ a_{84\text{-}100}\,,\ a_{79\text{-}83}\,,\ a_{57\text{-}78}.$$

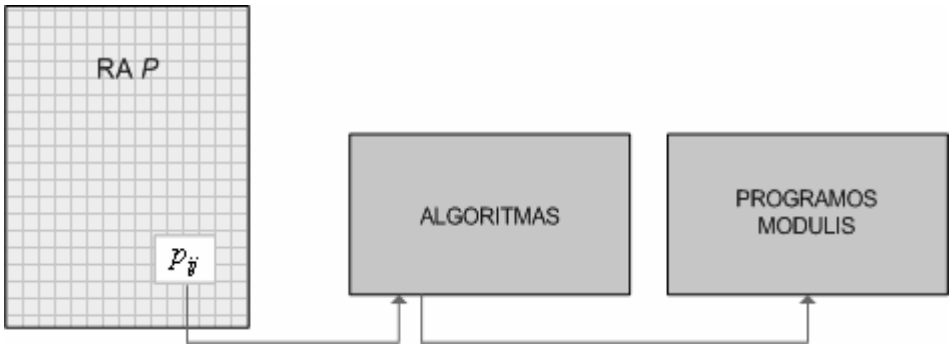
Taip, galima užtikrinti, kad konkrečiam uždaviniui spręsti būtų pateiktas bet koks reikiamas reliacinių aibių kiekis ir bet kokia tvarka.

Gali būti, kad tarp turimų duomenų bus dvi ar kelios aibės, turinčios tokius pačius identifikatorius, ir šių aibių visos atributų reikšmės vienodos. Tuomet turime nuspręsti, ar tai klaida, ar tai duomenų perteklius. Tačiau gali būti ir kitoks atvejis, kai yra dvi ar kelios vienodai identifikuotos aibės, kurių viena ar kelios atributų reikšmės skiriasi. Tada naudotojas turi žiūrėti, ar čia nėra padaryta klaida. Jeigu klaida, tai aibę su klaidingomis atributų reikšmėmis reikia pašalinti. Kitu atveju tai gali būti probleminės srities ypatybė. Tada (ir tik tada) greta tradicinių identifikatorių a b d yra sukuriamas papildomas identifikatorius, kuriame nurodomos vienos arba kelių atributų reikšmių koordinatės iš identifikuotos RA. Tai leidžia papildomai identifikuoti aibę, nes pagal tai galima iškviesti arba nukopijuoti identifikatoriuje pateiktose koordinatėse saugomas reikšmes (detaliau apie tai aprašyta 3.3 poskyryje).

III. *Duomenų apdorojimo projektavimo modelis*

Bazinėje algoritminių priklausomybių kodų RA (*P*) kiekvienas kodas turi savo algoritmą, o kiekvienas algoritmas – jį realizuojantį programinį modulį (3.4. pav.).

Algoritminės priklausomybės (toliau – AP) nurodo, kaip galima apdoroti RA pateiktus duomenis, nes kiekviena algoritminė priklausomybė turi savo realizavimo algoritmą ir programinį to algoritmo realizavimo modulį (3.4 pav.).



3.4 pav. *Algoritminės priklausomybės struktūra*

Algoritminės priklausomybės skirstomos į grupes (žr. 4.1 poskyryje). Visų AP grupių aibė ir algoritminių priklausomybių skaičius grupėse yra atviras papildymams. Tai teigiama todėl, kad jei sprendžiamam uždaviniui prireikia naujų algoritminių priklausomybių ir jas realizuojančių modulių, nes turima jų visuma negali atlikti tam tikrų uždaviniui spręsti reikalingų veiksmų, tada sukuriama nauja algoritminė priklausomybė ir ją realizuojantis modulis. Jie įtraukiami į jau turimas algoritminių priklausomybių ir programinių modulių aibes. Taip nauja algoritminė priklausomybė kartu su kitomis padeda realizuoti naujojo uždavinio sprendimo algoritmą.

Sudaroma tokia AP kodų struktūrą, kuri adekvati reliacinių aibių struktūroms:

$$r = \begin{pmatrix} A_1 & A_2 & \dots & A_j & \dots & A_n \\ c_{11} & c_{12} & \dots & c_{1j} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2j} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ c_{i1} & c_{i2} & \dots & c_{ij} & \dots & c_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mj} & \dots & c_{mn} \end{pmatrix}, \qquad (3.8)$$

$$P = \begin{pmatrix} <p_{11}><p_{12}> & \dots & <p_{1j}> & \dots & <p_{1n}> \\ <p_{21}><p_{22}> & \dots & <p_{2j}> & \dots & <p_{2n}> \\ \dots\dots & \dots & \dots & \dots & \dots \\ <p_{i1}><p_{i2}> & \dots. & <p_{ij}> & \dots. & <p_{in}> \\ \dots\dots & \dots & \dots & \dots & \dots \\ <p_{m1}><p_{m2}> & \dots & <p_{mj}> & \dots & <p_{mn}> \end{pmatrix}, \qquad (3.9)$$

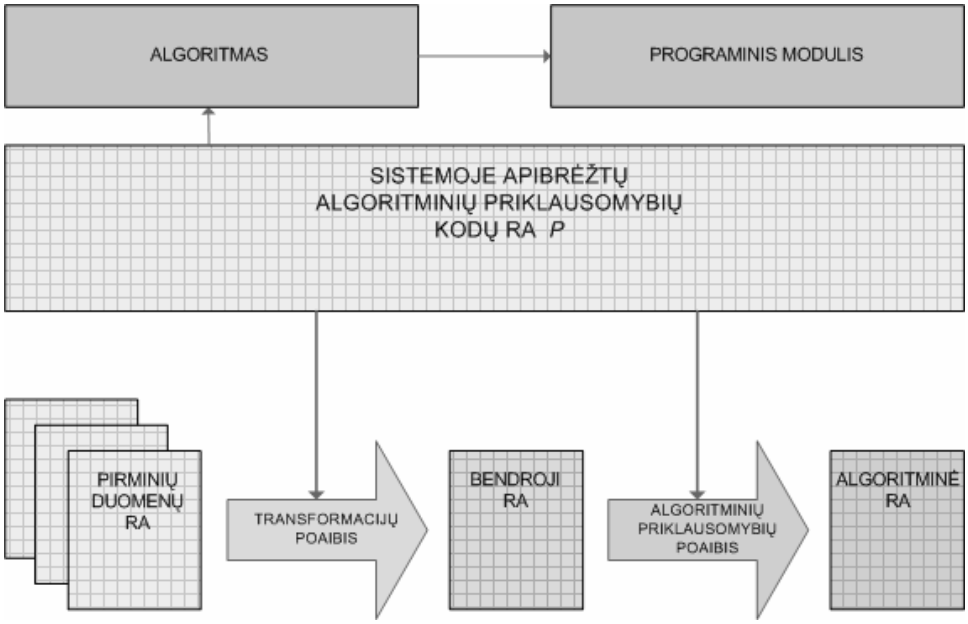
čia r – yra duomenų reliacinė aibė; P – algoritminių priklausomybių kodų RA. Reliacinėje aibėje P atributų pavadinimai nerašomi, nes jie visi yra vienodi, t. y. atributų reikšmių algoritminės priklausomybės.

Reliacinės aibės, sudarytos iš duomenų, yra žinomas ir plačiai naudojamas (ypač sudarant reliacines duomenų bazes) duomenų modelis. Tačiau RA, sudaryta iš algoritminių priklausomybių kodų, yra viena iš šiame darbe siūlomų teorinių naujovių, nes nepavyko aptikti jokioje literatūroje tokio sprendimo. Reikia pabrėžti, kad ši naujovė turi teigiamų ypatumų – taikant tą pačią adaptyviąją duomenų apdorojimo projektavimo technologiją galima apdoroti įvairius, reliacinėmis aibėmis išreikštus, duomenis.

Taigi norint spręsti uždavinį, reikia apibrėžti algoritmines priklausomybes tarp atributų reikšmių, esančių toje pačioje RA ar įvairiose RA, kai apibrėžiamos algoritminės transformacijų priklausomybės. Norint suformuoti taikomojo uždavinio algoritmą, reikia sukurti RA ne iš pirminių duomenų, o iš algoritminių priklausomybių kodų RA (P). Kitaip tariant, iš jau tuo metu turimos sistemos visų algoritminių priklausomybių kodų RA P atrenkamos reikiamos AP. Taip sudaroma algoritminė RA (toliau – ARA) konkrečiam uždaviniui spręsti, kuri yra P poaibis. ARA vienareikšmiškai lemia atributų reikšmių dalyvavimą tam tikroje duomenų apdorojimo operacijoje, nes kreipiantis į norimą kodą galima iškviesti tuo kodu apibrėžtą algoritminę priklausomybę ir ją realizuojantį programinį modulį (3.5 pav.). Vėliau programiškai einant ARA tam tikra tvarka ir kryptimi, pagal algoritminių priklausomybių kodus iškviečiami programos moduliai ir taip sprendžiamas uždavinys. Kiekvienam taikomajam uždaviniui spręsti parenkamos vis skirtingos ARA ir skirtinga tvarka bei būdai, kaip eiti konkrečia ARA sprendžiant tą uždavinį.

Praktikoje gali pasitaikyti, kad viena ARA bus sunku realizuoti sudėtingą uždavinį. Todėl pastaruoju atveju uždaviniui spręsti siūloma naudoti ARA eilutę, kurioje gretimas ARA eilutėje jungia savitos algoritminės priklausomybės, pavadintos išorinėmis algoritminėmis priklausomybėmis (ψ).

ARA korektiškumą patikrinti labai sudėtinga. Pirmiausia ARA turinį nusako nagrinėjamas taikomojo uždavinio algoritmas, o taikomojo uždavinio algoritmą diktuoja probleminė sritis. Probleminę sritį galima nustatyti tik išanalizavus dalykinę sritį.



3.5 pav. *Algoritminės RA sudarymo schema*

IV. *Duomenų agregavimo modelis*

Perkėlos iš iškviestųjų RA atributų reikšmių į vieną RA mechanizmas pavadintas transformacijomis, o sukurtoji RA – pirminiais taikomojo uždavinio sprendimo duomenimis.

Taikant sukurtą duomenų išrinkimo modelį galima atrinkti konkrečiam uždaviniui spręsti reikiamas reliacines aibes ir pateikti jas reikiama tvarka. Toliau sukuriamas ir pateikiamas duomenų agregavimo modelis (t. y. duomenų transformacijos), kuriuo galima perkelti reliacinių aibių reikiamas atributų reikšmes į vieną bendrą RA. Ši bendroji RA, t. y. dvimatė lentelė, gali neturėti visų RA būdingų ypatumų. Pavyzdžiui, gali neturėti atributų, o tik jų reikšmes. Atrinktą reikšmę dvimatėje lentelėje gali visiškai identifikuoti tos lentelės konkrečios reikšmės koordinatės. Tik duomenis apdorojančio uždavinio algoritmas turi „žinoti“, kokiose lentelės koordinatėse koks duomuo (atributo reikšmė) yra.

Reikia pabrėžti, kad programiniai moduliai naudojasi koordinatėmis. Daugkartinis to paties modulio naudojimas dar nereiškia, kad jis kiekvieną kartą naudos vis tas pačias koordinates iš pirminės dvimatės lentelės. Tas pats principas galioja tiek transformacijoms, tiek visiems kitiems programų moduliams. Diskretiesiems programų moduliams vykdyti būtina nurodyti tik tam vykdymui reikalingas duomenų koordinates iš pirminės konkretaus uždavinio lentelės, suformuotos RA identifikavimo ir transformacijų būdu.

Bendrąja RA gali sudaryti ir iš dalies apdoroti pirminiai duomenys, nes atliekant transformacijas galima realizuoti tiek aritmetinius, tiek loginius veiksmus. Taip pat pasitelkus transformacijas galima keisti pirminių duomenų RA struktūrą ir turinį.

V. *RA išsamumo ir korektiškumo tikrinimas*

Pateikus uždaviniui spręsti reikalingų duomenų reliacinių aibių identifikatorius, galima iškviesti reikiamus duomenis iš pirminių duomenų šaltinio. Taigi akivaizdu, kad duomenų šaltinyje neradus RA, kurios identifikatoriai yra pateiktoje konkretaus uždavinio identifikatorių sekoje, yra nustatoma, kokių reliacinių aibių uždaviniui spręsti trūksta. Kitaip tariant, nustatoma, kad pirminiai duomenys yra neišsamūs.

Tikrinant duomenų išsamumą kiekvienoje iš spręsti pateiktų RA, nustatoma, ar visi uždaviniui reikalingi kortežai yra kiekvienoje pateiktoje RA, kadangi prie RA schemos yra fiksuota tvarka, nurodytas ir raktinis atributas *K*, ir to atributo reikšmės:

$$c'_{1j}, c'_{2j}, c'_{3j}, \ldots, c'_{mj}.$$

Taigi atributų reikšmių transformacijos metu nepaliečiami tie kortežai, kurių konkrečiam uždaviniui spręsti nereikia, bet jeigu RA nerandamas kortežas iš sekos, tada nesunkiai nustatoma, kad duomenų trūksta ir tiksliai kokių kortežų konkrečioje RA nepateikta. Taip pat raktinių atributų priskyrimas garantuoja vienareikšmišką identifikaciją ir pertekliaus nebuvimą.

Duomenų korektiškumą galima užtikrinti atributų reikšmėms priskiriant savąsias algoritmines atributų reikšmių priklausomybes. Pavyzdžiui, ta pati reikšmė *c* negali vienu ir tuo pačiu metu apvalinama kaip skaičius ir būti tekstine reikšme. Deja, projektuotojas negali užtikrinti visų duomenų korektiškumo, nes kai kurių duomenų korektiškumą gali apibrėžti tik naudotojas. Naudotojui apibrėžus norimas sąlygas, projektuotojas turi formaliai, o vėliau ir programiškai reaguoti į tokį apibrėžimą.

VI. *Mechanizmas, kuriuo galima plėsti algoritminę reliacinę aibę ir algoritminių priklausomybių kodų RA, jeigu jos nepakanka konkretiems uždaviniams spręsti*

Kadangi algoritminių priklausomybių grupių aibė ir jų skaičius grupėse yra atviras papildymams, tai pagal poreikius galima lengvai papildyti reikalingomis AP. Tarkime, kad su turima algoritminių priklausomybių kodų RA negalima išspręsti norimo uždavinio, tada sukuriama nauja algoritminė priklausomybė ir ją realizuojantis modulis, kurie įtraukiami į jau turimas algoritminių priklausomybių ir programinių modulių aibes. Tada ši algoritminė priklausomybė kartu su kitomis apims ir naujo uždavinio sprendimo algoritmo realizavimą. Taip pat atsiradusi nauja algoritminė priklausomybė, naudojama kartu su kitomis AP, gali pateikti iš anksto nenumatytų efektų.

Trumpai apibrėžus adaptyviosios duomenų apdorojimo projektavimo technologijos kūrimo etapus, toliau pateikiama technologijos naudojimo, sprendžiant taikomojo pobūdžio uždavinius, seka.

I. *Iškviečiamos RA, kurių reikės konkretaus uždavinio duomenims apdoroti*

Pirmiausia būtina nustatyti, kokių duomenų reikia pasirinktam uždaviniui spręsti ir kokiose RA tie duomenys yra pateikti. Tada iš pirminių duomenų visumos pateikus reikiamus identifikatorius išrenkamos reliacinės aibės, kurios reikalingos sprendžiamam uždaviniui. Galimybė pateikti identifikatorius norima tvarka leidžia išrinkti ir pateikti duomenis uždaviniui spręsti reikiama seka, o identifikatorių indeksai užtikrina, kad bus išrinktas reikiamas duomenų skaičius.

II. *Iš atrinktųjų RA duomenys transformuojami į vieną bendrą RA*

Realizuojant duomenų transformacijas, atrinktų atributų reikšmės perkeliamos į vieną bendrą RA. Taip pat transformacijos leidžia pakeisti pirminių duomenų reliacinių aibių struktūras ir turinį, nes atliekant transformacijas galima realizuoti tiek aritmetinius, tiek loginius veiksmus. Aritmetinius veiksmus galima atlikti realizuojant aritmetines transformacijas, o loginius – sąlygines. Detaliau apie tai aprašyta 3.5 poskyryje.

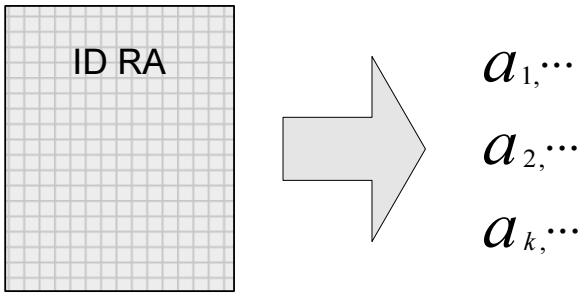
III. *Sudaroma uždavinio sprendimo ARA*

Suformavus vieną RA, skirtą konkrečiam uždaviniui spręsti, ir tarp atributų apibrėžus šio uždavinio algoritmines priklausomybes, yra sudaroma ARA arba ARA eilutė, sujungta išorinėmis algoritminėmis priklausomybėmis ψ (detaliau apie tai – 4.5 poskyryje). Realizavus ARA arba ARA eilutėje pateiktų algoritminių priklausomybių programinius modulius gaunamas sprendžiamo uždavinio galutinis rezultatas

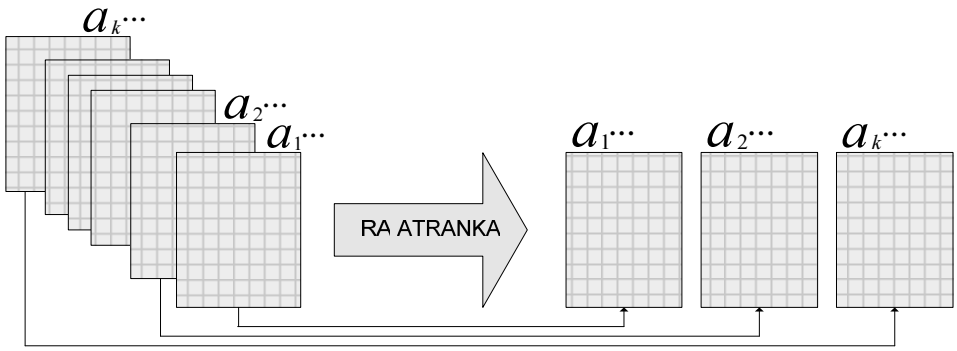
3.2. Duomenų išrinkimas ir agregavimas

Taikant duomenų išrinkimo modelį iš pirminių duomenų visumos galima išrinkti duomenis, reikalingus konkrečiam uždaviniui spręsti, reikiama eile ir reikiamą jų kiekį. Duomenų atranką ir reikiamą jų kiekį užtikrina identifikavimo metodai [4]. Tai atliekama iš identifikatorių (toliau – ID) RA atrinkus tik tuos identifikatorius, kurie prisikirti konkrečiam uždaviniui spręsti reikalingoms duomenų reliacinėms aibėms (3.6 pav.).

Naudojant tuos identifikatorius, iš pirminių duomenų RA visumos atrenkamos norimos RA ir pateikiamos norima tvarka (3.7 pav.). RA pateikimas priklauso nuo identifikatorių išrinkimo sekos, o tai reguliuoja projektuotojas, atsižvelgdamas į sprendžiamojo uždavinio algoritmą, kuris nurodo, kokia tvarka duomenys turi būti pateikti apdoroti.



3.6 pav. *Reikiamų reliacinių aibių identifikatorių išrinkimas*

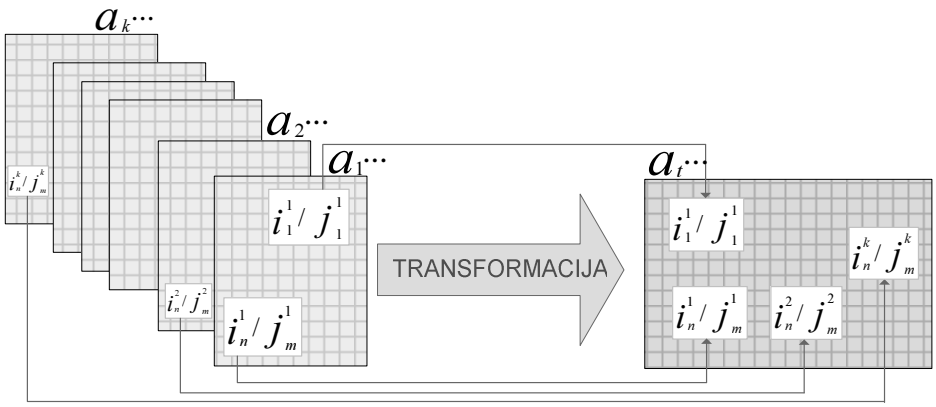


3.7 pav. *Reliacinių aibių atranka iš pirminių duomenų visumos*

Duomenų agregavimo modelis dėl transformacijų leidžia atrinktus duomenis perkelti į vieną bendrą RA. Transformacijos leidžia ne tik iš atrinktų RA perkelti pavienes atributų reikšmės, reikalingas pasirinktam uždaviniui spręsti (3.8 pav.), bet ir atlikti aritmetinius bei loginius veiksmus su norimomis atributų reikšmėmis. Tad galima teigti, kad transformacijos padeda keisti pirminių duomenų RA struktūrą ir turinį.

Sprendžiant kitą taikomojo pobūdžio uždavinį tuo pačiu principu, apdoroti pateikiamos naujai suformuotos RA iš atrinktų duomenų, kurie reikalingi tik šiam uždaviniui spręsti. Kitaip tariant, adaptyvieji duomenų modeliai leidžia pateikti duomenis kaitaliojant jų formaliąsias išraiškas.

Šiame poskyryje pateiktas tik RA atrinkimo principas pagal vieną faktorių, t. y. pagal RA schemos kodą. Realus RA išrinkimo modelis nagrinėjamas kituose šio skyriaus poskyriuose, kur identifikuojant RA dalyvauja ir kiti identifikatoriaus požymiai, tokie kaip: subjekto kodas, nurodantis, kokiam subjektui priklauso RA, ir laiko kodas, nurodantis duomenų galiojimo laiko tarpą arba momentą.



3.8 pav. *Duomenų transformacija*

3.3. Duomenų išrinkimo modelis

Taikomojo pobūdžio uždaviniams spręsti reikia, kad duomenys apdorojimui jiems būtų pateikti tam tikra tvarka arba eile. Tam tikslui apibrėžiama RA identifikatorių aibė, sudaryta tuo pačiu būdu kaip duomenų apie objektą RA. Šis duomenų struktūrų identifikavimas iš duomenų visumos, išreiktos RA, suteikia galimybę surinkti duomenis konkrečiam uždaviniui projektuoti ir spręsti pateikti reikiama eile bei reikiamą jų kiekį [4]. Tai leidžia kontroliuoti duomenų išsamumą, sutrumpinti uždavinio sprendimo trukmę ir sąnaudas, nes bet kokiam taikomajam uždaviniui, kurio pirminiai duomenys ir duomenys po apdorojimo yra RA, tinka tas pats modelis, t. y. analogiškas algoritmas.

Duomenų išsamumo užtikrinimas pasiekiamas trimis būdais, kurių naudojimas uždaviniams yra būtinas, jeigu uždavinio algoritmas apskritai tokio išsamumo reikalauja. Pirmiausia turi būti užtikrinama, kad uždaviniui spręsti būtų pateikiamos visos reikalingos RA. Antra, kad kiekviena iš pateiktųjų RA turėtų uždaviniui spręsti reikalingus visus kortežus. Trečiasis būdas užtikrina, kad kortežuose būtų visi reikiami c_{ij}.

Taigi RA gali būti identifikuotos trijų bendrųjų identifikatorių *ABD* atributų reikšmėmis [4]:

- RA schemos kodu – *a*,
- subjekto kodu – *b*,
- laiko faktoriaus, t. y. laiko momento arba laiko tarpo, kodu – *d*.

RA schemos kodas *a* turi būti visada. Jeigu duomenų lentelei identifikuoti pakanka tik kodo *a*, tai reškia, kad tos lentelės duomenys c_{ij} nepriklauso konkrečiam subjektui ir nepriklauso nuo laiko veiksnio. Tai ši RA yra sudaryta iš įvairių konstantų ir (arba) normatyvinių duomenų, nes normatyviniai duomenys

yra santykiškai pastovūs. Subjekto atributas *B* reiškia duomenų priklausomybę realiam subjektui, kurio duomenys pateikiami duomenų lentelėje. Laiko veiksnys atributas *D* nurodo duomenų kilmės, galiojimo ar panašaus laiko momentą (datą) arba tarpą (nuo – iki). Atributai žymimi didžiosiomis raidėmis, o atributų reikšmės tomis pačiomis mažosiomis raidėmis. Taigi, *a* yra ne atributas, o atributo reikšmė, t. y. pačios konkrečios RA schemos pavadinimas. Analogiški principai taikomi *b* ir *d*.

Tarkime, turime įmonės tam tikrų parduotuvių duomenų lentelę (3.1 lentelė), kurioje pateiktas tam tikrų metų pirmo ketvirčio mėnesiais gautas pelnas:

3.1 lentelė. *Įmonės tam tikrų parduotuvių pirmojo metų ketvirčio pelną pelno duomenys mėnesiais* [80A]

Parduotuvės Nr.	Sausio mėn.	Vasario mėn.	Kovo mėn.
1	13000 Lt	10000 Lt	11000 Lt
2	10000 Lt	11000 Lt	13000 Lt
3	9000 Lt	8000 Lt	10000 Lt

Žinodami, kad uždaviniui spręsti bus reikalingi tam tikri duomenys, esantys RA apie visas įmonės parduotuves, identifikuojame šios lentelės schemą. Taigi lentelės schemos kodu *a* koduojami 3.1 lentelėje surašyti atributai. Kita identifikatoriaus dalimi turėtų būti subjektas, kurio duomenims skirta lentelė. Šiuo atveju tai yra įmonės tam tikros parduotuvių grupės (1, 2, 3) kodas. Ir trečioji identifikatoriaus dalis būtų bendras visų duomenų požymis – laiko tarpas, šiuo atveju – metų pirmasis ketvirtis.

Taigi šios duomenų struktūros identifikavimo modelį sudarys:

1. Identifikatoriaus schema:
   a – pačios lentelės, kaip savarankiško duomenų darinio, kodas;
   B – subjekto atributas;
   D – laiko faktoriaus atributas.
2. Lentelės schema – jos atributų ir jų kodų sąrašas (3.2 lentelė).

3.2 lentelė. *Duomenų lentelės schema*

Parduotuvės Nr.	Sausio mėn.	Vasario mėn.	Kovo mėn.
C_1	C_2	C_3	C_4

Tada lentelės atributai ir atributų reikšmės bus tokie (3.3 lentelė):

3.3 lentelė. *Atributai ir atributų reikšmės*

C_1	C_2	C_3	C_4
c_{11}	c_{12}	c_{13}	c_{14}
c_{21}	c_{22}	c_{23}	c_{24}
c_{31}	c_{32}	c_{33}	c_{34}
---	---	---	---

Lentelės *a* tipo duomenų c_{ij} (i = 1, 2, 3, ..., j = 1, 2, 3, 4) skaičius priklauso nuo to, kelių parduotuvių ir kurių mėnesių duomenys apie pelną bus reikalingi. Neatsižvelgiant į duomenų c_{ij} gausumą, kompiuterio atmintyje lentelės schemą pakanka turėti tik vieną, o duomenis atmintyje laikyti atskirai nuo schemos tik su jų identifikatoriais [80A]. Tada duomenų struktūrai nustatyti turime vieną įrašą:

$$a, B, D; C_1, C_2, C_3, C_4.$$

Atributų reikšmės turės parodytąjį pavidalą, o jų kiekį lemia skirtingų b ir d skaičius:

$$<a, b, d; c_{ij}>.$$

Tokia formalia išraiška galima pateikti bendriausią lentelės sandarą, kuri laikoma kompiuterio atmintyje.

Taigi a, b, d ir c_{ij} yra bendru pavidalu pateiktos atributų $a, B, D, C_1, C_2, C_3, C_4$ reikšmės, skirtingos kiekvienam reikšmių aibės elementui – savarankiškai identifikuotai duomenų lentelei, neturinčiai tiesiogiai priskirtos schemos. Ta schema priskiriama visoms atributų reikšmėms, sulyginus ir nustačius, kad abi struktūras identifikuojantys kodai a sutampa.

Duomenų lentelėje kortežai gali būti išdėstyti įvairia tvarka: pagal kurio nors atributo skaitinių reikšmių didėjimą arba mažėjimą, pagal tekstinių atributų reikšmių pirmųjų raidžių abėcėlę ir t. t. Suprantama, kad nuo kortežų išdėstymo skirtingumo duomenys nesikeičia, jei bus sudaryta galimybė vieną eilutę atskirti nuo kitos. Tam tikslui paprastai vienas iš atributų nurodomas kaip rakto atributas, o jo reikšmės tampa eilučių raktais, t. y. identifikatoriais. Jei vienos atributo reikšmės nepakanka, nes yra pasikartojančių rakto atributo reikšmių, tai rakto atributais nurodomi du ar daugiau atributai. Pateiktoje *a* lentelėje rakto atributas priskirtas C_1.

Lentelių turinio ir jų identifikatorių įvairovė nagrinėtina teoriniu bei praktiniu aspektais dėl kiekybinių bei sandaros skirtingumų. Teoriniu požiūriu

atributų C_j ir jų reikšmių c_{ij} aibės, sudarančios lentelės turinį, yra begalinės, nes atspindi realaus pasaulio įvairovę.

Lentelių identifikatorių *A, B* ir *D* bei šių atributų reikšmių aibės yra skirtingos savo prigimtimi ir skaitlingumu. Lentelių schemų identifikatorių aibė $\{a\}$ gali būti neriboto dydžio, nes teoriškai įmanoma kontroliuoti vis naujas ir kitokias atributų schemas įvairiausiems realaus pasaulio objektams, procesams, reiškiniams ir pan.

Visa tai rodo, kad teoriniu požiūriu schemų aibė $\{a\}$ nėra baigtinė savo elementų skaičiumi. Kiekvienas iš šių *a* identifikuoja vis kitokią atributų schemą, kuri gali būti užpildoma neribotą kiekį kartų, atributų reikšmėmis c_{ij}. Tokie užpildymai priklauso vėl nuo neriboto realų pasaulį atspindinčio objektų ir subjektų kiekio *b*.

Visų minėtų lentelių įvairovę ir gausą radikaliai papildo identifikatoriaus dalis – laiko veiksnys. Laiko veiksnys – tai konkretaus laiko momento fiksavimas arba laiko tarpo nurodymas, kuriuose turi prasmę c_{ij}. Teoriniu požiūriu kiekvienam realiai galinčiam egzistuoti laiko tarpui arba momentui galima konstruoti vis naujas lenteles iš visų anksčiau paminėtų begalinių duomenų lentelių aibių.

RA identifikatoriai gali būti taip pat išreikšti reliacine aibe, kurios schema yra ši:

$$R\,(X) = <A_i\ \ B_j\ \ D_k>, \tag{3.10}$$

čia: A_i – RA schemų atributo pavadinimas: B_j – subjektų atributo pavadinimas; D_k – laiko veiksnių atributo pavadinimas.

Taigi visas RA, kuriose yra duomenys apie įmonės X veiklą, galime identifikuoti RA, kuri vadinama X:

$$X = \begin{pmatrix} A_i & B_j & D_k \\ a_1 & b_1 & d_1 \\ a_1 & b_1 & d_2 \\ .... & ... & ... \\ a_2 & b_1 & d_1 \\ .... & ... & ... \end{pmatrix}. \tag{3.11}$$

Jeigu tam tikram uždaviniui spręsti reikalinga viena savarankiška schema, du subjektai ir du fiksuoti skirtingi laiko momentai, tada visa tai galime išreikšti RA identifikatorių reikšmių kodais [80A]:

$$a_1, b_1, b_2, d_1, d_2.$$

Skirtingos RA gali būti:

$$X = \begin{pmatrix} A_i & B_j & D_k \\ a_1 & b_1 & d_1 \\ a_1 & b_2 & d_1 \\ a_1 & b_1 & d_2 \\ a_1 & b_2 & d_2 \end{pmatrix}. \qquad (3.12)$$

Konkrečiam uždaviniui spręsti gali būti reikalingi ne visi identifikatoriai, o tik jų dalis. Tokiu atveju reikalingų identifikatorių pateikimą galima organizuoti patogesniu būdu [80A], [78A]:

$$a_1, b_{1\text{-}2,} d_{1\text{-}2} = a_1b_1d_1,\ a_1b_2d_1,\ a_1b_1d_2,\ a_1b_2d_2\ .$$

Nors čia nesiskiria abiejų pateikimo būdų rezultatai, bet nesunku pastebėti, kad šiuo būdu lengva didelius RA identifikatorių kiekius pateikti glausta forma, o uždaviniui spręsti galima išskleisti identifikatorius automatiškai ir pagal tokį išskleidimą atrinkti reikiamas RA [80A]:

$$a_{1\text{-}30}b_{50\text{-}100}d_{2\text{-}4} = a_1b_{50}d_2,\ a_1b_{51}d_2,\ \ldots\ , a_{30}b_{100}d_4.$$

Akivaizdu, kad bet kokiems tokio tipo skleidimams pakanka turėti vieną ir tik vieną programinį modulį [4], kuris leistų išskleisti RA identifikatorius.

Glaustoje identifikatorių pateikimo formoje galima *a*, *b ir d* įvairiai keisti vietomis (pavyzdžiui, *abd, adb, bad*, *bda*, *dab*, *dba*), pateikti indeksų identifikatorius dalimis, pavyzdžiui, $a_{1\text{-}30} = a_{1\text{-}10}, a_{10\text{-}30}$. Taigi konkrečiam uždavinio sprendimui galime užtikrinti pateikimą bet kokio reikiamo RA kiekio ir bet kokia reikiama RA tvarka [4]. Taigi akivaizdu, kad duomenų šaltinyje neradus RA, kurios identifikatoriai yra pateiktoje konkretaus uždavinio identifikatorių sekoje, nustatoma, kokių uždaviniui spręsti RA trūksta.

Reikia pabrėžti, kad RA duomenys gali būti identifikuoti nebūtinai trimis kodais. Jų gali būti ir mažiau, ir daugiau. Pavyzdžiui, turime duomenis, identifikuotus toliau pateikta identifikatorių eilute:

$$\{b_1\ ,\ b_2\ ,\ b_3\ ,\ b_4\ \},\ \{\ d_1\ ,\ d_2\ \},\ \{\ a_1\ ,\ a_2\ \}.$$

Tarkime, kad aibės duomenų struktūros kodui a_2 papildomų kodų *b* ir *d* nereikia, tai identifikatorių RA atsiranda kortežų su tuščiomis atributų reikšmėmis, kurios žymimos ženklu Ø (žr. 3.13).

Kortežu a_2 su dviem tuščiomis reikšmėmis gali būti identifikuota normatyvų ir žinynų informacija, kuriai subjekto ar laiko veiksniai neturi jokios reikšmės.

$$X = \begin{pmatrix} B_j & D_k & A_i \\ b_1 & d_1 & a_1 \\ \varnothing & \varnothing & a_2 \\ b_1 & d_2 & a_1 \\ \varnothing & \varnothing & a_2 \\ b_2 & d_1 & a_1 \\ \varnothing & \varnothing & a_2 \\ b_2 & d_2 & a_1 \\ \varnothing & \varnothing & a_2 \\ b_3 & d_1 & a_1 \\ \varnothing & \varnothing & a_2 \\ b_3 & d_2 & a_1 \\ \varnothing & \varnothing & a_2 \\ b_4 & d_1 & a_1 \\ \varnothing & \varnothing & a_2 \\ b_4 & d_2 & a_1 \\ \varnothing & \varnothing & a_2 \end{pmatrix} \quad (3.13)$$

Viską apibendrinus, galima teigti, kad duomenys, kurių schema ir jos atributai turi realias atributų reikšmes, gali būti identifikuojami [80A]:

- *a* – tik schemos identifikatoriaus atributo reikšme;
- *a, b* – *a* ir subjekto identifikatoriaus atributo reikšme;
- *a, d* – *a* ir laiko veiksnio kodo atributo reikšme;
- *a, b, d* – *a*, subjekto ir laiko veiksnio identifikatorių atributų reikšmėmis.

Gali būti, kad tokio pobūdžio identifikavimo nepakanka, t. y. atsiranda daugiau nei viena lentelė, turinti tuos pačius *a*, *b* ir *d*. Suprantama, kad tai vieniems uždaviniams spręsti gali netrukdyti, bet bus ir tokių uždavinių, kuriems būtina „žinoti“, kiek ir kokių lentelių yra su vienodais *a*, *b*, *d*. Tai gali atsitikti dėl paprastų priežasčių kaupiant duomenis nesudėtingiems uždaviniams spręsti. Pavyzdžiui, tegul apie tam tikrą operaciją technologiniame procese duomenys pateikiami duomenų lentelėmis. Visiškai nesvarbu, ar tie duomenys vienodi, ar ne, svarbu juos turėti ir žinoti, kiek kartų įvyko operacija. Jeigu operacijos vyksmo pradžia matuojama sveikaisiais laiko vienetų skaičiais (juos apvalinant), tai dvi operacijos per mažesnę suminę trukmę nei laiko matavimo vienetas rodys, kad dvi operacijos įvyko vienu ir tuo pačiu laiko momentu *d*. Praktikoje tokios operacijos pavyzdžiu gali būti du telefono pokalbiai, įvykę tarp tų pačių abonentų

per minutę, jeigu laiko momento skaičius apvalinamas vienos minutės tikslumu, skaičiaus trupmeninę dalį atmetant. Tokiu atveju galima identifikatoriuje nurodyti vieną ar kelias duomenų koordinates lentelėje ir į identifikatorių (vietoje koordinačių) įrašyti tų koordinačių reikšmę, t. y. atributo reikšmę. Papildomi duomenys identifikatoriuje leis vienareikšmiškai identifikuoti bet kokią duomenų lentelę. Koordinatės lentelėje nurodomos grafos numeriu *j* arba eilutės ir grafos numeriu *ij*.

Tarkime, kad turime dvi vienodais identifikatoriais pažymėtas reliacines aibes:

$$a_2\, b_5\, d_1 \text{ ir } a_2\, b_5\, d_1.$$

Pasirinkę vieną iš aibių, prie jos turimų identifikatorių nurodome reikiamas tos aibės atributo koordinates kaip papildomą identifikatorių:

$$a_2\, b_5\, d_1\, (\, 2/5\,).$$

Šiuo atveju buvo pasirinktas pirmos RA antro kortežo penktame domene esantis atributas, kurio reikšmė – 6. Iškvietus tokią RA, ji yra identifikuojama iškvietus arba nukopijavus identifikatoriuje pateiktą atributo koordinačių reikšmę:

$$a_2\, b_5\, d_1\, (\, 6\,).$$

Reikia pabrėžti, kad žymint RA atributo reikšmės koordinates, indeksas *i* dažnai gali neturėti prasmės. RA kortežai gali būti išdėstyti bet kokia tvarka. Kaip jau buvo minėta, nuo to duomenų turinys nesikeičia, nes visos atributų reikšmės, esančios viename atributo domene, yra to paties atributo reikšmės.

3.4. Duomenų pateikimas ir jų išsamumas

Norint nustatyti ar visi duomenys, reikalingi konkrečiam uždaviniui spręsti, yra duomenų šaltinyje, tenka tikrinti duomenų išsamumą. Tai gali būti atliekama trimis lygmenimis.

1. RA lygmeniu.
2. Kortežų lygmeniu.
3. Atributų reikšmių lygmeniu.

Pirmu lygmeniu duomenų trūkumas yra nustatomas tuomet, kai duomenų šaltinyje, aprašančiame objektą, nerandame visų reliacinių aibių, reikalingų sprendžiamam uždaviniui. Šis trūkumas aptinkamas pateikiant konkrečiam uždaviniui spręsti reikiamų reliacinių aibių identifikatorių eilutę. Jeigu pagal kurį nors vieną iš eilutėje pateiktų identifikatorių nerandama jį atitinkanti RA, tuomet tai pranešama naudotojui. Naudotojas, norėdamas išspręsti uždavinį, turi papildyti pirminių duomenų šaltinį reikiama RA.

Tikrinant duomenų išsamumą kiekvienoje iš sprendimui atrinktų RA, galima nustatyti, ar visi reikalingi kortežai pasirinktam uždaviniui išspręsti yra

kiekvienoje iš atrinktų RA. Tai yra lengvai nustatoma, kadangi iš anksto yra siūloma prie RA schemos fiksuota tvarka nurodyti ir raktinį atributą, ir to atributo reikšmes [78A], [79A]:

- schema – R (K, L, M, N);
- raktinis atributas su raktais – K ($c_{1j} \neq c_{2j} \neq c_{3j} \neq ... \neq c_{mj}$).

Raktiniu atributu pasirenkamas vienas atributas iš visų reliacinėje aibėje esančių atributų. Šio atributo reikšmės tampa kortežų raktais, t. y. kortežų identifikatoriais. Jeigu sprendžiant konkretų uždavinį konkretaus rakto identifikuojamas kortežas tuščias, tada akivaizdu, kad duomenų trūksta ir kokie tai duomenys.

Taigi atributų reikšmių transformacijos metu nepaliečiami tie kortežai, kurie konkrečiam uždaviniui spręsti nereikalingi, bet jeigu RA nerandamas kortežas iš sekos

$$c_{1j}, c_{2j}, c_{3j}, ..., c_{mj},$$

tada nesunkiai nustatoma, kad duomenų trūksta ir kokie tiksliai kortežai konkrečioje RA nepateikti.

Jeigu atsitinka taip, kad RA yra du ar daugiau to pačio rakto ir vienodų (sutampančių) kitų atributų reikšmių kortežai, tai visi kortežai išbraukiami iš RA, paliekant tik vieną iš jų. Akivaizdu, kad buvo be reikalo dubliuojami vieni ir tie patys duomenys. Jeigu esant tam pačiam dviejų kortežų raktui, kitos atributų reikšmės skiriasi, tai akivaizdu, kad pateikiami duomenys yra klaidingi. Šiuo atveju būtina kreiptis į pirminį duomenų šaltinį, kadangi klaidą gali rasti ir ištaisyti tik duomenų savininkas [78A], [80A].

Trečiu lygmeniu duomenų išsamumas apibrėžiamas, kai esančiame korteže yra nustatoma, kad trūksta atskirų atributų reikšmių arba jos yra, bet neatitinka iš anksto nustatytų reikalavimų. Tai gali būti patikrinama atliekant nuosavų atributų reikšmių algoritminių priklausomybių programinių modulių realizaciją.

3.5. Duomenų agregavimo modelis

Duomenų agregavimo modelis dėl transformacijų leidžia reikiamus duomenis, kurie yra skirtingose RA, pateikti vienoje RA.

Transformacija – tai atributų reikšmių perkėlimas iš tam tikro RA skaičiaus ir įvairių schemų į vieną RA. Ši bendroji RA gali nebeturėti daugelio anksčiau apibrėžtų, naudotojui suvokiamų, savybių, o būti tiesiog dvimate duomenų lentele, kurios realų duomenį taikomojo uždavinio algoritmas naudoja kaip koordinačių i / j reikšmę. O koordinatės nurodo reikšmės vietą pirminių duomenų lentelėje. Taigi kortežo ir domeno sąvokos čia gali būti tik formalios, t. y. kortežo raktui nebecharakteringi visi jo elementai, o domeno atributo reikšmė gali nebūti lentelės stulpelyje esančiomis reikšmėmis. Kitaip tariant, gali būti, kad ši lentelė

neturės atributų, nes į tą patį stulpelį gali būti transformuota daug skirtingų reikšmių. Todėl reikiamas duomuo randamas tik žinant jo koordinates.

Taikydami transformacijų metodus ir būdus iš RA, kurios suformuotos iš atrinktų duomenų konkrečiam uždaviniui spręsti, galime transformuoti reikiamus duomenis į vieną reliacinę aibę [4], [76A], [77A], [78A], [79A]. Ją gali sudaryti iš dalies apdoroti duomenys, nes atliekant transformacijas galima realizuoti aritmetinius ir loginius veiksmus. Būtent transformacijos padeda suformuoti naują RA, kuri gali skirtis savo turiniu ir struktūra nuo tų RA, iš kurių imami duomenys, t. y. nuo duomenų šaltinių. Naudodami turimus programinius modulius, kuriais realizuojamos transformacijų formulės, ir tik formulėse pagal poreikius keisdami transformacijoje dalyvaujančių aibių atributų reikšmių koordinates, galime nesunkiai suformuoti kitokią, taikomajam uždaviniui reikalingą rezultatinę RA. Tai leidžia teigti, kad aprašytasis duomenų pateikimo būdas turi akivaizdžių adaptyvumo požymių.

3.5.1. Transformacijų tipai ir rūšys

Reliacinių aibių transformacija leidžia iš įvairių RA sudaryti pirminių duomenų aibę konkrečiam uždaviniui spręsti. Aibių transformacijos išreiškiamos formule, kuri išrenka iš RA tik tas atributų reikšmes ir išdėsto tokia tvarka, kad derėtų su taikomojo uždavinio algoritmu ir tą algoritmą realizuojančios programas galėtų išspręsti šį uždavinį. Jeigu RA probleminėje srityje keičiasi, tai pakanka pakeisti transformacijos formulę, o taikomojo uždavinio algoritmas ir programa lieka nepakitę. Kadangi, atliekant transformaciją, gali būti realizuojami aritmetiniai ir loginiai veiksmai, todėl duomenis priimanti RA gali būti sudaryta visiškai arba iš dalies iš visai naujų atributų reikšmių c_{ij}, kurių duomenų šaltinyje (žr. 3.5.2 poskyrį) nebuvo, nes jie gauti po nurodytų veiksmų [4], [76A].

RA, iš kurios transformuojami duomenys, vadinama duomenų šaltiniu, o į kurią transformuojami – duomenis priimančia RA. Jeigu priimančios RA duomenų koordinates joje pažymėsime i ir j, o duomenų šaltinio koordinates k ir l, tai duomenys bus transformuojami iš koordinačių $< k / l >$ į koordinates $< i / j >$. RA koordinatės, iš kurių imami duomenys, ir RA koordinatės, į kuriuos įvedami duomenys, turi būti vienodai nutolę nuo savosios RA pradžios. Koordinatės $< i / j >$ ir $< k / l >$ naudojamos dėl to, kad gali prireikti naudoti tam tikras RA su iš anksto nustatyta kortežų raktų eile ir jų kiekiu. Šiuo atveju galima akivaizdžiau fiksuoti duomenų trūkumą arba neišsamumą, jeigu konkrečioje RA trūksta kortežo, kuris uždaviniui spręsti yra būtinas. Kitais atvejais RA koordinatėms pažymėti pakanka l ir j, t. y. parodyti tik c nutolimą nuo kortežo pradžios.

RA transformacijos pagal tipus gali būti suskirstytos į [2], [76A]:

- besąlygines,
- sąlygines,

- aritmetines (rezultatines).

Sąlyginės transformacijos pagal rūšis dar gali būti skirstomos į:

- sąlygines ištisines,
- sąlygines ciklines.

Visose išvardytose transformacijose, išskyrus besąlyginę, sąlygos duomenims palyginti nurodomos šiais simboliais: $=, \neq, >, <, \geq, \leq$.

Sąlyginėse transformacijose yra lyginamos koordinačių reikšmės ir transformacija atliekama tik tada, jeigu sąlyga patenkinta.

Transformacijose naudojamos RA koordinatės gali būti kelių rūšių:

– koordinatės, iš kurių imami duomenys $< k / l >$;
– koordinatės, į kurias įkeliami duomenys $< i / j >$;
– koordinatės, kurių reikšmės sulyginamos pagal nustatytą sąlygą $\{k / l\}, \{ i / j \}$.

Šios koordinatės jungiamos į paprastąsias aibes, o ne į tvarkiąsias, nes lyginamųjų koordinačių skaičiai gali labai skirtis. Pavyzdžiui, vienose koordinatėse pateiktą reikšmę galime lyginti su dideliu skaičiumi duomenų šaltinio reikšmių ir jeigu sąlyga bus patenkinta, galėsime atlikti transformaciją. Šiuo atveju šaltinio duomenų koordinačių išdėstymo eilė visiškai neturi reikšmės.

Taigi bendroji transformacijos formulės struktūra bus tokia [82A]:

$$\{ k / l \}^n < k / l >^n \quad \xrightarrow{\theta} \quad < i / j >^n \{ i / j \}^n. \tag{3.14}$$

Čia lyginamos koordinatėse $\{ i / j \}$ ir $\{ k / l \}$ pateiktos reikšmės; θ apibrėžtas vienas iš anksčiau pateiktų lyginimo sąlygos simbolių. Kai sąlyga patenkinta, duomenys perkeliami iš koordinačių $< k / l >$ į koordinates $< i / j >$. Rodykle nurodyta duomenų transformacijos kryptis. Indeksas *n* reiškia aibės eilės numerį. Vienu metu šie indeksai gali būti visi skirtingi arba visi vienodi. Jei *n* reikšmės skirtingos, tai duomenys, kad būtų įvykdyta sąlyga θ, lyginami vienose aibėse, o duomenų transformacija – kitose. Jei *n* reikšmės vienodos, tai duomenys lyginami ir transformuojami toje pačioje RA, t. y. pakeičiamas vienos RA struktūros duomenų išdėstymas.

3.5.2. Besąlyginės transformacijos

Besąlyginės transformacijos atveju sąlygos θ formulėje nebūna, kaip ir koordinačių, kurių reikšmės turi atitikti θ. Tad besąlyginės transformacijos formulė bus tokia [82A]:

$$p^t(< k / l >^n \quad \longrightarrow \quad < i / j >). \tag{3.15}$$

Šioje išraiškoje $< i / j >$ indeksas *n* neturi prasmės, nes duomenis priimanti RA visada yra viena.

Visose transformacijose duomenys gali būti perteikiami iš vienos aibės į kitą trimis būdais [82A]:

- atliekant Dekarto sandaugą;
- atliekant transformaciją kortežuose;
- atliekant transformaciją domenuose.

Nustatome transformacijos užrašymo schemą [78A], [79A]:

[Duomenų šaltinis – RA] $\longrightarrow$ [Duomenis priimanti RA] = [Transformacijos rezultatas – RA]

Naudodami užrašymo schemą pateikiame Dekarto besąlyginės transformacijos RA pavyzdį, kuris apskaičiuotas pagal formulę [78A], [79A]:

$$p^{t}(<1,2> \rightarrow <3,4>). \tag{3.16}$$

$$\begin{bmatrix} \underline{K \quad L} \\ a \quad b \\ c \quad d \end{bmatrix} \longrightarrow \begin{bmatrix} \underline{M \quad N} \\ e \quad f \\ g \quad h \end{bmatrix} = \begin{bmatrix} \underline{M \quad N \quad K \quad L} \\ e \quad f \quad a \quad b \\ e \quad f \quad c \quad d \\ g \quad h \quad a \quad b \\ g \quad h \quad c \quad d \end{bmatrix}. \tag{3.17}$$

Tradicinės Dekarto sandaugos galimybės išplečiamos formulėmis [78A], [79A]:

$$p^{t}(<1,2> \rightarrow <1,3>), \tag{3.18}$$

$$p^{t}(<1,2> \rightarrow <2,4>), \tag{3.19}$$

nes RA rezultatas pakeičia net RA schemą:

$$\begin{bmatrix} \underline{K \quad L} \\ a \quad b \\ c \quad d \end{bmatrix} \longrightarrow \begin{bmatrix} \underline{M \quad K \quad N \quad L} \\ e \quad 0 \quad f \quad 0 \\ g \quad 0 \quad h \quad 0 \end{bmatrix} = \begin{bmatrix} \underline{M \quad K \quad N \quad L} \\ e \quad a \quad f \quad b \\ e \quad c \quad f \quad d \\ g \quad a \quad h \quad b \\ g \quad c \quad h \quad d \end{bmatrix}. \tag{3.20}$$

Galimybė keisti RA schemą ypač vertinga, nes išplečia duomenų manipuliavimo galimybes, nedidinant programinių modulių skaičiaus ir sudėtingumo. Toks pavyzdys gali būti galimybė koordinačių nuorodomis formulėje $p^{t}(<2> \rightarrow <3>)$ gauti rezultatą [78A]:

$$\begin{bmatrix} \underline{K \quad L} \\ a \quad b \\ c \quad d \end{bmatrix} \longrightarrow \begin{bmatrix} \underline{M \quad N} \\ e \quad f \\ g \quad h \end{bmatrix} = \begin{bmatrix} \underline{M \quad N \quad L} \\ e \quad f \quad b \\ e \quad f \quad d \\ g \quad h \quad b \\ g \quad h \quad d \end{bmatrix}. \qquad (3.21)$$

Transformacija korteže atliekama sujungiant vienodai nutolusius nuo RA pradžios kortežus į vieną kortežą [78A]:

$$\begin{bmatrix} \underline{D \quad E \quad F} \\ s_1 \quad r_1 \quad z_1 \\ t_1 \quad u_1 \quad v_1 \end{bmatrix} \longrightarrow \begin{bmatrix} \underline{A \quad D \quad B} \\ r \quad t \quad s \\ u \quad z \quad v \end{bmatrix} = \begin{bmatrix} \underline{A \quad D \quad B \quad F} \\ r \quad s_1 \quad s \quad z_1 \\ u \quad t_1 \quad v \quad v_1 \end{bmatrix}. \qquad (3.22)$$

Besąlyginės transformacijos korteže atliktos pagal formulę:

$$p^{t}(< 1,3 > \rightarrow < 2,4 >). \qquad (3.23)$$

Jeigu išrinkimo koordinatės duomenų šaltinyje ir duomenų įrašymo koordinatės priimančioje RA sutampa, tai būtina sąlyga, kad būtų atlikta besąlyginė duomenų transformacija domenuose. O pakankama sąlyga yra ta, kad p^{t} atitiktų transformacijos domenuose algoritmą.

$$p^{t}(< 1,2,3 > \rightarrow < 1,2,3 >), \qquad (3.24)$$

$$\begin{bmatrix} \underline{K \quad L \quad M \quad L} \\ a_1 \quad b_1 \quad e_1 \quad f_1 \\ c_1 \quad d_1 \quad g_1 \quad h_1 \end{bmatrix} \longrightarrow \begin{bmatrix} \underline{K \quad L \quad N} \\ a \quad b \quad e \\ c \quad d \quad g \end{bmatrix} = \begin{bmatrix} \underline{K \quad L \quad M} \\ a \quad b \quad e \\ c \quad d \quad g \\ a_1 \quad b_1 \quad e_1 \\ c_1 \quad d_1 \quad g_1 \end{bmatrix}. \qquad (3.25)$$

Pateikti duomenų besąlyginės transformacijos pavyzdžiai tik bendrąja prasme iliustruoja galimybę transformacijos metu keisti reliacinių aibių struktūrą ir turinį. Prireikus transformacijų galimybės gali būti keičiamos ir plėtojamos.

Taigi programiškai realizuota transformacijų formule, keičiant joje tik duomenų šaltinio ir priimančios aibės koordinates, galime iš bet kokių parametrų ir turinio RA suformuluoti rezultatinę RA, reikalingą taikomajam uždaviniui vienais ir tais pačiais transformacijos realizavimo programiniais moduliais.

3.5.3. Sąlyginės transformacijos

Sąlyginė transformacija reiškia duomenų perkėlimą, t. y. transformaciją iš duomenų šaltinio, kurį sudaro neapibrėžtas kiekis RA, į vieną duomenis priimančią RA tik tuo atveju, jeigu patenkinta sąlyga θ.

Pagrindinė sąlyginės transformacijos samprata analogiška besąlyginei transformacijai, nors turi kelis ypatumus. Pagrindinis ypatumas tas, kad sąlyginės transformacijos formulė gali turėti tik lyginamųjų duomenų koordinates, t. y. neturėti duomenų persiuntimo koordinačių. Tai reiškia, kad duomenys visiškai netransformuojami, o tik lyginami tarp savęs nustatant, ar tenkinama sąlyga θ, kuri apibrėžiama minėtais simboliais, ar netenkinama. Tai pagrindinis tokios transformacijos rezultatas. Ši galimybė ypač svarbi tikrinant duomenų korektiškumą, surandant klaidas duomenyse ir pan. Tokiam atvejui pakanka formulės:

$$p^{t}(\{k/l\} \xrightarrow{\theta} \{i/j\}). \tag{3.26}$$

Kitas ypatumas yra tas, kad ar sąlyga θ patenkinama, tikrinama tol, kol nustatoma, kad ta sąlyga patenkinta. Tada tolimesnis tikrinimas nutraukiamas ir atliekama transformacija. Tokia transformacija vadinama sąlygine. Jeigu, aptikus sąlygos patenkinimą ir atlikus transformaciją, toliau ieškoma, ar nėra duomenų, tenkinančių sąlygą θ, tai ši transformacija vadinama sąlygine ištisine transformacija.

Sąlygine cikline transformacija vadinama tokia sąlyginė ištisinė transformacija, kai atliekama daugiau kaip viena sąlyginė ištisinė transformacija tame pačiame duomenų šaltinyje. Pasibaigus vienai sąlyginei transformacijai arba vienam ciklui, lyginamųjų koordinačių reikšmė pakeičiama kita ir pradedama paieška θ patenkinimo nuo duomenų šaltinio pradžios ir t. t.

Jeigu sąlyga θ išreiškiama vienu iš ženklų:

$$=, \neq, >, <, \geq, \leq$$

ir duomenų lyginimo koordinatės bus dvi, pvz.,{2} ir {4}, tai θ patenkinimo arba nepatenkinimo rezultatas labai aiškus, nes yra vienareikšmis. Jeigu koordinačių, kurių reikšmės tarpusavyje lyginamos, yra daugiau nei dvi, tai θ patenkinimo rezultatus būtina aptarti atskirai, nes jie gali būti traktuojami įvairiai, nes lyginimo procesas yra sudėtingesnis.

Jeigu lyginamos koordinačių {3,1,2}→ {4,3,1} reikšmės, remiantis apibrėžta sąlyga „=“, tai RA

$$r' = \begin{bmatrix} a & n & m \\ k & l & b \\ d & k & n \end{bmatrix}, \quad r = \begin{bmatrix} k & a & m & l \\ l & m & k & b \\ a & k & b & c \end{bmatrix}, \tag{3.27}$$

kortežai $<k\ l\ b>$ ir $<l\ m\ k\ b>$ atitinka sąlygą. Norint patikrinti sąlygą „=“, nebūtina lyginti tik dvi atributų reikšmes. Lyginamų reikšmių skaičius gali būti bet koks, bet tokių lyginamųjų reikšmių koordinačių skaičius vienoje ir kitoje lygybės pusėje turi sutapti.

Jeigu sąlyga yra „≠“, tai ji tenkinama tada, kai bent vienu atveju lyginamųjų koordinačių reikšmės nesutampa. Jeigu θ yra viena iš:

$$>, <, \geq, \leq,$$

tai tikrinti paprasta, nes tarpusavyje visada lyginamos tik dvi atributų reikšmės.

Iki šiol apibrėžtos duomenų lyginimo sąlygos taikomos skaitinėms atributų reikšmėms. Loginiai palyginimai „$\vee$“ (Arba), „$\wedge$“ (Ir), „$\neg$“ (Ne) taikytini ne tik skaitinėms, bet ir tekstinėms atributų reikšmėms.

Jeigu θ yra loginė operacija Arba „$\vee$“, o duomenų šaltinio *r*' koordinačių {2/1} reikšmė pagal šią sąlygą bus lyginama su RA *r* antrojo kortežo

$$<l\ m\ k\ b>,$$

atributų reikšmėmis. Tai išraiška bus

$$\{2/1\} \xrightarrow{\vee} \{1,3,4\} \qquad (3.28)$$

ir šiuo atveju koordinačių {2/1} reikšmė atitiks sąlygą „$\vee$“ tada, kai tas turinys bus arba *l*, arba *k*, arba *b*, tuo atveju bus transformuojamas visas kortežas $<k\ l\ b>$.

Loginė operacija Ir „$\wedge$“ bus atliekama analogiškai, tik skirsis sąlyga.

3.5.4. Aritmetinės transformacijos

Sąlyginės ištisinė ir ciklinė transformacijos fiksuoja tam tikram laikui duomenis priimančioje aibėje, juos palygina pagal sąlygą θ su šaltinio duomenimis ir atlieka transformaciją. Aritmetinės transformacijos mechanizmas yra toks pat. Tik šioje transformacijoje neatsiranda naujas kortežas. Čia duomenys lyginami pačiame duomenų šaltinyje. Lyginamųjų koordinačių reikšmėms, t. y. raktinėms atributo reikšmėms skirtingose RA, sutapus tam tikros skaitinės atributų reikšmės sudedamos arba atimamos ir pernešamos į paskutinę duomenų šaltinio aibę, kuri tampa rezultatine. Likusių lyginimo neatitinkančių kortežų į rezultatą galima traukti arba ne. Remiantis tuo, aritmetinė transformacija skirstoma į aritmetinę jungiamąją ir aritmetinę nejungiamąją.

Aritmetinė jungiamoji transformacija atliekama taip, kad duomenų RA, kortežas ar jų dalys, jeigu sąlyga „=“ nepatenkinama, vis tiek yra įtraukiami į RA – rezultatą. Aritmetinė jungiamoji transformacija atliekama įrašant ne vienus duomenis į atitinkamas koordinates, kad ten buvę duomenys išsitrintų, o kad transformuojami duomenys ir esami duomenys sumuotųsi. Praktiškai dažniausiai taikoma sumavimo operacija, bet galimos ir kitos aritmetinės operacijos.

Jeigu duomenų šaltinis yra dvi RA:

$$\begin{bmatrix} K & L & N & M \\ \hline k & 1 & 3 & 2 \\ c & 2 & 4 & 1 \\ b & 4 & 5 & 3 \end{bmatrix}, \begin{bmatrix} K & L & N & M \\ \hline a & 2 & 1 & 3 \\ c & 1 & 3 & 1 \\ d & 4 & 2 & 5 \end{bmatrix} \tag{3.29}$$

ir aritmetinės jungiamosios transformacijos formulė yra:

$$p\,'(\{\ 1\ \}<2,3,4> \xrightarrow{+\atop =} \{\ 1\ \}<2,3,4>), \tag{3.30}$$

tai pirmosioms dviem RA atlikus aritmetinę jungiamąją transformaciją, gausime:

$$\begin{bmatrix} K & L & N & M \\ \hline k & 1 & 3 & 2 \\ c & 2 & 4 & 1 \\ b & 4 & 5 & 3 \end{bmatrix}, \begin{bmatrix} K & L & N & M \\ \hline a & 2 & 1 & 3 \\ c & 1 & 3 & 1 \\ d & 4 & 2 & 5 \end{bmatrix} \xrightarrow{+\atop =} \begin{bmatrix} K & L & N & M \\ \hline a & 2 & 1 & 3 \\ c & 3 & 7 & 2 \\ d & 4 & 2 & 5 \\ k & 1 & 3 & 2 \\ b & 4 & 5 & 3 \end{bmatrix}. \tag{3.31}$$

Esant tai pačiai transformacijos formulei aritmetinė nejungiamoji transformacija bus:

$$\begin{bmatrix} K & L & N & M \\ \hline k & 1 & 3 & 2 \\ c & 2 & 4 & 1 \\ b & 4 & 5 & 3 \end{bmatrix}, \begin{bmatrix} K & L & N & M \\ \hline a & 2 & 1 & 3 \\ c & 1 & 3 & 1 \\ d & 4 & 2 & 5 \end{bmatrix} \xrightarrow{+\atop =} \begin{bmatrix} K & L & N & M \\ \hline a & 2 & 1 & 3 \\ c & 3 & 7 & 2 \\ d & 4 & 2 & 5 \end{bmatrix}. \tag{3.32}$$

Esant tai pačiai transformacijos formulei, bet atlikę aritmetinę jungiamąją transformaciją su atimties veiksmu, gausime:

$$\begin{bmatrix} K & L & N & M \\ \hline k & 1 & 3 & 2 \\ c & 2 & 4 & 1 \\ b & 4 & 5 & 3 \end{bmatrix}, \begin{bmatrix} K & L & N & M \\ \hline a & 2 & 1 & 3 \\ c & 1 & 3 & 1 \\ d & 4 & 2 & 5 \end{bmatrix} \xrightarrow{-\atop =} \begin{bmatrix} K & L & N & M \\ \hline a & 2 & 1 & 3 \\ c & -1 & -1 & 0 \\ d & 4 & 2 & 5 \\ k & 1 & 3 & 2 \\ b & 4 & 5 & 3 \end{bmatrix}. \tag{3.33}$$

Kadangi duomenų šaltinis dažniausia sudarytas ne iš vienos ar dviejų RA, o iš kelių, tai toliau pateikiamas aritmetinės jungiamosios transformacijos pavyzdys iliustruoja tokį atvejį:

$$p'(\{\ 1\ \}<2> \xrightarrow{+\atop=} \{\ 1\ \}<2>), \tag{3.34}$$

$$\begin{bmatrix} \underline{K \quad L} \\ a \quad 2 \\ k \quad 5 \end{bmatrix}, \begin{bmatrix} \underline{K \quad L} \\ l \quad 6 \\ a \quad 2 \end{bmatrix}, \begin{bmatrix} \underline{K \quad L} \\ k \quad 4 \\ l \quad 1 \end{bmatrix}, \tag{3.35}$$

$$\begin{bmatrix} \underline{K \quad L} \\ k \quad 1 \\ a \quad 3 \end{bmatrix}, \begin{bmatrix} \underline{K \quad L} \\ l \quad 3 \\ b \quad 4 \end{bmatrix}, \begin{bmatrix} \underline{K \quad L} \\ k \quad 3 \\ l \quad 1 \end{bmatrix} \xrightarrow{+\atop=} \begin{bmatrix} \underline{K \quad L} \\ k \quad 13 \\ l \quad 11 \\ a \quad 7 \\ b \quad 4 \end{bmatrix}. \tag{3.36}$$

Dažnai gali būti reikalinga speciali transformacija, kuri sukuria pačioje transformavimo pradžioje tuščią priimančią RA. Ji naudojama tada, kai transformuojamos įvairių schemų RA, o priimančioji RA nesutampa nė su viena iš transformuojamų RA schemų.

Visos transformacijos (besąlyginė, sąlyginė, sąlyginė ištisinė ir sąlyginė ciklinė), išskyrus aritmetinę, naudojamos pirminių duomenų, reikalingų konkrečiam uždaviniui spręsti, RA suformuoti. Naudodami aritmetinę transformaciją galime naršyti po duomenų šaltinį ir keisti jų turinį. Gauti informaciją apie tai, kokių duomenų trūksta, ką reikia papildyti ar pan.

3.6. Trečiojo skyriaus išvados

Šiame skyriuje pateikiami adaptyvieji duomenų išrinkimo ir agregavimo modeliai, t. y duomenų identifikavimas ir transformacijos. Duomenų RA identifikavimas leidžia sukurti programines priemones, kurios pagal formalų kreipinį iš duomenų šaltinio gali iškviesti konkrečiam uždaviniui reikalingas reliacines aibes ir pateikti jas tam uždaviniui reikalinga seka. Akivaizdu, kad konkrečiam uždaviniui gali reikėti ne visų pateiktose aibėse esančių duomenų. Reikalingų atributų reikšmes išrinkti ir jų reikšmes išdėstyti reikiama seka padeda RA transformacijos. Taigi duomenys iš duomenų šaltinio yra agreguojami į pirminių duomenų lentelę konkrečiam uždaviniui spręsti. Bet kokiam taikomajam uždaviniui spręsti naudojamos tos pačios formalios identifikatorių ir transformacijų išraiškos. Skirtingiems uždaviniams naudojami tik skirtingi identifikatorių ir transformacijų formalių išraiškų parametrai. Taigi ir identifikatoriai, ir transformacijos patvirtina adaptyvumo principą, nes keičiantis duomenų struktūroms ir turiniui programos išlieka stabilios.

4

Duomenų apdorojimo projektavimo modelis

4.1. Algoritminės duomenų priklausomybės ir jų elementai

Taikant duomenų išrinkimo modelį ir atrinkus tam tikram uždaviniui spręsti reikalingas duomenų reliacines aibes, naudojant transformacijas sudaroma nauja reliacinė aibė, naujų atributų reikšmių kombinacijų lygmeniu. Tada taikomasis uždavinys sprendžiamas kaip tam tikra algoritminių priklausomybių tarp duomenų realizaciją, kadangi tarp atributų apibrėžiamos algoritminės priklausomybės. Vadinasi, šį uždavinį sprendžiant šie atributai yra siejami, t. y. vienas nuo kito priklauso. Vienam keičiantis gali keistis ar nesikeisti ir kiti atributai. Be to, šios algoritminės priklausomybės gali kisti. Vienam projektuojamam apdorojimui jos gali būti vienokios tarp tų pačių atributų, kitam – jos gali būti kitokios. Taip tolygiai plečiama sukaupta algoritminių priklausomybių algoritmus realizuojančių programinių modulių sistema, kuri ilgainiui gali realizuoti daugelį ar net visus atsirandančius taikomojo pobūdžio uždavinius.

Teoriniu požiūriu algoritmine duomenų priklausomybe laikomas algoritmas, kuris nurodo, kaip sprendžiant tam tikrą uždavinį bus apdorojami duomenys, kurie yra toje pačioje RA [76A], [77A], [82A]. Šio darbo 3.5 poskyryje aprašytos

transformacijos yra priskirtos prie išskirtinio pobūdžio algoritminių priklausomybių, nes transformacijų algoritmas dažniausiai nurodo, kaip bus apdorojami duomenys, kurie pateikti ne vienoje RA. Pavyzdžiui, kai uždaviniui spręsti reikalingi duomenys nėra vienoje ir toje pačioje RA, tai transformacijos algoritmo realizacija iš bet kokio RA rinkinio suformuoja reikiamą pirminių duomenų RA konkrečiam uždaviniui spręsti.

Algoritminės priklausomybės organizuojamos į struktūrą, kuri adekvati reliacinių aibių struktūroms ir gali būti išreikšta eilute:

$$P = (<p_1>, <p_2>, \dots <p_j>, <p_n>).$$

Tačiau algoritminių priklausomybių kodų RA geriau vaizduoti dvimate lentele todėl, kad didesnes algoritminių priklausomybių kodų RA, vizualiai išreikštas eilute, labai sunku suvokti. Todėl siūloma jas sudaryti dvimate lentele:

$$P_{fiksuota} = \begin{pmatrix} <p_{11}><p_{12}> & \dots & <p_{1j}> & \dots & <p_{1n}> \\ <p_{21}><p_{22}> & \dots & <p_{2j}> & \dots & <p_{2n}> \\ \dots & \dots & \dots & \dots & \dots \\ <p_{i1}><p_{i2}> & \dots & <p_{ij}> & \dots & <p_{in}> \\ \dots & \dots & \dots & \dots & \dots \\ <p_{m1}><p_{m2}> & \dots & <p_{mj}> & \dots & <p_{mn}> \end{pmatrix}. \quad (4.1)$$

Tokiu atveju atsiranda domeno ir kortežo koordinatės. Algoritminių priklausomybių kodų RA atributų pavadinimai nerašomi, nes visų atributų pavadinimai yra vienodi, t. y. atributų reikšmių algoritminės priklausomybės.

Bet kuri algoritminių priklausomybių išraiška $<p_{ij}>$ parodo, kad atributo reikšmė turi ne vieną AP ir jos turi būti išdėstytos griežtai nustatyta tvarka. Tad $<p_{ij}>$ yra tvarkioji algoritminių priklausomybių kodų RA.

Kiekviena AP turi kodą, savo realizavimo algoritmą ir programinį to algoritmo realizavimo modulį. Duomenys šioje duomenų apdorojimo sistemoje faktiškai yra RA atributų reikšmės – c_{ij} Kiekvienai atributo reikšmei arba bet kuriam RA reikšmių poaibiui parenkama tvarkioji algoritminių priklausomybių reliacinė aibė, kurios elementų algoritmų realizavimo programiniai moduliai ir atlieka duomenų apdorojimą [82A]:

$$<p_{ij}> \to <c_{ij}> \to <c'_{ij}>,$$

čia: $<p_{ij}>$ – algoritminių priklausomybių aibė; $<c_{ij}>$ – pirminių duomenų aibė; $<c'_{ij}>$ – uždavinio sprendimo rezultatas.

Duomenų struktūrų ir jų apdorojimo sistemą galima pavaizduoti toliau pateiktomis formaliomis konstrukcijomis:

Tegu duota papildymams neuždara duomenų algoritminių priklausomybių aibė:

$$\{ p_1, p_2, \ldots, p_n, \ldots \}.$$

Šių algoritminių priklausomybių algoritmus realizuoja taip pat papildymams atvira programinių modulių aibė:

$$\{m_1, m_2, \ldots, m_n, \ldots\}.$$

Abiejų aibių atvirumas reiškia, kad sprendžiant duomenų apdorojimo uždavinius gali prisireikti naujų p ir m, nes ankstesnė jų visuma negali atlikti tam tikrų uždaviniui spręsti reikalingų veiksmų. Tada sukuriama nauja algoritminė priklausomybė p_{n+1} ir ją realizuojantis modulis m_{n+1}, kurie ne tiesiogiai atlieka veiksmus, kurių nebuvo galima atlikti, o papildžius turimas aibes nauja algoritmine priklausomybe ir programiniu moduliu, kartu su kitomis p_j ir m_j ($1 \le j \le n$) apima ir naujo uždavinio sprendimo algoritmo realizavimą. Tad galime teigti, kad visų algoritminių priklausomybių grupių aibė ir algoritminių priklausomybių kiekis grupėje yra atviras papildymams.

Identifikuojant algoritminių priklausomybių kodų reliacines aibes (apie tai aprašyta 3.3 poskyryje) pakanka schemos kodo a, nes nei nuo subjekto ir nei nuo laiko veiksnių šios RA nepriklauso.

Šiuo tyrimo etapu algoritminės priklausomybės yra skirstomos į penkias grupes [76A], [77A], [82A]:

1. RA transformacijų AP – p^t. Šiai grupei priskiriami visi transformacijų būdai aprašyti 3.5 poskyryje.

2. Skaičiavimo-infologinės AP – p^{si}. Ši grupė skirstoma į du glaudžiai tarp savęs susietus, bet atskirus pogrupius. Skaičiavimo AP pogrupis naudojamas įvairiems aritmetiniams veiksmams aprašyti, gaunant ir tų veiksmų rezultatus. Infologinės AP pogrupis naudojamas įvairių aritmetinių veiksmų rezultatams sugretinti, tikrinant įvairių procesų suderinamumą ir korektiškumą.

3. Nuosavos atributų reikšmių AP – p^a. Tai grupė labai įvairių, gerokai viena nuo kitos besiskiriančių algoritminių priklausomybių, kurios palyginti pastovios. Tai reiškia, kai konkreti atributo reikšmė naudojama skirtingiems uždaviniams spręsti, atributo reikšmė gali būti kitokia. Net ir tokios atributo reikšmės algoritminę priklausomybę, kai atributo reikšmė – skaičius ar tekstas, negalime traktuoti kaip visiškai pastovios. Nes jeigu atributo reikšmę apibrėžtume kaip tekstą, tada ši reikšmė negalėtų dalyvauti aritmetinėje operacijoje. Vieną uždavinį sprendžiant gali reikėti, kad turimos atributų reikšmės būtų traktuojamos kaip skaičiai, o kitą – kaip tekstas.

4. Programinių ėjimų AP – $p^{\rightarrow}$. ARA yra aibė, sudaryta iš algoritminių priklausomybių kodų ir vienareikšmiškai lemia duomenų atributų reikšmių dalyvavimą apdorojimo operacijoje. Tai atliekama kreipiantis į ARA esančios AP kodą ir taip iškviečiant šią AP realizuojantį programinį modulį. Atsižvelgiant į

konkretaus uždavinio sprendimo eigą, į AP kodus galima kreiptis reikiama seka. Tai atlikti leidžia programinių ėjimų AP, kadangi jos nurodo programinio ėjimo pradžią ir kryptį. Programiniai ėjimai algoritminėje reliacinėje aibėje yra skirstomi į nuosekliuosius ir nenuosekliuosius.

5. Išorinės AP – p^{i}. Išorinės algoritminių reliacinių aibių priklausomybės reguliuoja apdorojamų duomenų ARA tarpusavio ryšius su gretimomis ARA. Šie ryšiai neatsiejami nuo apdorojamųjų duomenų RA. Išorinė algoritminė priklausomybė reikalinga tam, kad itin sudėtingus uždavinius galėtų projektuoti ne vienas projektuotojas, o keli vienu metu, tuo pagreitindami sudėtingo uždavinio sprendimą.

4.2. Skaičiavimo-infologinės algoritminės duomenų priklausomybės

Skaičiavimo-infologinės AP – tai algoritminės duomenų priklausomybės, kurių algoritmo realizavimo rezultatas yra skaičiai. Kadangi šie skaičiai tarpusavyje gali būti lyginami, tai galima numatyti ar tie skaičiai turi būti lygūs, ar nelygūs ir pan. Todėl akivaizdu, kad dvi algoritminės priklausomybės gali sudaryti vieną infologinę skaičiavimo AP, kurioje skaičiavimo rezultatai yra lyginami tarp savęs vienu iš ženklų:

$$=, \neq, >, <, \geq, \leq .$$

Skaičiavimo AP elementai formulėje reiškiami RA atributų reikšmių koordinatėmis, kur koordinačių reikšmės yra skaičiai, o koordinatės tarpusavyje jungiamos aritmetiniais ženklais:

$$+,\ -,\ :,\ \times \text{ ir kt.}$$

Pirmasis šios grupės algoritminių priklausomybių pogrupis vadinamas skaičiavimo AP ir žymimas p^{+}, o antrasis – infologinėmis AP ir žymimas $p^{=}$. Bendroji pirmojo pogrupio algoritminių priklausomybių išraiška yra [78A]:

$$p^{+}(A\ \psi\ A\ \psi\ [c]\ \psi\ ...\rightarrow A), \tag{4.2}$$

čia: A – atributo reikšmės koordinatės reliacinėje aibėje; ψ – vienas iš skaičiavimo operacijų ženklų; $\rightarrow$ – nuoroda į koordinates, kur turi būti pateiktas skaičiavimo rezultatas; [c] – skaitinė konstanta (konstantos naudojimo pavyzdžiu gali būti procento skaičiavimas).

Konstantos naudojimo pavyzdys priklausomybės eilutėje gali būti procento skaičiavimas:

$$k_1 : k_2 \times [100] \rightarrow k_3 , \tag{4.3}$$

čia atributo koordinačių k_1 reikšmė yra skaičius, k_2 – kitas skaičius. Į koordinates k_3 įkeliamas skaičius, nurodantis, kokį procentą sudaro antrasis skaičius, lyginant jį su pirmuoju.

Pateikiamas ir kitas pavyzdys, kuris leis geriau suvokti skaičiavimo algoritminės priklausomybės esmę. Tegu duota aritmetinė išraiška:

$$\begin{gathered} 5 + 1 + 2 \times (4 \times 2 + (19 - 8)) + (12 : 3 + 5) = \\ 5 + 1 + 2 \times (4 \times 2 + 11) + (4 + 5) = \\ 5 + 1 + 2 \times (8 + 11) + 9 = \\ 6 + 2 \times 19 + 9 = \\ 6 + 38 + 9 = 53. \end{gathered} \tag{4.4}$$

Pirminiai skaičiavimo duomenys pagal pateiktą išraišką tegu yra RA:

$$\begin{array}{l} \\ \text{1 įrašas} \\ \text{2 įrašas} \end{array} \begin{bmatrix} A_0 & A_1 & A_2 & A_3 & A_4 \\ B_1 & 4 & 12 & 5 & 2 \\ B_2 & 3 & 1 & 19 & 8 \end{bmatrix}, \tag{4.5}$$

Šioje RA atributas A_0 nurodytas kaip raktinis atributas, taigi atributo reikšmės B_1, B_2 yra eilučių raktai, t. y. įrašų identifikatoriai. Todėl turime galimybę vieną įrašą atskirti nuo kito. Kadangi įrašų ilgis dirbtinai padarytas pastoviu, tai ieškant lengva juos rasti, nes kiekvieno įrašo pradžia (pradedant antruoju įrašu duomenų sankaupoje) yra už vienodo baitų skaičiaus. Taip pat lengvai galime rasti reikiamą atributo reikšmę pagal jos koordinates c_{ij}.

Taigi prieš tai pateikti aritmetiniai veiksmai apskaičiuojami realizuojant skaičiavimo algoritminių priklausomybių formules.

$$p^+ (3/4 - 3/5 \to 3/4, \quad 2/3 : 3/2 \to 2/3), \tag{4.6}$$

$$\begin{bmatrix} A_0 & A_1 & A_2 & A_3 & A_4 \\ B_1 & 4 & 4 & 5 & 2 \\ B_2 & 3 & 1 & 11 & 8 \end{bmatrix}. \tag{4.7}$$

$$p^+ (2/2 \times 2/5 \to 2/2, \ 2/3 + 2/4 \to 2/3), \tag{4.8}$$

$$\begin{bmatrix} A_0 & A_1 & A_2 & A_3 & A_4 \\ B_1 & 8 & 9 & 5 & 2 \\ B_2 & 3 & 1 & 11 & 8 \end{bmatrix}. \tag{4.9}$$

$$p^+ (2/4 + 2/3 \to 2/4, \quad 2/2 + 2/4 \to 2/2), \tag{4.10}$$

$$\begin{bmatrix} A_0 & A_1 & A_2 & A_3 & A_4 \\ B_1 & 19 & 9 & 6 & 2 \\ B_2 & 3 & 1 & 11 & 8 \end{bmatrix}. \tag{4.11}$$

$$p^{+}\,(2/5 \times 2/2 \rightarrow 2/5), \tag{4.12}$$

$$\begin{bmatrix} A_0 & A_1 & A_2 & A_3 & A_4 \\ B_1 & 38 & 9 & 6 & 2 \\ B_2 & 3 & 1 & 11 & 8 \end{bmatrix}. \tag{4.13}$$

$$p^{+}\,(2/4 + 2/2 + 2/3 \rightarrow 2/4), \tag{4.14}$$

$$\begin{bmatrix} A_0 & A_1 & A_2 & A_3 & A_4 \\ B_1 & 38 & 9 & 53 & 2 \\ B_2 & 3 & 1 & 11 & 8 \end{bmatrix}. \tag{4.15}$$

Skaičiavimo rezultatas 53, kuris pateiktas koordinatėse 2/4.

Bendrojoje infologinių algoritminių priklausomybių išraiškoje duomenų lyginimo simbolis gali būti įrašytas tik vieną kartą vietoj bet kurio skaičiavimo operacijos ženklo ψ. Pavyzdžiui,

$$p^{=}(2/3 + 2/2 > 3/4 \times 3/2), \tag{4.16}$$

šių koordinačių reikšmės:

$$9 + 38 > 11 \times 3. \tag{4.17}$$

Kadangi šioje grupėje glaudžiai siejasi skaičiavimo operacijos ir informacijos logika, išreiškianti realaus pasaulio objektus ir procesus, todėl aprašytos algoritminės priklausomybės vadinamos skaičiavimo-infologinėmis priklausomybėmis.

4.3. Nuosavos atributų reikšmių algoritminės priklausomybės

Atsižvelgiant į tai, kad konkrečios atributo reikšmės gali kisti sprendžiant skirtingus uždavinius, buvo išskirtos atributų reikšmių algoritminės priklausomybės, kurios pavadintos nuosavomis. Kaip jau buvo minėta, šią grupę sudaro viena nuo kitos labai besiskiriančios algoritminės priklausomybės, kurios tik gana palyginti sąlyginai pastovios. Detaliau šiame poskyryje aprašomos tik dažniausiai praktikoje pasitaikančios nuosavos atributų reikšmių algoritminės priklausomybės ir jų formalios išraiškos.

I. Atributo reikšmė gali būti tapati vienai iš reikšmių, pateiktų pačios algoritminės priklausomybės formalioje išraiškoje:

$$p^{a}(c \vee \{c_i\}). \tag{4.18}$$

Pavyzdžiui, atributo reikšmė, apibūdinanti asmenį pagal lytį:

$$p^{a}(c \vee \{\text{vyr.; mot.}\}). \tag{4.19}$$

Apibrėžus šią algoritminę priklausomybę galima pagreitinti duomenų įvedimą. Tarkime, suvedame įrašus apie darbuotojus ir atribute „lytis“ turime įrašyti atitinkamą darbuotojo lytį. Realizavus šią algoritminę priklausomybę galime „Enter“ klavišu atkartoti prieš tai įvestą reikšmę naujos reikšmės vietoje.

II. Atributo reikšmės tipo nustatymas. Pavyzdžiui, reikšmė c yra skaičius – $p^{a}(s)$ arba atributo reikšmė yra tekstas – $p^{a}(t)$. Atributo reikšmę tikslinga apibrėžti kaip tekstą tik tada, kad tiksliai žinome, kad ji nedalyvaus aritmetinėje operacijoje. Laiko periodą arba laiko momentą nusakančius skaičius (datas) vienu atveju galima traktuoti kaip skaičius, o kitu – kaip tekstą. Pavyzdžiui, gimimo data gali būti traktuojama kaip tekstas, o amžiaus nustatymo operacijoje ta pati data turi dalyvauti kaip skaičius.

Jeigu atributo reikšmę apibrėšime kaip tekstą, tai jos vietoje gali būti įrašytas tiek tekstas, tiek skaičius, tiek simbolis. Įvedus atributo reikšmę, kurios duomenų tipas yra tekstinis, kitais programiniais moduliais bus atliekamas šios reikšmės duomenų tipo palyginamas su duomenų šaltinyje esančios atributo reikšmės tipu. Lyginamiems tipams sutapus, bus leidžiama įrašyti norimą atributo reikšmę. Jeigu deklaruojama, kad toje vietoje turi būti skaičius, bet joje norime įrašyti raidę arba simbolį, tai galutinis vartotojas informuojamas, kad tai yra klaida. Pavyzdžiui, jeigu norime kaupti duomenis apie valstybinius mašinų numerius, tai pirmuosius tris atributo reikšmės simbolius apibrėžiame kaip turinčius tekstinį duomenų tipą, o likusius tris – kaip skaitmeninį.

$$p^{a}(c = \mathit{tttsss}), \tag{4.20}$$

čia: c – skaičius; t – tekstinis duomenų tipas; s – skaitmeninis duomenų tipas.

III. Atributo reikšmė c gali būti apibrėžiama skaitiniame intervale nuo c_1 iki c_2:

$$p^{a}(c_1 \le c \le c_2). \tag{4.21}$$

Ši algoritminė priklausomybė gali būti plačiai naudojama duomenims, kurių skaitinių reikšmių kitimas galimas tam tikrame intervale. Pavyzdžiui, galime apriboti darbuotojo valandininko per dieną dirbtų valandų skaičių intervalu [0;8]:

$$p^{a}(0 \le c_{23} \le 8). \tag{4.22}$$

Jeigu darbuotojas per dieną dirbo daugiau nei aštuonias valandas, tai už viršytą valandų skaičių turėtų būti mokami viršvalandžiai. Kitaip tariant, įaršomai atributo reikšmei nepatekus į intervalo ribas, gaunamas pranešimas apie neatitikimus arba (ir) atliekama kita apibrėžta operacija neatitikimo atveju.

IV. Algoritminė priklausomybė

$$p^{a}(s) \tag{4.23}$$

nurodo s reikšmę, kokiu tikslumu gali būti apvalinamas skaičius. Pavyzdžiui, jei s = 1000, tai apvaliname tūkstančiais, s = 100 – apvaliname šimtais, s = 0,1 – apvaliname skaičiaus vienos dešimtainės tikslumu ir t. t.

Šios ir prieš šią ėjusios AP formali išraiška nėra tapati, nes p^{a} skiriasi savo viršutinio indekso a reikšme, pagal kurią ir atpažįstama procedūra bei programinis jos realizacijos modulis.

V. Algoritminė priklausomybė gali nurodyti dešimtainio kablelio (taško) vietą skaičiuje:

$$p^{a}(s), \tag{4.24}$$

čia s yra skaičius, nurodantis, kiek ženklų po kablelio turi turėti atributo reikšmė. Pavyzdžiui, s = 0 – sveikasis skaičius, s = 1 – skaičiaus tikslumas yra viena dešimtoji vieneto dalis ir t. t.

VI. Reliacinės aibės kortežų rūšiavimo algoritminė priklausomybė pagal j-ojo domeno reikšmių didėjimą, mažėjimą arba šabloną T:

$$p^{a}(j<), \quad p^{a}(j>), \quad p^{a}(j, T), \tag{4.25}$$

čia T = *K, KJ,* 12, 36, ... ir t. t. Tai reiškia, kad į pirmąsias RA kortežų vietas surenkami tokie kortežai, kurie j-ajame domene turi reikšmes *K*, po to *KJ*, 12, 36 ir t. t.

Pažymėtina, kad ir išvystytose, ilgai eksploatuotose sistemose būtent nuosavų atributų reikšmių algoritminių priklausomybių skaičių ir įvairovę tenka nuolat plėsti, nes duomenų savybių įvairovė praktiškai neribojama.

VII. Atributo reikšmei gali būti apibrėžiama skaičiaus iškraipymo kontrolė pagal papildomą skaitmenį

$$p^{a}(Xs). \tag{4.26}$$

Prie skaičiaus X prirašomas tik šio skaičiaus absoliučiam didumui ir skaitmenų išdėstymui būdingas skaitmuo, kuris gaunamas pagal formulę (s – paskutinis skaitmuo iš jau šiuo skaitmeniu papildyto skaičiaus):

$$Xs = 2^{1}s_1 + 2^{2}s_2 + 2^{3}s_3 + \ldots \quad , \tag{4.27}$$

čia s_1, s_2, ... yra skaičiaus X skaitmenys, pateikti natūralia eilės tvarka.

VIII. Algoritmine priklausomybe

$$p^a(\text{STOP}, c\,\theta\,(i/j)) \quad \text{arba} \quad p^a(\text{STOP}, c\,\theta\,(j)) \tag{4.28}$$

galime nurodyti, kad duomenų apdorojimas yra sustabdomas, jei yra tenkinama sąlyga θ, kurių galimi variantai pilnai atitnka tranformacijose taikomas sąlygas. Ši AP leidžia vartotojui įsikišti į automatinį uždavinio sprendimą norint koreguoti duomenis ar jų apdorojimo kryptis.

IX. Atributo reikšmė gali būti lygi vienai iš normatyvinės RA domeno j reikšmių:

$$p^a(c \vee \{j\}). \tag{4.29}$$

Ši AP naudojama tikrinti, kad pateikiant reikšmę nebūtų padaryta mechaninės klaidos. Šiuolaikiniuose kompiuteriuose ši operacija dažniausiai atliekama vedant kursorių specialiais klavišais per normatyvinius duomenis. Bet tuo atveju, kai normatyvinės RA kortežų skaičius labai didelis, racionaliau atributo reikšmę pateikti klaviatūra, o aprašomos algoritminės priklausomybės procedūrą atlikti jos programiniu moduliu.

Tarkime, turime normatyvinę duomenų lentelę apie bibliotekoje kaupiamų knygų identifikavimo kodus. Suteikiant naujai knygai kodą ar tikrinant esamų knygų kodus kyla būtinybė, kad jis sutaptų su vienu iš normatyvinėje lentelėje pateiktų kodų.

Šią AP galima apibrėžti ir tokia išraiška:

$$p^a(c \vee \{j, j'\}), \tag{4.30}$$

čia j' reiškia, kad suradus normatyvinės informacijos RA domene j, reikšmei c tapačią reikšmę, nesitenkinama tokio fakto konstatavimu, o į formuojamą duomenų struktūrą iškviečiama normatyvinė to paties kortežo reikšmė, esanti domene j'.

X. Atributo reikšmė (pradedant pirmąja) pačiame domene gali būti kartojama tol, kol sistema gauna išorinį signalą, kad ta reikšmė keisis:

$$p^a(=j). \tag{4.31}$$

Pažymėtina, kad ir išvystytose, ilgai eksploatuotose sistemose būtent nuosavų atributų reikšmių algoritminių priklausomybių skaičių ir įvairovę tenka nuolat plėsti, nes duomenų savybių įvairovė praktiškai neribojama

4.4. Algoritminių duomenų priklausomybių reliacinė aibė

Algoritminių priklausomybių RA yra aibė, sudaryta iš atributų reikšmių algoritminių priklausomybių kodų, kurie atsižvelgiant į sprendžiamą uždavinį atitinkamai parenkami iš sistemoje turimos algoritminių priklausomybių kodų RA. Sprendžiant konkretų uždavinį jam spręsti suformuotoje duomenų RA r atributai yra siejami atsižvelgiant į šį uždavinį, todėl suprantama, kad RA r yra susieta su algoritminių priklausomybių aibe P_r. Algoritminių priklausomybių formaliose išraiškose turi būti nurodytos koordinatės, kuriose yra tuo momentu sprendžiamo uždavinio atributų reikšmės c_{ij} iš RA r. Algoritminių priklausomybių RA dar vadinama algoritmine reliacine aibe – ARA. Ji vienareikšmiškai lemia atributų reikšmių dalyvavimą tam tikroje duomenų apdorojimo operacijoje, kadangi kreipiantis į norimą kodą galima iškviesti tuo kodu apibrėžtą algoritminę priklausomybę ir ją realizuojantį programinį modulį.

Taigi sukūrus ARA konkrečiam uždaviniui spręsti ir programiškai einant tam tikra tvarka ir kryptimi pagal algoritminių priklausomybių kodus yra iškviečiami programiniai moduliai ir sprendžiamas uždavinys. Kiekvienam taikomajam uždaviniui spręsti parenkamos vis skirtingos ARA ir skirtinga tvarka bei būdai, kaip eiti konkrečia ARA sprendžiant tą uždavinį. Jeigu uždavinys sudėtingas ir jį sunku realizuoti viena ARA, tai naudojama ARA eilutė. Šioje eilutėje gretimas ARA jungia specifinės algoritminės priklausomybės, kurios vadinamos išorinėmis algoritminėmis priklausomybėmis.

Tarkime, kad konkrečios probleminės srities duomenų aibės yra

$$\{ r_1, r_2, \ldots, r_t, \ldots \}.$$

Duomenų apdorojimo uždaviniai:

$$\{ z_1, z_2, \ldots, z_t, \ldots \}.$$

Tada patį paprasčiausią, elementarų uždavinį galima užrašyti taip:

$$z_1 = P_r \rightarrow r,$$

čia z_1 – uždavinio pavadinimas; P_r – algoritminė reliacinė aibė; r – reliacinė duomenų aibė.

RA r esančioms atributų reikšmėms $<c_{ij}>$ apdoroti parenkamos reikalingos algoritminės priklausomybės, o jų kodai $<p_j>$ surašomi į ARA P_r. Tam tikra tvarka programiškai einant per ARA P_r ir vykdant atitinkančius algoritminių priklausomybių programinius modulius sprendžiamas uždavinys z_1. Tokio elementaraus uždavinio pavyzdys gali būti RA r duomenų korektiškumo patikrinimas.

Sudėtingesnis uždavinys už elementarųjį yra tada, kai tie patys duomenų algoritminių priklausomybių programiniai moduliai apdoroja daugelį tos pačios

duomenų RA schemos egzempliorių (t. y. apdoroti naudoja tą pačią schemą su skirtingų subjektų *b* ir skirtingų laiko faktorių *d*). Tokio uždavinio formalus užrašymas gali būti

$$z_a \rightarrow P_a\,,\, r_a(b_k\,,\, d_l\,).$$

Čia uždavinio kodas, ARA kodas ir RA *r* schemos kodas yra *a*, kur egzempliorių skaičius priklauso nuo subjektų ir laiko veiksnių *k* ir *l* kombinacijų skaičiaus.

Pats sudėtingiausias duomenų apdorojimo uždavinys sprendžiamas tada, kai dėl uždavinio algoritmo sudėtingumo, duomenų įvairovės ir gausos tenka naudoti ne vieną, o ištisą aibę skirtingų ARA, taip pat RA skirtingų schemų.Kiekviena iš schemų turi iš anksto nežinomą skirtingų egzempliorių skaičių. Tada formali uždavinio išraiška bus tokia:

$$z = X < P\ \psi\ P\ \psi \ldots\ \psi\ P\ \psi\ P >.$$

Čia *X* yra RA identifikatorių aibė, kurie naudojami visam uždaviniui *z* spręsti. Tiems duomenims (t. y. atributų reikšmėms c_{ij}) apdoroti naudojama eilutė išorinių algoritminių priklausomybių ψ, kurios nustato informacinius santykius tarp eilutėje greta einančių *P*.

4.5. Išorinės algoritminės duomenų priklausomybės

Galima sudaryti vieną algoritminių priklausomybių kodų RA praktiškai nesudėtingam uždaviniui spręsti. Jeigu uždavinys yra sudėtingas, į vieną ARA nepaprastai sunku derinti įvairių algoritminių priklausomybių kodus su visiškai skirtingomis realizacijomis naudojant daugelį įvairių schemų, subjektų ir laiko tarpo RA, todėl sprendžiant sudėtingą uždavinį sprendimas supaprastinamas sukuriant ARA eilutę. Kiekviena ARA sprendžia tam tikrą sudėtingo uždavinio dalį. Tada atsiranda būtinybė ir visas ARA, ir dalinius sprendimus sujungti į vieno ir to paties uždavinio galutinį sprendimą. Todėl naudojama tarpinė išorinė algoritminė priklausomybė, kuri jungia eilutėje dvi greta esančias ARA. Pačias išorines algoritmines priklausomybes žymime ψ ir galime jas klasifikuoti į keletą priklausomybių, kurias gali realizuoti išoriniai programiniai moduliai.

Išorinių algoritminių priklausomybių gali būti labai daug. Jos gali nurodyti informacinius santykius tarp eilutėje greta esančių ARA šiais klausimais:

- keli *P* naudojami vienos schemos RA *r* apdoroti;
- kelis skirtingų schemų RA *r* apdoroja viena *P*;
- kokiu apdorojimo momentu *r* RA keičiama kita;
- kokiu apdorojimo momentu ARA keičiama kita *P*;
- kaip sujungiami tarpiniai apdorojimo rezultatai tarpinių rezultatų duomenų lauke, t. y. ar bus iš dalies perdengiami duomenys, ar bus užrašomi ant turimų duomenų viršaus;
- kaip naudojami duomenų apdorojimo rezultatai ir kt.

Pradžioje kol kas apibrėžtos tik kelios išorinės algoritminės priklausomybės, tačiau sistemai tobulėjant išorinių AP gali atsirasti naujų.

Taigi pirmiausia įvedami trys įvardinti duomenų laukai, iš kurių: pirmajame yra duotuoju momentu nagrinėjama ARA, antrajame – duotuoju momentu nagrinėjama RA, trečiajame saugomi tarpiniai apdorojimo rezultatai, kol išsekus ARA eilutei trečiajame lauke bus gautas galutinis uždavinio sprendimo rezultatas. Toliau, atsižvelgiant į tai, kokią išorinė AP yra apibrėžta ARA eilutėje, t. y. kokius informacinius santykius ši išorinė AP apibrėžia, atitinkamai bus keičiami duomenys tuose trijuose laukuose (4.1, 4.2 pav.).

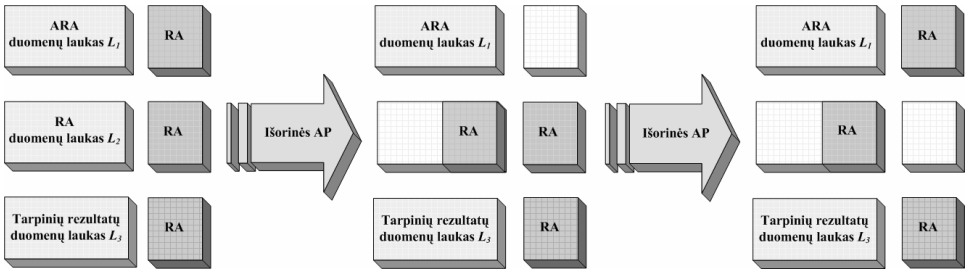
Tarkime turime ARA eilutę:

$$\text{ARA}\ \ \psi_1\ \ \text{ARA}\ \ \psi_2\ \ \text{ARA},$$

čia ψ_1, ψ_2 yra pažymėtos skirtingo pobūdžio išorinės AP. Detaliau aiškinti išorines AP naudojami tokie žymenys:

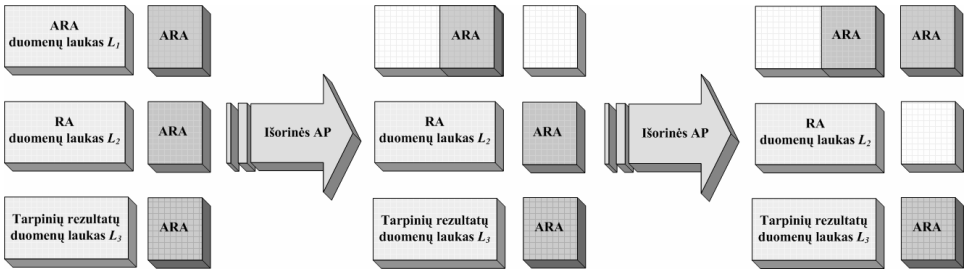
- kintančios ARA ir RA → $A\tilde{RA}$ ir $\tilde{RA}$;
- ASA ir SA liekančios pastoviomis → $\overline{ARA}$ ir $\overline{RA}$;
- tarpiniai apdorojimo rezultatai → $c^{\rightarrow}$;
- laukas, kuriame duotuoju momentu yra nagrinėjama ARA → L_1;
- laukas, kuriame duotuoju momentu yra nagrinėjama RA → L_2;
- laukas, kuriame saugomi tarpiniai ARA ir RA apdorojimo rezultatai arba galutinis uždavinio sprendimo rezultatas → L_3.

1. Išorinės AP ψ_1 algoritmas gali nurodyti, kad lauke L_1 iškviesta iš eilutės ARA lieka vis ta pati ($\overline{ARA}$), o apdoroti į lauką L_2 kviečiamos vis kitos tos pačios schemos RA ($\tilde{RA}$). Gauti duomenų apdorojimo rezultatai ($c^{\rightarrow}$) atsimenami lauke L_3. Scheminis vaizdas pateiktas 4.1 pav.



4.1 pav. *Išorinės AP nurodyta duomenų apdorojimo seka (I variantas)*

2. AP ψ_2 apibrėžia, kad lauke L_2 iškviesta ir apdorojama viena ir ta pati RA ($\overline{RA}$), naudojant į lauką L_1 iš eilutės kviečiamas vis kitas ARA ($A\tilde{RA}$). Tarpiniai rezultatai ($c^{\rightarrow}$) atsimenami L_3.



4.2 pav. *Išorinės AP nurodyta duomenų apdorojimo seka (II variantas)*

3. Duomenų šaltinyje atsiradus naujai RA su kita schema, kaip įprasta, keičiasi ir ARA. Įvykus tam pasikeitimui reikia grįžti vėl į pirmą ir antrą punktą ir žiūrėti, ar reikia keisti RA ir ARA.

4.6. Programinių ėjimų algoritminės duomenų priklausomybės

Programinių ėjimų AP yra žymimos $p^{\rightarrow}$ ir nurodo, kokia seka bus kreipiamasi į algoritminėje reliacinėje aibėje pateiktus AP kodus, pagal kuriuos atitinkamai bus iškviečiami jas realizuojantys programiniai moduliai.

Tuo atveju, kai realizuojamos nuosavos atributų reikšmių algoritminės priklausomybės, programinis ėjimas – jo pradžia ir kryptis dažniausiai nėra svarbūs. Kitu atveju, kadangi daugelis algoritminių priklausomybių naudoja duomenis, esančius įvairiose RA koordinatėse, būtina programinį ėjimą pradėti reikiamose ARA koordinatėse ir tęsti reikiama kryptimi. Į tai neatsižvelgus, duomenų apdorojimo procesas tampa nekorektiškas arba neįmanomas. Pavyzdžiui, tegu į RA koordinates k / l bus išsaugomas skaičiavimo algoritminės priklausomybės p_1^{+} duomenų apdorojimo rezultatas. Jeigu koordinačių k / l reikšmė reikalinga kitai algoritminės infologinės priklausomybės $p_2^{=}$ realizacijai, tai šių algoritminių priklausomybių realizavimo eilė būtinai turi būti p_1^{+}, $p_2^{=}$. Atvirkštinė realizavimo eilė $p_2^{=}$, p_1^{+} negalima, kadangi AP $p_2^{=}$ dar nebus duomens koordinatėse k / l.

Tarkime, yra algoritminė RA:

$$P = \begin{pmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{pmatrix}. \qquad (4.32)$$

Galima išskirti aštuonias programinio ėjimo kryptis pradedant tokius ARA kampinėse koordinatėse:

$$p^{\rightarrow} = p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7 \quad p_8 \quad p_9 ; \tag{4.33}$$

$$p^{\rightarrow} = p_3 \quad p_2 \quad p_1 \quad p_6 \quad p_5 \quad p_4 \quad p_9 \quad p_8 \quad p_7 ; \tag{4.34}$$

$$p^{\rightarrow} = p_7 \quad p_8 \quad p_9 \quad p_4 \quad p_5 \quad p_6 \quad p_1 \quad p_2 \quad p_3 ; \tag{4.35}$$

$$p^{\rightarrow} = p_9 \quad p_8 \quad p_7 \quad p_6 \quad p_5 \quad p_4 \quad p_3 \quad p_2 \quad p_1 ; \tag{4.36}$$

$$p^{\rightarrow} = p_1 \quad p_4 \quad p_7 \quad p_2 \quad p_5 \quad p_8 \quad p_3 \quad p_6 \quad p_9 ; \tag{4.37}$$

$$p^{\rightarrow} = p_3 \quad p_6 \quad p_9 \quad p_2 \quad p_5 \quad p_8 \quad p_1 \quad p_4 \quad p_7 ; \tag{4.38}$$

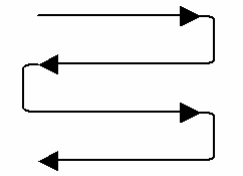
$$p^{\rightarrow} = p_7 \quad p_4 \quad p_1 \quad p_8 \quad p_5 \quad p_2 \quad p_9 \quad p_6 \quad p_3 ; \tag{4.39}$$

$$p^{\rightarrow} = p_9 \quad p_6 \quad p_3 \quad p_8 \quad p_5 \quad p_2 \quad p_7 \quad p_4 \quad p_1 . \tag{4.40}$$

Suprantama, kad, parinkus programinio ėjimo pradžia bet kurias kitas koordinates, krypčių skaičius padidėtų, bet praktika rodo, kad ir šie parinkimai labai reti. Dažniausiai pasitaikantys ėjimai yra:

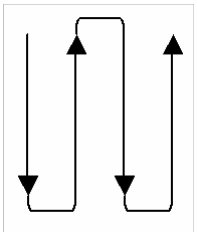
- Pradedant viršutinio kortežo kairiuoju elementu ir einant į dešinę:

$$p^{\rightarrow} = p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7 \quad p_8 \quad p_9 , \tag{4.41}$$



- Pradedant viršutinio kortežo kairiuoju elementu ir t einant žemyn:

$$p^{\rightarrow} = p_1 \quad p_4 \quad p_7 \quad p_2 \quad p_5 \quad p_8 \quad p_3 \quad p_6 \quad p_9 , \tag{4.42}$$



Algoritminei RA P_r, kai

$$P_r < p_1 \quad p_2 \quad p_3 >,$$

programinis ėjimas kartojamas tiek kartų kryptimi

$$p_1 \quad p_2 \quad p_3 ,$$

kiek kortežų turi apdorojamoji ARA r , todėl programinius ėjimus galime užrašyti

$$p^{\rightarrow} = p_1 \; p_2 \; p_3 \; \ldots \; p_1 \; p_2 \; p_3 \; \ldots p_1 \; p_2 \; p_3 \; \ldots, \qquad (4.43)$$

arba

$$p^{\rightarrow} = p_1 \; p_1 \; p_1 \; \ldots \; p_2 \; p_2 \; p_2 \; \ldots p_3 \; p_3 \; p_3 \; \ldots . \qquad (4.44)$$

Galimi ir nenuoseklūs programiniai ėjimai, kurie iš esmės nekeičia algoritminių priklausomybių realizavimo principo. Pavyzdžiui, jų reikia tada, kai įvedant duomenis į RA įsivelia klaidų. Jei RA matmenys gana dideli, tai RA nepateikiama iš pradžių, o nurodomos tos algoritminės RA vietos, kur įvyko klaida. Tad automatiškai pergeneruojamos algoritminių priklausomybių aibės ir toliau p realizuojamos programiškai.

Visur, nagrinėjant $p^{\rightarrow}$, algoritminės priklausomybės buvo žymimos p_1, p_2, p_3, ... tik trumpumo dėlei. Faktiškai kiekviena jų yra tvarkioji algoritminių priklausomybių aibė $< p >$.

4.7. Automatinis uždavinių sprendimų valdymas

Automatinis uždavinių sprendimų valdymas skirstomas į du etapus. *Pirmuoju etapu* galima patikrinti automatinį duomenų šaltinį papildžiusios RA duomenų teisingumą ir korektiškumą. Kai į pirminių duomenų šaltinį įvedama nauja RA, jai priskirti identifikatoriai bus užfiksuoti identifikatorių RA. Tada automatiškai bus iškviečiama tam tikra programinių ėjimų algoritminė priklausomybė, kuri nurodys, kokia tvarka ir kokia eile reikia tikrinti duomenų teisingumą, korektiškumą ir pan. Kitaip tariant, bus nurodyta, kokias algoritmines priklausomybes reikia iškviesti ir kokia seka jos turi būti realizuojamos. Tokio tipo algoritminės priklausomybės su RA pavyzdžiais yra aprašytos 4.6 poskyryje. Tarkime, kad į duomenų šaltinį naujai įvestos RA tam tikro atributo reikšmės būtinai turi būti tam tikrame skaitiniame diapazone. Skaitinių reikšmių atitiktį apibrėžtam intervalui galima patikrinti realizavus vieną iš nuosavų atributų reikšmių algoritminių priklausomybių. Tuo remiantis galima teigti, kad vieną kartą suprojektuotas duomenų kontrolės tikrinimas atliekamas automatiškai ir tą automatišką atlikimą inicijuoja naujų duomenų RA atsiradimas duomenų šaltinyje.

Antruoju etapu atliekamas uždavinių, skirtų konkrečiam naudotojui, sprendimo valdymas. Priklausomai nuo situacijos tai gali būti atliekama keliais būdais:

1. Norimo uždavinio sprendimo vykdymą iškviečia pats sistemos naudotojas, nurodydamas tik uždavinio kodą. Pagal nurodytą kodą sistema automatiškai pradeda vykdyti konkretaus uždavinio sprendimą.

2. Kai uždavinio sprendimo vykdymas priklauso nuo analizės rezultatų esant tam tikram laiko momentui. Pavyzdžiui, uždaviniui vykdyti gali būti numatyta tam tikra diena. Atėjus apibrėžtam momentui automatiškai bus pereinama prie apibendrintosios duomenų užklausos vykdymo. Ji bus skleidžiama iki konkrečių RA identifikatorių. Duomenų šaltinyje radus visas tam uždaviniui spręsti reikalingas RA, transformacijos leis formuoti pirminius duomenis ir toliau bus sprendžiamas uždavinys sprendimas, t. y. pagal konkrečiam uždaviniui priskirtą programinių ėjimų algoritminę priklausomybę iš visos algoritmines priklausomybes realizuojančios programinių modulių aibės bus iškviečiami ir realizuojami tik sprendžiamam uždaviniui skirti programiniai moduliai.

3. Konkrečiam uždaviniui spręsti suformuota reikalingų reliacinių aibių identifikatorių aibė. Pagal identifikatorius galime nustatyti, kokių aibių trūksta. Duomenų šaltinį papildžius nauja RA, ji bus identifikuojama ir identifikatoriai bus užfiksuoti RA identifikatorių aibėje. Tuo atveju, kai duomenų šaltinis bus papildytas visomis reikalingomis RA, t. y. tam tikro laikotarpio arba tam tikro objekto, arba abiejų pažymių kartu, bus automatiškai pereinama prie reikiamų RA iškvietimo ir transformacijomis atliekamo konkretaus uždavinio pirminių duomenų formavimo. Po to bus sprendžiamas uždavinys. Šiuo atveju sistemos naudotojas tiesiogiai neprisideda prie uždavinio sprendimo iškvietimo. Sprendžiama automatiškai.

Minėti veiksmai jokiu būdu nepašalina algoritminių priklausomybių ir jų modulių šablonų kaupimo ir kitų būdų, palengvinančių uždavinių projektavimą. Akivaizdu, kad disertacijoje aprašyti metodai ir būdai labai praplečia iki šiol taikytų adaptacijos metodų teoriją ir galimybes.

4.8. Ketvirtojo skyriaus išvados

Šiame skyriuje pateiktos algoritminės duomenų priklausomybės ir jas realizuojantys programiniai moduliai sudaro adaptyvųjį duomenų apdorojimo projektavimo modelį. Taikant šį modelį, naudojant vienus ir tuos pačius algoritmines priklausomybes realizuojančius programinius modulius ir keičiant tik jų naudojimo seką bei kiekius, galima spręsti skirtingų naudotojų skirtingų probleminių sričių taikomuosius uždavinius, kurių duomenys yra išreikšti RA. Tai parodo naudojamų metodų efektyvumą.

Skirtingų projektuotojų skirtingose probleminėse srityse AP aibė gali būti plečiama ir papildoma visiškai kitokiomis naujomis algoritminėmis priklausomybėmis, kaip įvairių matematinių reiškinių realizavimo algoritminiai programiniai moduliai, ekonominių procesų ar kitokie programiniai moduliai. Todėl skirtingose probleminėse srityse algoritminių priklausomybių aibės gali

būti kitokios. Tai išplečia sukurtos adaptyviosios duomenų apdorojimo projektavimo technologijos naudojimo galimybes.

Konkrečioje probleminėje srityje programinių modulių naudojimo poaibių kombinacijos gali būti tipinės tai probleminiai sričiai. Tokiu būdu atsiranda galimybė automatizuoti projektavimą, kadangi kiekvieną kartą nebereikia nurodyti vis naujos algoritminių priklausomybių sekos, nes galima identifikuoti tokias sekas ir pakartotinai jas naudoti.

Išorinės algoritminės priklausomybės koncepcija leidžia sudėtingų uždavinių algoritminių priklausomybių aibes skaidyti į mažesnes aibes ir pateikti jas eilute, kurioje gretimos algoritminių priklausomybių aibės sujungiamos išorine algoritmine priklausomybe. Tai padeda greičiau suprojektuoti sudėtingą uždavinį, nes kiekvieną algoritminių priklausomybių aibę gali realizuoti skirtingi projektuotojai, dirbdami tuo pat metu.

5

Adaptyviosios technologijos taikymas. Pavyzdžiai

5.1. Bendrieji taikomojo pobūdžio uždavinių dėsningumai

Šiame skyriuje pateikta adaptyviosios duomenų apdorojimo projektavimo technologijos taikymo praktikoje pavyzdžių. Jie gali būti laikomi vienu ir tik vienu duomenų apdorojimo uždaviniu. Kitaip tariant, taikant vieną ir tą patį principą galima atlikti skirtingų taikomojo pobūdžio uždavinių sprendimus, kur tiek pirminiai duomenys, tiek rezultatai yra išreikšti reliacinėmis aibėmis. Tai atliekama trimis pagrindiniais etapais. Pirmiausia yra realizuojamas duomenų išrinkimo modelis, leidžiantis iš duomenų šaltinio atrinkti konkrečiam uždaviniui reikiamas RA, kuriose yra duomenų (t. y. atributų reikšmių) šiam uždaviniui spręsti. Kadangi RA yra kuriamos savo struktūra ir turiniu kaip tam tikros probleminės srities duomenų struktūros, tai konkrečiam uždaviniui spręsti jos turi ir konkrečiam taikymui nereikalingų arba perteklinių duomenų. Todėl tuo pačiu etapu toliau taikomas duomenų agregavimo modelis, kuris transformacijų dėka leidžia sudaryti vieną duomenų RA, kurioje perkeltos tik konkrečiam uždaviniui spręsti reikalingos atributų reikšmės. Kitu, t. y. antru, etapu taikomas duomenų apdorojimo projektavimo modelis. Juo remiantis iš visos algoritminių priklausomybių aibės atrenkamos tik tos AP, kurios reikalingos pagal konkretaus

uždavinio sprendimo algoritmą. Kartu atrenkami ir šias AP realizuojantys programiniai moduliai, kurie ir sudaro uždavinio sprendimo programą. Taigi paskutiniu trečiuoju etapu, realizavus konkretaus taikomojo pobūdžio sprendimui sudarytą programą apdorojami duomenys, kurie yra vienoje RA. Šios programos projektavimas pagrįstas struktūrizavimo principu, nes ji sudaroma iš modulių, kurie vienas nuo kito yra nepriklausomi ir į visumą gali būti jungiami įvairiomis kombinacijomis.

Kadangi konkretaus uždavinio sprendimas realizuojamas keičiant RA identifikavimo, RA transformacijų, algoritminių priklausomybių ir jas realizuojančių programinių modulių parinkimo parametrus. Todėl akivaizdu, kad tai efektyviau už tiesioginį kiekvieno uždavinio sprendimo programavimą.

Numatomas ir kitoks aprašytos adaptyviosios duomenų apdorojimo projektavimo technologijos taikymo metodas, kuris iš esmės pagrįstas tuo pačiu veikimo principu, tačiau jį taikant taikomojo pobūdžio uždavinio programa sudaroma automatiškai. Toliau šiame poskyryje apibrėžiami adaptyviosios duomenų apdorojimo projektavimo technologijos kūrimo ir naudojimo etapai, kurie pagrįsti tik ką minėtu metodu.

Adaptyviosios duomenų apdorojimo projektavimo technologijos kūrimo etapai:

I. Sudaroma tuščia RA, kuri yra bendra bet kokio taikomojo uždavinio pirminiams duomenims (5.1 pav.). Šios RA struktūra nėra tokia pati, kaip tipinės RA, kadangi neturi atributų ir atitikties tarp atributų ir jų reikšmių. Taip yra todėl, kad į šią RA atliekant transformacijas, perkeliamos konkrečiam taikomajam uždaviniui spręsti reikiamos atributų reikšmės. Vadinasi, tame pačiame stulpelyje gali būti pateikiamos skirtingų atributų reikšmės iš pirminių duomenų šaltinio. Šios RA rangas (domenų skaičius) nėra ribotas, tačiau eilė (kortežų skaičius) priklauso nuo adaptyviojoje duomenų apdorojimo technologijoje turimų algoritminių priklausomybių skaičiaus. Kitaip tariant, prie kiekvieno šios RA kortežo turi būti pateiktas tam tikros algoritminės priklausomybės kodas (pavadinimas), kuris bus kortežo identifikatoriumi, t. y. raktu.

II. Šios RA kortežo raktas vienareikšmiškai lemia, pagal kokią algoritminę priklausomybę apdorojami tame kortеže pateikti duomenys ir koks programinis modulis tai realizuoja. Rakto sąsaja su algoritmines priklausomybes realizuojančių programinių modulių aibe realizuojama per valdymo operaciją, kuri pagal kortežo raktą (t. y. algoritminės priklausomybės kodą) automatiškai vykdyti pakviečia AP realizuojantį programinį modulį.

III. Algoritmines priklausomybes realizuojantys programiniai moduliai turi atitikti toliau išvardytus jiems keliamus bendruosius reikalavimus:

- Programinis modulis turi turėti įvardytą duomenų lauką (t. y. duomenų koordinates pirminių duomenų šaltinyje), kuriame pateikti duomenys, reikalingi realizuojant tą programinį modulį.

- Programinis modulis turi turėti įvardytą duomenų lauką (t. y. duomenų koordinates), kuriame bus pateikiami gauti rezultatai atlikus operaciją. Šis duomenų laukas gali būti nurodytas arba rezultatų RA (RRA), arba pirminių duomenų šaltinyje. Atsižvelgiant į tai, rezultatai yra išsaugomi viename iš jų.

IV. Kadangi programiniame modulyje gali būti nurodyta gautus rezultatus įsiminti rezultatų RA, tai būtina iš anksto sudaryti tokią tuščią RA (5.1 pav.). Šios RRA konkretūs atributų pavadinimai bus apibrėžti sprendžiant konkretų taikomąjį uždavinį, t. y. atsižvelgiant į sprendžiamo uždavinio poreikius.

V. Pirminių duomenų RA su programinių modulių aibe yra susieta naudojant valdymo operaciją, kuri kreipiasi į kiekvieną iš eilės einantį RA kortežo raktą, pagal kurį nustato, kokia algoritminė priklausomybė turi būti realizuota (5.1 pav.). Atsižvelgiant į tai, iš visos programinių modulių aibės valdymo operacija automatiškai vykdyti pakviečia šią AP realizuojantį programinį modulį. Tokiu pačiu principu vykdyti automatiškai iškviečiami visi reikiami programiniai moduliai, kurių visuma sudaro konkretaus taikomojo uždavinio sprendimo programą.

Taigi konkretus taikomasis uždavinys, naudojant sukurtą adaptyviąją duomenų apdorojimo projektavimo technologiją, sprendžiamas taip:

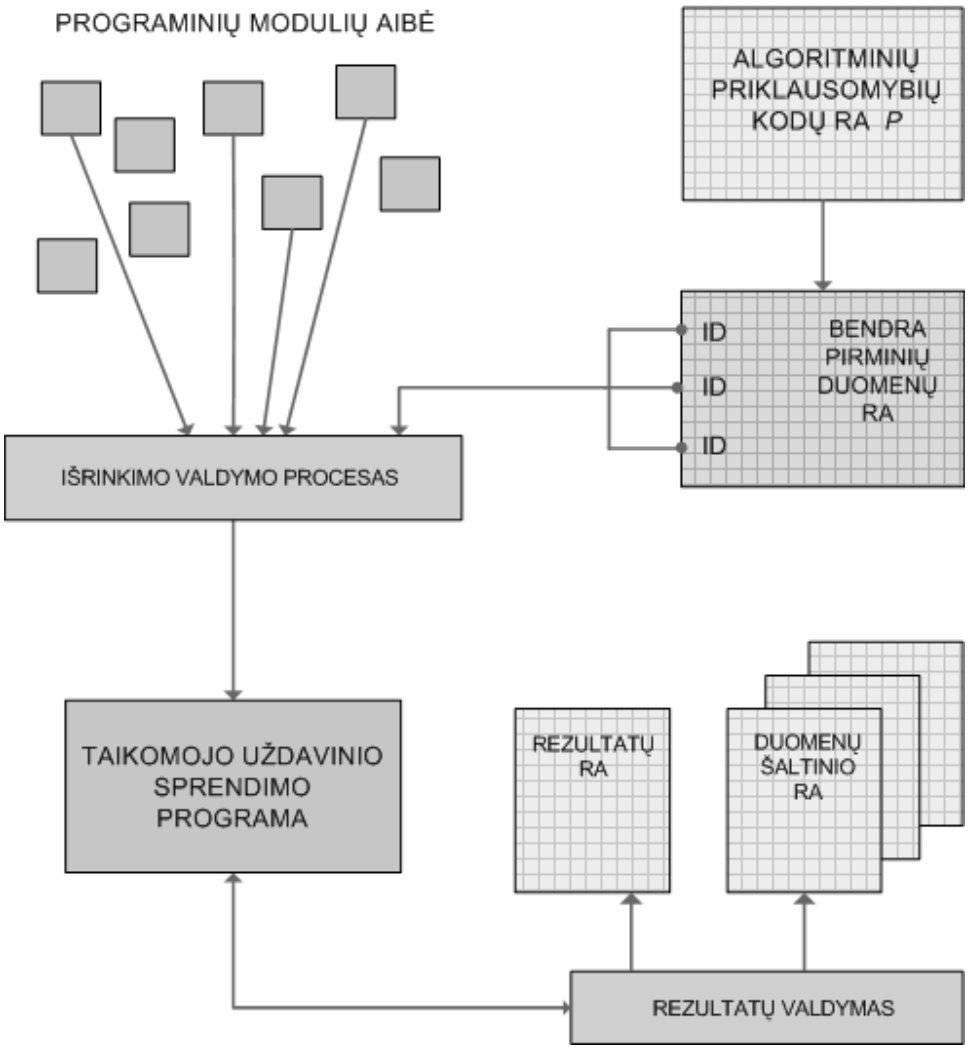
I. Atsižvelgiant į probleminę sritį sudaromas konkretaus uždavinio sprendimo algoritmas.

II. Remiantis sudarytu algoritmu ir naudojant duomenų išrinkimo modelį iš visų pirminių duomenų RA atrenkamos tik šiam uždaviniui spręsti reikalingos RA. Tai atlikti leidžia RA identifikatoriai.

III. Remiantis sudarytu algoritmu ir taikant transformacijas, t. y. naudojant duomenų agregavimo modelį, į jau turimą tuščią pirminių duomenų RA (5.1 pav.) iš atrinktų RA perkeliamos reikiamos atributų reikšmės. Šios reikšmės, priklausomai nuo to, kokios algoritminės priklausomybės joms taikomos, atitinkamai perkeliamos į tam tikrus tų algoritminių priklausomybių kodais identifikuotus kortežus.

IV. Turint užpildytą pirminių duomenų RA, paleidžiama valdymo operacija, kuri iš visos algoritmines priklausomybes realizuojančių programinių modulių aibės pagal pirminių duomenų RA kortežų identifikatorius automatiškai vykdyti reikiama tvarka atrenka, iškviečia ir paleidžia tik tuos programinius modulius, kurie reikalingi šiam uždaviniui išspręsti (5.1 pav.).

V. Atrinktų programinių modulių visuma sudaro konkretaus taikomojo uždavinio sprendimo programą, kurią realizavus gaunami šio uždavinio sprendimo rezultatai (5.1 pav.).



5.1 pav. *Bendra taikomojo pobūdžio uždavinių sprendimo schema*

Tad galima teigti, kad taikomojo uždavinio programa yra automatiškai sudaroma kaip visų algoritminių priklausomybių programinių modulių aibės poaibis, todėl skirtingiems taikomiesiems uždaviniams spręsti nereikia kurti naujų programų. Taikant sukurtą adaptyviąją duomenų apdorojimo projektavimo technologiją, galima spręsti įvairius taikomojo pobūdžio uždavinius RA kontekste. Technologijos adaptyvumas sudaro galimybę atlikti įvairius skaičiavimus tik kitaip aprašius jau turimos technologijos mechanizmą, t. y. algoritminės priklausomybės išliks tos pačios, keisis tik bendrasis konkretaus uždavinio duomenų apdorojimo algoritmas, kuris ir paskatins uždavinį išspręsti.

Įvairiems uždaviniams spręsti naudojami tie patys algoritminių priklausomybių algoritmai ir fiziškai tie patys tų priklausomybių realizavimo programiniai moduliai. Sprendžiant vienokio pobūdžio uždavinius turimas algoritmines priklausomybes atrinksime vienaip, o sprendžiant kitokio pobūdžio uždavinius – kitaip. Tai leidžia plėsti sprendžiamų uždavinių nomenklatūrą nekuriant naujų programinių priemonių, o manipuliuojant turimais algoritminių priklausomybių programiniais moduliais. Konkrečiam uždaviniui realizuoti prireikus naujų algoritminių priklausomybių, turimų algoritminių priklausomybių RA gali būti lengvai papildyta. Papildymas leis ne tik išspręsti šį uždavinį, bet ir praplės turimos adaptyviosios technologijos galimybes, nes technologija, papildyta nauja algoritmine priklausomybe, su kitomis, jau esančiomis algoritminėmis priklausomybėmis duoda naujų, tiesiogiai neplanuotų galimybių. Tai reiškia, kad algoritminių priklausomybių aibė nėra uždara papildymams.

5.2. Medžiagų ir detalių atsargų kontrolės uždavinys

Šiame poskyryje, remiantis adaptyviąja duomenų apdorojimo projektavimo technologija, detaliai aprašytas uždavinio, susijusio su sandėlyje esančių atsargų kontrole, sprendimas. Turint sandėlyje kaupiamų atsargų didelę nomenklatūrą, atsiranda poreikis stebėti ir kontroliuoti turimų atsargų kiekius pagal jas identifikuojančias raktinių atributų reikšmes, kad vyktų efektyvus darbas versle ar kitoje veikloje. Pavyzdžiui, neretai būtina žinoti, ar sandėlyje yra pakankamas kiekis tam tikrų medžiagų, ar sandėlis neperkrautas medžiagomis, kurios mažiau naudojamos nei kitos ir pan. [76A].

Taigi, spręsdami atsargų kontrolės uždavinį, visų pirma naudodami pagrindinę RA, kuri vaizduoja sandėlyje einamuoju metu kaupiamų atsargų kiekius, t. y. einamąją sandėlio būklę, ir taikydami besąlyginę transformaciją, kuri bus atlikta kortežuose, padarome identišką pagrindinės RA kopija. Šiuo atveju bendroji besąlyginės transformacijos formulė yra:

$$p^{t}(< k / l >) \longrightarrow < i / j >), \tag{5.1}$$

čia: $< k / l >$ – koordinatės, iš kurių imami duomenys; $< i / j >$ – koordinatės, į kurias įkeliami duomenys. Šiuo atveju RA, iš kurios imami duomenis, yra tik viena, todėl išraiškoje $< k / l >$ indeksas n neturi prasmės.

RA kopijai priskiriame identifikatorius, kurie iš duomenų visumos leis atrinkti būtent šią RA. Šie identifikatoriai bus tokie patys, kaip ir pagrindinės RA, tik turės ženklinimą, kuris identifikuos kopiją. Taigi bus priskiriamas RA kaip savarankiško duomenų darinio kodas a'. Subjekto kodas B' identifikuos subjektą, kurio duomenims skirta RA. Šiuo atveju tai bus kodas, kuriuo nustatysime, būtent kokio sandėlio duomenys pateikti RA. Neatmetama galimybė, kad gali būti keli skirtingi tos pačios veiklos sandėliai. Laiko

momentą, kai buvo užfiksuoti RA pateikti duomenys apie atsargų kiekius, identifikuos laiko veiksnio kodas D'. Objektams, apie kuriuos RA yra pateikti duomenys, priskiriami raktiniai atributai, kurių reikšmės bus tų objektų identifikatoriai. Šiuo atveju kiekviena medžiaga ir detalė turi savo kodą ir pavadinimą, kurie identifikuos kiekvieną iš jų. Taigi, atributai „Gaminio kodas“ ir „Gaminys“ bus raktiniai atributai. Taip pat nustatoma ir koduojama lentelės schema, t. y. koduojami tam tikra eile surašyti atributai.

Kitas etapas yra periodiškas pagrindinės RA, kurioje bus fiksuojami medžiagų kiekiai tam tikru laiko momentu, praėjus vienodiems laiko periodams, kopijų darymas. Taigi viena kopija atspindės objektų kiekius, esančius sandėlyje tam tikru laiko momentu, praėjus apibrėžtam laiko periodui. Galutinis naudotojas pagal gamybos poreikius sprendžia, kokį laiko periodą apibrėžti. Taip pat jis nusprendžia ir kopijų darymo trukmę tam, kad susidarytų pakankamas kopijų skaičius, kad būtų galima spręsti apie sandėlio atsargų eikvojimą arba naudojimą ir papildymą laiku, kad tų atsargų tam tikram laiko periodą nepritrūktų. Suprantama, kad kuo didesnis laiko tarpų skaičius fiksuojamas RA kopijomis, tuo atsargų kiekio rezervas nustatomas patikimiau.

Kiekvienai padarytai kopijai priskiriami identifikatoriai. Reikia pažymėti, kad visas RA kopijas identifikuojantys kodai a' bus vienodi, nes jie identifikuos tą pačią RA. Taip pat ir subjektų kodai bus tokie patys, kadangi šiuo atveju naudojama RA, kurioje kaupiami duomenys apie vieno ir to pačio sandėlio atsargas. Skirsis RA kopijų laiko veiksnio kodai, kadangi skirtingas kodas d' identifikuos vis kitokį laiko momentą, kurio metu buvo užfiksuoti atsargų kiekiai. Taigi, galima teigti, kad vienas laiko veiksnio kodas identifikuos vieną kopiją. Skirtingas medžiagas ir detales identifikuojančios raktinių atributų reikšmės išliks tos pačios, o sandėlį papildžius naujomis dar neturėtomis medžiagomis ar detalėmis, jiems bus priskirti tie patys raktiniai atributai, skirsis tik jų reikšmės. Tuo atveju, jeigu tam tikra medžiaga ar detalė bus likviduota, t. y. jos sandėlyje nebeliks, tai ją identifikuojančios raktinių atributų reikšmės nebus niekam kitam priskirtos. Atsargų visiško likvidavimo atveju galutiniam naudotojui gali būti išsiųstas pranešimas informuojantis apie nulinį tos medžiagos ar detalės atsargos kiekį.

Taigi turėdami galutinio naudotojo apibrėžtą kopijų skaičių ir naudojant duomenų identifikatorius iš turimos duomenų visumos, t. y. iš visų turimų kopijų, išrenkame visus duomenis, susijusius su kiekvienos medžiagos kiekiu, kuris buvo periodiškai fiksuotas tam tikru laiko momentu praėjus vienodiems laiko periodams. Tarkime, kad sprendžiamam uždaviniui reikalingi duomenys apie visas vieno sandėlio atsargas, kurių turimas kiekis buvo fiksuojamas kas mėnesį ištisus metus. Taigi duomenys iš visų kopijų būtų išrenkami pateikus tokią identifikatorių seką:

$$a_1\, b_1\, d_{1\text{–}12}\,.$$

Naudodami atrinktus duomenis ir taikydami besąlyginę transformaciją, juos transformuojame iš kortežų į kortežus naujos RA, kurios duomenis naudosime tolesniems skaičiavimams:

$$p^{t}(< k / l >^{n} \longrightarrow < i / j >), \tag{5.2}$$

čia indeksas n reiškia aibės eilės numerį. Kadangi duomenų transformacija vykdoma iš keliolikos skirtingų aibių į vieną, tai $< k / l >$ koordinatėms, iš kurių perkeliami duomenys, indeksas n turi prasmę, o $< i / j >$ – ne.

Kitu etapu taikant algoritminę skaičiavimo priklausomybę sudedami kiekvienos medžiagos atskirai užfiksuoti visi kiekiai, siekiant apskaičiuoti bendrą kiekvienos medžiagos kiekį per visą stebėjimo periodą. Sudėties veiksmams taikomos skaičiavimo algoritminės priklausomybės bendroji formulė yra:

$$p^{+}(A_1 + A_2 + ... + A_n \rightarrow A_{n+1}), \tag{5.3}$$

čia A_n – atributo reikšmės koordinatės reliacinėje aibėje; + – sumavimo operacijos ženklas; → – nuoroda į koordinates, kur turi būti skaičiavimo rezultatas.

Gauti rezultatai atsimenami pagal atitinkamą medžiagą tos pačios, su kuria dirbame, RA naujame domene. Kitiems skaičiavimams vykdyti naudojama nuosava atributų reikšmių algoritminė priklausomybė p^{a}, kuri suskaičiuotų kiek turime kopijų. Taip rasime dydį, iš kurio dalinsime apskaičiuojamų medžiagų kiekių vidurkius, kadangi, kaip jau buvo minėta, viena kopija identifikuoja vieną laiko momentą. Šiuose skaičiavimuose taikytinos nuosavos atributų reikšmių algoritminės priklausomybės formulė yra:

$$p^{a}(count\ d^{/} \rightarrow A_k), \tag{5.4}$$

čia $count\ d^{/}$ reiškia, kad bus suskaičiuoti visi $d^{/}$ identifikatoriai ir gautas rezultatas bus išsaugotas RA atributo reikšmės koordinatėse A_k ; → – nuoroda į koordinates, kur turi būti pateiktas skaičiavimo rezultatas.

Paskutiniu etapu taikydami skaičiavimo algoritminę priklausomybę ir joje naudodami prieš tai skaičiavimuose gautus rezultatus, t. y. apskaičiuotą bendrą kiekvienos medžiagos kiekį ir daliklį, apskaičiuojame kiekvienos atskirai sandėlyje kaupiamos medžiagos kiekio vidurkį:

$$p^{+}(A_{n+1} / A_k \rightarrow A_{n+2}), \tag{5.5}$$

čia: A_n – atributo reikšmės koordinatės reliacinėje aibėje; A_k – kitos atributo reikšmės koordinatės RA, tik ši atributo reikšmė išreikšta bendrą kopijų skaičių, iš kurių buvo imami medžiagų kiekiai; / – dalybos ženklas; → – nuoroda į koordinates, kur turi būti pateiktas skaičiavimo rezultatas.

Naudodami apskaičiuotus vidurkius sudarome normatyvinę medžiagų kiekių RA, kurios duomenimis remdamiesi galime automatiškai kontroliuoti einamuoju

momentu sandėlyje esančių medžiagų kiekius ir jų kitimą. Pavyzdžiui, jeigu tam tikros medžiagos kiekio reikšmė yra lygi ir (arba) mažesnė už tos pačios medžiagos kiekio reikšmę, pateiktą normatyvinių duomenų RA, tai ją identifikuojanti raktinio atributo reikšmė iškeliama į RA, kur kaupiami duomenys apie medžiagas neatitikusias sąlygas. Tokiu būdu galutinis naudotojas turi informaciją apie medžiagas, kurių atsargas reikia papildyti. Kontrolė atliekama sulyginus pagrindinėje RA, t. y. šiuo metu atspindinčioje einamąjį sandėlio būklę, pateiktų atsargų raktinių atributų reikšmes su normatyvinės duomenų RA pateiktų atsargų identifikatoriais, t. y. raktinių atributų reikšmėmis. Joms sutapus, t. y. suradus tą pačią medžiagą, atliekama duomenų kontrolė naudojant sąlyginę transformaciją be transformacijos, kur atliekamas tik duomenų lyginimas:

$$p^{t}(\{\, k/l \,\}^{n} \xrightarrow{\geq} \{\, i/j \,\}^{n}). \qquad (5.6)$$

Čia lyginamos koordinačių $\{\, i/j \,\}$ ir $\{\, k/l \,\}$ reikšmės, šiuo atveju $\{\, i/j \,\}$ yra normatyvinės RA, $\{\, k/l \,\}$ – pagrindinės RA; simboliu $\geq$ apibrėžta lyginimo sąlyga. Jei sąlyga patenkinta, lyginamos kitos koordinatės, jei nepatenkinta – išvedama aibė tų medžiagų, kurių kiekis neatitiko sąlygos. Indeksas n reiškia aibės eilės numerį.

Bėgant laikui seną RA kopijų archyvą papildome naujomis kopijomis. Naudojant pastarųjų duomenis atnaujinama turima normatyvinių duomenų RA. Tai leidžia be didelių pastangų vykdyti sandėlyje turimų medžiagų ir detalių atsargų kontrolę, užtikrinti efektyvų darbą versle.

Toliau šiame poskyryje pateikiamas pavyzdys su konkrečiais duomenimis leis detaliau iliustruoti medžiagų ir detalių atsargų kontrolės uždavinio sprendimą.

Tarkime, kad kiekių kontrolę vykdysime atsargoms, esančioms viename sandėlyje. Šio sandėlio einamasis būvis pateiktas RA, kurios struktūra yra:

$$\left[\begin{array}{cccccc} A_1 & A_2 & \dots & A_j & \dots & A_n \\ \hline c_{11} & c_{12} & \dots & c_{1j} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2j} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ c_{i1} & c_{i2} & \dots & c_{ij} & \dots & c_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mj} & & c_{mn} \end{array}\right] \qquad (5.7)$$

Elementai A_j (čia $1 \leq j \leq n$) yra vadinami atributais; elementai c_{ij} – atributų reikšmėmis.

Detalizuojant atsargų kontrolės uždavinio sprendimą reliacines aibes pateiksime duomenų lentelėmis, kadangi lentelė yra labai akivaizdus ir parastas, žmogaus suvokimui duomenų rinkinys.

Taigi tarkime, kad šiuo metu sandėlio pagrindinėje RA saugomi tam tikri duomenys (5.1 lentelė).

5.1 lentelė. *Sandėlio pagrindinė RA ir joje užfiksuoti duomenys*

Detalės kodas	Detalė	Kiekis sausio 1 d.
DIN 931	Varžtas	1107
DIN 826	Varžtas	1010
DIN 827	Varžtas	300
DIN 796	Varžtas	570
DIN 790	Varžtas	952
DIN 789	Varžtas	820

Šiam duomenų rinkiniui priskiriame jį identifikuojančius identifikatorius: RA kodą a_1, sandėlio kodą b_1 ir laiko momentą, kuriuo metu buvo užfiksuoti detalių kiekiai, identifikuojantį kodą – d_1. Detales identifikuojančiais, t. y. raktiniais, atributais pasirenkame atributus „Detalės kodas“ ir „Detalė“, šių atributų reikšmės identifikuos kiekvieną detalę atskirai. Taip pat užkoduojame lentelėje tam tikra eile surašytus atributus, t. y. lentelės schemą. Tad lentelės schema su jos atributų ir jų kodų sąrašu bus:

Detalės kodas	Detalė	Kiekis sausio 1 d.
C_1	C_2	C_3

Dabar turėdami visikai identifikuotą pagrindinę RA ir galutinio naudotojo apibrėžtą laiko periodą, kuriam praėjus, periodiškai tuo pačiu metu pagrindinėje RA yra fiksuojami sandėlyje kaupiamų detalių kiekiai, galime daryti pagrindinės RA kopijas.

Tarkime, kad galutinis naudotojas nusprendė detalių kiekius fiksuoti vienerius metus kiekvieno mėnesio pirmą dieną. Tada, praėjus tiems metams, turėsime dvylika pagrindinės RA kopijų su jas nustatančiais identifikatoriais. Kaip jau minėjau, visi identifikatoriai sutaps, skirsis tik laiko momentus identifikuojantys kodai. Taip pat kopijų identifikatoriai turės ženklinimą, kuris leis kopijas atskirti nuo pagrindinės RA.

Kopijos bus daromos taikant besąlyginę transformaciją, atliekamą kortežuose. Tarkime, darome sausio pirmą dieną pagrindinėje RA užfiksuotų duomenų, kurie šiuo atveju pateikti 5.1 lentelėje, kopiją. Į bendrąją besąlyginės

transformacijos formulę įrašę perkeliamų duomenų, esančių pagrindinėje RA, koordinates ir nurodę naujai sukurtos RA koordinates, į kurias duomenys turi būti transformuojami, padarome identišką pagrindinės RA kopiją. Transformuojame duomenis ne tik iš atributo, kuriame užfiksuoti detalių kiekiai, bet ir iš atributų „Detalės kodas“, „Detalė“:

$$p^{t}_{1,1}(<1/1, 1/2, 1/3> \longrightarrow <1/1, 1/2, 1/3>), \quad (5.8)$$

$$p^{t}_{1,2}(<2/1, 2/2, 2/3> \longrightarrow <2/1, 2/2, 2/3>), \quad (5.9)$$

$$p^{t}_{1,3}(<3/1, 3/2, 3/3> \longrightarrow <3/1, 3/2, 3/3>), \quad (5.10)$$

$$p^{t}_{1,4}(<4/1, 4/2, 4/3> \longrightarrow <4/1, 4/2, 4/3>), \quad (5.11)$$

$$p^{t}_{1,5}(<5/1, 5/2, 5/3> \longrightarrow <5/1, 5/2, 5/3>), \quad (5.12)$$

$$p^{t}_{1,6}(<6/1, 6/2, 6/3> \longrightarrow <6/1, 6/2, 6/3>). \quad (5.13)$$

Dėl aiškumo kiekvienos aprašomos algoritminės priklausomybės kodas yra papildytas indeksu. Pirmas indekso skaičius nurodo AP pobūdį bendrąja prasme, o antras – išskiria konkrečią AP šio uždavinio aprašytame sprendime.

Gauta RA kopija ir jos lentelės schema turės tokius identifikatorius:

$$a'_1, b'_1, d'_1; C_1, C_2, C_3.$$

Turėdami tokį vieną bendrą įrašą galime surasti duomenis ir jiems priskirti lentelės schemą. Taip pat nepriklausomai nuo duomenų c_{ij} gausumo, kompiuterio atmintyje lentelės schemą pakanka turėti tik vieną, o duomenis atmintyje laikyti atskirai nuo schemos tik su jų identifikatoriais.

Taigi naudodami tą pačią besąlyginę transformaciją, tik transformuojamų ir priimančių koordinačių vietose įrašę perkeliamų duomenų koordinates padarome kiekvieno mėnesio pirmą dieną pagrindinėje RA fiksuotų duomenų kopijas.

Toliau dėl analogiškų besikartojančių veiksmų pateikiami tik detalūs paskutinės kopijos duomenys, kuri apima visus duomenis užfiksuotus kiekvieno mėnesio pirmą dieną per ištisus stebėjimo metus. Paskutinė kopija apima visų stebėjimo metų rezultatus, kadangi sąmoningai buvo daromos identiškos, o ne dalinės pagrindinės RA kopijos. Tačiau pateikus paskutinės kopijos identifikatorius iš duomenų visumos išrinks tik laiko faktoriaus kodu d_{12} identifikuoto laiko momento užfiksuotus duomenis, t. y. šiuo atveju duomenis, fiksuotus gruodžio 1 d. (5.2 lentelėje išskirti spalva).

Paskutinės, t. y. dvyliktos, pagrindinės RA kopijos detalūs duomenys pateikti 5.2 lentelėje.

Šios RA kopijos identifikatoriai ir lentelės schemą identifikuojantys kodai yra:

$$a'_1, b'_1, d'_{12}; C_1, C_2, C_{14}.$$

5.2 lentelė. *Pagrindinės RA dvylikta kopija ir joje užfiksuoti duomenys*

Detalės kodas	**Detalė**	**Kiekis sausio 1 d.**	**Kiekis vasario 1 d.**	**Kiekis kovo 1 d.**	**Kiekis balandžio 1 d.**	**Kiekis gegužės 1 d.**	**Kiekis birželio 1 d.**	**Kiekis liepos 1 d.**	**Kiekis rugpjūčio 1 d.**	**Kiekis rugsėjo 1 d.**	**Kiekis spalio 1 d.**	**Kiekis lapkričio 1 d.**	**Kiekis gruodžio 1 d.**
C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}
DIN 931	Varžtas	1107	589	935	677	1030	954	764	893	1013	879	900	951
DIN 826	Varžtas	1010	700	954	134	764	478	937	998	1056	567	599	635
DIN 827	Varžtas	300	504	753	593	497	974	803	690	400	478	645	755
DIN 796	Varžtas	570	770	1074	480	276	479	198	389	654	889	1009	1000
DIN 790	Varžtas	952	594	798	678	985	1098	983	785	598	895	707	695
DIN 789	Varžtas	820	892	1092	953	750	467	798	1053	1000	820	1100	935

Toliau pateikiami visų padarytų kopijų identifikatoriai ir jiems pateikti naudojamų lentelių schemas nustatantys kodai:

Pirma kopija saugo sausio 1 d. fiksuotus duomenis:
$a'_1, b'_1, d'_1; C_1, C_2, C_3$.
Antra kopija saugo vasario 1 d. fiksuotus duomenis:
$a'_1, b'_1, d'_2; C_1, C_2, C_4$.
Trečia kopija saugo kovo 1 d. fiksuotus duomenis:
$a'_1, b'_1, d'_3; C_1, C_2, C_5$.
Ketvirta kopija saugo balandžio 1 d. fiksuotus duomenis:
$a'_1, b'_1, d'_4; C_1, C_2, C_6$.
Penkta kopija saugo gegužės 1 d. fiksuotus duomenis:
$a'_1, b'_1, d'_5; C_1, C_2, C_7$.
Šešta kopija saugo birželio 1 d. fiksuotus duomenis:
$a'_1, b'_1, d'_6; C_1, C_2, C_8$.
Septinta kopija saugo liepos 1 d. fiksuotus duomenis:
$a'_1, b'_1, d'_7; C_1, C_2, C_9$.
Aštunta kopija saugo rugpjūčio 1 d. fiksuotus duomenis:
$a'_1, b'_1, d'_8; C_1, C_2, C_{10}$.

Devinta kopija saugo rugsėjo 1 d. fiksuotus duomenis:
a'_1, b'_1, d'_9; C_1, C_2, C_{11}.
Dešimta kopija saugo spalio 1 d. fiksuotus duomenis:
a'_1, b'_1, d'_{10}; C_1, C_2, C_{12}.
Vienuolikta kopija saugo lapkričio 1 d. fiksuotus duomenis:
a'_1, b'_1, d'_{11}; C_1, C_2, C_{13}.
Dvylikta kopija saugo gruodžio 1 d. fiksuotus duomenis:
a'_1, b'_1, d'_{12}; C_1, C_2, C_{14}.

Kitas etapas yra pateikus identifikatorių seką $a'_1, b'_1, d'_{1\text{-}12}$ iš kopijose laikomos duomenų visumos atrinkti duomenis, kurie buvo fiksuoti per $d'_{1\text{-}12}$ kodu identifikuotus laiko momentus, apibrėžiančius kiekvienos detalės kodą, pavadinimą ir kiekį. Kadangi atrinkti duomenys yra skirtingose RA, tai taikydami besąlyginę transformaciją, atliekamą kortežuose, juos perkeliame į vieną naują RA:

$$p^t(< k / l >^n \longrightarrow < i / j >), \tag{5.14}$$

čia $< k / l >$ yra koordinatės, iš kurių imami duomenys šiuo momentu, čia bus nurodomi reliacinių aibių kopijose identifikatoriais atrinktų duomenų koordinatės; indeksas n reiškia aibės eilės numerį; $< i / j >$ – koordinatės, į kurias įnešami duomenys. Išraiškoje $< i / j >$ indeksas n neturi prasmės, nes duomenis priimanti RA visada yra tik viena.

Taigi visų atrinktų duomenų besąlyginė transformacija į vieną RA bus atlikta programiškai realizavus toliau pateiktas transformacijų išraiškas:

$p^t{}_{1,7}$ ((<1/1, 1/2, 1/3 >^{1} ⟶ < 1/1, 1/2, 1/3>),
(< 2/1, 2/2, 2/3 >^{1} ⟶ < 2/1, 2/2, 2/3>),
(< 3/1, 3/2, 3/3 >^{1} ⟶ < 3/1, 3/2, 3/3 >),
(< 4/1, 4/2, 4/3 >^{1} ⟶ < 4/1, 4/2, 4/3 >),
(< 5/1, 5/2, 5/3 >^{1} ⟶ < 5/1, 5/2, 5/3 >),
(< 6/1, 6/2, 6/3 >^{1} ⟶ < 6/1, 6/2, 6/3 >),
(<1/4 >^{2} ⟶ < 1/4>), (<2/4 >^{2} ⟶ < 2/4>),
(<3/4 >^{2} ⟶ < 3/4>), (<4/4 >^{2} ⟶ < 4/4>),
(<5/4 >^{2} ⟶ < 5/4>), (<6/4 >^{2} ⟶ < 6/4>),
(<1/5 >^{3} ⟶ < 1/5>), (<2/5 >^{3} ⟶ < 2/5>),
(<3/5 >^{3} ⟶ < 3/5>), (<4/5 >^{3} ⟶ < 4/5>),
(<5/5 >^{3} ⟶ < 5/5>), (<6/5 >^{3} ⟶ < 6/5>),
(<1/6 >^{4} ⟶ < 1/6>), (<2/6 >^{4} ⟶ < 2/6>),

$$
\begin{aligned}
&(<3/6>^{4} \longrightarrow <3/6>), (<4/6>^{4} \longrightarrow <4/6>),\\
&(<5/6>^{4} \longrightarrow <5/6>), (<6/6>^{4} \longrightarrow <6/6>),\\
&(<1/7>^{5} \longrightarrow <1/7>), (<2/7>^{5} \longrightarrow <2/7>),\\
&(<3/7>^{5} \longrightarrow <3/7>), (<4/7>^{5} \longrightarrow <4/7>),\\
&(<5/7>^{5} \longrightarrow <5/7>), (<6/7>^{5} \longrightarrow <6/7>),\\
&(<1/8>^{6} \longrightarrow <1/8>), (<2/8>^{6} \longrightarrow <2/8>),\\
&(<3/8>^{6} \longrightarrow <3/8>), (<4/8>^{6} \longrightarrow <4/8>),\\
&(<5/8>^{6} \longrightarrow <5/8>), (<6/8>^{6} \longrightarrow <6/8>),\\
&(<1/9>^{7} \longrightarrow <1/9>), (<2/9>^{7} \longrightarrow <2/9>),\\
&(<3/9>^{7} \longrightarrow <3/9>), (<4/9>^{7} \longrightarrow <4/9>),\\
&(<5/9>^{7} \longrightarrow <5/9>), (<6/9>^{7} \longrightarrow <6/9>),\\
&(<1/10>^{8} \longrightarrow <1/10>), (<2/10>^{8} \longrightarrow <2/10>),\\
&(<3/10>^{8} \longrightarrow <3/10>), (<4/10>^{8} \longrightarrow <4/10>),\\
&(<5/10>^{8} \longrightarrow <5/10>), (<6/10>^{8} \longrightarrow <6/10>),\\
&(<1/11>^{9} \longrightarrow <1/11>), (<2/11>^{9} \longrightarrow <2/11>),\\
&(<3/11>^{9} \longrightarrow <3/11>), (<4/11>^{9} \longrightarrow <4/11>),\\
&(<5/11>^{9} \longrightarrow <5/11>), (<6/11>^{9} \longrightarrow <6/11>),\\
&(<1/12>^{10} \longrightarrow <1/12>), (<2/12>^{10} \longrightarrow <2/12>),\\
&(<3/12>^{10} \longrightarrow <3/12>), (<4/12>^{10} \longrightarrow <4/12>),\\
&(<5/12>^{10} \longrightarrow <5/12>), (<6/12>^{10} \longrightarrow <6/12>),\\
&(<1/13>^{11} \longrightarrow <1/13>), (<2/13>^{11} \longrightarrow <2/13>),\\
&(<3/13>^{11} \longrightarrow <3/13>), (<4/13>^{11} \longrightarrow <4/13>),\\
&(<5/13>^{11} \longrightarrow <5/13>), (<6/13>^{11} \longrightarrow <6/13>),\\
&(<1/14>^{12} \longrightarrow <1/14>), (<2/14>^{12} \longrightarrow <2/14>),\\
&(<3/14>^{12} \longrightarrow <3/14>), (<4/14>^{12} \longrightarrow <4/14>),\\
&(<5/14>^{12} \longrightarrow <5/14>), (<6/14>^{12} \longrightarrow <6/14>)).
\end{aligned}
\tag{5.15}
$$

Uždaviniui spręsti atrinktus duomenis transformavus į vieną RA ir pritaikius jiems sudėties skaičiavimo algoritminę priklausomybę, apskaičiuojamas bendras kiekvienos detalės kiekis per visą, šiuo atveju metus trukusį, stebėjimo periodą. Detalūs kiekvienos detalės bendro kiekio skaičiavimai atrodytų taip:

$p^{+}_{2,8}$ (1/3 + 1/4 + 1/5 + 1/6 + 1/7 + 1/8 + 1/9+ 1/10 + 1/11 + 1/12 + 1/13 + 1/14 → 1/15), (5.16)

$p^{+}_{2,9}$ (2/3 + 2/4 + 2/5 + 2/6 + 2/7 + 2/8 + 2/9+ 2/10 + 2/11 + 2/12 + 2/13 + 2/14 → 2/15), (5.17)

$p^{+}_{2,10}$ (3/3 + 3/4 + 3/5 + 3/6 + 3/7 + 3/8 + 3/9+ 3/10 + 3/11 + 3/12 + 3/13 + 3/14 → 3/15), (5.18)

$p^{+}_{2,11}$ (4/3 + 4/4 + 4/5 + 4/6 + 4/7 + 4/8 + 4/9+ 4/10 + 4/11 + 4/12 + 4/13 + 4/14 → 4/15), (5.19)

$p^{+}_{2,12}$ (5/3 + 5/4 + 5/5 + 5/6 + 5/7 + 5/8 + 5/9+ 5/10 + 5/11 + 5/12 + 5/13 + 5/14 → 5/15), (5.20)

$p^{+}_{2,13}$ (6/3 + 6/4 + 6/5 + 6/6 + 6/7 + 6/8 + 6/9+ 6/10 + 6/11 + 6/12 + 6/13 + 6/14 → 6/15). (5.21)

Kiekvienos detalės skaičiavimai atliekami atskirai. Išraiškoje naudojami detalės kiekį apibrėžiančių atributų reikšmių koordinatės RA. Gautas sumavimo rezultatas atsimenamas toje pačioje RA remiantis išraiškoje pateikta nuoroda į rezultato koordinates. Taigi skaičiuojant pirmos detalės, t. y. varžto, kurio kodas yra DIN 931, bendrą kiekį per visą stebėjimo periodą bus sudedamos pirmo kortežo atributų reikšmės, kurios yra pateiktos pradedant trečiu ir baigiant keturioliktu domenuose imtinai. Sumavimo rezultatas bus išsaugotas pirmo kortežo penkioliktame domene.

Atlikus šią skaičiavimo algoritminę priklausomybę turimos RA duomenys pateikti 5.3 lentelėje.

5.3 lentelė. *RA su apskaičiuotais duomenimis*

Detalės kodas	Detalė	Kiekis sausio 1 d.	Kiekis vasario 1 d.	Kiekis kovo 1 d.	Kiekis balandžio 1 d.	Kiekis gegužės 1 d.	Kiekis birželio 1 d.	Kiekis liepos 1 d.	Kiekis rugpjūčio 1 d.	Kiekis rugsėjo 1 d.	Kiekis spalio 1 d.	Kiekis lapkričio 1 d.	Kiekis gruodžio 1 d.	Bendras kiekis
DIN 931	Varž-tas	1107	589	935	677	1030	954	764	893	1013	879	900	951	10692
DIN 826	Varž-tas	1010	700	954	134	764	478	937	998	1056	567	599	635	8832
DIN 827	Varž-tas	300	504	753	593	497	974	803	690	400	478	645	755	7392
DIN 796	Varž-tas	570	770	1074	480	276	479	198	389	654	889	1009	1000	7788
DIN 790	Varž-tas	952	594	798	678	985	1098	983	785	598	895	707	695	9768
DIN 789	Varž-tas	820	892	1092	953	750	467	798	1053	1000	820	1100	935	10680

Toliau taikydami nuosavą atributų reikšmių algoritminę priklausomybę suskaičiuojame visus kopijas identifikuojančius d' identifikatorius. ir gautą rezultatą, t. y. kopijų kiekį, išsaugome RA atributo reikšmės koordinatėse A_k :

$$p^{a}{}_{3,14}\,(count\ d' \rightarrow A_k). \tag{5.22}$$

Analizuojamu atveju gautas rezultatas yra skaičius dvylika. Dabar, turėdami visus reikiamus duomenis, kad apskaičiuotume kiekvienos sandėlyje kaupiamos detalės vidutinį kiekį, taikome skaičiavimo algoritminę priklausomybę p^{+}:

$$p^{+}{}_{4,15}\,(A_{n+1} / A_k \rightarrow A_{n+2}). \tag{5.23}$$

Išraiškoje pateiktas atributo reikšmės, kuri nurodo skaičiuojamos detalės bendrą kiekį, koordinatės A_{n+1}. A_k koordinatėse yra išsaugota reikšmė, nurodanti prieš tai apskaičiuotą kopijų skaičių – 12. Gautas skaičiavimo rezultatas išsaugomas po nuorodos → apibrėžtose koordinatėse A_{n+2}. Šiuo atveju tos pačios RA šešioliktame domene. Taigi varžto, kurio kodas yra DIN 931, vidutinis skaičius bus:

$$10706 / 12 = 891. \tag{5.24}$$

Tai buvo apskaičiuota pagal išraišką:

$$p^{+}{}_{4,15}\,(1/15 / 12 \rightarrow 1/16). \tag{5.25}$$

Taikydami tą pačią skaičiavimo algoritminę priklausomybę suskaičiuojame ir kitų detalių vidurkius:

$$p^{+}{}_{4,16}\,(2/15 / 12 \rightarrow 2/16), \qquad 8832 / 12 = 736. \tag{5.26}$$
$$p^{+}{}_{4,17}\,(3/15 / 12 \rightarrow 3/16), \qquad 7392 / 12 = 616. \tag{5.27}$$
$$p^{+}{}_{4,18}\,(4/15 / 12 \rightarrow 4/16), \qquad 7788 / 12 = 649. \tag{5.28}$$
$$p^{+}{}_{4,19}\,(5/15 / 12 \rightarrow 5/16), \qquad 9768 / 12 = 814. \tag{5.29}$$
$$p^{+}{}_{4,20}\,(6/15 / 12 \rightarrow 6/16), \qquad 10680 / 12 = 890. \tag{5.30}$$

Naudodami gautus rezultatus sudarome detalių kiekių normatyvinę RA. Tai atliekama taikant besąlyginę transformaciją, kurią naudodami perkelsime reikalingus duomenis iš kortežų į naujai sukurtos RA kortežus. Transformuojame duomenis, ne tik atspindinčius detalių kiekių vidurkius, bet ir atributus, su jų reikšmėmis, identifikuojančius gaminį, t. y. „Detalės kodas“ ir „Detalė“. Reikalingus duomenis atrenkame pateikus a'_1, b'_1; C_1, C_2, C_{16} identifikatorių seką. Naudodami atrinktus duomenis realizuojame besąlyginę transformaciją:

$$p^{t}{}_{5,21}\,(<1/1, 1/2, 1/16> \longrightarrow <1/1, 1/2, 1/3>), \tag{5.31}$$
$$p^{t}{}_{5,22}\,(<2/1, 2/2, 2/16> \longrightarrow <2/1, 2/2, 2/3>), \tag{5.32}$$
$$p^{t}{}_{5,23}\,(<3/1, 3/2, 3/16> \longrightarrow <3/1, 3/2, 3/3>), \tag{5.33}$$
$$p^{t}{}_{5,24}\,(<4/1, 4/2, 4/16> \longrightarrow <4/1, 4/2, 4/3>), \tag{5.34}$$
$$p^{t}{}_{5,25}\,(<5/1, 5/2, 5/16> \longrightarrow <5/1, 5/2, 5/3>), \tag{5.35}$$
$$p^{t}{}_{5,26}\,(<6/1, 6/2, 6/16> \longrightarrow <6/1, 6/2, 6/3>). \tag{5.36}$$

Naujai sukurtai RA priskiriame ją identifikuojančius identifikatorius. Kadangi tai yra normatyvinių duomenų RA, vadinasi, joje pateikti duomenys

nepriklauso konkrečiam subjektui ir taip pat nepriklauso nuo laiko veiksnio. Taigi šiai RA identifikuoti pakanka tik RA schemos kodo *a*. Šiuo atveju tegu tai bus a_2. Detales identifikuojančios raktinių atributų reikšmės išliks tos pačios, t. y. raktiniai atributai nesikeis. Duomenims pateikti lentelės schemą naudosime tą pačią, kurią turime, tik pagal kodus išrinksime reikiamus atributus. Šiuo atveju tai būtų C_1, C_2, C_{16}. Normatyvinių duomenų lentelė su atributų reikšmėmis ir kodais pateikta 5.4 lentelėje. Turėdami normatyvinių duomenų lentelę galime pagal poreikius automatiškai atlikti atsargų, esančių dabartiniu metu sandėlyje, kontrolę. Tarkime, kad 5.5 lentelėje yra pateikta RA, kurioje saugomi duomenys apie šiuo metu sandėlyje esančius atsargų kiekius.

5.4 lentelė. *Normatyvinių duomenų lentelė*

Detalės kodas	**Detalė**	**Vidurkis**
C_1	C_2	C_{16}
DIN 931	Varžtas	891
DIN 826	Varžtas	736
DIN 827	Varžtas	616
DIN 796	Varžtas	649
DIN 790	Varžtas	814
DIN 789	Varžtas	890

5.5 lentelė. *Šiuo metu sandėlyje esantys atsargų kiekiai*

Detalės kodas	**Detalė**	**Vidurkis**
C_1	C_2	C_{16}
DIN 826	Varžtas	700
DIN 931	Varžtas	1527
DIN 827	Varžtas	943
DIN 796	Varžtas	541
DIN 790	Varžtas	948
DIN 789	Varžtas	1092

Šioje RA pateiktų detalių raktinių atributų reikšmės sulyginamos su normatyvinėje lentelėje pateiktų detalių raktinių atributų reikšmėmis. Tai

atliekama taikant sąlyginę transformaciją be transformacijos. Tarkime pirmos 5.5 lentelėje pateiktos detalės raktinių atributų reikšmės lyginamos su normatyvinėje lentelėje pateiktomis pirmos detalės raktinių atributų reikšmėmis:

$$p^{t}_{6,27}(\{\ 1/1,\ 1/2\ \}^{2} \xrightarrow{=} \{\ 1/1,\ 1/2\ \}^{1}). \qquad (5.37)$$

Dėl aiškumo vietoj išraiškoje pateiktų atributų reikšmių koordinačių RA, įrašysime tų atributų konkrečias reikšmes:

$$\{\text{DIN 826, Varžtas}\} \xrightarrow{=} \{\text{DIN 931, Varžtas}\}. \qquad (5.38)$$

Akivaizdu, kad lyginimo sąlyga nebuvo iki galo patenkinta, kadangi neatitiko atributų „Detalės kodas“ pateiktos reikšmės. Vadinasi, bus atliekami tokie palyginimai tol, kol bus surasta ta pati detalė, t. y. kai abiejose RA detalę identifikuojančios raktinių atributų reikšmės iki galo sutaps:

$$p^{t}_{6,28}(\{\ 1/1,\ 1/2\ \}^{2} \xrightarrow{=} \{\ 2/1,\ 2/2\ \}^{1}), \qquad (5.39)$$

$$\{\text{DIN 826, Varžtas}\} \xrightarrow{=} \{\ \text{DIN 826, Varžtas}\}. \qquad (5.40)$$

Identifikavus tą pačią detalę, tos detalės einamuoju momentu kiekį išreiškiančios atributo reikšmė lyginama su normatyvinėje lentelėje pateikta reikšme. Pagal 5.4 lentelėje pateiktus duomenis tos pačios detalės kiekį išreiškiančios reikšmės koordinatės RA, kurioje užfiksuoti sandėlio atsargų kiekiai einamuoju momentu, yra 1/3, o normatyvinių duomenų RA – 2/3. Taigi, taikant sąlyginę transformaciją šiose koordinatėse pateiktos atributų reikšmės bus lyginamos. Jei palyginimo sąlyga bus patenkinta, t. y. einamuoju metu sandėlyje šios detalės kiekis bus mažesnis negu nurodyta normatyvinėje duomenų RA, tai šios detalės duomenys bus transformuojami į atskirą RA, kuri bus pateikiama galutiniam vartotojui. Tokiu būdus galutinis naudotojas automatiškai gaus informaciją apie detales, kurių atsargos sandėlyje yra mažesnės ir (arba) lygios už normose nustatytas. Šiuo atveju bendroji sąlyginės transformacijos išraiška bus:

$$p^{t}\{\ k/l\ \}^{n} < k/l >^{n} \xrightarrow{\leq} < i/j >^{n} \{\ i/j\ \}^{n}. \qquad (5.41)$$

Čia lyginami koordinačių $\{i/j\}$ ir $\{k/l\}$ turiniai; $\leq$ apibrėžtas lyginimo sąlygos simbolių. Kai sąlyga patenkinta, duomenys pernešami iš koordinačių $< k/l >$ į koordinates $< i/j >$. Rodykle nurodyta duomenų transformacijos kryptis. Indeksas n reiškia aibės eilės numerį.

Ši išraiška su įrašytomis analizuojamos detalės kiekius apibrėžiančių reikšmių koordinatėmis RA atrodytų taip:

$$p^{t}_{7,29}(\{\ 1/3\ \}^{2} < 1/1,\ 1/2,\ 1/3 >^{2} \xrightarrow{\leq} \{2/3\ \}^{1} < 1/1,\ 1/2,\ 1/3 >^{3}). \qquad (5.42)$$

Kadangi 1/3 koordinatėse pateikta reikšmė yra 700, o 2/3 – 736, tai akivaizdu, kad lyginimo sąlyga patenkinta. Kitaip tariant, einamuoju metu sandėlyje šios detalės kiekis yra mažesnis negu nurodyta normatyvinėje duomenų RA, todėl šios detalės duomenys bus transformuojami į atskirą RA, kuri bus pateikta galutiniam naudotojui.

Jei palyginimo sąlyga atitinka, tai ta pati kontrolės procedūra atliekama kitas detalės atžvilgiu ir tai kartojama tol, kol bus įvykdyta visų sandėlyje pateiktų atsargų kiekių kontrolė. Laiką, kuriam praėjus bus atliekama kita atsargų kontrolė, pasirenka galutinis naudotojas atsižvelgdamas į veikos poreikius.

Išanalizavus šio uždavinio sprendimą, toliau reikia sudaryti algoritminę RA šiam uždaviniui spręsti. Remiantis 4.4 poskyryje aprašyta teorija, uždaviniui spręsti gali būti sudaryta arba viena ARA, arba ištisa aibė skirtingų ARA, tarpusavyje sujungtų išorinėmis algoritminėmis priklausomybėmis, t. y. ARA eilutė. Kadangi šio uždavinio algoritmas nėra sudėtingas, tai jam sprendimui pakanka sudaryti vieną ARA.

Aprašytam uždaviniui spręsti buvo apibrėžtos šios algoritminės priklausomybės:

$p^{t}_{1,1}, p^{t}_{1,2}, p^{t}_{1,3}, p^{t}_{1,4}, p^{t}_{1,5}, p^{t}_{1,6}, p^{t}_{1,7}, p^{+}_{2,8}, p^{+}_{2,9}, p^{+}_{2,10}, p^{+}_{2,11}, p^{+}_{2,12}, p^{+}_{2,13}, p^{a}_{3,14}, p^{+}_{1,15}, p^{+}_{1,16}, p^{+}_{1,17}, p^{+}_{1,18}, p^{+}_{1,19}, p^{+}_{1,20}, p^{t}_{5,21}, p^{t}_{5,22}, p^{t}_{5,23}, p^{t}_{5,24}, p^{t}_{5,25}, p^{t}_{5,26}, p^{t}_{6,27}, p^{t}_{6,28}, p^{t}_{7,29}.$

Taigi ARA M bus sudaroma iš visų šių algoritminių priklausomybių išrinkus nepasikartojančių (tai galima atpažinti iš AP kodo indekse pateikto pirmojo skaičiaus) AP kodus:

$$M = \left(p^{t}_{1}\ p^{+}_{2}\ p^{a}_{3}\ p^{+}_{4}\ p^{t}_{5}\ p^{t}_{6}\ p^{t}_{7}\ \right).$$

Šioje ARA M programinių ėjimų kryptis ir tvarka apibrėžiama programinių ėjimų algoritmine priklausomybe $p^{\rightarrow}$:

$$p^{\rightarrow} = p^{t}_{1}, p^{+}_{2}, p^{a}_{3}, p^{+}_{4}, p^{t}_{5}, p^{t}_{6}, p^{t}_{7}. \qquad (5.43)$$

Programiškai einant turima ARA M ir iškviečiant algoritminių priklausomybių kodus iškviečiant AP realizuojančius programinius modulius, atliksime uždavinio sprendimą.

Aprašytas medžiagų ir detalių atsargų kontrolės uždavinys bus sprendžiamas pagal algoritminės priklausomybės $p^{\rightarrow}$ nurodytą tvarką bei kryptį, programiškai einant algoritmine RA M ir besikreipiant į algoritminių priklausomybių kodus, kuriuos naudojant iškviečiami šias AP realizuojantys programiniai moduliai.

5.3. Kainų koregavimo uždavinys

Sandėliavimo institucija gauna medžiagų, detalių, agregatų ir pan., kurių kainos gali keistis. Pristatyti į institucijos sandėlius tie patys produktai arba tos pačios prekės yra fiziškai visiškai identiškos ir jų raktinių atributų reikšmės vienodos, t. y. priskirti kodai vienodi, tačiau jų kainos, palyginti su sandėlyje turimų prekių kainomis, yra skirtingos. Tam tikrais laikotarpiais kai kurių prekių, esančių vienu metu tame pačiame sandėlyje, kainos gali net labai smarkiai skirtis, todėl adaptyvioji duomenų apdorojimo projektavimo technologija gali, manipuliuojant vienais ir tais pačiais algoritminių priklausomybių moduliais, pasiūlyti įvairius kainų reguliavimo mechanizmus. Šiame darbe siūlomas vidurkinis kainų reguliavimo interaktyvus mechanizmas, t. y. kiekvienam momentui, kai sandėlis papildomas prekėmis, kurių kodai tie patys, bet kaina skiriasi.

Tarkime, kad institucijos X viename iš sandėlių šiuo metu yra saugomos tam tikros detalės. Papildę tą patį sandėlį detalių atsargomis sulyginsime turimų ir gautų detalių, kurių kodai vienodi, kainas. Radę kainų skirtumus apskaičiuosime šių detalių kainų vidurkius, kad sureguliuotume, t. y. šiuo atveju – suvienodintume turimų ir gautų to pačio kodo detalių kainas.

Taigi sandėlyje turimų detalių duomenys yra kaupiami vadinamojoje pagrindinėje RA (5.6 lentelė).

5.6 lentelė. *Pagrindinė RA ir jos duomenys apie sandėlyje esančias detales (pirma RA)*

Detalės kodas	**Detalė**	**Kiekis**	**Vieneto kaina, Lt**
C_1	C_2	C_3	C_4
DIN 931	Varžtas	20000	1,2
DIN 826	Varžtas	13600	1,5
DIN 827	Varžtas	12000	1,4
DIN 796	Veržlė	17200	0,5
DIN 790	Veržlė	9000	0,7
DIN 789	Veržlė	18000	0,4

Kaip jau buvo minėta 3.3 poskyryje, turimą duomenų struktūrą būtina identifikuoti, t. y. priskirti RA kodą, subjekto kodą ir lentelės atributų kodus. Todėl šiai RA ir jai pateikti naudojamos lentelės schemai identifikuoti priskirti tokie identifikatoriai ir atributų kodai:

$$a_1, b_1; C_1, C_2, C_3, C_4 .$$

Atributai „Detalės kodas“ ir „Detalė“ pasirenkami kaip raktiniai atributai, kurių reikšmės vienareikšmiškai identifikuos kiekvieną RA pateiktą objektą.

Duomenys apie sandėlį papildžiusias detales užfiksuoti kitoje RA (5.7 lentelė).

5.7 lentelė. *RA ir jos duomenys apie sandėlį papildžiusias detales (antra RA)*

Detalės kodas	**Detalė**	**Kiekis**	**Vieneto kaina, Lt**
C_1	C_2	C_3	C_4
DIN 826	Varžtas	20000	1,8
DIN 931	Varžtas	30000	1,2
DIN 827	Varžtas	13000	1,6
DIN 796	Veržlė	10000	0,7
DIN 790	Veržlė	9000	0,7
DIN 789	Veržlė	18000	0,4

Šiai duomenų struktūrą identifikuoti priskirta tokia seka:

$$a_2, b_1; C_1, C_2, C_3, C_4 .$$

Reikia pabrėžti, kad šios RA ir pagrindinės RA identifikatorių sekos skiriasi tiktai RA kodu a_2, kadangi kiekvienas savarankiškas duomenų darinys turi turėti unikalų, t. y. tik jį identifikuojantį, kodą. Subjekto atributo kodai b sutaps, nes identifikuoja tą patį sandėlį. Duomenims pateikti apie sandėlį papildžiusias detales bus naudojama ta pati lentelė, kaip ir pagrindinės RA duomenims pateikti, kadangi abiejų RA atributų pavadinimai yra identiški. Taip pat nesiskirs ir detales identifikuojantys raktiniai atributai. Suprantama, kad vienodų detalių raktinių atributų reikšmės skirtingose RA bus vienodos.

Norint kainas koreguoti, visų pirma reikia surasti tą pačią detalę tiek pagrindinėje RA, tiek sandėlį papildžiusių detalių RA. Tai realizuosime taikydami sąlyginę transformaciją, t. y. bus sulyginamos sandėlį papildžiusių detalių raktinių atributų reikšmės su pagrindinėje RA pateiktų detalių raktinių atributų reikšmėmis. Dėl aiškumo, kiekvienos aprašomos algoritminės priklausomybės kodas yra papildytas indeksu, kurio pirmas skaičius nurodo algoritminės priklausomybės pobūdį bendrąja prasme, o antras skaičius išskiria konkrečią AP šio uždavinio aprašytame sprendime.

$$p^{t}{}_{1,1}\{\ 1/1,\ 1/2\ \}^2 < 1/1,\ 1/2,\ 1/3,\ 1/4 >^2 <1/3,\ 1/4>^1 \quad \xrightarrow{=} \quad \{\ 1/1,\ 1/2\ \}^1 < 1/1,\ 1/2,\ 1/3,\ 1/4,\ 1/5,\ 1/6>^3, \tag{5.44}$$

$p^{t}_{1,2}$ { 1/1, 1/2 }^{2} < 1/1, 1/2, 1/3, 1/4 >^{2} <2/3, 2/4>^{1} $\xrightarrow{=}$ { 2/1, 2/2 }^{1} < 1/1, 1/2, 1/3, 1/4, 1/5, 1/6>^{3}. (5.45)

Pirmoje sąlyginės transformacijos išraiškoje pateiktos sandėlį papildžiusių detalių RA (antra aibė) atributų reikšmių koordinatės, kurių reikšmės pagal sąlygą yra lyginamos su reikšmėmis, kurių koordinatės yra pateiktos pirmoje RA (pagrindinėje RA). Kad būtų aiškiau, vietoj išraiškoje apibrėžtų lyginamų atributų reikšmių koordinačių pateikiamos tų atributų konkrečios reikšmės:

{DIN 826, Varžtas} $\xrightarrow{=}$ {DIN 931, Varžtas}, (5.46)

{DIN 826, Varžtas} $\xrightarrow{=}$ {DIN 826, Varžtas}. (5.47)

Kadangi pirmoje išraiškos eilutėje lyginamųjų atributų reikšmės nepilnai patenkina sąlygą, t. y. atributų „Detalės kodas“ reikšmės yra nelygios, todėl tos pačios atributų reikšmės, paimtos iš sandėlį papildžiusių detalių RA, toliau lyginamos su kitomis pagal eilę einančiomis, raktinių atributų reikšmėmis, paimtomis iš pagrindinės RA (5.6 lentelė). Analizuojamu atveju, $p^{t}_{1,2}$ sąlyginės transformacijos išraiškoje lyginamos atributų reikšmės visiškai atitinka lyginimo sąlygą, vadinasi, surasta ta pati detalė. Taigi patenkinus sąlyginės transformacijos išraiškoje apibrėžtą sąlygą, bus atliekama duomenų, kurių koordinatės pateiktos išraiškoje, transformacija į naują RA (5.8 lentelė). Šiuo atveju bus transformuotos visos pirmo kortežo atributų reikšmės iš antros RA į trečios RA koordinates 1/1, 1/2, 1/3, 1/4 ir antro kortežo trečio ir ketvirto atributų reikšmės iš pagrindinės RA į trečios RA 1/5, 1/6 koordinates. Taigi trečią naująją RA sudarys kiti atributai su jų reikšmėmis (5.8 lentelė): „Detalės kodas“, „Detalė“, „Kiekis“, „Vieneto kaina“ iš antros RA ir „Kiekis“, „Vieneto kaina“ – iš pirmos RA.

5.8 lentelė. *Laikina tarpinių rezultatų RA (trečia RA)*

Detalės kodas	**Detalė**	**Kiekis**	**Vieneto kaina, Lt**	**Kiekis**	**Vieneto kaina, Lt**
C_1	C_2	C_3	C_4	C_5	C_6
DIN 826	Varžtas	20000	1,8	13600	1,5

Tokia duomenų struktūra šiuo atveju nebus identifikuojama, kadangi joje esantys duomenys bus naudojami tik tarpiniams skaičiavimams atlikti ir įsiminti. Suprantama, kad galime jai priskirti identifikatorius ir saugoti ją kaip tarpinių rezultatų RA.

Toliau, norėdami sužinoti, ar sandėlį papildžiusios detalės, varžto DIN 826, kaina skiriasi nuo sandėlyje turimos tos pačios detalės kainos, palyginsime trečioje RA turimų atributų „Vieneto kaina“ reikšmes. Tai įgyvendinsime naudodami sąlyginę transformaciją be transformacijos, kur lyginimo sąlyga apibrėžiama simboliu $\neq$. Analizuojamu atveju sąlyginės transformacijos išraiška bus apibrėžta taip:

$$p^{t}{}_{2,3}\{\ 1/4\ \}^3 \xrightarrow{\neq} \{\ 1/6\ \}^3. \qquad (5.48)$$

Akivaizdu, kad koordinatėse 1/4 pateikta reikšmė 1,8 (Lt) nėra lygi 1/6 koordinatėse pateiktai reikšmei 1,5 (Lt). Kadangi išraiškoje apibrėžta sąlyga yra patenkinta, vadinasi, sandėlį papildžiusios ir turimos detalės kaina skiriasi. Todėl reikia apskaičiuoti šios detalės kainų vidurkį apskaičiavimus ir suvienodinti kainas. Tuo atveju, kai sąlyga nepatenkinta, t. y. vienodų detalių kainos nesiskiria, bus vykdomos kitos operacijos. Šiame poskyryje tai pateikta toliau.

Norint sužinoti vidutinę detalės kainą, visų pirma būtina apskaičiuoti sandėlį papildžiusio kiekio bendrą kainą ir sandėlyje esančio kiekio bendrą kainą. Tai atliksime sudauginę kiekį iš kainos, o daugybos operaciją realizuosime taikydami skaičiavimo algoritminę priklausomybę p^{+}:

$$p^{+}{}_{1,4}(A_3 \times A_4 \rightarrow A_7), \qquad (5.49)$$

čia: A_3 , A_4 – atributų reikšmių, su kuriomis bus atliekama daugybos operacija, koordinatės RA; A_7 – koordinatės, kur bus atsimenamas gautas rezultatas. Šiuo atveju rezultatas bus užfiksuotas tos pačios RA, iš kurios buvo imami duomenys, naujame domene, t. y. atribute pavadinimu „Papildžiusio kiekio kaina“, kadangi pirmiausia bus skaičiuojama sandėlį papildžiusios detalės kiekio bendra kaina. Taigi, sandėlį papildžiusio kiekio bendra kaina apskaičiuojama taip:

$$p^{+}{}_{1,4}(1/3 \times 1/4 \rightarrow 1/7), \qquad 20000 \times 1{,}8 = 36000. \qquad (5.50)$$

Kad būtų aiškiau, kokie tiksliai duomenys naudojami ir kokie gauti, taikant tokią skaičiavimo algoritminę priklausomybę, greta pateikti skaičiavimai, kur vietoj atributų koordinačių įrašytos konkrečios atributų reikšmės.

Taikydami tą pačią algoritminę skaičiavimo priklausomybę, tik jos išraiškoje apibrėžę kitas atributų reikšmių koordinates, apskaičiuojame sandėlyje turimo kiekio bendrą kainą:

$$p^{+}{}_{1,5}(1/5 \times 1/6 \rightarrow 1/8), \qquad 13600 \times 1{,}5 = 20400. \qquad (5.51)$$

Gautą rezultatą išsaugome trečios RA aštuntame naujai sukurto atributo, kurio pavadinimas „Esamo kiekio kaina“, koordinatėse.

Laikina, tarpinių rezultatų RA, papildyta gautais rezultatiniais duomenimis, pateikta (5.9 lentelėje):

5.9 lentelė. *Tarpinių rezultatų RA su naujai apskaičiuotais duomenimis (trečia RA)*

Detalės kodas	Detalė	Kiekis	Vieneto kaina, Lt	Kiekis	Vieneto kaina, Lt	Papildžiusio kiekio kaina, Lt	Esamo kiekio kaina, Lt
C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
DIN 826	Varžtas	20000	1,8	13600	1,5	36000	20400

Turėdami visus reikalingus duomenis vienoje RA, toliau skaičiuojama detalės DIN 826 vidutinės kainą. Tai atliksime realizavę tokią aritmetinę išraišką:

$$(a + b) / (c + d) = \times \; ; \qquad (5.52)$$

čia: a – sandėlį papildžiusio kiekio kaina; b – sandėlyje esančio kiekio kaina; c – detalių kiekis, papildęs sandėlį; d – turimas sandėlyje detalių kiekis; x – detalės, kurios duomenys buvo naudojami skaičiuoti, vidutinė kaina.

Norint atlikti išraiškoje pateiktus aritmetinius veiksmus, šią išraišką apibrėžiame skaičiavimo algoritminių priklausomybių formulėmis:

$$p^{+}_{2,6}\,(1/7 + 1/8 \rightarrow 1/9), \qquad 36000 + 20400 = 56400. \qquad (5.53)$$

$$p^{+}_{2,7}\,(1/3 + 1/5 \rightarrow 1/10), \qquad 20000 + 13600 = 33600. \qquad (5.54)$$

$$p^{+}_{3,8}\,(1/9 \,/\, 1/10 \rightarrow 1/11), \qquad 56400 \,/\, 33600 = 1{,}678. \qquad (5.55)$$

Gauti rezultatai užfiksuoti tos pačios RA, iš kurios buvo imami duomenys, naujai sukurtuose domenuose (5.10 lentelė).

5.10 lentelė. *Tarpinių rezultatų RA (trečia RA)*

Detalės kodas	Detalė	Kiekis	Vieneto kaina, Lt	Kiekis	Vieneto kaina, Lt	Papildžiusio kiekio kaina, Lt	Esamo kiekio kaina, Lt	Bendro kiekio kaina, Lt	Bendras kiekis	Vieneto kainos vidurkis, Lt
C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}
DIN 826	Varžtas	20000	1,8	13600	1,5	36000	20400	56400	33600	1,678

Apskaičiavus vidutinę detalės vieneto kainą, būtina pakoreguoti tos detalės duomenis, esančius pagrindinėje RA. Todėl naudodami detalės raktinių atributų

reikšmes, pateiktas trečioje RA, turime surasti tą pačią detalę pagrindinėje RA. Identifikavus tą pačią detalę, reikia perkelti atributų „Bendras kiekis“ ir „Vieneto kainos vidurkis“ reikšmes iš trečios RA į atributų „Kiekis“ ir „Vieneto kaina“ reikšmių koordinates, pateiktas pagrindinėje RA. Tad aprašytus veiksmus atlikti galėsime realizavę sąlyginę transformaciją, kurią naudodami ne tik surasime tą pačią detalę, bet ir perkelsime norimas atributų reikšmes, „užtrindami“ prieš tai buvusias reikšmes. Remiantis analizuojamos detalės duomenimis, sąlyginės transformacijos išraiška yra ši:

$$p^{t}_{1,9}\{\ 1/1,\ 1/2\ \}^{3} < 1/10,\ 1/11 >^{3} \xrightarrow{=} \{\ 1/1,\ 1/2\ \}^{1} < 1/3,\ 1/4 >^{1}, \qquad (5.56)$$

$$p^{t}_{1,10}\{\ 1/1,\ 1/2\ \}^{3} < 1/10,\ 1/11 >^{3} \xrightarrow{=} \{\ 2/1,\ 2/2\ \}^{1} < 2/3,\ 2/4 >^{1}. \qquad (5.57)$$

Kaip jau buvo minėta, norint rasti tą pačią detalę, turi visiškai sutapti lyginamųjų raktinių atributų reikšmės, t. y. sąlyginės transformacijos sąlyga turi būti patenkinta. Kadangi pirmoje išraiškoje sąlyga nebuvo patenkinta, todėl toliau taikome tą pačią sąlyginę transformaciją kitoms, iš eilės einančioms pagrindinės RA raktinių atributų reikšmių koordinatėms. Suprantama, kad išraiškoje atitinkamai keičiamos ir atributo reikšmės koordinatės, į kurias, patenkinus sąlygą, bus transformuojami duomenys. Taigi antroje sąlyginės transformacijos išraiškoje lyginamųjų atributų reikšmės visiškai atitinka sąlygą, nes išraiškoje pateiktų koordinačių reikšmės yra lygios. Vadinasi, trečios RA koordinatėse 1/10, 1/11 pateiktos atributų reikšmės bus perkeltos į 2/3, 2/4 koordinates pagrindinėje RA. Atlikus šias operacijas pagrindinė RA saugos toliau pateiktus duomenis (5.11 lentelė).

5.11 lentelė. *Pagrindinė RA, atitinkamai pakoregavus detalės DIN 826 duomenis (pirma RA)*

Detalės kodas	**Detalė**	**Kiekis**	**Vieneto kaina, Lt**
C_1	C_2	C_3	C_4
DIN 931	Varžtas	20000	1,2
DIN 826	Varžtas	33600	1,678
DIN 827	Varžtas	12000	1,4
DIN 796	Veržlė	17200	0,5
DIN 790	Veržlė	9000	0,7
DIN 789	Veržlė	18000	0,4

Atributo „Vieneto kaina“ reikšmėms būtų tikslinga apibrėžti keletą nuosavų atributų reikšmių algoritminių priklausomybių, kurias galėtų būtų nurodyti: dešimtainio kablelio (taško) vieta skaičiuje; kokiu tikslumu apvalinamas skaičius

ir pan. Ar šios algoritminės priklausomybės bus apibrėžtos, ar ne, tai turėtų spręsti galutinis naudotojas, atsižvelgiant į poreikius.

Tad siūlomas vidurkinių kainų reguliavimo būdas detaliai apibrėžiamas remiantis tik vienos detalės hipotetiniais duomenimis, kadangi dėl analogiškų besikartojančių veiksmų nėra tikslinga aprašyti šio būdo kitoms detalėms. Suprantama, kad atsižvelgiant į pasirinktą detalę, atitinkamai keisis ir atributų reikšmių koordinatės aukščiau apibrėžtose algoritminių priklausomybių išraiškose.

Sandėlį papildžiusioms detalėms, kurių kainos nesiskiria nuo sandėlyje esančių detalių kainų, vadinamasis kainų reguliavimo mechanizmas bus taikomas ne visas. Kitaip tariant, visos šiame poskyryje apibrėžtos operacijos iki detalės papildžiusio ir esamo kiekio vieneto kainų palyginimo, kuris atliekamas naudojant trečioje RA pateiktus duomenis, bus taip pat realizuojamos. Tačiau nustačius, kad sandėlį papildžiusios detalės kaina nesiskiria nuo sandėlyje turimos tos pačios detalės kainos, bus praleidžiamos skaičiavimo algoritminės priklausomybės, kurios apskaičiuoja: papildžiusio kiekio kainą, esamo kiekio kainą, bendro kiekio kainą ir vieneto kainos vidurkį. Toliau realizacijoje dalyvaus algoritminė skaičiavimo priklausomybė, apskaičiuojanti atributo „Bendras kiekis“ reikšmę. Ši reikšmė, atliekant sąlyginę transformaciją, bus perkelta į pagrindinę RA, „užtrinant“ atribute „Kiekis“ esančią reikšmę. Toliau, naudojant sandėlį papildžiusios detalės DIN 931 (5.7 lentelė) hipotetinius duomenis, pateikiamas konkretus pavyzdys, iliustruojantis šį atvejį. Dėl aiškumo vietoj kai kurių išraiškose apibrėžtų atributų reikšmių koordinačių pateikiamos konkrečios tų atributų reikšmės:

$$p^{t}_{1,11}\{\ 2/1,\ 2/2\ \}^{2} < 2/1,\ 2/2,\ 2/3,\ 2/4 >^{2} <1/3,\ 1/4>^{1} \xrightarrow{=} \{\ 1/1,\ 1/2\ \}^{1} < 1/1,\ 1/2,\ 1/3,\ 1/4,\ 1/5,\ 1/6>^{3}. \tag{5.58}$$

$$\{\text{DIN 931, Varžtas}\} \xrightarrow{=} \{\text{DIN 931, Varžtas}\}. \tag{5.59}$$

$$p^{t}_{2,12}\{\ 1/4\ \}^{3} \xrightarrow{\neq} \{\ 1/6\ \}^{3}, \tag{5.60}$$

$$1{,}2 \neq 1{,}2. \tag{5.61}$$

$$p^{+}_{2,13}(1/3 + 1/5 \rightarrow 1/10), \tag{5.62}$$

$$30000 + 20000 = 50000. \tag{5.63}$$

$$p^{t}_{1,14}\{\ 1/1,\ 1/2\ \}^{3} < 1/10 >^{3} \xrightarrow{=} \{\ 1/1,\ 1/2\ \}^{1} < 1/3 >^{1}, \tag{5.64}$$

$$\{\text{DIN 931, Varžtas}\} \xrightarrow{=} \{\text{DIN 931, Varžtas}\}. \qquad (5.65)$$

5.12 lentelė. *Pagrindinė RA, atitinkamai pakoregavus detalės DIN 931 duomenis (pirma RA)*

Detalės kodas	Detalė	Kiekis	Vieneto kaina, Lt
C_1	C_2	C_3	C_4
DIN 931	Varžtas	50000	1,2
DIN 826	Varžtas	33600	1,678
DIN 827	Varžtas	12000	1,4
DIN 796	Veržlė	17200	0,5
DIN 790	Veržlė	9000	0,7
DIN 789	Veržlė	18000	0,4

Tuo atveju, kai sandėlį papildžiusios detalės, naudojant jos identifikatorius, nerandame pagrindinėje RA, tuomet galutinis naudotojas turėtų priimti vieną iš sprendimų:

- ar detalės kodas ir pavadinimas nėra klaidingi;
- ar pristatyta detalė, kurios iki šiol nebuvo sandėlyje.

Pirmuoju atveju turi būti pataisytas pirminis dokumentas ne adaptyviojoje technologijoje, o antruoju – duomenys apie detalę be pakitimų turi būti transformuojami į pagrindinę RA (5.1 lentelė), papildant ją nauju įrašu. Transformacija atliekama taikant besąlyginę transformaciją domenuose, kurios bendroji formulė yra:

$$p^{t}{}_{3}(< k/l > \longrightarrow < i/j >). \qquad (5.66)$$

Šiuo atveju koordinačių $< k/l >$ vietoje bus nurodyti sandėlį pirmą kartą papildžiusios detalės visų atributų reikšmių koordinatės iš antros RA (5.2 lentelė), o $< i/j >$ – naujai sukurto kortežo koordinatės pagrindinėje RA (5.1 lentelė), į kurią bus transformuojami duomenys iš antros RA.

Detaliai išanalizavus galimus šio uždavinio sprendimo atvejus, toliau būtina sudaryti vadinamąją ARA šiam uždaviniui spręsti. Remiantis 4.4 poskyryje aprašyta teorija, uždaviniui spręsti gali būti sudaryta arba viena ARA, arba ištisa aibė skirtingų ARA, sujungtų išorinėmis algoritminėmis priklausomybėmis, t. y. ARA eilute. Kadangi šio uždavinio algoritmas nėra sudėtingas, tai jam spręsti pakanka sudaryti vieną ARA. Priklausomai nuo situacijos išdėsčius šios ARA poaibius reikiama eile ir juos realizavus programiniais moduliais, bus gautas uždavinio sprendimo rezultatas.

Analizuojamas kainų koregavimo uždavinys, atsižvelgiant į situaciją, gali būti sprendžiamas atitinkamai parenkant vieną iš trijų sprendimų:

1. Tuo atveju, kai sandėlį papildžiusių detalių ir sandėlyje jau esančių, tais pačiais kodais užfiksuotų, detalių kaina skiriasi, kainoms suvienodinti, reikia atlikti duomenų apdorojimo operacijas tokia seka:

$$p^{t}_{1,1}, p^{t}_{1,2}, p^{t}_{2,3}, p^{+}_{1,4}, p^{+}_{1,5}, p^{+}_{2,6}, p^{+}_{2,7}, p^{+}_{3,8}, p^{t}_{1,9}, p^{t}_{1,10}.$$

2. Jeigu tų pačių kodų detalių kainos nesiskiria, tai duomenų apdorojimo operacijų seka bus:

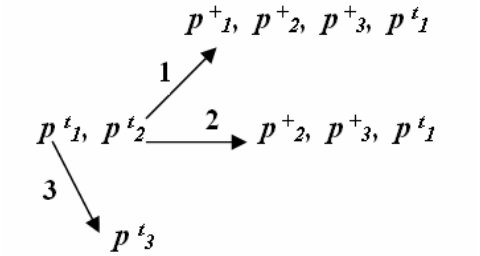
$$p^{t}_{1,11}, p^{t}_{2,12}, p^{+}_{2,13}, p^{t}_{1,14}.$$

3. Sandėlį papildžiusios detalės neradus pagrindinėje RA ir naudotojui nusprendus, kad duomenys nėra klaidingi, visa informacija, susijusi su nauja detale, bus perkeliama į pagrindinę RA taikant šias algoritmines priklausomybes:

$$p^{t}_{1,x}, p^{t}_{3,x}.$$

Čia pateiktose algoritminių priklausomybių koduose indeksų antroje pozicijoje nėra konkretaus skaičiaus, nes aprašytame uždavinyje šis atvejis su konkrečios detalės duomenimis nebuvo aprašytas.

Tad atmetus visą informaciją, susijusią su šiame uždavinyje pateiktais duomenimis, aprašytam kainų koregavimo uždaviniui išspręsti bus taikomos šios algoritminės priklausomybės atitinkamai, t. y. pagal apibrėžtus galimus sprendimus, parenkant jų seką:



Analizuojamo uždavinio ARA *K* bus sudaroma naudojant visų šių algoritminių priklausomybių kodus:

$$K = \begin{pmatrix} p^{t}_{1}\ p^{t}_{2}\ p^{+}_{1}\ p^{+}_{2} p^{+}_{3} \\ p^{t}_{1}\ p^{+}_{2}\ p^{+}_{3}\ p^{t}_{1}\ p^{t}_{3} \end{pmatrix}. \tag{5.67}$$

Norint nurodyti ėjimų kryptis, pateiktoje algoritminėje RA *K*, yra apibrėžiamos trys programinių ėjimų algoritminės priklausomybės $p^{\rightarrow}_{1}, p^{\rightarrow}_{2}$ ir $p^{\rightarrow}_{3}$. Kiekviena iš jų nurodo šiam uždaviniui spręsti numatomą programinių ėjimų seką:

$$p^{\rightarrow}_{1} = p^{t}_{1}, p^{t}_{2}, p^{+}_{1}, p^{+}_{2}, p^{+}_{3}, p^{t}_{1;} \tag{5.68}$$

$$p^{\rightarrow}_{2} = p^{t}_{1}, p^{t}_{2}, p^{+}_{3}, p^{t}_{1;} \tag{5.69}$$

$$p^{\rightarrow}_{3} = p^{t}_{3.} \tag{5.70}$$

Tad tam tikra tvarka ir kryptimi programiškai einant per turimą ARA *K* ir pagal algoritminių priklausomybių kodus iškviečiant AP realizuojančius programinius modulius, bus sprendžiamas uždavinys.

5.4. Detalių, agregatų ir produktų laiko išteklių uždavinys

Laiko išteklių uždavinys yra ypač aktualus tuomet, kai turime reikalų su detalėmis, agregatais ar produktais, kurių galiojimo laikas priklauso nuo sandėliavimo ir eksploatacijos sąlygų. Suprantama, kad kai kurios detalės, agregatai ar produktai, atsižvelgiant į tai, kokiomis sąlygomis yra sandėliuojami ar eksploatuojami, atitinkamai turės tam tikrą naudojimo laiką, t. y. vienomis eksploatacijos sąlygomis ta pati detalė, agregatas ar produktas gali būti naudojamas vienokią laiko trukmę, o kitomis – kitokią, pavyzdžiui, mažesnę. Tam tikros detalės, agregato ar produkto naudojimo laiko išteklių, kuris priklauso nuo eksploatacijos sąlygų, turi nustatyti specialistas – gamintojas. Todėl šį uždavinį sprendžiant naudojamas koeficientas, kuris leidžia atitinkamai apskaičiuoti detalės, agregato ar produkto naudojimo laiką esant tam tikrai sandėliavimo ar eksploatacijos sąlygai. Turėdami detalės, agregato ar produkto bendrą eksploatacijos laiką, t. y. suminį laiką visomis eksploatacijos sąlygomis, galime palyginti šį laiką su gamintojo apibrėžtu bendrojo naudojimo laiko rezervu. Jeigu bendras eksploatacijos laikas viršija arba yra lygus bendrojo naudojimo laiko rezervui, tada būtina nutraukti šias detalės, agregato ar produkto naudojimą. Turint nedidelius kiekius detalių, agregatų ar produktų, kurių naudojimo laikas priklauso nuo sandėliavimo ar eksploatacijos sąlygų, naudotojui nėra sunku stebėti jų galiojimo laiką. Tačiau naudotojui yra gana sunku tai stebėti, kai yra dideli kiekiai detalių, agregatų ar produktų, o ypač sunku tuo metu, kai jie naudojami eksploatuojant kitus objektus, kurių kiekiai taip pat dideli.

Analizuojamas detalių, agregatų ar produktų laiko išteklius uždavinio sprendimas, naudojant laiko išteklių apskaitos duomenis, gali būti realizuojamas tokia eiga. Tarkime, kad turime hipotetinius duomenis apie orlaivius, kurių detalių ir agregatų naudojimo laikas priklauso nuo orlaivių eksploatacijos sąlygų. Tuo metu būtina sudaryti dvi reliacines aibes. Pirmoji RA (5.13 lentelė) turės duomenis apie orlaivius ir jų eksploatacijos laiką tam tikromis sąlygomis. Antroji RA (5.14 lentelė) bus užpildyta duomenimis apie detalių ir agregatų, kurie naudojami šiuose orlaiviuose, naudojimo laiką, įvertinus orlaivių eksploatacijos sąlygas. Šios orlaivių detalių ir agregatų naudojimo aplinkybės nėra tikri eksploatacijos parametrai. Tai uždavinio adaptacijos galimybėmis rodyti skirti duomenys. Realioje aplinkoje orlaivių detalių ir agregatų eksploatacija turi daug didesnę aibę skirtingų sąlygų.

5.13 lentelė. *RA ir jos duomenys apie orlaivius ir jų eksploatacijos laiką esant tam tikromis temperatūros režimo sąlygom*

Orlaivio kodas	**Orlaivio modelis**	**Detalės (agregato) kodas**	**Eksploatacijos sąlyga A: > – 40 ° C, h**	**Eksploatacijos sąlyga B: ≤ – 40 ° C ir ≤ + 30 ° C, h**	**Eksploatacijos sąlyga C: > + 30 °C, h**
C_1	C_2	C_3	C_4	C_5	C_6
OR547	Boeing 737-300	DIN 931	100	8567	44
OR547	Boeing 737-300	DIN 826	100	8567	44
OR548	Boeing 737-500	DIN 827	170	8359	20
OR549	Boeing 737-500	DIN 796	30	4830	32

Pateikta RA identifikuojama RA kodu a_1 ir subjekto kodu b_1, kuris šiuo atveju identifikuoja sandėlį. Orlaivius identifikuos raktinio atributo „Orlaivio kodas“ reikšmės. Šios RA atributo „Orlaivio modelis“ reikšmes apibrėžia gamintojas, o atributus „Orlaivio kodas“, „Detalės (agregato) kodas“, „Eksploatacijos sąlyga A: > – 40 °C“, „Eksploatacijos sąlyga B: ≤ – 40 °C ir ≤ + 30 °C“ ir „Eksploatacijos sąlyga C: > + 30 °C“ – turi nustatyti naudotojas. Suprantama, kad orlaivių eksploatacijos sąlygos turi būti nustatomos specialistų, remiantis techninėmis orlaivių charakteristikomis.

RA yra užfiksuoti duomenys (5.14 lentelė) apie detales ir agregatus pagal tai, kokiame tiksliai orlaivyje jie naudojami, taip pat pateikiamas gamintojo nustatytas bendras detalės ar agregato naudojimo laiko išteklius. Ši reikšmė nurodo tam tikros detalės ar agregato naudojimo trukmę, neatsižvelgiant į eksploatacijos sąlygas.

Šiai RA priskiriami tokie identifikatoriai: a_2 b_1. Detales ir agregatus identifikuojančiu atributu priskiriamas atributas „Detalės (agregato) kodas“. Atributų „Orlaivio kodas“ ir „Detalės (agregato) kodas“ reikšmes nustato naudotojas, atributų „Detalė (agregatas)“ ir „Bendras laiko išteklius“ – gamintojas, o atributo „Išteklių koeficientas“ – specialistas. Šį koeficientą specialistas turi parinkti atsižvelgęs į konkrečias tam tikros detalės ar agregato eksploatacijos sąlygas, kadangi atsižvelgiant nuo šias sąlygas gali kisti ir konkrečios detalės ar agregato eksploatacijos laikas. Kitaip tariant, eksploatacijos laikas tam tikromis sąlygomis yra kintamas dydis, kuris atitinkamai priklauso nuo koeficiento. Todėl į šio RA įvestas atributas „Bendras eksploatacijos laikas“, kurio reikšmes reikia apskaičiuoti sudauginus pasirinktos sąlygos atributų „Išteklių koeficientas“ ir „Eksploatacijos laikas“ reikšmes.

5.14 lentelė. *RA apie detalių ir agregatų naudojimo trukmę, įvertinus eksploatacijos sąlygas*

Detalė (agregato) kodas	Detalė (agregatas)	Bendras laiko išteklius, h	Orlaivio kodas	Eksploatacijos sąlygos A			Eksploatacijos sąlygos B			Eksploatacijos sąlygos C		
				Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h	Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h	Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h
C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}
DIN 931	Sraigtas	13160	OR547	1,1			1			1,2		
DIN 826	Stabdžių kaladėlės	8760	OR547	1,3			1			1,5		
DIN 827	Propeleris	26280	OR548	1,1			1			1,1		
DIN 796	Padangos	52560	OR549	1,3			1			1,5		

Toliau sulyginsime orlaivių RA (5.13 lentelė) raktinių atributų reikšmes su detalių ar agregatų RA (5.14 lentelė) pateiktomis, orlaivius ir detales (agregatus) identifikuojančiomis raktinių atributų reikšmėmis. Šioms reikšmėms sutapus, t. y. suradę tą patį orlaivį ir tą pačią detalę (agregatą) skirtingose RA, atliksime duomenų, nurodančių, kiek valandų tam tikromis sąlygomis identifikuotas orlaivis buvo eksploatuojamas, transformaciją. Tai realizuosime taikydami sąlyginę transformaciją p^t. Dėl aiškumo kiekvienos aprašomos algoritminės priklausomybės kodas yra papildytas indeksu. Pirmas indekso skaičius nurodo AP pobūdį bendrąja prasme, o antras išskiria konkrečią AP šio uždavinio aprašytame sprendime.

$$p^{t}_{1,1}\{\ 1/1,\ 1/3\ \}^1 < 1/4,\ 1/5,\ 1/6 >^1 \ = \longrightarrow \{\ 1/4,\ 1/1\ \}^2 < 1/6,\ 1/9,\ 1/12 >^2. \quad (5.71)$$

Remiantis šia sąlyginės transformacijos išraiška bus lyginamos 1/1, 1/3 koordinatėse užfiksuotos atributų reikšmės iš pirmos RA su 1/4, 1/1 koordinatėse užfiksuotomis reikšmėmis iš antros RA. Šioms reikšmėms sutapus, t. y. jeigu jos atitiks išraiškoje apibrėžtą sąlygą, bus atliekama duomenų, užfiksuotų pirmos RA 1/4, 1/5, 1/6 koordinatėse, transformacija į antros RA koordinates 1/6, 1/9, 1/12.

Kad būtų aiškiau, vietoj išraiškoje apibrėžtų lyginamų atributų reikšmių koordinačių pateikiamos tų atributų konkrečios reikšmės:

$$\{\text{OR547, DIN 931}\} \xrightarrow{=} \{\text{OR547, DIN 931}\}. \qquad (5.72)$$

Analizuojamu atveju, $p^{t}_{1,2}$ sąlyginės transformacijos išraiškoje lyginamos atributų reikšmės visiškai atitinka lyginimo sąlygą, vadinasi, bus atlikta duomenų transformacija. Tuo pačiu principu, tik keisdami atributų koordinates, transformuosime ir visas kitas atributų „Eksploatacijos sąlyga A: > – 40 °C“, „Eksploatacijos sąlyga B: ≤ – 40 °C ir ≤ + 30 °C“ ir „Eksploatacijos sąlyga C: > + 30 °C“ reikšmes:

$$p^{t}_{1,2}\{\ 2/1, 2/3\ \}^{1} < 2/4, 2/5, 2/6 >^{1} \xrightarrow{=} \{\ 2/4, 2/1\ \}^{2} < 2/6, 2/9, 2/12 >^{2}, \qquad (5.73)$$

$$p^{t}_{1,3}\{\ 3/1, 3/3\ \}^{1} < 3/4, 3/5, 3/6 >^{1} \xrightarrow{=} \{\ 3/4, 3/1\ \}^{2} < 3/6, 3/9, 3/12 >^{2}, \qquad (5.74)$$

$$p^{t}_{1,4}\{\ 4/1, 4/3\ \}^{1} < 4/4, 4/5, 4/6 >^{1} \xrightarrow{=} \{\ 4/4, 4/1\ \}^{2} < 4/6, 4/9, 4/12 >^{2}. \qquad (5.75)$$

Reliacinė aibė, skirta detalių ir agregatų naudojimo trukmei užfiksuoti, su transformuotais duomenimis realizavus $p^{t}_{1,2}$, $p^{t}_{1,3}$, $p^{t}_{1,4}$ algoritmines priklausomybes pateikiama 5.15 lentelėje. Išsaugojus turimoje RA duomenis, kurie gauti realizavus pastarąsias skaičiavimo AP, turėsime RA, kuri pateikta 5.16 lentelėje.

Norint sužinoti ar detalei (agregatui) gamintojo priskirtas bendras naudojimo laiko išteklius nėra viršytas arba lygus, pirmiausia, būtina apskaičiuoti bendrą eksploatacijos laiką visomis eksploatacijos sąlygomis. Tai atliksime atributo „Išteklių koeficientas“ reikšmę padauginus iš atributo „Eksploatacijos laikas“ reikšmės. Šį aritmetinį veiksmą realizuosime naudodami skaičiavimo algoritminę priklausomybę p^{+}:

$$p^{+}_{2,5}(1/5 \times 1/6 \rightarrow 1/7), \qquad 1{,}1 \text{ x } 100 = 110. \qquad (5.76)$$

Čia 1/5 ir 1/6 yra atributų reikšmių, su kuriomis atliekama daugybos operacija, koordinatės , o 1/7 – tai koordinatės, kur bus įsimenamas gautas rezultatas. Kad būtų aiškiau, kokie tiksliai duomenys dalyvauja ir kokie gauti, taikant šią skaičiavimo algoritminę priklausomybę, greta pateikti skaičiavimai, kur vietoj atributų koordinačių įrašytos konkrečios atributų reikšmės.

Taikydami tą pačią skaičiavimo algoritminę priklausomybę, tik jos išraiškoje apibrėžę kitas atributų reikšmių koordinates, apskaičiuojame kitų turimų detalių ir agregatų bendrą eksploatacijos trukmę tam tikroms eksploatacijos sąlygomis:

$$p^{+}_{2,6}(1/8 \times 1/9 \rightarrow 1/10), \qquad 1 \times 8567 = 8567, \qquad (5.77)$$

$$p^{+}_{2,7}(1/11 \times 1/12 \rightarrow 1/13), \qquad 1{,}2 \times 44 = 53, \qquad (5.78)$$

$$p^{+}_{2,8}(2/5 \times 2/6 \rightarrow 2/7), \qquad 1{,}3 \times 100 = 130, \qquad (5.79)$$

$$p^{+}_{2,9}(2/8 \times 2/9 \rightarrow 2/10), \quad 1 \times 8567 = 8567, \quad (5.80)$$

$$p^{+}_{2,10}(2/11 \times 2/12 \rightarrow 2/13), \quad 1{,}5 \times 44 = 66, \quad (5.81)$$

$$p^{+}_{2,11}(3/5 \times 3/6 \rightarrow 3/7), \quad 1{,}1 \times 170 = 187, \quad (5.82)$$

$$p^{+}_{2,12}(3/8 \times 3/9 \rightarrow 3/10), \quad 1 \times 8359 = 8359, \quad (5.83)$$

$$p^{+}_{2,13}(3/11 \times 3/12 \rightarrow 3/13), \quad 1{,}1 \times 20 = 22, \quad (5.84)$$

$$p^{+}_{2,14}(4/5 \times 4/6 \rightarrow 4/7), \quad 1{,}3 \times 30 = 39, \quad (5.85)$$

$$p^{+}_{2,15}(4/8 \times 4/9 \rightarrow 4/10), \quad 1 \times 4830 = 4830, \quad (5.86)$$

$$p^{+}_{2,16}(4/11 \times 4/12 \rightarrow 4/13), \quad 1{,}5 \times 32 = 48. \quad (5.87)$$

Išsaugoję turimoje RA duomenis, kurie gauti realizavus šias skaičiavimo algoritmines priklausomybes, turėsime RA, kuri pateikta 5.16 lentelėje.

5.15 lentelė. *RA apie detalių ir agregatų naudojimo trukmę su transformuotais duomenimis*

Detalė (agregato) kodas	Detalė (agregatas)	Bendras laiko išteklius, h	Orlaivio kodas	Eksploatacijos sąlygos A			Eksploatacijos sąlygos B			Eksploatacijos sąlygos C		
				Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h	Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h	Išteklių koeficientas	Išteklių koeficientas	Eksploatacijos laikas, h
C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}
DIN 931	Sraigtas	13160	OR547	1,1	100		1	8567		1,2	44	
DIN 826	Stabdžių kaladėlės	8760	OR547	1,3	100		1	8567		1,5	44	
DIN 827	Propeleris	26280	OR548	1,1	170		1	8359		1,1	20	
DIN 796	Padangos	52560	OR549	1,3	30		1	4830		1,5	32	

5.16 lentelė. *RA apie detalių ir agregatų naudojimo trukmę su naujais duomenimis*

Detalė (agregato) kodas	Detalė (agregatas)	Bendras laiko išteklius, h	Orlaivio kodas	Eksploatacijos sąlygos A			Eksploatacijos sąlygos B			Eksploatacijos sąlygos C		
				Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h	Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h	Išteklių koeficientas	Išteklių koeficientas	Bendras eksploatacijos laikas, h
C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}
DIN 931	Sraigtas	13160	OR547	1,1	100	110	1	8567	8567	1,2	44	53
DIN 826	Stabdžių kaladėlės	8760	OR547	1,3	100	130	1	8567	8567	1,5	44	66
DIN 827	Propeleris	26280	OR548	1,1	170	187	1	8359	8359	1,1	20	22
DIN 796	Padangos	52560	OR549	1,3	30	39	1	4830	4830	1,5	32	48

Atskirai įvertinus ir apskaičiavus kiekvienos detalės ar agregato naudojimo laiką tam tikromis eksploatacijos sąlygomis, liko apskaičiuoti bendrą tam tikros detalės ar agregato naudojimo trukmę ir gautą rezultatą palyginti su tai pačiai detalei ar agregatui gamintojo apibrėžtu naudojimo bendruoju laiko ištekliumi.

Norint apskaičiuoti tam tikros detalės ar agregato bendrą naudojimo laiką visomis eksploatacijos sąlygomis, būtina realizuoti skaičiavimo algoritminę priklausomybę, kuri atliktų sudėties aritmetinę operaciją:

$$p^{+}_{3,17}(1/7 + 1/10 + 1/13 \rightarrow 1/14), \qquad 110 + 8567 + 53 = 8730. \tag{5.88}$$

Pagal šią išraišką sudėtis bus atliekama su atributų, kurių koordinatės 1/7, 1/10 ir 1/13, reikšmėmis ir gautas rezultatas įsimenamas, šiuo atveju tos pačios RA, iš kurios imami duomenys, naujame domene, t. y. atribute pavadinimu „Suminis eksploatacijos laikas”. Detalūs kitų, tos pačios RA, detalių ir agregatų bendro naudojimo laiko skaičiavimai atrodo taip (5.17 lentelė):

$$p^{+}_{3,18}(2/7 + 2/10 + 2/13 \rightarrow 2/14), \qquad 130 + 8567 + 66 = 8763, \tag{5.89}$$

$$p^{+}_{3,19}(3/7 + 3/10 + 3/13 \rightarrow 3/14), \qquad 187 + 8359 + 22 = 8568, \tag{5.90}$$

$$p^{+}_{3,20}(4/7 + 4/10 + 4/13 \rightarrow 4/14), \qquad 39 + 4830 + 48 = 4917. \tag{5.91}$$

5.17 lentelė. *Detalių ir agregatų RA, užfiksavus paskutinius skaičiavimų rezultatus*

Detalė (agregato) kodas	Detalė (agregatas)	Bendras laiko išteklius, h	Orlaivio kodas	Eksploatacijos sąlygos A			Eksploatacijos sąlygos B			Eksploatacijos sąlygos C			Suminis eksploatacijos laikas, h.
				Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h	Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h	Išteklių koeficientas	Eksploatacijos laikas, h	Bendras eksploatacijos laikas, h	
C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}
DIN 931	Sraig-tas	13160	OR547	1,1	100	110	1	8567	8567	1,2	44	53	8730
DIN 826	Stabdžių kaladėlės	8760	OR547	1,3	100	130	1	8567	8567	1,5	44	66	8763
DIN 827	Propele-ris	26280	OR548	1,1	170	187	1	8359	8359	1,1	20	22	8568
DIN 796	Padan-gos	52560	OR549	1,3	30	39	1	4830	4830	1,5	32	48	4917

Paskutiniu šio uždavinio sprendimo etapu reikia palyginti duomenis. Tada bus aišku, ar tam tikros detalės (agregato) bendras naudojimo laikas, gautas įvertinus visas jos eksploatacijos sąlygas, neviršija gamintojo apibrėžto bendro naudojimo laiko rezervo. Duomenų kontrolė atliekama naudojant sąlyginę transformaciją:

$$p^{t}_{4,21}\{1/14\}^2 < 1/1,\ 1/4 >^2 \xrightarrow{\geq} \{1/3\}^2 < 1/1,\ 1/2 >^3, \qquad (5.92)$$

$$\{8730\} \xrightarrow{\geq} \{13160\}. \qquad (5.93)$$

Čia lyginamos a_2 RA pirmo kortežo koordinačių 1/l4 ir 1/3 reikšmės. Kadangi sąlyga nepatenkinta, t. y. gamintojo nustatytas naudojimo laiko išteklius neviršytas, duomenys nebus transformuojami į naują RA. Toliau bus atliekama ta pati sąlyginė transformacija su kitų kortežų tų pačių atributų reikšmių koordinatėmis:

$$p^{t}_{4,22}\{2/l\,4\}^2 < 2/1,\ 2/\cdot\ ^2 \xrightarrow{\geq} \{2/3\}^2 < 1/1,\ 1/2 >^3, \qquad (5.94)$$

$$\{8763\} \xrightarrow{\geq} \{8760\}. \qquad (5.95)$$

Šiuo atveju sąlyga patenkinta, todėl a_2 reliacinės aibės 2/1, 2/4 koordinatėse pateiktos reikšmės bus transformuojamos į naujai sudarytos (šiuo atveju trečios) RA koordinates 1/1, 1/2. Tai reiškia, kad šios detalės ar agregato naudojimo laikas viršijo gamintojo apibrėžtą bendrą naudojimo laiko išteklių. Todėl šios detalės ar agregato duomenys buvo transformuoti į aibę tų detalių ir agregatų, kurių naudojimą reikia nutraukti. Tokiu būdu naudotojas gauna informaciją apie detales ir agregatus, kurias būtina pakeisti.

Atlikus atributų reikšmių, kurių koordinatės buvo pateiktos $p^{t}_{4,22}$ išraiškoje, transformaciją, toliau baigsime vykdyti sąlyginę transformaciją su kituose kortežuose pateiktų tų pačių atributų reikšmių koordinatėmis:

$$p^{t}_{4,23}\{3/\text{l}\,4\}^2 < 3/1,\ 3/4 >^2 \xrightarrow{\geq} \{3/3\}^2 < 2/1,\ 2/2 >^3, \tag{5.96}$$

$$\{8568\} \xrightarrow{\geq} \{26280\}; \tag{5.97}$$

$$p^{t}_{4,24}\{4/\text{l}\,4\}^2 < 4/1,\ 4/4 >^2 \xrightarrow{\geq} \{4/3\}^2 < 2/1,\ 2/2 >^3, \tag{5.98}$$

$$\{4917\} \xrightarrow{\geq} \{52560\}. \tag{5.99}$$

Realizavus šias algoritmines priklausomybes turimų duomenų kontrolė bus baigta.

Analizuojamu atveju orlaivių detalių ir agregatų laiko išteklių uždaviniui spręsti buvo naudojamos tokios algoritminės priklausomybės:

$p^{t}_{1,1}$, $p^{t}_{1,2}$, $p^{t}_{1,3}$, $p^{t}_{1,4}$, $p^{+}_{2,5}$, $p^{+}_{2,6}$, $p^{+}_{2,7}$, $p^{+}_{2,8}$, $p^{+}_{2,9}$, $p^{+}_{2,10}$, $p^{+}_{2,11}$, $p^{+}_{2,12}$, $p^{+}_{2,13}$, $p^{+}_{2,14}$, $p^{+}_{2,15}$, $p^{+}_{2,16}$, $p^{+}_{3,17}$, $p^{+}_{3,18}$, $p^{+}_{3,19}$, $p^{+}_{3,20}$, $p^{t}_{4,21}$, $p^{t}_{4,22}$, $p^{t}_{4,23}$, $p^{t}_{4,24}$.

Tuo metu, šio uždavinio ARA L bus:

$$L = \left(p^{t}_{1}\ p^{+}_{2}\ p^{+}_{3}\ p^{t}_{4}\right). \tag{5.100}$$

Šioje algoritminėje RA L programiniai ėjimai turi būti vykdomi tokia seka:

$$\vec{p} = p^{t}_{1}, p^{+}_{2}, p^{+}_{3}, p^{t}_{4}. \tag{5.101}$$

Tad aprašytas orlaivių detalių ir agregatų laiko išteklių uždavinys bus sprendžiamas programiškai einant algoritminių priklausomybių aibe L, pagal AP $\vec{p}$ nurodytą ėjimų kryptį ir tvarką, ir besikreipiant į algoritminių priklausomybių kodus, kuriuos naudojant bus iškviečiami jas realizuojantys programiniai moduliai.

5.5. Penktojo skyriaus išvados

Taikant sukurtą adaptyviąją duomenų apdorojimo projektavimo technologiją šiame skyriuje aprašyti sandėlio apskaitos, buhalteriniai ir detalių laiko išteklių taikomojo pobūdžio uždavinių sprendimai algoritminių priklausomybių lygmeniu. Pateikti sprendimai parodo, kad aprašytiems uždaviniams spręsti taikomos tos pačios AP, tik skirtingi jų kiekiai ir tvarka. Uždavinių sprendimuose keičiasi tik bendrasis konkretaus uždavinio duomenų apdorojimo algoritmas, kuris ir daro įtaką uždavinio išsprendimui. Taigi įvairiems uždaviniams spręsti naudojami tie patys algoritminių priklausomybių algoritmai ir fiziškai tie patys tų priklausomybių realizavimo programiniai moduliai.

Tad taikydami adaptyviąją duomenų apdorojimo projektavimo technologiją, kuri naudodama vienus ir tuos pačius algoritmines priklausomybes realizuojančius programinius modulius, įvairiomis jų naudojimo sekomis ir skirtingais kiekiais, nekurdami naujų programų galime išspręsti įvairius taikomojo pobūdžio uždavinius, kurių pirminiai ir rezultatiniai duomenys yra reliacinės aibės.

6

Bendrosios išvados

1. Sukurta adaptavimo metodika, skirtingai negu daugelyje mokslo darbų, orientuota ne į objektinį modelį, o į reliacines duomenų aibes. Tai susiaurina adaptyvumo principų pritaikymo sferą, tačiau leidžia praktiškai išspręsti bet kuriuos taikomuosius uždavinius, kurių duomenys išreikšti reliacinėmis aibėmis.
2. Sukurta duomenų modelių adaptavimo taikomiesiems uždaviniams spręsti metodika, kuri leidžia automatizuotai projektuoti taikomuosius uždavinius ir automatiškai juos spręsti, kai duomenų struktūros ir turinys atitinka uždavinio pradines sprendimo sąlygas.
3. Sukurta algoritminių duomenų priklausomybių koncepcija ir realizuotos programinės algoritminės priklausomybės, kurios reliacinių aibių plotmėje leidžia papildyti algoritminių priklausomybių aibę naujomis, tai probleminei sričiai charakteringomis algoritminėmis priklausomybėmis. Nauji uždaviniai gali būti sprendžiami iki 2-jų kartų efektyviau, nes pakanka parinkti jau egzistuojančias duomenų sudarymo bei apdorojimo algoritmines priklausomybes ir nustatyti jų naudojimo eilę.
4. Realizuotieji adaptyvumo principai sudaro galimybes spręsti skirtingus taikomojo pobūdžio uždavinius tomis pačiomis algoritminėmis priklausomybėmis ir jas realizuojančiais programiniais moduliais, ir realizuoti pagrindinį adaptyvumo principą – skirtingiems taikomiesiems uždaviniams spręsti naudoti tų pačių algoritminių priklausomybių ir jas realizuojančių programinių modulių aibių skirtingus poaibius.

5. Disertacijoje realizuoti skirtingų taikomųjų uždavinių pavyzdžiai algoritminių priklausomybių lygmeniu, kurie patvirtina pateiktojo adaptavimo gebėjimus.

6. Darbe pateikto tyrimo rezultatai ir galimybės nesietinos su konkrečia DBVS ar programavimo kalba. Tai leidžia praplėsti siūlomo adaptyvaus taikymo sferą ir keitimąsi duomenimis su egzistuojančiomis duomenų bazėmis, nes galima sudaryti algoritminę priklausomybę, galinčią reliacinę aibę transformuoti į DB struktūrą.

Literatūros sąrašas

[1] Abdelmeged, A.; Lieberherr, K. J. Recursive Adaptive Computations Using Perobject Visitors. In *Proceedings of the 22nd ACM SIGPLAN Conference on Object Oriented Programming Systems and Applications Companion,* Montreal, 2007, p. 825–826.

[2] Adomėnas, P. G. Data Functional Feature Sets and their Adaptability. In *Proceedings of V East–European Conference on Advances in Databases and Information Systems*, Vilnius, 2001, p. 131–140.

[3] Адоменас, П.-Г. *Адаптивные модели и программы.* Вильнюс: Мокслас, 1992.

[4] Adomėnas, P. G.; Čiučelis, A. Data Aggregation Sets in Adaptive Data Model. *Informatica*, 2002, Vol. 13, No. 4, p. 381–392.

[5] Bekke, J. H. *Semantic Data Modeling*. Prentice Hall, 1992.

[6] Blando, L. The Eel Compiler Using Adaptive Object Oriented Programming and Demeter/Java [žiūrėta 2007 05 08]. Prieiga per internetą: <http://www.blando.info/luis/eelc/eelc v20.htm>

[7] Booch, G.; Cummings, B. *Object Oriented Analysis and Design with Applications*. Addison-Wesley Professional, 1994.

[8] Cacace, F.; Lamperti, G. *Advanced Relational Programming: Mathematics and Its Applications*. Kluwer, 1996.

[9] Chadwick, B.; Skotiniotis, T.; Lieberherr, K. J. Functional Visitors Revisited. Technical Report NU-CCIS-06-03, Northeastern University, Boston, 2006.

[10] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 1970, Vol. 13, No. 6, p. 377–387.

[11] Codd, E. F. *The relational model for database management: version 2.* Addison-Wesley Professional, 1990.

[12] Cohen, G.; Chase, J.; Kaminsky, D. Automatic Program Transformation with JOIE. In *Proceedings of USENIX Annual Technical Symposium*, 1998, p. 167–178.

[13] Čaplinskas, A. *Programų sistemų inžinerijos pagrindai (I ir II dalys)*. Matematikos ir informatikos institutas, 1998.

[14] Date, C. J. *Database in Depth.* O'Reilly, 2005.

[15] Date, C. J. *The Database Relational Model: A Retrospective Review and Analysis*. Addison-Wesley Professional, 2001.

[16] Demeter Research Group. Online Material on Adaptive Programming and Demeter [žiūrėta 2007 05 08]. Prieiga per internetą: <http://www.ccs.neu.edu/research/demeter>

[17] Elmasri, R.; Navathe, S. B. *Fundamentals of Database Systems.* Addison-Wesley Professional, 2007.

[18] Fowler, M. *Analysis Patterns: Reusable Object Models*. Addison-Wesley Professional, 1996.

[19] Freeman, E.; Sierra, K.; Bates, B. *Head First Design Patterns*. O'Reilly, 2004.

[20] Gamma, E.; Vlissides, J. *Design patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley Professional, 1995.

[21] Garcia-Molina, H.; Ullman, J. D.; Widom, J. *Database system implementation*. Prentice Hall, 2000.

[22] Haeuer, A.; Saake, G. *Datenbanken, Konzepte und Sprachen*. International Thomson Publishing Group, 1995.

[23] Hernandez, M. J. *Database Design for Mere Mortals™: A Hands-On Guide to Relational Database Design*. Addison Wesley Professional, 2003.

[24] Holland, I. M.; Lieberherr, K. J. Object-Oriented Design. *Journal ACM Computing Surveys*, 1996, Vol. 28, No. 1, p. 273–275.

[25] Hunt, A.; Thomas, D. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999.

[26] Yourdan, E.; Constantine, L. *Structured Design.* Prentice-Hall, 1979.

[27] Jay, C. B.; Palsberg, J. The Essence of the Visitor Pattern. In *Proceedings of 22nd International Computer Software and Applications Conference (COMPSAC)*, Vienna, 1998, p. 9–15.

[28] Keszenheimer, K.; Lieberherr, K. J. Incremental testing of adaptive software. Technical Report NU-CCS-94-22, Northeastern University, Boston, 1994.

[29] Kroha, P. Adaptive Programming. University of Technology Chemnitz, Germany [žiūrėta 2007 05 08]. Prieiga per internetą: <http://www.datakon.cz/datakon04/d04_it_kroha.pdf>

[30] Lämmel, R.; Visser, E.; Visser, J. Strategic programming meets adaptive programming. In *Proceedings of Aspect-Oriented Software Development (AOSD)*, Boston, 2003, p. 168–177.

[31] Lieberherr, K. J. *Adaptive Object Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing Company, 1996.

[32] Lieberherr, K. J. Workshop on adaptable and adaptive software. *Journal of OOPS Messenger*, 1995, Vol. 6, No. 4, p.149–154.

[33] Lieberherr, K. J.; Lorenz, D. H.; Mezini, M. Building modular object-oriented systems with reusable collaborations. In *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, 2000, p. 821–829.

[34] Lieberherr, K. J.; Orleans, D.; Ovlinger, J. Aspect-Oriented Programming with Adaptive Methods. *Journal of Communications of the ACM*, 2001, Vol. 44, Iss. 10, p. 39–41.

[35] Lieberherr, K. J.; Patt-Shamir, B.; Pradhan, S. An Efficient Compiler for Adaptive Programs. Technical Report NU-CCS-97-03, Northeastern University, Boston, 1997.

[36] Lieberherr, K. J.; Xiao, C. Minimizing dependency on class structures with adaptive programs. *Collection of International Symposium on Object Technologies for Advanced Software*, 1993, Vol. 742, p. 424–441.

[37] Lieberherr, K. J. Component enhancement: An adaptive reusability mechanism for groups of collaborating classes. In *Processing of 12th World Computer Congress*, Madrid, 1992, p. 179–185.

[38] Lieberherr, K. J. Controlling the Complexity of Software Designs. In *Proceedings of International Conference on Software Engineering (ICSE),* Edinburgh, 2004, p. 2–11.

[39] Lieberherr, K. J. From transience to persistence in object-oriented programming: architectures and patterns. *Journal of ACM Computing Surveys*, 1996, Vol.28, Iss. 4, p. 691–700.

[40] Lieberherr, K. J. *The Art of Growing Adaptive Object-Oriented Software.* PWS Publishing Company, 1995.

[41] Lieberherr, K. J.; Lorenz, D. *Coupling Aspect-Oriented and Adaptive Programming.* Book chapter in: *Aspect-Oriented Software Development*. Addison Wesley Professional, 2004.

[42] Lieberherr, K. J.; Orleans, D. Preventive Program Maintenance in Demeter/Java. In *Proceedings of International Conference on Software Engineering (ICSE)*, Boston, 1997, p. 604–605.

[43] Lieberherr, K., J.; Patt-Shamir, B.; Orleans, D. Traversals of object structures: Specification and efficient implementation. *Journal of ACM Transactions on Programming Languages and Systems*, 2004, Vol. 26, No. 2, p. 370–412.

[44] Lieberherr, K. J.; Silva-Lepe, I.; Xiao, C. Adaptive object-oriented programming using graph-based customization. *Journal of Communications of the ACM*, 1994, Vol. 37, No. 5, p. 94–101.

[45] Lieberherr, K. J.; Wand, M. Traversal semantics in object graphs. Technical Report NU-CCS-2001-05, Northeastern University, Boston, 2001.

[46] Lieberherr, K. J.; Holland, I. M.; Hürsch, W. L.; Silva-Lepe, I.; Xiao, C. Demeter Tools/C++. *Journal of OOPS Messenger*, 1993, Vol. 4, No. 2, p. 233–241.

[47] Lopes, C. V.; Lieberherr, K. J. Abstracting Process-to-Function Relations in Concurrent Object-Oriented Applications. *Journal of Lecture Notes in Computer*, 1994, Vol. 821, p. 81–99.

[48] Martin, R. *Agile Software Development: Principles, Patterns and Practices, Upper Saddle River*. Prentice Hall, 2002.

[49] Mätzel, K. U.; Bischofberger, W. R. Designing object systems for evolution. *Journal of Theory and Practice of Object Systems*, 1997, Vol. 3, Iss. 4, p. 265–283.

[50] McLaughlin, B.; Pollice, G.; West, D. *Head First Object-Oriented Analysis and Design*. O'Reilly, 2006.

[51] Meyer, B. *Object-Oriented software construction*. Prentice Hall, 2000.

[52] Melnik, S.; Garsie, H. Adaptive Algorithms for Set Containment Joins, *ACM Transactions on Database Systems (TODS)*, 2003, Vol. 28, Iss. 1, p. 56–99.

[53] Mezini, M.; Lieberherr, K. Adaptive Plug-and Play Components for Evolutionary Software Development. In *Proceedings of.Conference on Object-Oriented programming, Systems, Languages and Applications*, New York, 1998, p. 97–116.

[54] Narendra, K. S.; Annaswamy, A. M. *Stable Adaptive Systems*. Dover Publications, 2005.

[55] Norvig, P.; Cohn, D. Adaptive Software [žiūrėta 2007 05 08]. Prieiga per internetą: <http://norvig.com/adapaper-pcai.html>

[56] Orleans, D.; Lieberherr, K. J. DJ: Dynamic Adaptive Programming in Java. In *Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, Kyoto, 2001, p. 73–80.

[57] Palsberg, J. Class-graph Inference for Adaptive programs. *Journal of Theory and Practice of Object Systems,* 1997, Vol. 3, No. 2, p. 75–85.

[58] Palsberg, J.; Patt-Shamir, B.; Lieberherr, K. J. A new approach to compiling adaptive programs. *Journal of Science of Computer Programming*, 1997, Vol. 29, No. 3, p. 303–326.

[59] Palsberg, J.; Xiao, C.; Lieberherr, K. J. Efficient implementation of adaptive software. *Journal of ACM Transactions on Programming Languages and Systems*, 1995, Vol. 17, No. 2, p. 264–292.

[60] Riordan, R. M. *Designing Effective Database Systems.* Addison Wesley Professional, 2005.

[61] Robillard, M., P. Separation of Concerns and Software Components [žiūrėta 2007 05 08]. Prieiga per internetą: <http://www.idt.mdh.se/kurser/phd/CBSE/reports/Final_reports/report_06.pdf>

[62] Rojas, F.; Lieberherr, K. J. Applying Traversals Over Derived Edges. Technical Report NU-CCIS-03-12, Northeastern University, Boston, 2003.

[63] Sangal, N.; Farrell, E.; Lieberherr, K. J.; Lorenz, D. Interaction Schemata: Compiling Interactions to Code. In *Proceedings of the*

Technology of Object-Oriented Languages and Systems (TOOLS), Santa Barbara, 1999, p. 268–277.

[64] Seiter, L. M.; Hursch, W. L. Adaptive behavioral components: Bridging the class-module gap. Technical Report NU-CCS-95-13, College of Computer Science, Northeastern University, Boston, 1995.

[65] Seiter, L. M.; Palsberg, J.; Lieberherr, K. J. Evolution of Object Behavior Using Context Relations. *Journal of IEEE Transactions on Software Engineering*, 1998, Vol. 24, No.1, p. 79–92.

[66] Sherman, P. D.; Jose, S. Demeter in the Database. Paper (107-2007) of SAS Global Forum 2007 [žiūrėta 2007 05 08]. Prieiga per internetą: <http://www2.sas.com/proceedings/forum2007/107-2007.pdf>

[67] Skotiniotis, T.; Palm, J.; Lieberherr, K. J. Demeter Interfaces: Adaptive Programming Without Surprises. In *Proceedings of European Conference on Object-Oriented Programming*, Nantes, 2006, p. 477–500.

[68] Spit, M.; Brinkkemper, S.; Lieberherr, K. J. Integrating Adaptive Programming into Existing Object-Oriented Analysis and Design Methods: Do It Yourself Adaptiveness. In *Proceedings of OOIS,* Boston, 1996, p. 57–65.

[69] Stirewalt, K.; Dillon, L. K. Generation of visitor components that implement program transformations. In *Proceedings of ACM SIGSOFT Symposium on Software Reusability*, Toronto, 2001, p. 86–94.

[70] Sullivan, G.; Lieberherr, K. J. An Object-oriented Design Methodology. Technical Report NU-CCS-95-1, Northeastern University, Boston, 1995.

[71] Thiemann, P. Compiling Adaptive Programs by Partial Evaluation. In *Proceedings of the 9th International Conference on Compiler Construction, Lecture Notes In Computer Science* (Vol. 1781), 2000, p. 264–278.

[72] Ullman, J. D.; Widom, J. *A First Ciurse in database Systems*. Upper Saddle River, 2002.

[73] Weisfeld, M. *Object-Oriented Thought Process*. Sams, 2003.

[74] Wu, P.; Krishnamurthi, S.; Lieberherr, K. J. Traversing Recursive Object Structures: The Functional Visitor in Demeter. In *Proceedings of the Workshop on Software-Engineering Properties of Languages for Aspect Technologies*, Madrid., 2003, p. 236–248.

[75] Ханнсен, Г.; Ханнсен, Д. *Базы данных: разработка управление*. Бином, 1999.

Autorės publikacijų sąrašas disertacijos tema

Straipsniai mokslo žurnaluose, referuojamuose pripažintose tarptautinėse duomenų bazėse, kurių sąrašą sudarė Lietuvos mokslo taryba

[76A] Pliuskuvienė, B.; Adomėnas, P. G. An Adaptive Technology for Processing Data Relational Sets and its Application. *WSEAS Transactions on Information Science and Applications*, vol. 4, Iss. 4. Athens: WSEAS, 2007, p. 648–654. ISSN 1790-0832 [*INSPEC, Compendex*].

[77A] Pliuskuvienė, B.; Adomėnas, P. G. Duomenų algoritminių priklausomybių ir jų savybių kaita reliacinėse aibėse. *Lietuvos matematikos rinkinys*, t. 45. Vilnius: Matematikos ir informatikos institutas, 2005, p. 160–166. ISSN 0132-2818 [*MatSciNet, VINITI, Zentralblatt MATH*].

[78A] Pliuskuvienė, B.; Adomėnas, P. G. The aggregation of relational sets and the algorithmic dependencies of data. *WSEAS Transactions on Systems*, vol. 5, Iss. 4. Athens: WSEAS, 2006, p. 793–798. ISSN 1109-2777 [*INSPEC, Compendex*].

Straipsniai recenzuojamoje užsienio tarptautinių konferencijų medžiagoje, referuojamoje pripažintose tarptautinėse duomenų bazėse, kurių sąrašą sudarė Lietuvos mokslo taryba

[79A] Pliuskuvienė, B.; Adomėnas, P. G. The change of algorithmic data dependencies and their properties in relational sets. In *Proceedings of the 5th WSEAS International Conference on „Artificial Intelligence, Knowledge Engineering and Data Bases“ (AIKED 2006), held in Madrid on 15–17 February, 2006*. Madrid: WSEAS Press, 2006, p. 1–5. ISBN 960-8457-41-6 [*INSPEC, Compendex*].

[80A] Pliuskuvienė, B.; Adomėnas, P. G. The main adaptability principles for the process of handling relational data sets. In *Proceedings of the 6th WSEAS International Conference on „Applied Computer Science“ (ACS ’06), held in Tenerife on 16–18 December, 2006*. Tenerife: WSEAS, 2006, p. 380–384. ISBN 960-8457-57-2 [*INSPEC, Compendex*].

Straipsniai recenzuojamoje Lietuvos konferencijų

[81A] Pliuskuvienė, B.; Adomėnas, P. G. Duomenų struktūrų identifikavimo modelis. *„Informacinės technologijos 2005: aktualijos ir perspektyvos“ IV mokslinės praktinės konferencijos pranešimų medžiaga, įvykusios Alytuje gegužės 25 d., 2005*. Alytus: Alytaus kolegija, 2005, p. 157–162 ISBN 9955-9779-0-6.

[82A] Pliuskuvienė, B.; Adomėnas, P. G.; Sliesoraitytė, I. Reliacinių aibių agregavimas ir jų apdorojimo programų generavimas. *„Informacinės technologijos 2007“ konferencijos pranešimų medžiaga, įvykusios Kaune sausio 31 d., 2007.* Kaunas: Technologija, 2007, p. 351–355 ISSN 1822-6337.

Birutė Pliuskuvienė

ADAPTYVŪS DUOMENŲ MODELIAI PROJEKTAVIME

Daktaro disertacija
Technologijos mokslai, informatikos inžinerija (07T)

2008 05 10. 8 sp. l. Tiražas 20 egz.
Vilniaus Gedimino technikos universiteto
leidykla „Technika“, Saulėtekio al. 11,
LT-10223 Vilnius, *http://leidykla.vgtu.lt*
Spausdino UAB „Baltijos kopija“,
Kareivių g. 13B, LT-09109 Vilnius
www.kopija.lt