UNIVERZITET U BEOGRADU

FAKULTET ORGANIZACIONIH NAUKA

Marko V. Petrović

RAZVOJ PROCESA EKSTRAKCIJE, TRANSFORMACIJE I PUNJENJA PODATAKA SKLADIŠTA PODATAKA ZASNOVAN NA MODELOM VOĐENOM PRISTUPU

doktorska disertacija

Beograd, 2014.

UNIVERSITY OF BELGRADE

FACULTY OF ORGANIZATIONAL SCIENCES

Marko V. Petrović

A MODEL DRIVEN DEVELOPMENT APPROACH FOR THE DATA WAREHOUSE EXTRACT, TRANSFORM AND LOAD PROCESS

Doctoral Dissertation

Belgrade, 2014.

Mentor:

prof. dr Zoran Marjanović,

redovni profesor Fakulteta organizacionih nauka u Beogradu

Članovi komisije:

prof. dr Milica Vučković,
vanredni profesor Fakulteta organizacionih nauka u Beogradu

prof. dr Milija Suknović,
redovni profesor Fakulteta organizacionih nauka u Beogradu

dr Slađan Babarogić,
docent Fakulteta organizacionih nauka u Beogradu

prof. dr Vladan Jovanović,

redovni profesor Allen E. Paulson College of Engineering and Information Technology, Georgia Southern University

Datum odbrane:

_____

Datum promocije:

_____

Disertaciju posvećujem mojoj majci Ljiljani i ocu Vlastimiru

Zahvalnica

Izuzetnu i neizmernu zahvalnost želim da izrazim svojoj porodici, supruzi ***Tanji*** *i deci* ***Jovanu*** *i* ***Simoni*** *bez čije emotivne podrške ne bih ostvario ovakve rezultate.*

Zahvaljujem se mentoru ***Prof. dr Zoranu Marjanoviću*** *na ukazanom bezrezervnom poverenju i nesebičnoj podršci. Svojim iskustvom, stručnim savetima, a posebno ljudskim odnosom doprineo je da prevaziđem mnoge prepreke i dostignem cilj.*

Ogromnu zahvalnost dugujem ***Prof. dr Milici Vučković*** *koja mi je svih ovih godina bili velika podrška. Njeno iskreno zalaganje, posvećenost i spremnost da prenosi svoje profesionalno iskustvo i znanje doprineli su mom akademskom načinu razmišljanja. Ideje i stručni saveti koje mi je pružila tokom izrade disertacije direktno su doprineli njenom kvalitetu.*

Takođe želim da istaknem i izuzetno zalaganje ***dr Slađana Babarogića*** *čiji stručni saveti su mi u pojedinim teškim trenucima mnogo pomogli i razrešili razne nedoumice.*

Ovim putem želim da se zahvalim i ***Prof. dr Miliji Suknoviću*** *i* ***Prof. dr Vladanu Jovanoviću*** *na korisnim savetima i izuzetnoj saradnji koju smo ostvarili tokom izrade ovog rada.*

Želim da izrazim i veliku zahvalnost prijateljici i koleginici ***Nini Turajlić*** *na svesrdnoj podršci pri izradi ovog rada, zatim* ***mr Ivanu Bojičiću***, ***Ognjenu Turkoviću***, ***Ivanu Tamburiću*** *i* ***Milošu Nedeljkoviću*** *na kritičkom stavu prema svakoj iznetoj ideji, kao i svim prijateljima i kolegama sa Fakulteta organizacionih nauka koji su mi pružali podršku i reči ohrabrenja.*

Marko Petrović

Beograd, avgust 2014. godine

RAZVOJ PROCESA EKSTRAKCIJE, TRANSFORMACIJE I PUNJENJA PODATAKA SKLADIŠTA PODATAKA ZASNOVAN NA MODELOM VOĐENOM PRISTUPU

Rezime

U tezi je razmatran problem konceptualizacije i automatizacije razvoja ETL procesa i dato je originalno rešenje koje se zasniva na formalnoj specifikaciji ETL procesa i njenoj automatizaciji uz pomoć razvijenog specifičnog aplikacionog okvira.

U skladu sa *Domain-Specific Modeling* (DSM) pristupom, za formalnu specifikaciju definisano je nekoliko novih domensko-specifičnih jezika: jezik za specifikaciju operacija transformacija podataka (ETL-O), jezik za specifikaciju toka izvršavanja ETL procesa (ETL-P), jezik za specifikaciju izraza (ETL-E) i jezik za specifikaciju šablona operacija transformacija (ETL-T). Svaki od ovih jezika definiše koncepte koji su relevantni za specifični aspekt ETL procesa. Modelovanje ETL procesa zapravo se svodi na modelovanje određenog aspekta ETL procesa pomoću odgovarajućeg domensko-specifičnog jezika i na ovaj način se značajno smanjuje složenost modelovanja.

Implementacija specificiranih domensko-specifičnih jezika ostvarena je uvođenjem specifičnog aplikacionog okvira kao tehnološke podrške predloženoj formalnoj specifikaciji. Uvođenjem aplikacionog okvira značajno je podignut semantički nivo koji je implementaciono podržan i koji se može automatizovati. Implemetacija se zapravo zasniva na automatskoj transformaciji modela, formiranim u skladu sa odgovarajućim domensko-specifičnim jezikom, u izvrsni kôd aplikacionog okvira. Visok semantički nivo koji je implementaciono podržan aplikacionim okvirom, utiče na smanjenje broja koraka u razvoju ETL procesa, koji postaje više automatizovan i samim tim mnogo produktivniji.

Uvedena je i specifična ETL platforma (kao proširenje *Microsoft .NET platforme*) kojom je specificirana tehnološka podrška razvoju i izvršavanju modela ETL

procesa. Zapravo, ETL platformom se opisuje tehnološka podrška modelovanju (tj. formiranju modela zasnovanim na odgovarajućim domensko-specifičnim jezicima) i implementaciji domensko-specifičnih jezika. Tehnološka podrška modelovanju ostvarena je razvojem odgovarajućih softverskih alata (pre svega sintaksnih editora). Automatske transformacije modela, formiranih korišćenjem sintaksnih editora i samim tim zasnovanim na odgovarajućim domensko-specifičnim jezicima, u izvršni kôd aplikacionog okvira podržane su sa specifično razvijenim generatorima. Aplikacioni okvir podržan je skupom servisa koji omogućavaju izvršavanje i upravljanje izvršnim kôdom, generisanim iz formiranih modela.

U tezi je predstavljena opšta fizička softverska arhitektura ETL platforme, kao osnova za njenu implementaciju. Opšta fizička softverska arhitektura daje specifikaciju softverskog rešenja koje definiše osnovne komponente ETL platforme. Fizička implementacija opšte fizičke arhitekture realizovana je u *Microsoft .NET* implementacionom okruženju uz korišćenje softverskih komponenti za realizaciju komponenti iz opšte fizičke arhitekture.

U cilju validacije rešenja koje se predlaže u ovoj tezi sprovedno je više eksperimentalnih testiranja, a planiraju se i testiranja ovog rešenja u praksi u različitim domenima.

Ključne reči:
razvoj procesa ekstrakcije, transformacije i učitavanja (ETL) podataka; domensko-specifični jezik (DSL); razvoj vođen modelima (MDD)

Naučna oblast:
Tehničke nauke

Uža naučna oblast:
Informacioni sistemi

UDK broj:
004.4(043.3)

A MODEL DRIVEN DEVELOPMENT APPROACH FOR THE DATA WAREHOUSE EXTRACT, TRANSFORM AND LOAD PROCESS

Abstract

The problem of conceptualization and automatization of the ETL Process development is considered in this thesis and an original solution is proposed which is based on a formal specification of the ETL Process and its automatization with the use of a specifically developed application framework.

In accordance with the Domain-Specific Modeling (DSM) approach, a number of unique domain-specific languages are defined: a language for the specification of transformation operations (ETL-O), a language for control flow specification (ETL-P), a language for the specification of various logical and arithmetic expressions (ETL-E) and a language for the specification of transformation operation templates (ETL-T). Each of these languages define concepts that are relevant for specific aspect of ETL process. ETL process modeling is basically the modeling of specific aspects of the ETL process using the appropriate domain-specific language, which in effect significantly reduces the complexity of modeling.

The implementation of specified domain-specific languages is achieved through the introduction of a specific application framework as technological support to the given formal specification. By introducing the application framework the semantic level, which is technologically supported and can be automatized, is significantly elevated. The implementation is actually based on the automated transformation of the model, established in accordance with appropriate domain-specific language, in executable code. By elevating the semantic level and supporting it technologically, fewer steps will be needed to develop an ETL process and the development itself becomes more automated and therefore more productive.

Technological support to development and ETL process model execution is specified by a proposed original ETL Platform (envisaged as an upgrading of the

Microsoft .NET Platfrom). More precisely, the ETL Platform describes the technological support to modeling (e.g. forming of models based on appropriate domain-specific language) and domain-specific languages' implementation. Technological support to modeling is accomplished by the developed software tool (mainly syntax editors). Automated transformations of models (formed by using syntax editors and based on appropriate domain-specific languages) into executable code are supported by specifically developed generators. The application framework is supported by services that enable execution and managing of executable code generated from formed models.

In the thesis, the general physical software architecture of the ETL Platform, as the base for its implementation is presented. General physical software architecture gives a specification of the software solution which defines the basic components of the ETL Platform. The implementation of this general software architecture is realized in Microsoft .NET platform, using software components to realise components from the given general physical architecture.

In order to validate the solution proposed in this thesis, several experimental tests are undertaken, with plans for additional tests to be carried out in practice in different domains.

Keywords:
Extract-Transform-Load (ETL) process development, Domain-Specific Language (DSL), Model Driven Development (MDD)

Scientific Area:
Technical Sciences

Specific Scientific Area:
Information Systems

UDK Number:
004.4(043.3)

SADRŽAJ

1. UVOD

U ovom delu rada opisani su problem, predmet i ciljevi istraživanja doktorske disertacije. Shodno tome, definisane su opšta i posebne hipoteze koje će se istraživanjem potvrditi ili oboriti i na kraju je data struktura rada.

1.1. Predmet i cilj istraživanja

Problem koji će biti analiziran u okviru ovog rada je razvoj sistema skladišta podataka. Uzimajući u obzir prirodu problema koji se rešava ovim sistemima može se reći da problem istraživanja obuhvata oblasti kao što su: baze podataka, skladišta podataka, poslovna inteligencija i podrška odlučivanju.

Savremeno poslovanje je gotovo nemoguće zamisliti bez odgovarajuće podrške informacionih tehnologija. Napredak tehnologije, oštra konkurencija kao i česte promene u poslovnom okruženju uticali su da današnja poslovanja budu u velikoj meri zavisna od odgovarajuće podrške informacionih sistema. Ova podrška se pre svega odnosi na efikasno obavljanje rutinskih dnevnih zadataka, odnosno prikupljanje, skladištenje i obradu svih podataka koji su neophodni za nesmetano izvršavanje definisanih poslovnih procesa.

Efikasnost je potreban, ali ne i dovoljan uslov uspešnog poslovanja. Pored efikasnosti, neophodno je postići i efektivnost izvršavanja poslovnih procesa. Da bi se omogućilo postizanje efektivnosti poslovanja, potrebno je obezbediti odgovarajuću podršku višem menadžmentu, odnosno obezbediti efikasnost u procesu donošenja poslovnih odluka. Permanentna analiza poslovanja i evaluacija postavljenih ciljeva, kao i iznalaženje novih i boljih načina za ostvarivanje konkurentske prednosti su uslov uspešnog poslovanja.

Povećanje složenosti poslovanja i sve izraženija konkurencija koja vlada u njegovom okruženju, povećava pritisak na donosioce odluka *(Ponniah, 2011)*, bilo

da se donose operativne, taktičke ili strateške odluke. Potrebne su pravovremene i ispravne odluke kako bi se poslovanje što brže prilagodilo promenama i kako bi se pronašli novi načini sticanja konkurentske prednosti. Ono što karakteriše proces donošenja odluka u savremenim poslovanjima je postojanje velike količine podataka i kratki vremenski rokovi u kojima je odluke potrebno doneti *(Suknović & Delibašić, 2010)*. Imati prave informacije, u pravo vreme, na pravom mestu uz razumne troškove je preduslov donošenja ispravnih poslovnih odluka *(Jarke, Lenzerini, Vassiliou, & Vassiliadis, 2003)*.

Složenost problema koji se rešavaju taktičkim i strateškim odlukama, kao i značaj koji te odluke imaju na poslovanje sa jedne strane, i nemogućnost postojećih tehnologija i sistema da transformišu poslovne podatke u strateške informacije u obliku koji je pogodan za analize sa druge strane, dovela je do pojave posebne vrste sistema koji su danas poznati pod nazivom poslovna inteligencija (eng. *business intelligence*). Pojmom poslovna inteligencija označavaju se savremeni sistemi za podršku odlučivanju kao posebne grane u informacionim tehnologijama kojima je osnovni cilj unapređenje procesa donošenja odluka *(Suknović & Delibašić, 2010)*.

U okviru poslovne inteligencije razvili su se sistemi skladišta podataka (eng. *data warehousing systems*), kao posebna vrsta informacionih sistema, koji predstavljaju sponu između transakcionih i analitičkih sistema. Potreba za ovim sistemima proistekla je iz činjenice da transakciona i analitička obrada podataka ne mogu efikasno da koegzistiraju u istom okruženju *(Jarke, Lenzerini, Vassiliou, & Vassiliadis, 2003)*, kao i činjenice da postojeći transakcioni sistemi nisu u mogućnosti da obezbede strateške informacije *(Ponniah, 2011)*.

Sistemi skladišta podataka treba da omoguće prikupljanje poslovnih podataka, zatim njihovu transformaciju u odgovarajuće strateške informacije i smeštanje tako dobijenih strateških informacija u obliku koji je pogodan za sprovođenje analiza *(Poole, 2003)*. Za prikupljanje i transformaciju poslovnih podataka sprovodi se veliki broj različitih tipova aktivnosti koje se organizuju u odgovarajuće procese (eng. *extract-transform-load process*, u daljem tekstu ETL proces), dok se za

skladištenje strateških informacija koriste specijalizovane baze podataka koje se nazivaju skladišta podataka (eng. *data warehouse*).

U cilju dobijanja visoko kvalitetnih strateških informacija u obliku koji je pogodan za donošenje odluka, potrebno je sprovesti veći broj različitih tipova funkcija na podacima u odgovarajućem redosledu. Tipovi funkcija i redosled u kome se one izvršavaju definisan je procesom koji je u literaturi poznat pod nazivom ETL proces. ETL proces je opšte prihvaćen naziv za procese u okruženju sistema skladišta podataka, kojima se vrši ekstrahovanje i korekcija podataka, zatim njihova transformacija i konsolidacija i na kraju učitavanje u skladište podataka *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*.

Predmet ovog rada je razvoj ETL procesa kao najsloženijeg i najznačajnijeg dela u čitavom procesu razvoja sistema skladišta podataka. Osnovni cilj koji se želi postići ovim procesom je pravovremeno obezbeđivanje visoko kvalitetnih strateških informacija u obliku koji je pogodan za donošenje poslovnih odluka. Uzimajući ovo u obzir, način na koji se projektuju i implementiraju ovi procesi u velikoj meri utiče na kvalitet dobijenih strateških informacija, a samim tim i na upotrebljivost, odnosno uspeh celokupnog sistema skladišta podataka.

Razvoj ETL procesa je veoma složen i vremenski zahtevan proces koji zahteva značajna finansijska sredstva. U literaturi se navodi da je čak 80% vremena i uloženog napora u razvoju sistema skladišta podataka upravo posvećen razvoju ETL procesa *(Golfarelli, 2009; Kimball, Ross, Thornthwaite, Mundy, & Becker, 2010)*. Potrebno je veliko znanje, veštine i sposobnosti svih učesnika kako bi se rešili brojni problemi i rizici koji vladaju na projektu. Pored toga, na projektima se sprovodi veliki broj različitih tipova aktivnosti, koje zajedno sa njihovim rezultatima treba organizovati na odgovarajući način. Sve ovo ukazuje na neophodnost korišćenja odgovarajućeg metodološkog pristupa, odnosno formalizovanog načina na koji se ovi sistemi razvijaju.

U literaturi je predložen veći broj pristupa koji se bave razvojem ETL procesa. Međutim, i pored značajnog napretka, predloženi pristupi nisu doveli do očekivanih rezultata u rešavanju brojnih teškoća i problema kao što su visoki troškovi razvoja i održavanja, slaba produktivnost, neodgovarajuće ispunjene korisničkih zahteva itd. Navedeni problemi uzrokovani su pre svega velikom složenošću savremenih poslovnih sistema, čestim promenama u organizacionom i tehnološkom okruženju i sve većom potrebom za integracijom sa okruženjem.

Najsavremeniji pristup koji se predlaže u softverskoj industriji za rešavanje navedenih problema je razvoj vođen modelima (eng. *model driven development*, u daljem tekstu MDD). Osnovni cilj koji se želi postići je da se poveća produktivnost i skrati vreme potrebno za razvoj softverskog rešenja. Za postizanje ovog cilja, MDD propisuje izdizanje nivoa apstrakcije i korišćenje transformacija kojima se proces razvoja automatizuje. U osnovi ovog pristupa je korišćenje dobro poznate i dokazane tehnike apstrakcije, koja predstavlja najmoćniji intelektualni alat u savladavanju složenosti. Naime, jedini metodološki način savladavanja složenosti je upotreba apstrakcija, tj. svesno zanemarivanje detalja na nekom nivou apstrakcije kako bi se broj koncepata sa kojima se suočavamo redukovao na nivo savladive složenosti. U skladu sa tim, ovim pristupom se propisuje korišćenje modela kao primarnih softverskih artefakata kojima se vodi razvoj softvera.

U skladu sa navedenim, ovaj rad ima kao osnovnu tezu da je za rešavanje problema u razvoju softverskih rešenja kojima se rešava problem ETL procesa izuzetno važna upotreba apstrakcija kao jedinog metodološkog načina za savladavanje složenosti. Pored savladavanja složenosti, apstrakcije se koriste i za uočavanje razlika i sličnosti kako bi se delovi sistema sa istim funkcionalnostima mogli izdvojiti, specificirati i implementirati nezavisno od specifikacije i višestruko koristiti. Drugim rečima, pošto je potreban veći stepen automatizacije, potrebno je formalizovati znanja i stečena iskustva u obliku koji dozvoljava njihovo ponovno korišćenje. Ponovnim korišćenjem se dodatno utiče na produktivnost i efikasnost, čime se ujedno i smanjuju troškovi uvođenja ovakvih sistema. Jedan od predmeta istraživanja u ovom radu je i implementacija apstraktnih specifikacija koja treba značajno da podigne semantički nivo koji je tehnološki podržan i koji se može

automatizovati. Na taj način razvoj postaje više automatizovan i potreban je manji broj koraka u postupku realizacije apstraktnih specifikacija.

Kao što je i navedeno, problem koji se rešava u ovom radu je razvoj ETL procesa i rešenje koje se predlaže ima za cilj da učini takav razvoj u velikoj meri automatizovanim. Predloženo rešenje razvoja zasniva se na *Domain Specific Modeling* (DSM) pristupu i u skladu sa tim korišćene su formalne specifikacije ETL procesa i njihova automatska transformacija u specifičan aplikacioni okvir.

Za specifikaciju ETL procesa, predlaže se korišćenje odgovarajućih domensko-specifičnih jezika (eng. *domain specific language*, u daljem tekstu DSL) kojima su definisani koncepti relevantni za dati domen i koji nisu ekstenzije postojećih opštih jezika modelovanja, kao što su *Unified Modeling Language* (UML) i *Business Process Model and Notation* (BPMN), već su razvijeni kao novi jezici. Svaki od predloženih DSL jezika omogućava modelovanje određenog aspekta ETL procesa. Na ovaj način omogućeno je uvođenje veoma jednostavnih specifičnih jezika modelovanja koji se lako uče i primenjuju i koji sadrže samo one koncepte koji su relevantni za dati aspekt ETL procesa. Pored toga, dodatno se utiče na smanjenje složenosti modelovanja, jer se različiti aspekti ETL procesa nezavisno modeluju. Na ovom mestu je potrebno naglasiti da se u ovom radu ne razmatraju svi aspekti ETL procesa već samo oni najvažniji kao što je redosled izvršavanja aktivnosti ETL procesa i semantika aktivnosti. Specifikacije ostalih aspekata ETL procesa mogu se podržati uvođenjem novih domensko-specifičnih jezika.

Implementacija ETL procesa ostvarena je uvođenjem specifičnog aplikacionog okvira da bi se predložena specifikacija tehnološki podržala i omogućila automatizacija razvoja u skladu sa usvojenim DSM pristupom. Uvođenjem aplikacionog okvira značajno se podiže semantički nivo koji je implementaciono podržan, definisanjem specifičnih implementacionih koncepata bliskih konceptima realnog domena specificiranim u predloženim domensko-specifičnim jezicima. Pre svega, predložena je odgovarajuća softverska arhitektura za rešenja iz domena ETL procesa kojom su identifikovane osnovne komponente i veze između njih, a zatim je data njihova implementacija u izabranom tehnološkom okruženju u vidu

generičkih softverskih komponenti. Implementacija konkretnog ETL procesa omogućena je kroz proces kastomizacije ovih generičkih komponenti. Ovaj proces kastomizacije je automatizovan uvođenjem specifičnih generatora kojima se date specifikacije ETL procesa automatski prevode u odgovarajuće softverske komponente koje predstavljaju specijalizacije prethodno realizovanih generičkih komponenti. Bazirajući se na automatskim transformacijama između specifikacija ETL procesa i aplikacionog okvira povećava se i produktivnost i efikasnost razvoja ETL procesa.

Na osnovu svega navedenog, **cilj** ovog rada je konceptualizacija i automatizacija razvoja ETL procesa u skladu sa DSM pristupom.

1.2. Polazne hipoteze

Na osnovu analize dostupne literature i postavljenog predmeta i cilja istraživanja može se postaviti:

Opšta hipoteza:

- Moguća je automatizacija razvoja ETL procesa korišćenjem MDD pristupa.

Posebne hipoteze:

- Moguće je definisati specifičan jezik (metamodel) za specifikaciju toka ETL procesa.
- Moguće je definisati specifičan jezik (metamodel) za specifikaciju skupa operacija transformacija ETL procesa.
- Moguće je definisati automatske transformacije modela, zasnovanih na definisanim specifičnim jezicima, na osnovu kojih se može automatski generisati izvorni kod za izabranu ETL platformu.
- Predloženim rešenjem moguće je unaprediti postojeću industrijsku praksu.

1.3. Struktura rada

Doktorska disertacija je organizovana na sledeći način:

U prvom poglavlju su opisani su problem, predmet i ciljevi istraživanja doktorske disertacije. Definisane su opšta i posebne hipoteze koje će se istraživanjem potvrditi ili oboriti i na kraju je data struktura rada.

Drugim poglavljem je dat teorijski okvir istraživanja sa ciljem da se opišu i bliže odrede pojmovi iz oblasti kojom se bavi istraživanje. Razmatrani su pojmovi poslovna inteligencija i sistemi skladišta podataka, zatim je dat opis arhitekture i procesa razvoja sistema skladišta podataka i na kraju je razmatran pristup razvoju softvera vođen modelima.

U trećem poglavlju su razmatrane ključne karakteristike ETL procesa i predstavljena je taksonomija operacija transformacija podataka kao osnovnih elemenata ETL procesa.

Četvrtim poglavljem su razmatrana postojeća rešenja problema formalizacije i automatizacije razvoja ETL procesa. Dat je pregled relevantnih istraživanja sa ciljem da se prikažu dosadašnji rezultati i praksa u oblasti razvoja ETL procesa, a zatim je predstavljena analiza rezultata koji su ovim istraživanjima postignuta. Analizom su razmatrana rešenja problema konceptualizacije i automatizacije ETL procesa i shodno tome navedeni su nedostaci postojećih pristupa i opisani konkretni predlozi za poboljšanja.

U petom poglavlju predstavljeno je originalno rešenje, zasnovano na DSM pristupu, za problem konceptualizacije i automatizacije razvoja ETL procesa. Razmatrani su osnovni koncepti predloženog rešenja, a zatim je predstavljena specifična ETL platforma kojom je obezbeđena tehnološka podrška razvoju i izvršavanju modela ETL procesa.

U šestom poglavlju se razmatra problem formalne specifikacije ETL procesa. Opisan je predloženi konceptualni okvir ETL procesa, kao i osnovni koncepti neophodni za definisanje predloženih jezika modelovanja. U nastavku poglavlja

definisani su predloženi domensko-specifični jezici: jezik za specifikaciju operacija transformacija podataka (ETL-O), jezik za specifikaciju toka izvršavanja ETL procesa (ETL-P), jezik za specifikaciju izraza (ETL-E) i jezik za specifikaciju šablona operacija transformacija (ETL-T). Za svaki predloženi jezik, predstavljeni su relevantni koncepti za specifični aspekt ETL procesa i ograničenja u vezi njihovog korišćenja, zatim predlog odgovarajuće konkretne sintakse i na kraju kratak primer kojim se ilustruje korišćenje jezika.

Sedmo poglavlje bavi se problemom modelovanja ETL procesa. Predložen je originalni pristup modelovanju ETL procesa koji se zasniva na korišćenju definisanih domensko-specifičnih jezika. Posebna pažnja je posvećena savladavanju složenosti modelovanja ETL procesa i načinu na koji je to postignuto. U nastavku poglavlja je dat realan primer kojim je izvršena validacija predloženog pristupa modelovanju. Primerom je ilustrovano formiranje modela u skladu sa predloženim domensko-specifičnim jezicima.

U osmom poglavlju je data specifikacija softverskog rešenja. Predstavljena je opšta fizička softverska arhitektura ETL platforme kojom su specificirane neophodne softverske komponente, njihove uloge i odgovornosti, kao i odgovarajući interfejsi. Identifikovane softverske komponente su organizovane u dva sloja, formirajući na taj način razvojno i izvršno okruženje. Implementacija komponenti razvojnog i izvršnog okruženja je razmatrana u poglavljima koja slede.

Devetim poglavljem je predstavljeno razvojno okruženje predložene ETL platforme. Predstavljeni su razvijeni alati, kao i tehnologije koje su korišćene za njihovu implementaciju. U prvom delu poglavlja, razmatran je osnovni skup alata (*ETLProcessTool*, *ETLDataProcessTool*, *ETLExpressionTool* i *ETLTemplateTool*) kojima je obezbeđena tehnološka podrška formiranju modela ETL procesa u skladu sa definisanim domensko-specifičnim jezicima, dok je u drugom delu predstavljen alat (*ETLDocumentationTool*) za automatsko generisanje dokumentacije ETL procesa.

Deseto poglavlje se bavi implementacijom izvršnog okruženja predložene ETL platforme. Predstavljeni su osnovni servisi (*ETLProcessService*,

ETLDataProcessService i *ETLMetadataService*) predloženog izvršnog okruženja kojima je omogućeno izvršavanje i upravljanje formiranim modelima ETL procesa, a zatim je razmatrana njihova implementacija u izabranom tehnološkom okruženju (*Microsoft .NET* platforma). Dodatno, razmatrane su i specifične tehnologije *Microsoft .NET platforme* kojima su omogućene automatske transformacije modela i generisanje izvršnog kôda. Izvršena je analiza i za svaku tehnologiju su identifikovani mogući scenariji korišćenja u kontekstu implementacije predložene ETL platforme.

U jedanaestom poglavlju dat je zaključak, zatim stručni i naučni doprinosi, kao i pravci daljih istraživanja.

Na kraju rada je dat spisak korišćene literature.

2. TEORIJSKI OKVIR ISTRAŽIVANJA

U ovom poglavlju su obrađeni osnovni teorijski koncepti sa ciljem da se bliže odrede pojmovi i steknu znanja iz oblasti kojom se bavi istraživanje. Razmatrani su pojmovi poslovna inteligencija i sistemi skladišta podataka, zatim je dat opis arhitekture i procesa razvoja sistema skladišta podataka i na kraju je razmatran pristup razvoju softvera vođen modelima.

2.1. Poslovna inteligencija

Savremeno poslovanje je gotovo nemoguće zamisliti bez odgovarajuće podrške informacionih tehnologija. Napredak tehnologije, oštra konkurencija kao i česte promene u poslovnom okruženju uticali su da današnja poslovanja budu u velikoj meri zavisna od odgovarajuće podrške softverskih sistema. Ova podrška se pre svega odnosi na efikasno obavljanje rutinskih dnevnih zadataka, odnosno prikupljanje, skladištenje i obradu svih podataka koji su neophodni za nesmetano izvršavanje definisanih poslovnih procesa.

Efikasnost je potreban, ali ne i dovoljan uslov uspešnog poslovanja. Da bi poslovanje bilo uspešno, pored efikasnosti, neophodno je postići i efektivnost izvršavanja poslovnih procesa. Da bi se omogućilo postizanje efektivnosti poslovanja, potrebno je obezbediti odgovarajuću podršku višem menadžmentu, odnosno obezbediti efikasnost u procesu donošenja poslovnih odluka. Definisanje odgovarajuće strategije i ciljeva poslovanja, zatim njihovo praćenje i evaluacija su danas u prvom planu. Permanentna analiza poslovanja i evaluacija postavljenih ciljeva, kao i iznalaženje novih i boljih načina za ostvarivanje konkurentske prednosti su uslov uspešnog poslovanja.

Povećanje složenosti poslovanja i sve izraženija konkurencija koja vlada u njegovom okruženju, povećava pritisak na donosioce odluka *(Ponniah, 2011)*, bilo da se donose operativne, taktičke ili strateške odluke *(Suknović & Delibašić, 2010)*. Potrebne su pravovremene i ispravne odluke kako bi se poslovanje što brže prilagodilo promenama i kako bi se pronašli novi načini sticanja konkurentske

prednosti *(Ponniah, 2011)*. Ono što karakteriše proces donošenja odluka u savremenim poslovanjima je postojanje velike količine podataka i kratki vremenski rokovi u kojima je odluke potrebno doneti *(Suknović & Delibašić, 2010)*. Imati prave informacije, u pravo vreme, na pravom mestu uz razumne troškove je preduslov donošenja ispravnih poslovnih odluka *(Jarke, Lenzerini, Vassiliou, & Vassiliadis, 2003)*.

Mnogi su prepoznali značaj informacija u savremenom poslovanju. *Kimball* i *Ross* navode da upravo informacije čine jednu od osnovnih vrednosti organizacije *(Kimball & Ross, 2002)*. Način na koji dolaze do ovih informacija kao i stepen njihovog korišćenja u procesu sticanja znanja u velikoj meri utiče na donošenje ispravnih poslovnih odluka, a samim tim i na uspeh poslovanja. Ovo su upravo i dva osnovna cilja koji se žele postići *(Ponniah, 2011)*: (1) efikasna transformacija podataka u informacije i (2) sticanje znanja na osnovu dobijenih informacija kako bi mogle da se donesu validne poslovne odluke.

Slika 1. Tok donošenja odluka

U organizacijama se donose različite poslovne odluke i one se mogu klasifikovati na: (1) *operativne*, (2) *taktičke* i (3) *strateške odluke (Suknović & Delibašić, 2010)*. Nivo potrebnog znanja za njihovo donošenje se razlikuje, što je posledica prirode problema koji se njima rešava. *Operativne odluke* se smatraju najjednostavnijim u ovoj hijerarhiji odluka. Za njihovo donošenje ne postoje velike nepoznanice *(Suknović & Delibašić, 2010)* i one se odnose na obavljanje tekućeg poslovanja. Za razliku od njih, *taktičke* i *strateške odluke* zahtevaju viši stepen znanja i smatraju se složenijim.

Složenost odluke proizilazi iz *strukturiranosti problema*[1] koji se odlukom rešava, odnosno nivoa potrebnog znanja za njegovo rešavanje. Ukoliko donosilac odluke

[1] „Strukturiranost problema se može izjednačiti sa njegovom definisanošću, pri čemu se pod potpuno definisanim problemom podrazumeva: (1) da je problem jasan, (2) da su precizno

vlada znanjem u određenoj oblasti onda su za njega problemi iz te oblasti strukturirani *(Suknović & Delibašić, 2010)*. Iz ovoga proizilazi i jedan od osnovnih ciljeva odlučivanja, a to je da se omogući „*prevođenje problema odlučivanja iz nestrukturiranosti u strukturiranost*“ *(Suknović & Delibašić, 2010)*. Uzimajući u obzir definiciju strukturiranosti problema, ovo je jedino moguće ukoliko se donosiocima odluka pravovremeno obezbedi relevantan skup informacija neophodan za razumevanje posmatranog problema.

Ponniah razlikuje dva tipa informacija *(Ponniah, 2011)*: *operativne informacije* i *strateške informacije*. *Operativne informacije* se tiču tekućeg poslovanja i koriste se prilikom donošenja *operativnih odluka* koje se, u pogledu strukturiranosti problema koje rešavaju, smatraju potpuno strukturiranim *(Suknović & Delibašić, 2010)*. *Strateške informacije* se tiču definisanja i sprovođenja strategija i ciljeva poslovanja i koriste se tokom donošenja *taktičkih* i *strateških odluka*. U odnosu na strukturiranost problema koji rešavaju, ove odluke se smatraju polustrukturiranim i nestrukturiranim *(Suknović & Delibašić, 2010)*.

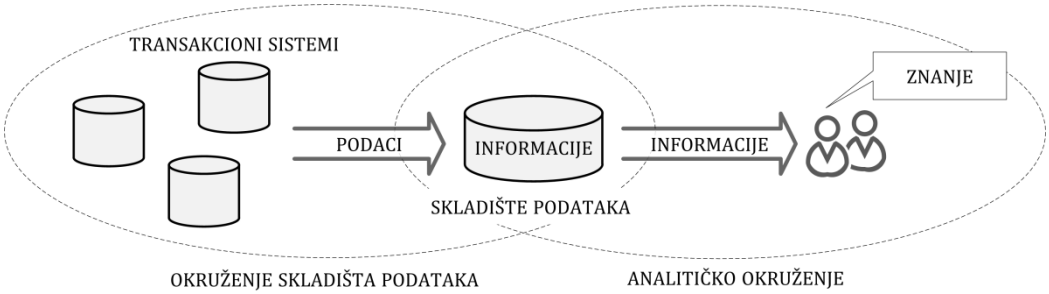
Složenost problema koji se rešavaju taktičkim i strateškim odlukama, kao i značaj koji te odluke imaju na poslovanje sa jedne strane, i nemogućnost postojećih tehnologija i sistema da transformišu poslovne podatke u strateške informacije u obliku koji je pogodan za analize sa druge strane, dovela je do pojave posebne vrste sistema koji su danas poznati pod nazivom *poslovna inteligencija* (eng. *business intelligence*). Pojmom *poslovna inteligencija* označavaju se savremeni sistemi za podršku odlučivanju kao posebne grane u informacionim tehnologijama. Njihov osnovni cilj je unapređenje procesa donošenja odluka i mogu se definisati kao *(Suknović & Delibašić, 2010)*:

„*Skup informacionih tehnologija, organizacionih pravila kao i znanja i veština zaposlenih u organizaciji udruženih u generisanju, zapisivanju, integraciji i analizi podataka sve sa ciljem da se dođe do potrebnog znanja za donošenje odluke.*“

definisani ulazni podaci i (3) da su poznati načini na koji se vrši analiza i dolazi do rešenja.“ *(Suknović & Delibašić, 2010)*

Cilj *sistema poslovne inteligencije* je da omogući donosiocima odluka da, kroz sprovođenje efikasnih i efektivnih analiza poslovnih podataka, steknu sveobuhvatan uvid u poslovanje i identifikuju ključne faktore koji na njega utiču. Na taj način stiče se neophodno znanje, koje je preduslov za donošenje pravovremenih i ispravnih odluka koje se tiču stvaranja dodatne vrednosti za organizaciju *(Rizzi, 2009)*.

Kao što je i prikazano (slika 2), *poslovnu inteligenciju* čine dva komplementarna okruženja *(Ponniah, 2011)*: (1) okruženje skladišta podataka i (2) analitičko okruženje. Okruženje skladištenja podataka čine alati i tehnologije kojima je omogućena transformacija i integracija poslovnih podataka u strateške informacije i njihovo skladištenje u obliku koji je pogodan za analizu. Analitičko okruženje čine alati i tehnologije kojima se omogućava sticanje znanja kroz sprovođenje različitih analiza nad strateškim informacijama.



Slika 2. Okruženje poslovne inteligencije *(Ponniah, 2011)*

U kontekstu poslovne inteligencije mogu se identifikovati tri vrste sistema: (1) *transakcioni sistemi* (eng. *online transaction processing OLTP systems*), (2) *sistemi skladišta podataka* (eng. *data warehousing systems*) i (3) *analitički sistemi* (eng. *online analytical processing OLAP systems*). *Transakcioni sistemi* omogućavaju efikasno izvršavanje poslovnih procesa i predstavljaju izvor poslovnih podataka, dok *analitički sistemi* omogućavaju analiziranje strateških informacija i sticanje neophodnog znanja kako bi mogle da se donose odluke. *Sistemi skladišta podataka* predstavljaju sponu između transakcionih i analitičkih sistema *(Suknović & Delibašić, 2010)*. Oni omogućavaju transformaciju i integraciju poslovnih podataka u strateške informacije i njihovo skladištenje u obliku koji je pogodan za analizu *(Ponniah, 2011)*.

Transakcioni i analitički sistemi u osnovi imaju različitu namenu. Svrha transakcionih sistema je da reflektuju ponašanje poslovnog sistema *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*, odnosno da omoguće automatizaciju i efikasno izvršavanje definisanih poslovnih procesa. Sa druge strane, svrha analitičkih sistema je da omoguće i olakšaju proces donošenja poslovnih odluka koje imaju za cilj stvaranje dodatne vrednosti za organizaciju *(Rizzi, 2009)*, odnosno postizanje efektivnosti u izvršavanju poslovnih procesa. Iako komplementarni u pogledu sveobuhvatne podrške poslovanju, ovi sistemi imaju suprotstavljene zahteve.

Potreba za *sistemima skladišta podataka*, proistekla je iz činjenice da transakciona i analitička obrada podataka na mogu efikasno da koegzistiraju u istom okruženju *(Jarke, Lenzerini, Vassiliou, & Vassiliadis, 2003)*, kao i činjenice da postojeći transakcioni sistemi nisu u mogućnosti da obezbede strateške informacije *(Ponniah, 2011)*.

Uzimajući u obzir da se u organizacijama paralelno obavlja veliki broj poslovnih procesa, kao i da je broj korisnika ovih sistema veliki, akcenat transakcionih sistema ja na efikasnom i preciznom izvršavanju transakcija[2] *(Reeves, 2009)*. Da bi se ovo omogućilo, u bazama podataka koje koriste transakcioni sistemi, podaci se smeštaju u normalizovanom obliku (najčešće se koristi treća normalna forma 3NF) čime se smanjuje redudansa podataka i odstranjuju tzv. „anomalije“ u ažuriranju *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*. Ovakav pristup svakako povećava performanse sistema kada je u pitanju ažuriranje podataka, ali kada je potrebno da se ti podaci pročitaju, odnosno interpretiraju kako bi se dobile korisne informacije, performanse su daleko lošije. Posebno u situacijama kada je potrebno interpretirati veliku količinu podataka, kao što je slučaj prilikom analize poslovanja, performanse sistema mogu biti značajno narušene, što svakako nije poželjno u transakcionim sistemima. Velika normalizovanost podataka se jedne strane i zahtev za jednostavnim izveštavanjem sa druge strane su suprotstavljeni

[2] „Pod transakcijom se podrazumeva niz operacija nad bazom podataka koja odgovara jednoj logičkoj celini posla u realnom sistemu.“ *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*

zahtevi *(Suknović & Delibašić, 2010)*. Pored toga, za dobijanje podataka, odnosno informacija iz baza podataka koristi se upitni jezik (*structured query language* SQL) koji zahteva informatička znanja, a koja donosioci odluka obično nemaju.

Automatizacija poslovnih procesa organizacije je evolutivni proces. Poslovni procesi se automatizuju fazno, obično prema prioritetnim funkcionalnim celinama organizacije. Celokupan proces je vremenski zahtevan i može rezultovati većim brojem transakcionih podsistema, potencijalno na različitim platformama, kojima se podržavaju različiti poslovni procesi organizacije. Posledica ovakvih situacija je da poslovni podaci nisu integrisani na jednom mestu, već smešteni u distribuiranim i heterogenim izvorima podataka, memorisani u različitim modelima podataka i opisani pomoću različitih šema, pa je sprovođenje analiza znatno otežano. Podaci se moraju prikupiti sa različitih lokacija, zatim konsolidovati i transformisati u oblik koji je pogodan za analizu.

Važna karakteristika transakcionih sistema je i da oni odražavaju trenutno stanje poslovnog sistema *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*. Održavanje konzistentnog stanja sistema je svakako preduslov za uspešno obavljanje poslovnih procesa, ali isto tako, odsustvo istorijskih podataka predstavlja veliko ograničenje u pogledu mogućnosti sprovođenja analize poslovanja. Naime, sagledavanje poslovanja u određenom vremenskom periodu je neophodno kako bi se identifikovali trendovi i pratile performanse poslovanja *(Reeves, 2009)*. Pored toga, u procesu donošenja strateških odluka veoma bitnu ulogu imaju i podaci iz okruženja, odnosno eksterni podaci. Ovi podaci svakako nisu u opsegu transakcionih sistema, ali su veoma bitni u procesu sticanja znanja koje je neophodno za donošenje strateških odluka.

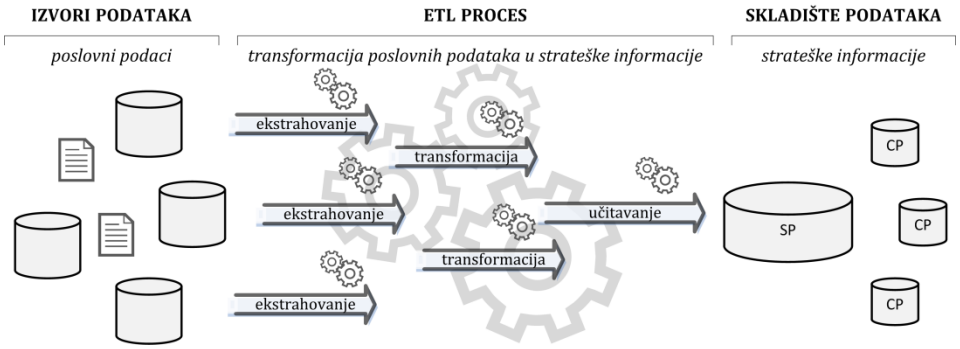
Iz svega što je do sada navedeno, jasno je da transakcioni sistemi ne mogu da odgovore na zahteve strateškog odlučivanja. Osnovni razlog leži u činjenici da su oni optimizovani da podrže efikasno izvršavanje poslovanja. Proširivanje njihovog opsega, odnosno dodeljivanje dodatnih odgovornosti transakcionim sistemima bi svakako ugrazilo njihovu osnovnu namenu. Upravo ovo je razlog uvođenja nove vrste sistema. Ovi sistemi imaju jasno definisanu odgovornost, a to je da obezbede

strateške informacije u obliku koji je pogodan za analizu, otkrivanje trendova i praćenje performansi poslovanja *(Ponniah, 2011)*.

2.2. Sistemi skladišta podataka

Sistemi skladišta podataka su se razvili u okviru poslovne inteligencije *(Ponniah, 2011)* sa jasno definisanim ciljem da se razdvoji transakciona i analitička obrada podataka. Potreba za ovim sistemima nastala je iz činjenice da transakciona i analitička obrada podataka imaju suprotstavljane zahteve i da ne mogu efikasno egzistirati unutar istog okruženja *(Jarke, Lenzerini, Vassiliou, & Vassiliadis, 2003)*. Uvođenjem ove nove vrste sistema u okruženje poslovne inteligencije uspostavljana je spona između transakcionih i analitičkih sistema *(Suknović & Delibašić, 2010)* i obezbeđeno njihovo efikasno izvršavanje.

Ovi sistemi treba da omoguće efikasno prikupljanje poslovnih podataka iz heterogenih izvora, zatim njihovu transformaciju u odgovarajuće strateške informacije i smeštanje tako dobijenih strateških informacija u obliku koji je pogodan za sprovođenje analiza *(Poole, 2003)*. Za prikupljanje i transformaciju poslovnih podataka koriste se *ETL procesi* (eng. *extract-transform-load ETL*), dok se za smeštanje strateških informacija koriste specijalizovane baze podataka koje se nazivaju *skladišta podataka* (eng. *data warehouse*).



Slika 3. Okruženje sistema skladišta podataka

Kao što slika prikazuje (slika 3), ulaz u ove sisteme predstavljaju poslovni podaci koji mogu dolaziti iz distribuiranih i heterogenih izvora. Ovi izvori poslovnih podataka mogu biti različite baze podataka i datoteke koje se koriste u poslovnim procesima, ali isto tako mogu dolaziti iz okruženja organizacije u vidu eksternih

podataka. Činjenica da potiču iz heterogenih izvora podataka, navodi na potrebu njihove transformacije i integracije kako bi se eliminisale sve nekonzistentnosti, bilo strukturne bilo semantičke, koje u njima postoje. Na kraju se skladišta podataka pune integrisanim podacima i stavljaju na raspolaganje analitičkim sistemima, što je razlog da se skladišta podataka nazivaju analitičkim bazama podataka *(Suknović & Delibašić, 2010)*. Koncept skladišta podataka prvi je uveo *Inmon* 90tih godina prošlog veka i od tada je u literaturi predstavljen veći broj definicija ovog koncepta. Sve ove definicije sa većim ili manjim uspehom opisuju skladište podataka i slažu se u jednom, a to je da skladišta podataka predstavlja bazu podataka u kojoj se prikupljaju, integrišu i strukturiraju svi podaci neophodni za sprovođenje različitih analiza poslovanja. U nastavku su navedene definicije skladišta podataka koje su dali vodeći autori u oblasti *Bill Inmon* i *Ralph Kimball*:

„*Skladište podataka je kopija transakcionih podataka specifično strukturiranih za potrebe upita i analiza.*“ *(Kimball & Ross, 2002)*

„*Skladište podataka je predmetno orijentisani, integrisani, trajni i vremenski orijentisani skup podataka u funkciji podrške odlučivanja menadžera.*“ *(Inmon W. H., 2005)*

Iako predstavnici različitih filozofija, odnosno pristupa u razvoju skladišta podataka, definicije koje su dala ova dva autora na najbolji način odražavaju prirodu skladišta podataka. Može se reći da se dopunjuju. *Kimball* je u svojoj definiciji dao funkcionalni pogled na skladište podataka, dok je *Inmon* kroz definiciju više sugerisano na to koje karakteristike ima skladište podataka.

Svakako, za bolje razumevanje neophodno je detaljnije analizirati karakteristike skladišta podataka koje proizilaze iz ovih definicija.

U skladištu podataka podaci se organizuju prema funkcionalnim područjima *(Inmon W. H., 2005)*, odnosno na način da su svi podaci koji se tiču određenog poslovnog područja povezani. Može se reći da ova predmetna organizacija podataka objedinjuje podatke iz različitih aplikacija *(Ponniah, 2011)*. Na ovaj način

omogućava se sveobuhvatan pogled na podatke koji su u vezi sa određenim poslovnim područjem ili događajem.

Druga bitna karakteristika skladišta podataka, a prema *Inmon*-u i najvažnija, je da su podaci u skladištu podataka integrisani *(Inmon W. H., 2005; Inmon, Strauss, & Neushloss, 2010)*. Naime, podaci u skladištu podataka potiču iz različitih izvora. To mogu biti različite baze podataka i datoteke koje se koriste unutar organizacije kao podrška izvršavanju poslovnih procesa, ali isto tako i različiti izvori iz okruženja organizacije. Ovi heterogeni izvori podataka su razvijani nezavisno, moguće korišćenjem različitih tehnologija i alata *(Ponniah, 2011)*, što ukazuje na potrebu transformacije i konsolidacije ovih podataka *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)* kako bi se otklonile sve potencijalne sintaksne i semantičke nekonzistentnosti među njima.

Za razliku od transakcionih sistema u kojima se podaci stalno menjaju kako bi odražavali tekuće stanje poslovanja *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*, kod skladišta podataka podaci imaju trajni karakter *(Inmon W. H., 2005; Inmon, Strauss, & Neushloss, 2010)*. Skladišta podataka odražavaju stanje poslovanja u vremenu. Podaci se periodično učitavaju i koriste, ali se ne menjaju. Skladište podataka se može posmatrati kao serija stanja poslovanja *(Inmon W. H., 2005)*, pri čemu je svako stanje uzeto u određenom trenutku vremena. Ove dve karakteristike, trajnost i vremenska orijentacija podataka, su posebno bitne u odnosu na potrebe donosilaca odluka. Na ovaj način omogućena je istorijska sekvenca aktivnosti i događaja *(Inmon W. H., 2005)* koja je veoma bitna za analize koje se tiču praćenja performansi i identifikovanja odgovarajućih trendova u poslovanju.

Još jedan veoma bitan aspekt skladišta podataka je pitanje granularnosti. Granularnost se odnosi na nivo detalja podataka smeštenim u skladištu podataka. Nizak nivo detalja je u vezi sa visokim nivoom granularnosti, i obrnuto visok nivo detalja je u vezi sa niskim nivoom granularnosti *(Inmon W. H., 2005)*.

Granularnost je kritično pitanje u projektovanju skladišta podataka, jer direktno utiče na količinu podataka koja se smešta u skladištu podataka, ali i na tipove upita

koji se mogu postaviti. Što je nivo granularnosti niži to je moguće postaviti raznovrsnije upite. Granularnost je ključna za postizanje ponovnog korišćenja, jer se podaci na nižem nivou granularnosti mogu koristiti u različite svrhe i potrebe. Ovo je posebno bitno u kontekstu budućih potreba organizacija. Stoga, zbog čestih promena u samoj organizaciji kao i u njenom okruženju, neophodno je njihovo planiranje *(Inmon W. H., 2005)*. Planiranjem promena, odnosno prihvatanjem da one predstavljaju realnost, povećava se fleksibilnost i prilagodljivost sistema.

Uvođenjem centralizovanog skladišta podataka, koje se još naziva i *primarno skladište podataka (Golfarelli & Rizzi, 2009)*, omogućavaju se brojne prednosti i razrešavaju brojni problemi u vezi sa korišćenjem podataka i informacija. Međutim, ovako centralizovan sistem vremenom može da postane isuviše složen za korišćenje, pa se pribegava decentralizaciji kroz uvođenje manjih celina kojima se lakše može upravljati. U kontekstu skladišta podataka, decentralizacija se postiže kroz uvođenje manjih, funkcionalno orijentisanih skladišta podataka koja se nazivaju *centri podataka* (eng. *data mart*). *Golfarelli* i *Rizzi* navode da je opseg ovih centara podataka skup podataka, odnosno informacija koji su relevantni za specifično poslovno područje, organizacionu jedinicu ili grupu korisnika *(Golfarelli & Rizzi, 2009)*.

2.3. Arhitektura sistema skladišta podataka

Tipična okruženja skladišta podataka se opisuju preko tokova informacija *(Poole, 2003)*. Pod njima se podrazumeva niz profinjavanja podataka, kojima se poslovni podaci prevode u strateške informacije. Proces profinjavanja podrazumeva preuzimanje podataka iz heterogenih izvora podataka, zatim višestruke transformacije tih podataka, njihovu integraciju i na kraju smeštanje u skladište podataka. Da bi se ovaj proces profinjavanja omogućio, potrebne su različite komponente koje su povezane na optimalan način. Komponente i njihova organizacija predstavljaju arhitekturu sistema skladišta podataka *(Ponniah, 2011)*.

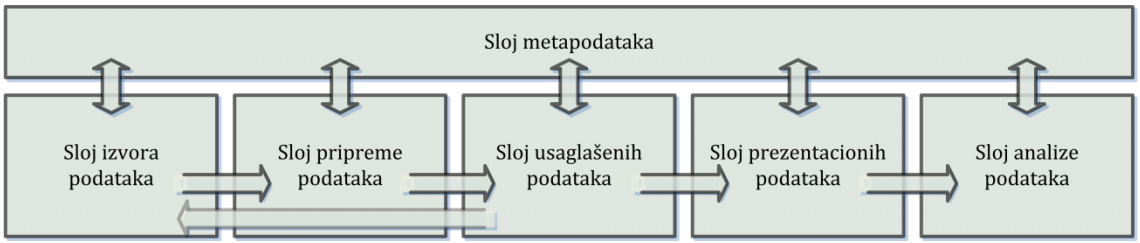
Arhitekturom je omogućeno sveobuhvatno sagledavanje i razumevanje različitih komponenti sistema skladišta podataka, odnosno organizacija procesa obrade i skladištenja podataka *(Reeves, 2009)*. Međusobno se razlikuju u odnosu na izbor

komponenti i načine na koje su te komponente organizovane sa ciljem da se dobiju optimalni rezultati *(Ponniah, 2011)*. Jasno definisana arhitektura je preduslov uspešnih projekata, jer omogućava integraciju rezultata brojnih aktivnosti koje se sprovode tokom projekta *(Reeves, 2009)*.

U kontekstu skladišta podataka treba razlikovati arhitekturu podataka i tehničku arhitekturu *(Reeves, 2009)*. Arhitektura podataka, odnosno logička arhitektura skladišta podataka se fokusira na podatke, odnosno njihovu strukturu i načine na koji će se koristiti, kao i komponente koje su neophodne za njihovu obradu i skladištenje. Tehničkom arhitekturom se definišu kolekcije potrebnih hardverskih i softverskih tehnologija i alata koje služe kao podrška razvoju, testiranju i korišćenju sistema skladišta podataka. Shodno tome, tehničkom arhitekturom se najčešće definišu tri okruženja *(Reeves, 2009)*: (1) razvojno, (2) testno i (3) produkciono okruženje.

2.3.1. Komponente sistema skladišta podataka

Arhitektura skladišta podataka se obično predstavlja preko različitih slojeva, pri čemu se podaci sa jednog sloja dobijaju iz podataka sa prethodnog. *(Jarke, Lenzerini, Vassiliou, & Vassiliadis, 2003)*. U skladu sa tim, mogu se identifikovati sledeće slojevi u opštoj logičkoj arhitekturi sistema skladišta podataka *(Kimball & Ross, 2002; Reeves, 2009; Golfarelli & Rizzi, 2009)*: (1) sloj izvora podataka, (2) sloj pripreme podataka, (3) sloj usaglašenih podataka, (4) sloj prezentacionih podataka, (5) sloj analize podataka i (6) sloj metapodataka.



Slika 4. Opšta logička arhitektura sistema skladišta podataka

Kao što je i prikazano (slika 4), prvi sloj u arhitekturi je sloj izvora podataka, odnosno sloj u kome se kreiraju podaci. Iako nije u domenu skladišta podataka *(Reeves, 2009)* predstavlja bitan sloj, jer svi podaci u skladištu podataka potiču iz

različitih heterogenih sistema koji su obuhvaćeni ovim slojem. Drugi sloj u arhitekturi je sloj pripreme podataka u kome se vrši ekstrahovanje i korekcija, zatim transformacija i konsolidacija podataka koji su dobijeni iz sloja izvora podataka. Transformisani i integrisani podaci se zatim učitavaju u sloj usaglašenih podataka. Ovaj sloj usaglašenih podataka predstavlja integrisanu kopiju transakcionih podataka *(Kimball & Ross, 2002)* i kao takav služi kao osnov za popunjavanje sloja prezentacionih podataka, ali isto tako služi kao podrška izvršavanju poslovnih procesa. Sloj prezentacionih podataka je ono što poslovni korisnici vide i koriste i što nazivaju skladište podataka. U njemu su smeštene strateške informacije koje korisnici koriste u različitim analizama. Ovim strateškim informacijama se pristupa uz pomoć odgovarajućih alata. Ovi alati i tehnologije za pristup strateškim informacijama su obuhvaćeni poslednjim slojem u arhitekturi sistema skladišta podataka.

2.3.1.1. Sloj izvora podataka

Sloj izvora podataka je prvi sloj u arhitekturi sistema skladišta podataka i predstavlja različite izvore podataka iz kojih se puni skladište podataka. Ovaj sloj nije u domenu sistema skladišta podataka, jer ista nemaju uticaj na sadržaj i format podataka koji se nalaze u ovim sistemima *(Reeves, 2009; Kimball & Ross, 2002)*. Izvori podataka mogu biti različite relacione ili objektne baze podataka, mrežne baze, kao i podaci u različitim datotekama *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*. Uopšte, izvori podataka obuhvataju heterogene sisteme u kojima se podaci kreiraju *(Reeves, 2009)*.

Izvorni podaci se mogu klasifikovati u četiri kategorije *(Ponniah, 2011)*: (1) produkcioni podaci, (2) interni podaci, (3) arhivirani podaci i (4) eksterni podaci. Pod produkcionim podacima se smatraju podaci koji potiču iz različitih transakcionih sistema koji se koriste kao podrška izvršavanju poslovnih procesa preduzeća. Internim podacima se smatraju različiti podaci koji su smešteni u individualnim dokumentima ili čak dokumentima koji se vode na nivou organizacionih jedinica, a koji nisu obuhvaćeni postojećim transakcionim sistemima. *Ponniah* skreće pažnju i na arhivirane podatke koji postoje u svakom

preduzeću i koji nastaju periodično, arhiviranjem produkcionih podataka koji nisu relevantni za tekuće izvršavanje poslovanja. Međutim, u kontekstu analiza poslovanja ovi podaci su od velikog značaja i svakako se moraju uzeti u obzir. Pod eksternim podacima podrazumevaju se svi oni podaci koji se ne mogu prikupiti unutar preduzeća i koji dolaze iz okruženja preduzeća.

2.3.1.2. Sloj pripreme podataka

Sloj pripreme podataka (eng. *data staging area*) je sloj u kome se izvodi skup procesa koji se nazivaju ETL procesi. U tehnološkom smislu, na ovom sloju se rešavaju problemi koji su tipični za distribuirane informacione sisteme *(Golfarelli & Rizzi, 2009)*. Sprovode se različite funkcije nad podacima kako bi se učinili korisnim u kontekstu skladišta podataka *(Reeves, 2009)*.

Ovaj sloj povezuje sloj izvora podataka i sloj prezentacionih podataka. U situacijama kada je arhitekturom predviđen sloj usaglašenih podataka, odnosno kod troslojnih arhitektura *(Golfarelli & Rizzi, 2009)* ovim slojem se povezuju sloj izvornih podataka i sloj usaglašenih podataka.

Na ovom sloju, podaci se čuvaju za potrebe ETL procesa i u obliku koji je pogodan za njihovo sprovođenje. Koriste se relacione tehnologije ili obične datoteke, kao i njihova kombinacija *(Kimball & Ross, 2002; Ponniah, 2011; Linstedt D. , 2010)*, a sve u cilju da se omogući efikasno izvršavanje ETL procesa. Pored podataka koji se obrađuju, na ovom sloju se čuvaju i referentni podaci kako bi se osigurao kvalitet podataka. *(Reeves, 2009)*

ETL procesi se smatraju najsloženijim u čitavom procesu razvoja sistema skladišta podataka *(Golfarelli & Rizzi, 2009; Ponniah, 2011)*. U literaturi se navodi da čak 70% vremena i uloženog napora na čitavom projektu odlazi na razvoj ETL procesa *(Golfarelli, 2009; Kimball, Ross, Thornthwaite, Mundy, & Becker, 2010)*. Ovim procesima se vrši ekstrahovanje i korekcija podataka, zatim njihova transformacija i konsolidacija i na kraju učitavanje u skladište podataka *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*.

2.3.1.3. Sloj usaglašenih podataka

Sloj usaglašenih podataka (eng. *reconciled data layer*) je sloj na kome se smeštaju podaci dobijeni nakon transformacije i integracije izvornih podataka, odnosno podaci koji predstavljaju rezultat izvršavanja ETL procesa. Ovaj sloj je karakterističan za troslojne arhitekture i njegovo uvođenje ima za posledicu da se popunjavanje sloja prezentacionih podataka ne vrši direktno iz izvora podataka, već indirektno preko ovog sloja *(Golfarelli & Rizzi, 2009)*.

Podaci na ovom sloju se skladište u repozitorijumu operativnih podataka (eng. *operational data store ODS*), koji može posmatrati kao integrisana kopija transakcionih podataka *(Kimball & Ross, 2002; Inmon, Strauss, & Neushloss, 2010)*. Podaci na ovom sloju se često ažuriraju kako bi odražavali tekuće stanje transakcionih sistema.

Prednost uvođenja ovog sloja podataka ogleda se u kreiranju opšteg referentnog modela operativnih podataka celokupnog preduzeća. Istovremeno, razdvaja se problem ekstrahovanja i integracije podataka od problema koji se odnose na popunjavanje skladišta podataka *(Golfarelli & Rizzi, 2009)*. Ovaj sloj se najčešće implementira kako bi se omogućilo izveštavanje u situacijama kada postojeći transakcioni sistemi nisu u mogućnosti da obezbede adekvatne izveštaje. Ovi izveštaji se tiču zahteva za odlučivanjem na taktičkom nivou *(Kimball & Ross, 2002)*.

Uvođenjem ovog sloja povećava se redudansa podataka i uvećava složenost sistema, pa *Kimball* i *Ross* navode da ovaj sloj treba uvoditi tek kada ni traksakcioni sistemi ni sistemi skladišta podataka ne mogu da zadovolje potrebe poslovanja *(Kimball & Ross, 2002)*.

2.3.1.4. Sloj prezentacionih podataka

Sloj prezentacionih podataka je ono što poslovna zajednica naziva skladištem podataka. To je jedino što poslovni korisnici vide i čemu pristupaju sa odgovarajućim alatima *(Kimball & Ross, 2002)*.

Na ovom sloju se nalaze strateške informacije kojima se može pristupiti direktno postavljanjem odgovarajućih upita. Za skladištenje ovih strateških informacija koristi se logički centralizovano skladište podataka *(Golfarelli & Rizzi, 2009)*. Njemu se može pristupati direktno, ali isto tako se može koristiti kao izvor za kreiranje odgovarajućih centara podataka (eng. *data mart*).

Komponenta skladište podataka se često naziva i primarno ili korporativno skladište podataka. Takvo skladište se ponaša kao sistem centralizovanog skladišta za sve podatke *(Golfarelli & Rizzi, 2009)*. Centri podataka se posmatraju kao male, lokalne, delimične replike primarnog skladišta koje su specifične za određeni aplikativni domen.

U praksi ne postoji konsenzus u pogledu organizacije ovog sloja *(Kimball & Ross, 2002; Inmon W. H., 2005)*. Različiti pristupi predlažu različita rešenja. Dok jedna rešenja predlažu postojanje jednog centralizovanog skladišta (*Inmon*-ov pristup), koje se može dalje replikovati u odgovarajuće centre podataka, dotle drugi predlažu postojanje samo usaglašenih centara podataka (*Kimball*-ov pristup).

Centri podataka koji se popunjavaju iz primarnog skladišta podataka se nazivaju *zavisni centri podataka*, dok se centri podataka koji se popunjavaju direktno iz izvora podataka nazivaju *nezavisni centri podataka (Golfarelli & Rizzi, 2009)*. Pored ova dva, u literaturi se navode i *usaglašeni centri podataka (Kimball & Ross, 2002)*. Kao i nezavisni centri podataka, usaglašeni centri podataka su karakteristični za pristupe koji ne podrazumevaju postojanje primarnog skladišta podataka. Razlika između ove dve vrste centara podataka se ogleda u obezbeđenju konzistentnog pogleda na informacije preduzeća *(Kimball & Ross, 2002)*. Za razliku od usaglašenih centara podataka, nezavisni centri podataka nose veliki rizik u pogledu ostvarivanja konzistentnog pogleda na informacije preduzeća.

Iako nisu neophodni, zavisni centri podataka mogu biti veoma korisni u sistemima skladišta podataka. Oni mogu poslužiti kao gradivni blokovi tokom inkrementalnog razvoja sistema skladišta podataka, ali i kao sredstvo za postizanje boljih performansi s obzirom da su manje od primarnih skladišta podataka. *(Golfarelli & Rizzi, 2009)*

2.3.1.5. Sloj analize podataka

Sloj analize podataka obuhvata skup alata i tehnologija kojima se poslovnim korisnicima omogućava pristup i analiza informacija koje su smeštene u sloju prezentacionih podataka. Ovim slojem je potrebno obezbediti efikasan i fleksibilan pristup strateškim informacijama kako bi se kreirali izveštaji, dinamički analizirale informacije i simulirali različiti hipotetički scenariji *(Golfarelli & Rizzi, 2009)*. Ovi alati mogu biti jednostavni, kao što su alati za postavljanje ad hoc upita, ali i veoma složeni kao što su alati za otkrivanje zakonitosti u podacima (eng. *data mining*) ili alati za simulacije (eng. *what-if analysis*) *(Kimball & Ross, 2002)*.

2.3.1.6. Sloj metapodataka

Metapodacima se nazivaju podaci kojima se definišu drugi podaci *(Golfarelli & Rizzi, 2009)*, odnosno podatak o podatku se naziva metapodatak *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*.

Metapodaci u sistemima skladišta podataka su slični *rečnicima podataka* (eng. *data dictionary*; *catalog*) koji se koriste u sistemima za upravljanje bazama podataka *(Ponniah, 2011)*. Rečnik podataka je deo baze podataka kojim se opisuje njena struktura, zatim pravila za očuvanje integriteta, indeksi, prava korišćenja i mnogi drugi bitni aspekti baze podataka. Uzimajući u obzir ulogu koju ima u bazama podataka, rečnik podataka se često naziva i metabaza podataka *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*.

Kao i kod baza podataka, metapodaci u sistemima skladišta podataka imaju važnu ulogu. Njima se precizno i u potpunosti opisuju svi bitni aspekti podataka u skladištu podataka, kako sa aspekta onih koji razvijaju skladišta podataka, tako i sa aspekta krajnjih korisnika skladišta podataka *(Ponniah, 2011; Kimball & Ross, 2002; Inmon, Strauss, & Neushloss, 2010)*.

Metapodaci zauzimaju ključnu poziciju u sistemima skladišta podataka *(Inmon, Strauss, & Neushloss, 2010)* kojima se omogućava izvršavanje, odnosno automatizacija različitih procesa *(Ponniah, 2011)*, kako tokom razvoja i održavanja, tako i tokom korišćenja skladišta podataka *(Object Management Group, 2003)*.

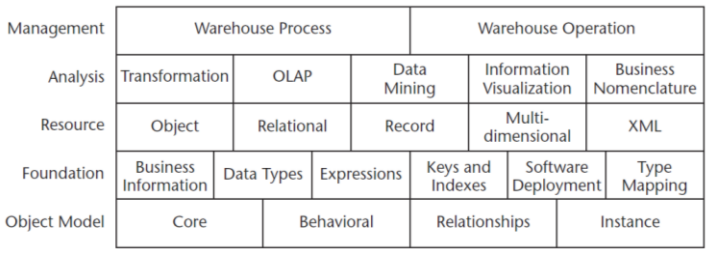
Metapodaci koji su generisani jednim procesom, se koriste u izvršavanju drugog procesa *(Ponniah, 2011)*. Naime, u procesu razvoja se koriste različiti alati, pri čemu svaki od njih prilikom kreiranja određenog dela skladišta podataka kreira i odgovarajuće metapodatke. Generisane metapodatke zatim koriste drugi alati kako bi kreirali druge delove skladišta podataka ili alati koji omogućavaju krajnjim korisnicima pristup podacima. Sve ovo jasno ukazuje da se u sistemima skladišta podataka koristi veći broj međusobno zavisnih alata i da je postizanje interoperabilnosti između njih od ključnog značaja.

Postizanje efektivne interoperabilnosti podrazumeva jednostavnu razmenu podataka između ovih različitih alata, odnosno jasnu i preciznu definiciju kako sintakse tako i semantike podataka koji se razmenjuju. S obzirom da se sintaksa i semantika podataka definiše preko metapodataka, potrebno je obezbediti standardizovan jezik za opisivanje i predstavljanje metapodataka, kao i standardizovan format za njihovu razmenu *(Poole, 2003)*.

Opšti metamodel skladišta podataka (eng. *common data warehouse metamodel CWM*) je standard koji je propisao Object Management Group (OMG). Osnovna svrha ovog metamodela je da omogući razmenu metapodataka, u domenu skladišta podataka i poslovne inteligencije, između različitih alata, platformi i repozitorijuma u distribuiranim heterogenim okruženjima *(Object Management Group, 2003)*. Ovim standardom se definiše jezik za opis metapodataka i mehanizam za njihovu razmenu *(Poole, 2003)*.

Predloženi standard se bazira na tri ključna standarda *Unified Modeling Language* (UML), *Meta Object Facility* (MOF) i XML Metadata Interchange (XMI), pri čemu za formalno predstavljanje metapodataka koristi UML, dok za njihovo skladištenje i razmenu koristi Extensible Markup Language (XML) *(Object Management Group, 2003)*. Uvođenjem ovog standarda, OMG je proširio postojeću arhitekturu metamodelovanja (MDA) sa svim onim konceptima koji su relevantni za domen skladišta podataka i poslovne inteligencije. Na ovaj način je omogućeno predstavljanje metapodataka u vidu formalnih modela koji su u skladu sa opštim

modelom skladišta podataka, što dalje omogućava razvijanje interoperabilnih baza podataka, aplikacija i alata na platformski nezavisan način *(Poole, 2003)*.



Slika 5. Slojevi CWM metamodela *(Object Management Group, 2003)*

Kao što je i prikazano (slika 5), CWM je sveobuhvatan metamodel koji se sastoji od većeg broja, blisko povezanih, metamodela koji su organizovani u odgovarajuće slojeve. Svaki od pripadajućih metamodela omogućava predstavljanje metapodataka u određenom poddomenu sistema skladišta podataka i poslovne inteligencije *(Object Management Group, 2003; Poole, 2003)*

2.3.2. Referentne arhitekture sistema skladišta podataka

Činjenica da ne postoji jedna najbolja arhitektura koja se može koristiti u svim situacijama *(Watson & Ariyachandra, 2005)* uslovila je pojavu većeg broja arhitektura. Ove različite arhitekture dobijene su različitim kombinovanjem osnovnih slojeva, čime su stvoreni alternativni načini ekstrahovanja, transformacije, učitavanja i skladištenja podataka u sistemima skladišta podataka *(Ariyachandra & Watson, 2010)*. U praksi su prepoznate sledeće arhitekture sistema skladišta podataka *(Reeves, 2009; Ariyachandra & Watson, 2010; Ponniah, 2011)*:

(1) arhitektura nezavisnih centara podataka (eng. *independent data mart arch.*),
(2) arhitektura usklađenih centara podataka (eng. *data mart bus architecture*),
(3) arhitektura zavisnih centara podataka (eng. *hub and spoke architecture*),
(4) centralizovana arhitektura (eng. *centralized architecture*) i
(5) federativna arhitektura (eng. *federated architecture*).

Navedene arhitekture su nastale iz prakse i različitih potreba na projektima, ali i kao rezultat korišćenja različitih pristupa u razvoju sistema skladišta podataka.

Nazivaju se još i *referentne arhitekture*, jer predstavljaju osnovu od koje se polazi u projektovanju sistema skladišta podataka *(Ariyachandra & Watson, 2010)*. Naime, na konkretnom projektu, u odnosu na postavljene zahteve, prvo se bira najpogodnija referentna arhitektura, a zatim se ona dalje prilagođava konkretnim potrebama.

Pravilan izbor referentne arhitekture je od velikog značaja za uspeh projekta. Ovaj izbor nije jednostavan i zahteva veliko iskustvo i odgovarajuća znanja koja se ogledaju u razumevanju preduslova koje svaka od ovih arhitektura postavlja, kao i u pogledu razumevanja prednosti i nedostataka koje svaka od njih nosi. Da bi se omogućilo bolje razumevanje i kontrola u procesu izbora odgovarajuće arhitekture, potrebno je identifikovati i razumeti faktore koji na taj izbor utiču *(Golfarelli & Rizzi, 2009; Ariyachandra & Watson, 2010)*.

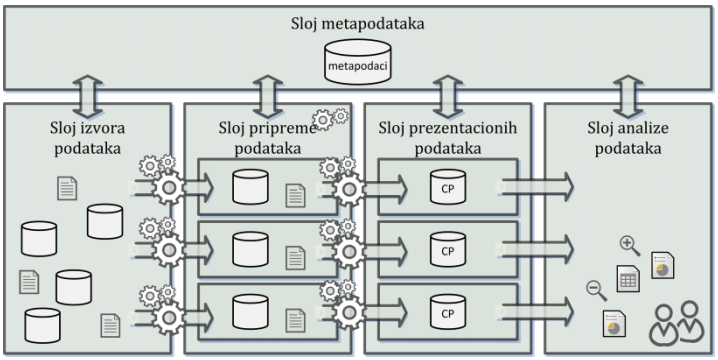
Ariyachandra i *Watson* su identifikovali jedanaest faktora koji se smatraju relevantnim: *(Watson & Ariyachandra, 2005; Ariyachandra & Watson, 2010)*: (1) međuzavisnost informacija koje se koriste u različitim organizacionim jedinicama, (2) potrebe višeg menadžmenta za informacijama, (3) hitnost uvođenja, (4) priroda posla korisnika, (5) ograničenja u pogledu resursa, (6) potreba za strategijskim pogledom pre implementacije, (7) uticaj eksperata, (8) kompatibilnost sa postojećim sistemima, (9) tehničke sposobnosti zaposlenih sektora informacionih tehnologija, (10) inicijator projekta i (11) tehnička pitanja.

Isti autori navode da određeni faktori imaju veći značaj u odnosu na ostale prilikom izbora, odnosno da različite kombinacije određenih faktora daje prednost izboru određene arhitekture. Pored ostalih rezultata, navode da faktori kao što su: potreba za strategijskim pogledom pre implementacije, tehničke sposobnosti zaposlenih sektora informacionih tehnologija i raspoloživost resursa imaju najveći uticaj na izbor arhitekture, dok hitnost uvođenja ima najmanji uticaj.

2.3.2.1. Arhitektura nezavisnih centara podataka

Arhitektura nezavisnih centara podataka (eng. *independent data mart architecture*) podrazumeva kolekciju različiti *centara podataka* koji se razvijaju nezavisno *(Rizzi,*

2008). Ovaj tip arhitekture nastaje u preduzećima gde organizacione jedinice razvijaju *centre podataka* za svoje potrebe *(Ponniah, 2011)*. Činjenica da *centri podataka* nisu integrisani, dovodi do situacija u kojima *centri podataka* imaju nekonzistentne definicije i standarde podataka. Može se reći, da ovako razvijeni *centri podataka* ne obezbeđuju „jednu verziju istine“, što znatno otežava i u nekim slučajevima onemogućava sprovođenje analiza koje zahtevaju različite centre podataka *(Ponniah, 2011)*.

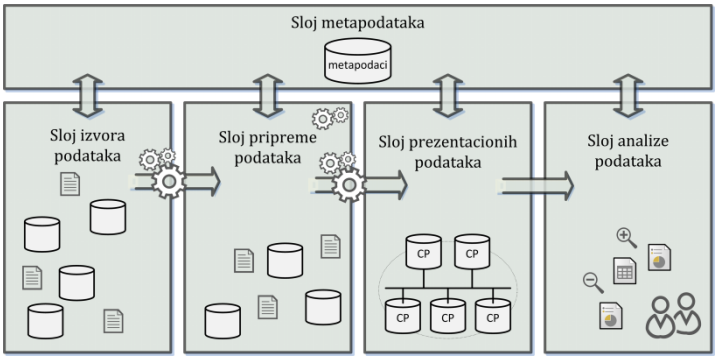


Slika 6. Arhitektura nezavisnih centara podataka

2.3.2.2. Arhitektura usklađenih centara podataka

Arhitektura usklađenih centara podataka (eng. *data mart bus architecure*) je tip arhitekture sličan *arhitekturi nezavisnih centara podataka* sa jednom značajnom razlikom koja se ogleda u činjenici da su razvijeni *centri podataka* usaglašeni. *Centri podataka* se razvijaju u skladu sa usaglašenim skupom *dimenzija* koji je dobijen na osnovu detaljne analize osnovnih procesa preduzeća. Na ovaj način obezbeđena je logička integracija *centara podataka (Rizzi, 2008)*, odnosno konzistentan pogled na informacije poslovanja. Ovaj tip arhitekture je nastao iz primene *odozdo-naviše pristupa* (eng. *bottom-up approach*) razvoja skladišta podataka koji je predložio *Ralph Kimball (Ponniah, 2011; Golfarelli & Rizzi, 2009)*.

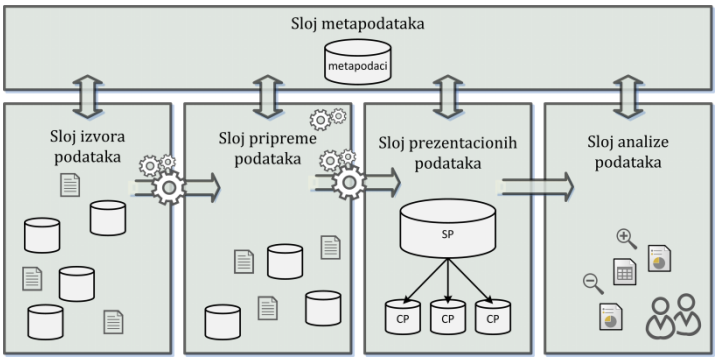
Kimbal i *Ross* navode da se korišćenjem ove arhitekture omogućava razvoj distribuiranih sistema skladišta podataka. Pored toga, ovom arhitekturom je omogućen razvoj na decentralizovan način koji je po njima realniji u odnosu centralizovan razvoj *(Kimball & Ross, 2002)*.



Slika 7. Arhitektura usklađenih centara podataka

2.3.2.3. Arhitektura zavisnih centara podataka

Arhitektura zavisnih centara podataka (eng. *hub and spoke architecture*) omogućava skalabilnost i proširivost, ali i sveobuhvatan pogled na informacije poslovanja *(Rizzi, 2008)*.

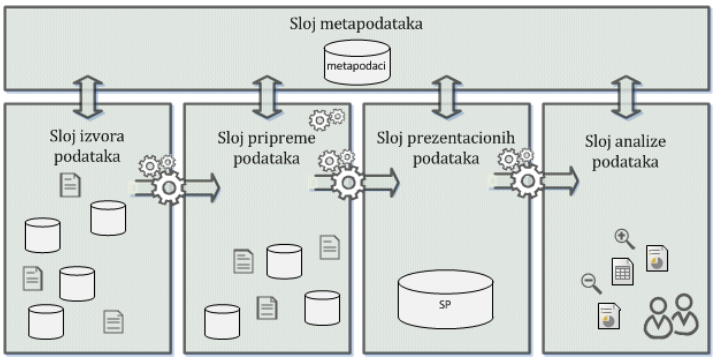


Slika 8. Arhitektura zavisnih centara podataka

Podaci na ovom sloju se čuvaju u normalizovanom obliku (*Inmon* predlaže da se koristi treća normalna forma 3NF) i služe kao izvor za centre podataka u kojima se sumirani podaci čuvaju u multidimenzionalnom obliku *(Golfarelli & Rizzi, 2009)*. *Centri podataka* razvijeni na ovaj način se nazivaju *zavisni centri podataka.* Mogu razvijati za različite namene, bilo za analitičke potrebe određenih organizacionih celina preduzeća, bilo za izvršavanje određenih specijalizovanih upita *(Ponniah, 2011)*. Korisnici obično pristupaju ovim *zavisnim centrima podataka*, ali je ovom arhitekturom dozvoljeno postavljanje upita i nad repozitorijumom usaglašenih podataka *(Golfarelli & Rizzi, 2009)*. Ovaj tip arhitekture je nastao iz primene *odozgo-nadole pristupa* (eng. *top-down approach*) za razvoj skladišta podataka koji je predložio *Bill Inmon (Ponniah, 2011)*.

2.3.2.4. Centralizovana arhitektura

Centralizovana arhitektura (eng. *centralized architecture*), slično kao i arhitektura zavisnih centara podataka, predviđa da se podaci čuvaju u normalizovanom obliku na najnižem nivou granularnosti *(Ponniah, 2011)*. *Rizzi* navodi da se ovaj tip arhitekture može posmatrati kao specifična implementacija *arhitekture zavisnih centara podataka*, s tom razlikom što su *sloj usaglašenih podataka* i *centri podataka* spojeni u jedan fizički repozitorijum *(Rizzi, 2008)*.



Slika 9. Centralizovana arhitektura

2.3.2.5. Federativna arhitektura

Federativna arhitektura (eng. *federated architecture*) je nastala iz potrebe da se postojeća *skladišta podataka* preduzeća integrišu, kako bi se stvorilo jedno okruženje kao podrška odlučivanju *(Rizzi, 2008)*. Ovakvim rešenjima se pribegava u situacijama kada nije ekonomski opravdano da se postojeća rešenja odbace *(Ponniah, 2011)* ili u situacijama kada se poslovanja integrišu pri čemu oba poslovanja imaju već razvijena *skladišta podataka*. Integracija ovih *skladišta podataka* se može izvršiti bilo virtuelno, bilo fizički uz korišćenje naprednih tehnika kao što su distribuirani upiti, ontologije, interoperabilnost zasnovana na meta podacima, itd *(Rizzi, 2008)*.

2.4. Proces razvoja sistema skladišta podataka

U ovom poglavlju je dat pregled relevantnih pristupa razvoja sistema skladišta podataka, a zatim je predstavljena njihova klasifikacija.

2.4.1. Pregled relevantnih pristupa razvoja

Razvoj sistema skladišta podataka, kao specifične vrste informacionog sistema, je veoma složen i vremenski zahtevan proces koji zahteva značajna finansijska sredstva. Potrebno je veliko znanje, veštine i sposobnosti svih učesnika kako bi se rešili brojni problemi i rizici koji vladaju na projektu. Pored toga, na projektima se sprovodi veliki broj različitih tipova aktivnosti, koje zajedno sa njihovim rezultatima treba organizovati na odgovarajući način. Sve ovo ukazuje na neophodnost korišćenja odgovarajućeg metodološkog pristupa, odnosno formalizovanog načina na koji se sistem skladišta podataka razvija *(Golfarelli & Rizzi, 2009)*.

Cilj metodoloških pristupa za razvoj sistema skladišta podataka je da omoguće organizacijama koje se bave razvojem da iznova i kvalitetno razvijaju sisteme skladišta podataka. Oni obezbeđuju disciplinovan pristup organizovanju i izvođenju projekata sa ciljem da se skladišta podataka realizuju sa definisanim planom i budžetom, kao i da zadovoljavaju definisani nivo kvaliteta. Uopšte, metodološkim pristupima za razvoj sistema skladišta podataka se omogućava savladavanje složenosti i postizanje uspeha na projektima.

Pod uspešnim projektom se podrazumeva projekat koji ispunjava sva očekivanja u vezi sa troškovima, planovima i kvalitetom, pri čemu kvalitet obuhvata željene funkcionalnosti i osobine. U kontekstu sistema skladišta podataka očekivanja se pre svega ogledaju u preciznim, pravovremenim i lako dostupnim informacijama koje omogućavaju donošenje strateških odluka. Pored toga, očekivanja su da rešenja budu stabilna i fleksibilna, tako da pored tekućih potreba budu u mogućnosti da podrže i buduće potrebe. Shodno tome, odluke tokom planiranja i izvođenja projekta se donose sa ciljem da se smanje troškovi i vreme izvršavanja, a da se pri tome postigne željeni, odnosno visok nivo kvaliteta.

Metodološki pristup se može posmatrati kao propisano rešenje kojim se definiše opšti proces razvoja, odnosno model procesa razvoja kojim se određuju aktivnosti koje se izvršavaju, uloge odgovorne za njihovo izvršavanje, zatim redosled izvršavanja aktivnosti, artefakti koji će biti razvijeni, kao i način procenjivanja njihovog kvaliteta *(Petrović, 2012)*. Osnovna premisa modela procesa razvoja je da će, u situacijama u kojima je model primenljiv, njegovo korišćenje kao procesa projekta voditi ka niskim troškovima, visokom nivou kvaliteta i smanjenju potrebnog vremena. Uopšteno, predstavlja sredstvo za dostizanje ciljeva projekta.

Za definiciju metodoloških pristupa koriste se tri koncepta: proces, modeli i arhitektura. Procesom se definiše skup aktivnosti koje se sprovode u odgovarajućem redosledu. Rezultati ovih aktivnosti su specifični modeli kojima se sistem predstavlja sa različitih tačaka gledišta (funkcionalni, projektni, implementacioni, itd.). Da bi se omogućilo lakše praćenje i razumevanje procesa razvoja vrši se njegovo strukturiranje, odnosno grupisanje povezanih tipova aktivnosti u odgovarajuće faze. Sve one zajedno čine životni ciklus razvoja. Kaže se da su to faze kroz koje se prolazi od trenutka kada je identifikovana potreba za sistemom do trenutka kada se on povlači iz upotrebe.

Jedan od osnovnih izazova je izbor odgovarajućeg metodološkog pristupa za razvoj skladišta podataka. Važno je napomenuti da nisu sve pristupi pogodni za određeni projekat, kao i da ne postoji jedan najbolji metodološki pristup za razvoj sistema skladišta podataka. Izbor pogrešnog pristupa može imati za posledicu neuspešan projekat. Određeni pristup na jednom projektu može dati odlične rezultate, dok na drugom projektu može dovesti do neuspeha. Ovo je posledica činjenice da nisu svi projekti isti. Oni mogu biti slični, ali se razlikuju okolnosti pod kojima se izvršavaju. Pored toga, metodološkim pristupom se definišu opšte smernice koje zahtevaju dalje prilagođavanje na konkretnim projektima, odnosno prilagođavanje konkretnim potrebama i uslovima kako bi se dobilo optimalno rešenje.

U literaturi i praksi je predložen veći broj pristupa koji se bave razvojem sistema skladišta podataka:

◊ *Inmon* navodi da se životni ciklus razvoja skladišta podataka značajno razlikuje od životnog ciklusa razvoja transakcionih sistema. Kod transakcionih sistema životni ciklus razvoja je vođen zahtevima, dok je životni ciklus razvoja skladišta podataka vođen podacima. On dalje navodi da razvoj skladišta podataka ne može biti vođen zahtevima, jer zahtevi postaju jasni tek pošto klijenti izvrše analizu upita nad skladištem podataka *(Inmon W. H., 2005)*. Naravno, ovakvo gledište podrazumeva da je skladište podataka razvijeno i popunjeno sa podacima, pa otuda i potreba da se prvo analiziraju izvori podataka čime se zahtevi korisnika stavljaju u drugi plan *(Golfarelli, 2010)*.

Životni ciklus razvoja sistema skladišta podataka koristi spiralni model razvoja koji započinje sa podacima. *Inmon* kao prvu fazu u procesu razvoja predlaže *(Inmon W. H., 2005)*: (1) fazu implementacije skladišta podataka (eng. *implement data warehouse*) iza koje sledi (2) integracija dostupnih podataka (eng. *integrate data*), (3) testiranje (eng. *test for bias*), (4) programiranje (eng. *programm against data*), (6) projektovanje (eng. *design DSS system*), (7) analiza rezultata (eng. *analyse results*) i kao poslednju navodi (8) fazu specifikacije zahteva (eng. *understand requirements*). Ovako dobijeni zahtevi, su prema *Inmon*-u precizni i jasni i predstavljaju osnov za dalje dorade i prilagođavanja sistema. Nakon toga započinje novi ciklus kojim se obrađuje drugi skup podataka.

◊ *Kimball* predlaže odozdo-nagore pristup, kojim se u odgovarajućim ciklusima implementiraju centri podataka. Ovi centri podataka se mogu implementirati nezavisno i predstavljaju sveobuhvatan pogled na poslovne procese preduzeća *(Kimball & Ross, 2002)*. U osnovi ovog pristupa je orijentacija na sveobuhvatne potrebe poslovanja. Integracija nezavisno razvijenih centara podataka omogućena je sveobuhvatnim planiranjem, odnosno izradom matrice skladišta podataka kojom se identifikuju opšte poslovne dimenzije preko kojih se centri podataka usaglašavaju *(Kimball, Ross, Thornthwaite, Mundy, & Becker, 2010)*.

Razvoj skladišta podataka se realizuje inkrementalno, pri čemu se svakim ciklusom razvija deo skladišta podataka odnosno određeni centar podataka. Životni ciklus je podeljen u više faza *(Kimball, Ross, Thornthwaite, Mundy, & Becker, 2010)*: (1) faza

planiranja (eng. *program/project planning*), (2) faza definisanja poslovnih zahteva (eng. *business requirements definition*), (3) faza projektovanja i implementacije, (4) faza isporuke (eng. *deployment*), (5) faza rasta (eng. *growth*) i (6) faza održavanja (eng. *maintenance*). Aktivnosti koje se sprovode tokom faze projektovanja i implementacije su podeljene u tri grupe, odnosno aktivnosti koje se bave tehnologijom, aktivnosti koje se bave podacima i aktivnosti koje se bave korisničkim aplikacijama. Aktivnosti pomenutih grupa se mogu izvršavati paralelno i u okviru svake grupe su dalje strukturirane u odgovarajuće faze.

◊ *Golfarelli* navodi da se proces razvoja sistema skladišta podataka može podeliti u tri faze *(Golfarelli, 2009)*: (1) planiranje skladišta podataka, (2) projektovanje i implementacija centara podataka i (3) održavanje i evolucija skladišta podataka.

Pod fazom planiranja skladišta podataka podrazumeva se određivanje opsega i ciljeva koji se projektom žele postići. Identifikuju se neophodni centri podataka i definišu prioriteti, odnosno redosled u kome će se oni realizovati. Pored centara podataka, definiše se i odgovarajuća arhitektura budućeg sistema. Pod arhitekturom se misli na logičku i fizičku arhitekturu sistema *(Reeves, 2009)*, pri čemu se logičkom arhitekturom definišu neophodne komponente i veze između njih, dok se fizičkom arhitekturom definišu softverske i hardverske potrebe.

Pod fazom projektovanja i implementacije centara podataka podrazumeva se iterativni razvoj centara podataka, pri čemu svaka iteracija prolazi kroz sledeće faze *(Golfarelli, 2009)*: (1) analiza zahteva, (2) konceptualno projektovanje, (3) logičko projektovanje, (4) projektovanje ETL procesa i (5) fizičko projektovanje.

Pod fazom održavanja i evolucije skladišta podataka podrazumevaju se aktivnosti koje se tiču optimizovanja performansi skladišta podataka, ali i aktivnosti koje se bave promenama u poslovnom domenu i promenama poslovnih zahteva.

◊ *Luján-Mora* i *Trujilo* su predložili objektno-orijentisani pristup koji su nazvali *Data Warehouse Engineering Process (Luján-Mora & Trujillo, 2005)*. Predloženi pristup se bazira na dobro poznatom i opšte prihvaćenom *Jedinstvenom procesu razvoja softvera* (eng. u*nified process,* u daljem tekstu UP), dok za modelovanje

koristi *Jedinstveni jezik modelovanja* (eng. *unified modeling language*, u daljem tekstu UML).

Autori navode da je predloženi pristup celovit i da omogućava razvoj kompletnog sistema skladišta podataka *(Luján-Mora & Trujillo, 2005)*, odnosno da omogućava podršku razvoju svih komponenti skladišta podataka i to na konceptualnom, logičkom i fizičkom nivou. U skladu sa ovim obezbeđeni su odgovarajući formalizmi, zasnovani na UML profilima, koji se koriste za modelovanje na svakom od pomenutih nivoa.

Predloženi pristup je prilagođeni UP za domen razvoja sistema skladišta podataka *(Luján-Mora & Trujillo, 2005)*. Kao takav, propisuje iterativno-inkrementalni proces razvoja za koji su definisane četiri upravljačke faze: (1) početna faza (eng. *inception*), (2) faza elaboracije (eng. *elaboration*), (3) faza konstrukcije (eng. *construction*) i (4) faza tranzicije (eng. *transition*). U kontekstu ovih faza, izvršava se veći broj iteracija u okviru kojih se izvršavaju različiti tipovi aktivnosti. Ove različite aktivnosti su dalje grupisane u odgovarajuće potprocese (eng. *workflows*) koji se različitim intenzitetom izvršavaju u okviru svake iteracije, a u zavisnosti od upravljačke faze kojoj iteracija pripada. Autori su proširili osnovni skup potprocesa i propisuju sledeće potprocese *(Luján-Mora & Trujillo, 2005)*: (1) prikupljanje zahteva (eng. *requirements*), (2) analiza zahteva (eng. *analysis*), (3) projektovanje (eng. *design*), (4) implementacija (eng. *implementation*), (5) testiranje (eng. *test*), (6) održavanje (eng. *maintenance*) i (7) razmatranje (eng. *post-development review*). Faza održavanja i razmatranja ne postoje originalno u UP, već su dodate kao specifične za razvoj skladišta podataka. U fazi održavanja sprovode se aktivnosti u vezi punjenja i osvežavanja skladišta podataka, dok se u fazi razmatranja vrši revizija projekta kako bi se identifikovala poboljšanja u vezi procesa razvoja na budućim projektima *(Luján-Mora & Trujillo, 2005)*.

◊ *Ponniah* navodi da su faze životnog ciklusa organizovane oko opšte arhitekture sistema skladišta podataka koju čine tri funkcionalne komponente: prikupljanje podataka (eng. *data acquisition*), skladištenje podataka (eng. *data storage*) i isporuka podataka (eng. *information delivery*) *(Ponniah, 2011)*. S obzirom na

kompleksnost sistema skladišta podataka predlaže korišćenje iterativnog pristupa. Kao faze u životnom ciklusu predlažu se tradicionalne faze: (1) planiranje (eng. *project plan*), (2) definisanje zahteva (eng. *requirements definition*), (3) projektovanje (eng. *design*), (4) implementacija (eng. *construction*), (5) isporuka (eng. *deployment*) i (6) održavanje (eng. *maintenance*). *Ponniah* dalje navodi da se aktivnosti koje se sprovode u fazama konstrukcije i implementacije mogu grupisati oko komponenti koje su definisane arhitekturom sistema.

Takođe, *Ponniah* navodi da je u razvoju sistema skladišta podataka izražen trend prihvatanja agilnih principa razvoja kojima se ohrabruje saradnja tehničke i poslovne zajednice i promoviše iterativno-inkrementalni razvoj.

◊ *Ambler* je predložio agilni, objektno-orijentisani pristup koji predstavlja prilagođenu verziju *Rational Unified Process* (RUP) za razvoj skladišta podataka *(Ambler, 2006)*. Kao takav propisuje iterativno-inkrementalni proces u kome su definisane četiri upravljačke faze: (1) početna faza (eng. *inception*), (2) faza elaboracije (eng. *elaboration*), (3) faza konstrukcije (eng. *construction*) i (4) faza tranzicije (eng. *transition*). Za svaku upravljačku fazu definiše se jedna ili više iteracija, kao i ciljevi koje treba postići. Svaka iteracija se tretira kao mini projekat, koji ima svoj početak i kraj i čijim se izvršavanjem dobija rezultat koji je za jedan korak bliži krajnjem rezultatu. Na osnovu ciljeva se ocenjuje napredak na projektu, odnosno utvrđuje se da li definisane iteracije konvergiraju ka rešenju. Definisane upravljačke faze se izvršavaju sekvencijalno i na kraju svake je definisan odgovarajući ključni događaj u kome se verifikuju rezultati faze, odnosno proverava da li su dostignuti postavljeni ciljevi. Tokom svake iteracije izvršavaju se različite aktivnosti koje su grupisane u odgovarajuće discipline. Vreme koje se provodi na izvršavanju aktivnosti određene discipline varira u zavisnosti od faze u kojoj se projekat nalazi. Definisano je devet disciplina, koje su dalje grupisane u tehničke i discipline za podršku. Tehničke discipline su: (1) modelovanje poslovanja, (2) prikupljanje zahteva, (3) projektovanje, (4) implementacija, (5) testiranje, (6) isporučivanje, dok su discipline podrške: (7) upravljanje projektom, (8) upravljanje konfiguraijom i promenama i (9) unapređenje procesa.

Kao i *Ponniah*, *Ambler* propisuje korišćenje agilnih koncepata i tehnika, jer kako navodi, postizanje visoko kvalitetnog rešenja koje može adekvatno da odgovori na promene, zahteva napuštanje tradicionalnih i usvajanje agilnih tehnika *(Ambler, 2006)*.

◊ ◊ ◊

Uzimajući u obzir kompleksnost sistema skladišta podataka, kao i činjenicu da se za njihovu realizaciju angažuju značajni resursi, kako u pogledu finansija tako i u pogledu vremena koje je potrebno za njihovu realizaciju, u industriji preovlađuje stav da je potrebno sprovoditi iterativni, odnosno iterativno-inkrementalni razvoj. Iterativni razvoj sistema skladišta podataka se pre svega ogleda u postepenom profinjavanju rešenja u svakom sledećem koraku razvoja, čime se brže dolazi do prvih rezultata. Na ovaj način omogućena je pravovremena verifikacija rešenja, ispravljanje grešaka, ali i adekvatno prilagođavanje promenama. Korisnici imaju priliku da koriste i testiraju rešenja, čime stiču iskustvo i razumevanje na osnovu kojih mogu da daju precizne i jasne zahteve. Ove povratne informacije i saradnja tehničke i poslovne zajednice su od neprocenjivog značaja za uspeh projekta *(Reeves, 2009)*. Upravo ovo je jedna od osnovnih vrednosti koja se neguje u agilnim pristupima. Agilne pristupe karakteriše fleksibilnost, za razliku od tradicionalnih pristupa u kojima se propisivanje vrši na visokom nivou detaljnosti. Ovim pristupima se propisuju vrednosti, principi i prakse koje se koriste u svakom aspektu razvoja i implementacije skladišta podataka *(Ponniah, 2011)*.

U industriju ne postoji konsenzus u pogledu metodoloških pristupa, ali se većina slaže da se u procesu razvoja sistema skladišta podataka mogu identifikovati sledeće faze *(Golfarelli, 2009)*: (1) analiza zahteva, (2) konceptualno modelovanje, (3) logičko modelovanje, (4) modelovanje ETL procesa i (5) fizičko modelovanje.

Analiza zahteva je ključna faza tokom koje se vrši identifikacija informacija koje su neophodne u procesu odlučivanja. Ova faza se smatra ključnom, jer od kvaliteta prikupljenih zahteva u velikoj meri zavisi uspeh celokupnog projekta. Identifikovani zahtevi se tokom faze konceptualnog modelovanja prevode u implementaciono nezavisan konceptualni model, koji se dalje tokom faze logičkog

modelovanja prevodi u odgovarajući logički model. Ovaj logički model u zavisnosti od korišćenog formalizma može biti relacioni, multidimenzionalni ili hibridni. Tokom modelovanja ETL procesa definišu se mapiranja između odgovarajućih logičkih modela izvora i skladišta podataka, kao i transformacije koje su neophodne kako bi se podaci iz izvornih sistema učitali u skladište podataka. Tokom fizičkog modelovanja logički model se prevodi i u odgovarajući fizički model kojim se obuhvataju sve specifičnosti izabranog tehnološkog okruženja.

2.4.2. Klasifikacija metodoloških pristupa

Metodološki pristupi razvoja sistema skladišta podataka se mogu svrstati u *(Golfarelli & Rizzi, 2009; Ponniah, 2011)*:

(1) odozgo-naniže pristupi (eng. *top-down approaches*) i
(2) odozdo-naviše pristupi (eng. *bottom-up approaches*).

Pristup odozgo-naniže podrazumeva razvoj skladišta podataka kao celine *(Golfarelli & Rizzi, 2009)*. Analiziraju se globalne potrebe preduzeća i na bazi njih projektuje i razvija sveobuhvatno skladište podataka *(Ponniah, 2011)*. Ovako razvijeno skladište podataka obezbeđuje sveobuhvatan pogled na informacije preduzeća čime se stvaraju preduslovi za ispunjenje tekućih, ali i potreba koje mogu nastati u budućnosti. Može se reći da se ovim pristupom postiže stabilno i visoko kvalitetno rešenje. Međutim, ovaj pristup pokazuje i ozbiljne nedostatke koji se pre svega ogledaju u visokim troškovima i vremenu koje je potrebno za realizaciju *(Golfarelli & Rizzi, 2009)*. Činjenica da implementacija može da počne tek nakon sveobuhvatne analize i projektovanja dovodi do toga da je potrebno dosta vremena kako bi se videli prvi rezultati. Posledica ovoga je da se gubi mogućnost rane verifikacije rešenja, ali i da korisnici gube poverenje i interesovanje čime se rizici na projektu dodatno povećavaju. Upravo ovo su i razlozi zbog kojih su brojni projekti označeni kao neuspešni *(Golfarelli & Rizzi, 2009)*.

Pristup odozdo-naviše podrazumeva inkrementalni razvoj centara podataka *(Ponniah, 2011)*. Centri podataka se razvijaju prema prioritetima, jedan po jedan u

skladu sa potrebama organizacionih jedinica preduzeća, odnosno potreba određenih grupa korisnika. Ovim pristupom se smanjuju troškovi i vreme koje je potrebno da se dođe do prvih rezultata, čime je omogućena rana verifikacija rezultata, pravovremeno otklanjanje grešaka, kao i mogućnost da se brže prilagođava promenama *(Golfarelli & Rizzi, 2009)*. Međutim, pored navedenih prednosti, ovom pristupu se kao najveća mana pripisuje fragmentacija podataka *(Ponniah, 2011)*, odnosno zanemarivanje sveobuhvatnih potreba preduzeća. Fokus na potrebe korisnika, uz zanemarivanje potreba poslovanja, rezultuje nezavisnim centrima podataka kojima se ne obezbeđuje konzistentan pogled na celokupno poslovanje *(Golfarelli, 2009)*. Na ovaj način direktno se utiče na stabilnost rešenja i mogućnost da se odgovori na buduće potrebe. Da bi se prevazišli navedeni problemi, u literaturi se predlaže povezivanje, odnosno usaglašavanje centara podataka *(Kimball, Ross, Thornthwaite, Mundy, & Becker, 2010)*. Ovo podrazumeva da se centar podataka koji je prvi razvijen tretira kao referentna tačka oko koje se naredni centar podataka razvija. Svaki naredni centar podataka se razvija korišćenjem konzistentnih podataka koji su već dostupni *(Golfarelli & Rizzi, 2009)*. Na ovaj način omogućeno je da se rezultati iteracija na projektu uklapaju u već postojeće rešenje, odnosno da svaki razvijeni centar podataka predstavlja logički podskup kompletnog skladišta podataka *(Ponniah, 2011)*.

Watson i *Ariyachandra* navode da su tokom vremena ovi pristupi postali veoma slični *(Watson & Ariyachandra, 2005)*. Naime, pristupi odozgo–nadole uvažavaju činjenicu da je inkrementalni razvoj i rana isporuka rešenja veoma bitan faktor uspeha, ali isto tako pristupi odozdo-nagore uvažavaju činjenicu da je sveobuhvatno sagledavanje potreba poslovanja, odnosno postojanje sveobuhvatnog plana razvoja i integracije inkrementalno razvijenih centara podataka veoma važno u pogledu obezbeđenja dugoročnog rešenja.

Dugogodišnje iskustvo u razvoju transakcionih sistema je pokazalo da razumevanje problema koji se rešava, odnosno identifikovanje svih relevantnih poslovnih zahteva predstavlja preduslov za postizanje uspeha na projektu. Može se reći da je razvoj transakcionih sistema vođen poslovnim zahtevima. Za razliku od transakcionih sistema, kod sistema skladišta podataka po ovom pitanju nije

postignut konsenzus *(Inmon W. H., 2005; Kimball, Ross, Thornthwaite, Mundy, & Becker, 2010)*. Naime, opšte je shvatanje da su poslovni zahtevi neizostavni u procesu razvoja sistema skladišta podataka, ali se njihova uloga i mesto u procesu razvoja razlikuje u različitim pristupima.

Uopšte, uzimajući u obzir ulogu i mesto koje poslovni zahtevi zauzimaju u procesu razvoja skladišta podataka, pristupi se mogu klasifikovati na:

(1) pristupi vođeni podacima,
(2) pristupi vođeni potrebama,
    (2.1) pristupi vođeni korisnicima,
    (2.2) pristupi vođeni ciljevima,
(3) kombinovani pristupi

U odnosu na ulogu i mesto koje poslovni zahtevi zauzimaju u procesu razvoja skladišta podataka, pristupi se mogu klasifikovati u dve osnovne kategorije *(Winter & Strauch, 2003; Corr & Stagnitto, 2011; Golfarelli, 2010; Reeves, 2009)*: (1) *pristupi vođeni podacima* (eng. *data-driven; supply-driven approaches*) i (2) *pristupi vođeni potrebama* (eng. *demand-driven; requirement-driven approaches*). Prvu kategoriju čine pristupi koji se polaze od analize dostupnih izvora podataka, dok drugu čine pristupi koji se polaze od potreba koje buduće skladište podataka treba da zadovolji.

Pristupi vođeni potrebama se dalje mogu klasifikovati na *(List, Bruckner, Machaczek, & Schiefer, 2002; Golfarelli, 2010)*: (1) *pristupi vođeni korisnicima* (eng. *user-driven approaches*) i (2) *pristupi vođeni ciljevima* (eng. *goal-driven approaches*). Pristupi vođeni korisnicima polaze od pojedinačnih zahteva korisnika koji se na kraju integrišu, dok pristupi vođeni ciljevima polaze od strategije poslovanja koja se progresivno profinjava *(Golfarelli, 2010)*. *Golfarelli* dalje navodi da će se rezultati ova dva pristupa razlikovati ukoliko korisnici nemaju jasnu sliku i razumevanje o poslovnoj strategiji i ciljevima organizacije.

Navedeni pristupi se smatraju osnovnim u razvoju sistema skladišta podataka. Njihovom analizom može se utvrditi da oni nisu međusobno isključivi, već da su

komplementarni *(List, Bruckner, Machaczek, & Schiefer, 2002)* i da se njihovom zajedničkom upotrebom mogu eliminisati nedostaci koji proističu iz samostalne, nezavisne primene *(Golfarelli, 2010)*. Na bazi ovog saznanja, u literaturi i praksi je razvijen veći broj pristupa koji na različite načine kombinuju osnovne pristupe, čineći na taj način posebnu grupu koja se naziva *kombinovani pristupi* (eng. *hybrid*; *mixed approaches*). *Romero* i *Abello* navode da oni najčešće započinju fazom u kojoj se identifikuju potrebe, iza koje sledi faza u kojoj se identifikuju podaci. *(Romero & Abello, 2010)*.

2.4.2.1. Pristupi vođeni podacima

Pristupi vođeni podacima (eng. *data-driven; supply-driven approaches*) se smatraju odozdo-naviše pristupima, koji se zasnivaju na analizi podataka koji su dostupni u izvornim sistemima *(List, Bruckner, Machaczek, & Schiefer, 2002; Golfarelli, 2010)*. Zahtevi u vezi sa podacima se dobijaju na osnovu analize modela podataka izvornih sistema ili preko fizičkog modela, tako što se biraju oni elementi koji se smatraju relevantnim za skladište podataka *(Ballard, Farrell, Gupta, Mazuela, & Vohnik, 2006)*. U skladi sa tim, *Ballard* navodi da je rezultat ovih pristupa informacija o tome šta se može dobiti.

Ovi pristupi su karakteristični za rane inicijative u kojima je učešće korisnika izbegavano *(Corr & Stagnitto, 2011)*. Učešće korisnika je ograničeno na izbor relevantnih podataka iz skupa podataka koji su dostupni u izvornim sistemima *(Golfarelli, 2010)*. U osnovi ovih pristupa je stav da se skladišta podataka ne mogu razvijati na osnovu zahteva, jer su oni poznati tek nakon što se podaci prikupe i analiziraju *(Inmon W. H., 2005)*.

Reeves navodi da je ovaj pristup nekada bio popularan, ali da je praksa pokazala da retko vodi ka uspehu *(Reeves, 2009)*. Kao osnovnu zamerku ovom pristupu navodi da je kompletan razvoj izvršen bez, ili uz veoma malo uključivanje korisnika, što rezultuje rešenjima u kojima su podaci organizovani na način koji ne obezbeđuje vezu sa stvarnim potrebama korisnika.

Takođe, ovaj pristup zahteva značajna ulaganja, jer je potrebno dosta vremena da se dođe do rešenja koje će imati vrednost za poslovanje *(Reeves, 2009; Corr & Stagnitto, 2011)*. Pored toga, povećan je rizik razvoja nevalidnog skupa zahteva kojima neće biti identifikovani ciljevi poslovanja i potrebe korisnika *(Ballard, Farrell, Gupta, Mazuela, & Vohnik, 2006)*. Ovo je pre svega posledica odsustva korisnika koji bi definisali upravljiv opseg projekta i definisali prioritete, ali i činjenice da su modeli podataka u transakcionim sistemima složeni i često veoma generički *(Corr & Stagnitto, 2011)*.

Golfarelli navodi da je primena ovog pristupa opravdana kada postoji detaljno znanje o izvorima podataka, odnosno kada modeli podataka izvornih sistema nisu kompleksni i kada ih karakteriše dobar nivo normalizacije *(Golfarelli, 2010)*. *Ballard* navodi da je ovim pristupom omogućeno da na samim počecima projekta bude poznato šta se može isporučiti *(Ballard, Farrell, Gupta, Mazuela, & Vohnik, 2006)*, čime se izbegavaju problemi u vezi sa prevelikim očekivanjima korisnika *(Reeves, 2009)*. Isto tako, skraćuje se i potrebno vreme, jer se ne sprovode intervjui i grupne sesije sa korisnicima.

2.4.2.2. Pristupi vođeni korisnicima

Pristupi vođeni korisnicima (eng. *user-driven approaches*) se smatraju odozdo-naviše pristupima, koji počinju sa identifikacijom zahteva, odnosno potreba korisnika skladišta podataka *(Ballard, Farrell, Gupta, Mazuela, & Vohnik, 2006; Golfarelli, 2010)*. Fokus je na onome što je stvarno potrebno, za razliku od pristupa vođenih podacima kojima je fokus na obezbeđivanju svih dostupnih podataka. Ovim pristupima se u kraćem vremenskom periodu obezbeđuje korisno skladište podataka, odnosno skladište podataka koje zadovoljava potrebe korisnika *(Ballard, Farrell, Gupta, Mazuela, & Vohnik, 2006)*.

Ovim pristupima se održava konstantna interakcija sa korisnicima čime se povećava stepen prihvatanja budućeg rešenja *(List, Bruckner, Machaczek, & Schiefer, 2002)*. Korisnici izuzetno cene ovaj pristup *(Golfarelli, 2010)*, jer su aktivno uključeni u proces i što mogu da razumeju razloge zbog kojih su određene odluke na projektu donete. Pored toga stiču utisak da su dali doprinos i da imaju

priliku da oblikuju budući sistem prema svojim potrebama. Međutim, održavanje konstantne interakcije sa korisnicima dovodi do toga da su ovi pristupi vremenski zahtevni *(Golfarelli, 2010)*. Potrebno je vreme da se sprovedu brojni sastanci i intervjui. Vođenje sastanaka zahteva određene veštine, ali u nekim slučajevima i najiskusniji imaju problem da održe sastanak u granicama predviđenog. Isto tako, uzimajući u obzir prirodu posla koji korisnici obavljaju, njima je teško da jasno i precizno opišu šta im je stvarno potrebno. Korisnici su svesni da se promene često dešavaju, kao i da probleme koje razmatraju treba posmatrati iz različitih perspektiva, pa imaju problem da se opredele koji im izveštaji tačno trebaju. Uopšte, korisnici imaju tendenciju da traže sve, pa čak i ono što im verovatno neće ni trebati.

Zahtevi prikupljeni na ovaj način iziskuju dodatni korak u kome će se izvršiti njihova integracija *(Golfarelli, 2010)*. *Golfarelli* navodi da cilj ovog koraka nije samo da se prikupljeni zahtevi objedine, već i da se izvrši njihovo usaglašavanje što nije lak posao. Naime, korisnici često imaju tendenciju da na probleme gledaju usko, odnosno da ih posmatraju isključivo iz svoje perspektive pri čemu zanemaruju širi kontekst celokupnog poslovanja.

Nakon izvršene integracije prikupljenih podataka, ovi pristupi kao sledeći korak podrazumevaju mapiranje zahteva sa podacima koji su dostupni u izvorima podataka *(Golfarelli, 2010; Ballard, Farrell, Gupta, Mazuela, & Vohnik, 2006)*. Činjenica da se ovo dešava na kraju može da dovede do neuspeha, odnosno do nemogućnosti ispunjenja identifikovanih potreba. Činjenica da ovo dovodi do razočarenja korisnika *(Golfarelli, 2010)*, koji su uložili veliki napor u definisanje zahteva, iziskuje upravljanje očekivanjima korisnika *(List, Bruckner, Machaczek, & Schiefer, 2002)*. Korisnicima se mora objasniti da određene zahteve nije moguće ispuniti, jer pojedine podatke nije moguće dobiti iz izvora podataka. I pored ovakvih situacija u kojima se radi na zahtevima koje nije moguće realizovati, smatra se da su koristi od ovakvog pristupa veće nego nedostaci *(Ballard, Farrell, Gupta, Mazuela, & Vohnik, 2006)*. *Ballard* navodi da se ohrabrivanjem korisnika da razmilja u širem kontekstu smanjuju šanse da se određene stvari zaborave ili zanemare.

Iako omogućavaju efikasan razvoj *(Corr & Stagnitto, 2011)*, primena ovih pristupa povećava rizik u pogledu adekvatnosti i stabilnosti budućeg skladišta podataka *(List, Bruckner, Machaczek, & Schiefer, 2002; Golfarelli, 2010)*. Osnovni razlog ovome, je što različiti pogledi korisnika mogu dovesti do rešenja koje ne odražava potrebe poslovanja u celini. Zanemarivanje ciljeva i potreba poslovanja, odnosno isključivo fokusiranje na tekuće potrebe korisnika rezultuje nefleksibilnim rešenjima kojima nije moguće zadovoljiti potrebe u budućnosti *(Ballard, Farrell, Gupta, Mazuela, & Vohnik, 2006)*.

2.4.2.3. Pristupi vođeni ciljevima

Pristupi vođeni ciljevima (eng. *goal-driven approaches*) se smatraju odozgo-nadole pristupima, koji polaze od ciljeva i strategija poslovanja *(List, Bruckner, Machaczek, & Schiefer, 2002; Golfarelli, 2010)*. U osnovi ovih pristupa je razumevanje da se za obezbeđivanje efikasnog procesa donošenja strateških odluka mora uzeti u obzir osnova svrha poslovanja, odnosno ciljevi koje poslovanje želi postići kao i strategije kojima se ti ciljevi postižu. Fokus na osnovnu svrhu poslovanja smanjuje rizik od zastarelosti rešenja *(Golfarelli, 2010)*, odnosno omogućava razvoj dugoročnog rešenja kojim će moći da se zadovolje potrebe u budućnosti.

Primenljivost ovih pristupa u velikoj meri zavisi od spremnosti top menadžmenta da učestvuje u procesu *(List, Bruckner, Machaczek, & Schiefer, 2002)*. Informacije se prikupljaju kroz sprovođenje intervjua sa top menadžerima, nakon čega se različite vizije konsoliduju. Ovako dobijena vizija, odnosno sveobuhvatna strategija poslovanja se prevodi u ključne indikatore performansi koji se mogu meriti *(Golfarelli, 2010)*. *Golfarelli* navodi da je potrebno veliko znanje i sposobnost projektanata kako bi se prikupljeni zahtevi na visokom nivou apstrakcije preveli u odgovarajuće indikatore.

Kod ovih pristupa polazi se identifikovanja ključni poslovnih procesa, jer se njima direktno doprinosi ispunjenju ciljeva poslovanja *(Böhnlein & Ulbrich-vom Ende, 2000)*. Za identifikovane poslovne procese se definišu ključni indikatori, koji se zatim profinjavaju i transformišu u odgovarajuće modele podataka.

Pristupi vođeni korisnicima polaze od pojedinačnih zahteva korisnika koji se na kraju integrišu, dok pristupi vođeni ciljevima polaze od strategije poslovanja koja se progresivno profinjava *(Golfarelli, 2010)*. *Golfarelli* dalje navodi da će se rezultati ova dva pristupa razlikovati, ukoliko korisnici nemaju jasnu sliku i razumevanje o poslovnoj strategiji i ciljevima poslovanja.

2.4.2.4. Kombinovani pristupi

Kombinovani pristupi (eng. *hybrid*;*mixed approaches*) su nastali na osnovu saznanja da osnovni pristupi nisu međusobno isključivi, već da su komplementarni *(List, Bruckner, Machaczek, & Schiefer, 2002)*. Njihovom zajedničkom upotrebom, odnosno kombinovanjem mogu se eliminisati nedostaci koji proističu iz samostalne, nezavisne primene *(Golfarelli, 2010)*.

U odnosu na način na koji kombinuju osnovne pristupe mogu se razlikovati *(Romero & Abelló, 2009)*: *sekvencijalno kombinovani pristupi* (eng. *sequential hybrid approaches*) i *isprepleteno kombinovani pristupi* (eng. *interleaved hybrid approaches*). Osnovna karakteristika prvih je da se osnovni pristupi sprovode nezavisno, a da se na kraju povezuju rezultati, dok se kod drugih osnovni pristupi sprovode paralelno.

2.5. Pristup razvoju softvera zasnovan na modelima

Na samim počecima razvoja softvera, softver je bio jednostavan, razvijali su ga pojedinci i korišćen je u istraživačke svrhe. Vremenom, softver pronalazi svoje mesto u gotovo svim porama života, bilo da su to lične potrebe ili potrebe organizacije. Danas je gotovo nemoguće zamisliti organizaciju koja svoje poslovanje ne obavlja uz pomoć softvera. Softver više nije prestiž, već realna potreba svakog poslovanja.

Organizacije više ne posmatraju softver samo kao alat za automatizaciju kojim se povećava efikasnost, već uviđaju njegovu stratešku ulogu u ostvarivanju konkurentske pozicije. Konkurencija diktira razvoj, pa samim tim organizacije zahtevaju više, zahtevi su složeniji i obimniji što za posledicu ima složeniji softver čija je realizacija znatno komplikovanija.

Istorijski gledano, softversku industriju prate sve složeniji problemi i sve učestalije promene u okruženju koje u velikoj meri otežavaju poslovanje i samim tim utiču na uspeh poslovanja. Sve ovo zahteva stalna prilagođavanja i iznalaženje novih načina razvoja koji će obezbediti veću efikasnost.

Kao i druge industrije i softverska industrija je u stalnom „pokretu". Iznalaženje novih načina razvoja softvera ogleda se u novim pristupima, metodologijama i tehnologijama kojima se omogućava efikasnije rešavanje problema. U današnje vreme broj tehnologija je veliki i njima se omogućava rešavanje veoma složenih problema koji su u prošlosti bili nezamislivi. Ovo se svakako može tretirati kao pozitivan trend, jer se sa novim tehnologijama omogućava efikasnije rešavanje problema, nude se nove mogućnosti i povećava produktivnost.

Međutim, ono što zabrinjava je činjenica da se ovako brzom ekspanzijom novih tehnologija, u kontekstu postojećih pristupa, značajno skraćuje životni vek rešenja. Novim tehnologijama zamenjuju se postojeće tehnologije, čime se indirektno i rešenja koja su njima realizovana čine zastarelim. Na ovaj način, posmatrano u dužem vremenskom periodu, umanjuje se vrednost uloženog rada čime se smanjuje produktivnost što nije u skladu sa osnovnim ciljem da se omogući efikasniji razvoj.

Uzimajući u obzir izložene probleme, može se reći da je softverska industrija iscrpela postojeće pristupe i da je dostigla kritičnu tačku (softverska kriza) u kojoj se moraju dati novi pravci razvoja. Postojeći princip „Sve je objekat" više nije dovoljan.

Istorijski gledano, softverska industrija se već susretala sa ovakvim situacijama. Na primer, 60-tih godina prošlog veka problem je bio da se omogući nezavisnost softvera od hardverske platforme ili 80-tih godina kada je izvršen prelaz sa proceduralnog na objektno-orijentisani pristup. Karakteristika svih ovih pristupa se ogleda u korišćenju apstrakcije kao osnovne tehnike u savladavanju složenosti. Uopšte, može se reći da je istorija razvoja softvera ustvari istorija podizanja nivoa apstrakcije na kome se radi *(Greenfield & Short, 2003)*.

Najnoviji pristup koji se predlaže u softverskoj industriji za rešavanje navedenih problema je razvoj vođen modelima (eng. *model driven development*, u daljem tekstu MDD). Kao i prethodni pristupi, osnovna karakteristika je da izdiže nivo apstrakcije na kojoj se radi na viši nivo. Može se reći da je osnovni motiv ovog pristupa da se dosadašnji princip „Sve je objekat“ zameni novim „Sve je model“ *(Bezivin, 2004; Bezivin, 2005)*.

Uvođenjem novog sloja apstrakcije, zanemaruju se detalji na nižim nivoima čime se zapravo ignoriše tehnologija koja se koristi prilikom implementacije softvera. Modeli postaju primarni softverski artefakti kojima se vodi razvoj softvera. Oni imaju veću upotrebnu vrednost i duži životni vek, čime se u velikoj meri doprinosi povećanju produktivnosti i efikasnosti razvoja.

2.5.1. Povećanje produktivnosti

Kao što je prethodno navedeno, povećanje produktivnosti je jedan od osnovnih ciljeva koji se želi postići u softverskoj industriji. U skladu sa tim potrebno je razumeti na koje načine je moguće povećati produktivnost. Njihovim razumevanjem omogućava se lakše sagledavanje i ocenjivanje različitih pristupa koji se predlažu.

Povećanje produktivnosti se može postići na dva načina *(Atkinson & Kuhne, 2003)*: (1) kratkoročnim povećanjem produktivnosti i (2) dugoročnim povećanjem produktivnosti. Pod kratkoročnim povećanjem produktivnosti misli se na povećanje vrednosti primarnih softverskih artefakata u smislu količine funkcionalnosti koja se na osnovu njih dobija. Što je količina izvršne funkcionalnosti, dobijene generisanjem iz primarnih softverskih artefakata veća, to je veća i produktivnost. Pod dugoročnim povećanjem produktivnosti misli se na produžavanje životnog veka primarnih softverskih artefakata. Što je životni vek artefakta duži, to je i vrednost uloženog rada u njegovo kreiranje veća. Shodno tome, cilj je da primarni softverski artefakti budu upotrebljivi u dužem vremenskom periodu.

Atkinson i *Kuhne* navode da je dugoročno povećanje produktivnosti daleko važnije i da se može smatrati strateškim ciljem. Za njegovo ostvarivanje, potrebno je da se smanji uticaj promena na softverske artefakte, odnosno da se softverski artefakti učine imunim na promene. Isti autori su identifikovali četiri osnovne kategorije promena *(Atkinson & Kuhne, 2003)*: (1) promena članova tima, (2) promena zahteva, (3) promena razvojne platforme i (4) promena izvršne platforme.

Navedeni tipovi promena ukazuju na najznačajnije probleme razvoja softvera i ciljeve koje treba postići. Shodno navedenim problemima može se reći da novim pristupima treba da se obezbedi *(Object Management Group, 2003)*: (1) interoperabilnost, (2) portabilnost i (3) ponovno korišćenje.

2.5.2. Model

Kreiranje uprošćene predstave sistema pre kreiranja samog sistema nije nova ideja. Ona je odavno poznata u inženjerskim disciplinama koje su dosta zrelije od softverske i predstavlja obaveznu praksu kojom je omogućena analiza karakteristika budućeg sistema, verifikacija projektantskih odluka i način da se te projektantske odluke prenesu onima koji će implementirati sistem. Ova uprošćena predstava sistema se naziva *model sistema*, a proces njenog kreiranja se naziva *modelovanje (Milicev, 2009)*.

U literaturi postoji veći broj definicija modela i sve se u osnovi slažu da model uvek predstavlja apstrakciju nečega što postoji u realnosti, da se razlikuje od te realnosti u pogledu nivoa detalja sa kojim je opisuje, kao i da se može iskoristiti kao primer na osnovu koga se može proizvesti nešto što postoji u realnosti *(Kuhne, 2006)*. U nastavku je navedeno nekoliko različitih definicija koje su date u literaturi, a zatim je data definicija koja će se koristiti za potrebe rada.

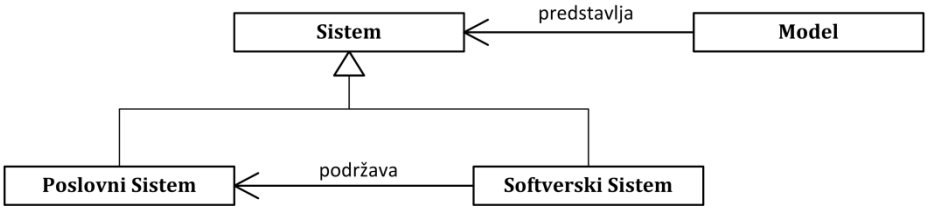
„Model je opis sistema ili nekog njegovog dela, koji je iskazan dobro definisanim jezikom. Dobro definisani jezik je onaj koji ima dobro definisanu sintaksu i semantiku tako da ga softverski sistem može automatski interpretirati“ *(Kleppe, Warmer, & Bast, 2003)*.

„Model je opis, odnosno specifikacija sistema i njegovog okruženja koji se izrađuje sa određenom svrhom. Najčešće se predstavlja kao kombinacija crteža i teksta, pri čemu se tekst može predstaviti određenim jezikom modelovanja ili prirodnim jezikom“ *(Object Management Group, 2003)*.

Za potrebe ovog rada koristiće se sledeća definicija:

Model je opis posmatranog sistema ili nekog njegovog dela koji je iskazan u određenom dobro definisanom jeziku modelovanja. Kreira se sa određenom namerom i predstavlja znanje koje se može interpretirati u odnosu na posmatrani sistem.

„Model je opis posmatranog sistema ili nekog njegovog dela […] Kreira se sa određenom namerom [...] “ što znači da se modelom želi analizirati određeni aspekt posmatranog sistema. U kontekstu razvoja softvera, modelima je omogućeno predstavljanje znanja kako o domenu problema (poslovni sistem) tako i o domenu rešenja (softverski sistem).



Slika 10. Odnos modela i sistema

Za posmatrani sistem, moguće je definisati veći broj modela, pri čemu svaki od njih predstavlja određeni aspekt sistema na odgovarajućem nivou detalja. Kao što je i predstavljeno (slika 10), odnos između *Modela* i *Sistema* je opisan vezom koja se naziva „*predstavlja*“ *(Kurtev, Bezivin, Jouault, & Valduriez, 2006)*.

„[…] predstavlja znanje koje se može interpretirati u odnosu na posmatrani sistem.“ što znači da omogućava da se na osnovu njega mogu identifikovati karakteristike posmatranog sistema i donositi odgovarajuće odluke. U kontekstu razvoja softvera, modelom poslovnog sistema omogućeno je sagledavanje i rezonovanje o domenu problema, odnosno razumevanje problema koji treba da se reši, i identifikovanje potreba. Isto tako, modelom softverskog sistema omogućeno je sagledavanje

domena rešenja, odnosno rezonovanje i donošenje projektantskih odluka u pogledu budućeg softverskog rešenja.

„[…] iskazan u određenom dobro definisanom jeziku modelovanja.“ što znači da se za kreiranje modela koristi određeni jezik modelovanja koji mora imati precizno i jasno definisanu sintaksu i semantiku. Prema *Milićevu*, neophodan je odgovarajući rečnik koji se može koristiti tokom modelovanja. Ovaj rečnik zapravo predstavlja definiciju jezika modelovanja, koga čini skup koncepata zajedno sa njihovom sintaksom i semantikom *(Milicev, 2009)*.

Jezici su osnova razvoja sistema. Koriste se različiti jezici, od jezika modelovanja visokog nivoa, kojima se apstrahuju implementacioni detalji, do jezika na nižim nivoima koji se baziraju na specifičnim implementacionim tehnologijama *(Clark, Sammut, & Willans, 2008)*. S obzirom da se modelom predstavljaju određeni aspekti posmatranog sistema, veoma je bitno da se koristi adekvatan jezik modelovanja, odnosno jezik koji podržava (sadrži) koncepte koji su semantički bliski posmatranom problemu *(Mernik, Heering, & Sloane, 2005; Kosar, Oliveira, Mernik, Pereira, Črepinšek, & Henriques, 2010)*. Izbor adekvatnog jezika modelovanja omogućava značajno povećanje produktivnosti *(Clark, Sammut, & Willans, 2008; Voelter, i drugi, 2013)*.

Osnovna klasifikacija jezika je ona po kojoj se jezici dele na: jezike opšte namene (eng. *general purpose languages*, u daljem tekstu GPL) i jezike specifične namene (eng. *domain specific languages*, u daljem tekstu DSL). Jezici opšte namene se mogu koristiti u različite svrhe, odnosno konceptima koji su podržani jezikom mogu se predstaviti različiti aspekti posmatranog sistema *(Kelly & Tolvanen, 2008)*. Za razliku od njih, jezici specifične namene (domensko-specifični jezici) imaju vrlo ograničenu primenu, ali su zato semantički bogati, odnosno dosta bliži posmatranom problemu *(Mernik, Heering, & Sloane, 2005; Voelter, i drugi, 2013)*.

Navedena klasifikacija je ortogonalna na ostale klasifikacije, kao što je klasifikacija na programske jezike i jezike modelovanja. Ova klasifikacija na programske jezike i jezike modelovanja je često korišćena klasifikacija kojom se pravi razlika u odnosu na mogućnosti izvršavanja. Međutim, na ovom mestu je bitno naglasiti da se sa

pojavom novih pristupa, kao što je MDD, ova razlika gubi, s obzirom da se programi tretiraju kao modeli i da neki jezici modelovanja mogu da se izvršavaju *(Kurtev, Bezivin, Jouault, & Valduriez, 2006; Voelter, i drugi, 2013)*

2.5.3. Uloga modela u postojećim pristupima

Pojavom *Jedinstvenog jezika modelovanja* (eng. *unified modeling language*, u daljem tekstu UML) stvorena je osnova za razvoj softvera koji se zasniva na modelima. Međutim, primena ovog jezika u realnosti nije imala veliki uticaj na razvoj softvera *(Greenfield & Short, 2003; Greenfield, Short, Cook, & Kent, 2004; Kelly & Tolvanen, 2008)*. U postojećim pristupima razvoju softvera, modeli se pretežno koriste za dokumentovanje poslovnih i softverskih sistema i kao sredstvo za ostvarivanje nedvosmislene komunikacije između učesnika na projektu. Modeli se pretežno kreiraju u fazi projektovanja i predstavljaju smernice o tome kako sistem treba da se implementira. Oni gube na vrednosti istog trenutka kada započne faza kodiranja i kako kodiranje odmiče njihova vrednost je sve manja *(Babarogić, 2004)*. Na ovaj način, modeli su vremenom sve dalji od stvarnog stanja stvari, čime gube svaku dalju upotrebnu vrednost.

Uzimajući u obzir ulogu modela u postojećim pristupima razvoja softvera, može se postaviti pitanje opravdanosti njihovog korišćenja. Posebno ako se uzme u obzir njihova mala upotrebna vrednost i kratak životni vek. Odnos uloženog rada i efekti koji se njihovom upotrebom dobijaju ne idu u prilog povećanja produktivnosti. Shodno tome, da bi upotreba modela imala opravdanje, neophodno je da se omogući njihovo aktivno učešće u procesu razvoja. Potrebno je da se omogući njihovo procesiranje, odnosno procesiranje informacija koje su u njima predstavljene kako bi se automatizovao razvoj *(Greenfield & Short, 2003)*.

2.5.4. Razvoj vođen modelima (MDD)

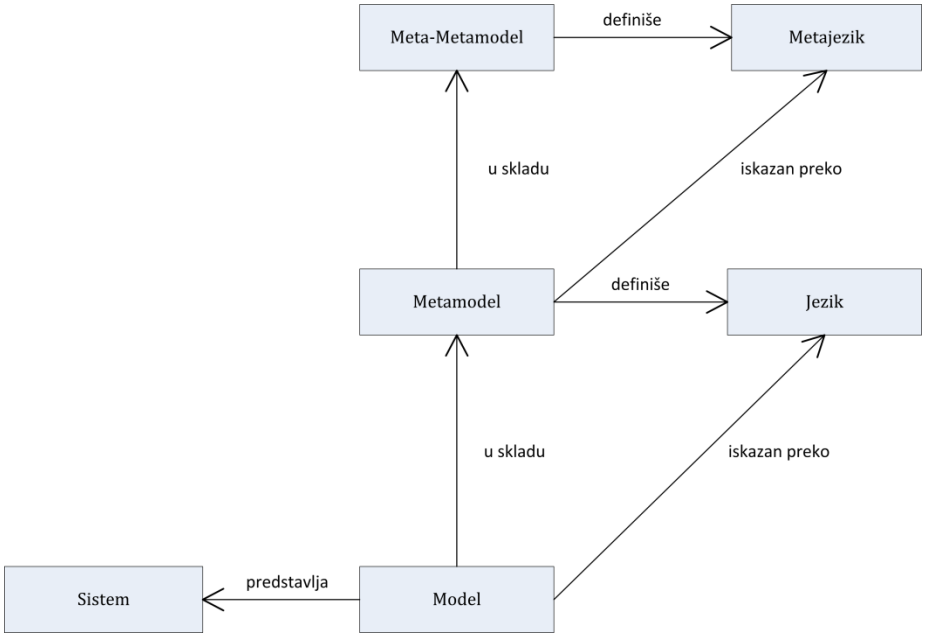
Razvoj vođen modelima (eng. *model driven development*, u daljem tekstu MDD) je jedan od najsavremenijih pristupa razvoju softvera kojim se fokus razvoja softvera pomera sa programskog koda na modele. Osnovni cilj koji se želi postići ovim pristupom je da se poveća produktivnost i skrati vreme potrebno za

implementaciju softverskog rešenja. Osnovna tehnika kojom se služi je apstrakcija pa se može reći da je razvoj vođen modelima pristup kojim se izdiže nivo apstrakcije kroz uvođenje koncepata koji su bliži domenu problema.

Vizija koja se želi ostvariti ovim pristupom je da se modeli mogu koristiti direktno u razvoju softvera. U skladu sa tim, podrazumeva da se sistem razvija preko skupa modela koji se sukcesivno transformišu. Transformacije zauzimaju centralno mesto.

Razvoj vođen modelima se bazira na ideji da najvažniji rezultat razvoja softvera nije programski kôd, već modeli. Glavni cilj razvoja vođenog modelima je da se automatizuje razvoj softvera kroz postupak uzastopnih transformacija modela, počev od modela koji predstavlja specifikaciju sistema, do modela koji predstavljaju detaljni opis fizičke realizacije iz kojih se dalje može automatski generisati programski kôd.

Model se može definisati u odnosu na internu organizaciju i u odnosu na moguću upotrebu, odnosno korišćenje *(Kurtev, Bezivin, Jouault, & Valduriez, 2006)*.

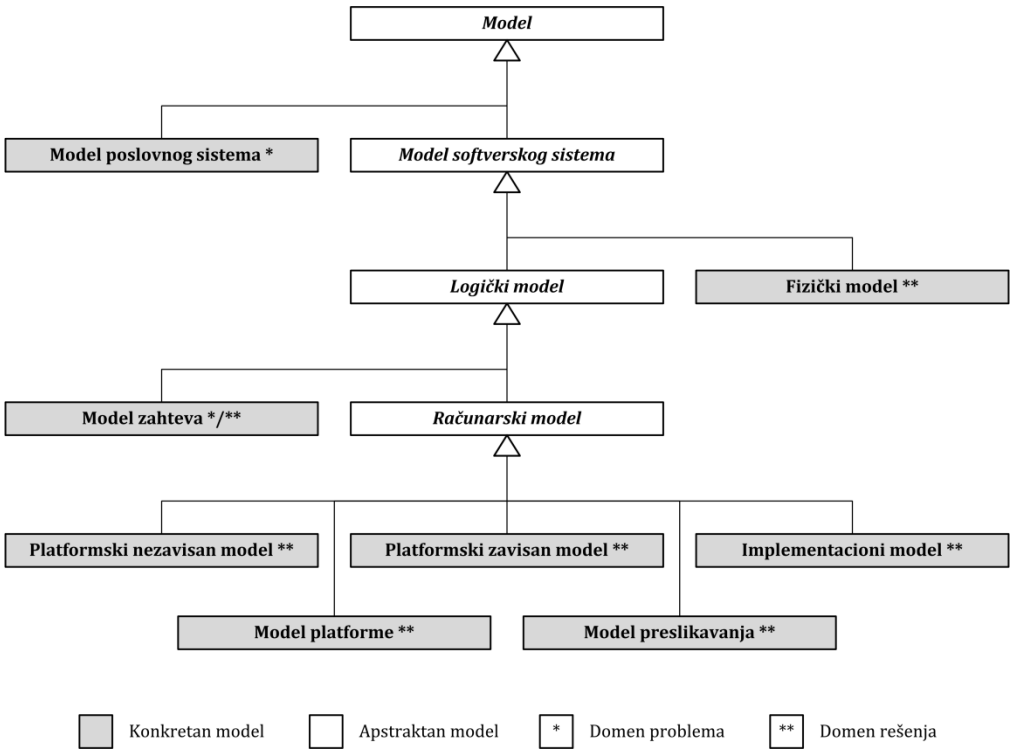


Slika 11. Organizacija modela *(Kurtev, Bezivin, Jouault, & Valduriez, 2006; Kuhne, 2006)*

Kao što je i prikazano (slika 11), odnos između modela, metamodela i meta-metamodela je predstavljen vezom „u skladu“ *(eng.* conformsTo*)*, dok je odnos

modela i sistema predstavljen vezom „predstavlja“ *(eng.* represents*). Kurtev* navodi da je organizaciona struktura modela ograničena modelom koji se naziva metamodel, koji je dalje ograničen modelom koji se naziva meta-metamodel. Za iskazivanje modela koriste se odgovarajući jezici modelovanja, pa su u skladu sa tim predstavljeni (slika 11) *Metajezik i Jezik (Kuhne, 2006). Metajezik* omogućava iskazivanje metamodela i definisan je meta-metamodelom, dok *Jezik* omogućava iskazivanje modela i definisan je metamodelom.

Definisanjem modela u odnosu na njegovo korišćenje, definiše se njegova svrha i veza sa realnim sistemom koga predstavlja. U skladu sa tim predstavljena je taksonomija modela (slika 12) prema kojoj se modeli mogu podeliti na: (1) modele poslovnog sistema i (2) modele softverskog sistema.



Slika 12. Taksonomija modela *(Frankel, 2002; Albin, 2003)*

Modeli poslovnog sistema opisuju aspekte posmatranog poslovanja bez obzira da li će svi ti aspekti biti automatizovani, odnosno realizovani softverskim sistemom. Osnovni cilj ovih modela je da obuhvate sve relevantne aspekte posmatranog poslovnog sistema, kako bi omogućili razumevanje problema koji treba rešiti. Tek

pošto se razume problem moguće je identifikovati potrebe koje budući softverski sistem treba da realizuje.

Model softverskog sistema opisuje aspekte softverskog sistema kojim se pruža podrška poslovnom sistemu, odnosno kojim se poslovni sistem automatizuje. Njihov opseg je uži od opsega *Modela poslovnog sistema*. Oni se mogu podeliti na logičke i fizičke modele, pri čemu se *Logičkim modelima* opisuje logika softverskog sistema, a *Fizičkim modelima* fizički artefakti i resursi koji se koriste tokom razvoja i izvršavanja (korišćenja) softverskog sistema.

Modeli zahteva (*Računarski nezavisni modeli*) predstavljaju kategoriju logičkih modela kojima se specificiraju identifikovane potrebe, odnosno zahtevi koje budući softverski sistem treba da automatizuje *(Object Management Group, 2003)*. Ovi modeli ne uključuju tehničke detalje i specificiraju se na osnovu znanja o poslovnom sistemu, odnosno na osnovu *Modela poslovnog sistema*. Njima se specificiraju sve funkcije i kvalitativni atributi koje budući softverski sistem treba da poseduje (ispuni).

Ovi modeli daju odgovor na pitanje „*Šta softverski sistem treba da radi?*“ i igraju važnu ulogu u premošćavanju jaza koji postoji između eksperata iz domena problema, odnosno domena poslovnog sistema i eksperata iz domena rešenja, tj. domena softverskog sistema *(Object Management Group, 2003)*.

Idealni razvoj softvera bi podrazumevao da se na osnovu *Modela zahteva* automatski izgeneriše softverski sistem. Međutim, semantički jaz koji postoji između *Modela zahteva* i softverskog sistema je prevelik, tako da je tako nešto teško uraditi. Da bi se ovo omogućilo potrebno je da se izvrše razne dorade i profinjavanja kojima se dobijaju modeli na nižem nivou apstrakcije i koji su semantički „bliži“ softverskom sistemu. Ovo u praksi podrazumeva kreiranje većeg broja modela na nižim nivoima apstrakcije kojim se *Model zahteva* postepeno prevodi u softverski sistem.

Kao i *Modeli zahteva*, *Računarski modeli* predstavljaju logičke modele, ali za razliku od *Modela zahteva* oni uključuju tehničke detalje. Njima se specificira realizacija i

implementacija softverskog sistema. Ove specifikacije se daju preko niza modela koji se definišu na različitim nivoima apstrakcije.

Razvoj softvera vođen modelima se zasniva na postepenom uvođenju detalja, pa je tako *Model zahteva* je na višem nivou apstrakcije od *Platformski nezavisnog modela*, koji je na višem nivou apstrakcije od *Platformski zavisnog modela*, a on dalje na višem nivou u odnosu na *Model implementacije*.

Platformski nezavisan model predstavlja dalju razradu *Modela zahteva* i kao takav je na nižem nivou apstrakcije. Iako je nezavisan od korišćene platforme njime se uključuju određeni tehnički detalji. Platformska nezavisnost je relativan pojam, jer da bi se za neki model moglo reći da je platformski nezavisan moraju se navesti tehnologije u odnosu na koje je nezavisan *(Frankel, 2002)*.

Platformski zavisni modeli se realizuju na osnovu *Platformski nezavisnih modela*. Oni predstavljaju njihovu dalju razradu, odnosno proširivanje detaljima koji se tiču izabrane tehnološke platforme (na primer, .NET, J2EE, itd.). Ovim modelima se mapiraju elementi platformski nezavisnih modela u određenu tehnološku platformu ili programski jezik. Ovi tipovi modela služe kao uputstva kojima se vodi implementacija softverskog sistema *(Albin, 2003)*.

Na osnovu *Platformski zavisnih modela* prelazi se na niži, implementacioni nivo koji je predstavljen *Implementacionim modelima*. U osnovi ovi modeli predstavljaju programe koji su napisani u određenom programskom jeziku i u opštem smislu se mogu tretirati kao modeli jer specificiraju softverski sistem.

Model poslovnog sistema i *Model zahteva* su rezultat aktivnosti kojima se modeluje domen problema i koja se naziva analiza, dok su ostali modeli rezultat aktivnosti kojima se modeluje domen rešenja i nazivaju se projektovanje i implementcija *(Frankel, 2002)*.

Profinjavanje, odnosno postepena dorada modela se može raditi manuelno, poluautomatski ili u potpunosti automatski. S obzirom da je osnovni cilj koji se želi postići tokom razvoja softvera postizanje visokog stepena produktivnosti,

automatska transformacija se nameće kao željeno rešenje. U tu svrhu se uvode *Model platforme* i *Model preslikavanja* kao osnov automatskih transformacija.

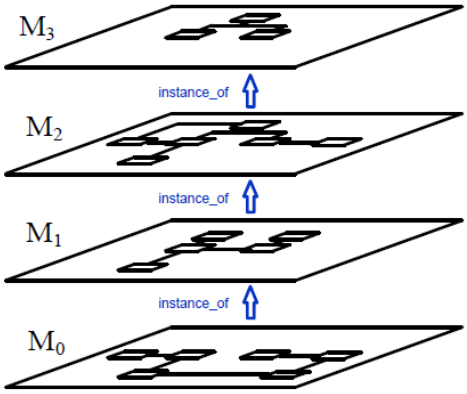
2.5.5. Arhitektura vođena modelima (MDA)

Arhitektura zasnovana na modelima (eng. *model driven architecture*, u daljem tekstu MDA) je originalni pristup razvoju softvera vođen modelima. Predložen je 2001. godine od strane *Object Managament Group* (OMG) kao najvećeg udruženja u softverskoj industriji. MDA se može definisati kao realizacija principa razvoja vođenog modelima preko skupa OMG standarda kao što su MOF, XMI, OCL, UML, CWM, SPEM, itd. *(Bezivin, 2005; Kurtev, Bezivin, Jouault, & Valduriez, 2006)*.

MDA predlaže definisanje modela na različitom nivou apstrakcije u vidu tri osnovne vrste modela *(Object Management Group, 2003)*: (1) računarski nezavisni modeli (eng. *computer independent models*, u daljem tekstu CIM), (2) platformski nezavisni modeli (eng. *platform independent models*, u daljem tekstu PIM) i (3) platformski specifični modeli (eng. *platform specific models*, u daljem tekstu PSM). Za modelovanje ove tri vrste modela koristi se UML kao osnovni jezik modelovanja *(Pilone & Pitman, 2005)*, ali se mogu definisati UML profili (eng. *UML profiles*), kao proširenja osnovnog UML jezika *(Object Management Group, 2003)*.

Softverski sistem se razvija preko skupa platformski nezavisnih modela koji se u fazi implementacije automatski transformišu u odgovarajuće platformski zavisne modele (eng. *model-to-model transformation* M2M transformation). Ovi platformski zavisni modeli se, takođe automatski transformišu u odgovarajući programski kôd (eng. *model-to-text transformation* M2T transformation).

MDA definiše četiri osnovna apstraktna nivoa na kome se modeli mogu nalaziti. Ovi nivoi su po MDA standardu imenovani od M0 do M3. Na svakom od ovih nivoa se nalaze modeli koji opisuju odgovarajući niži apstraktni nivo, odnosno koji predstavljaju instance modela višeg nivoa.



Slika 13. Apstraktni nivoi prema OMG MDA standardu *(Kuhne, 2006)*

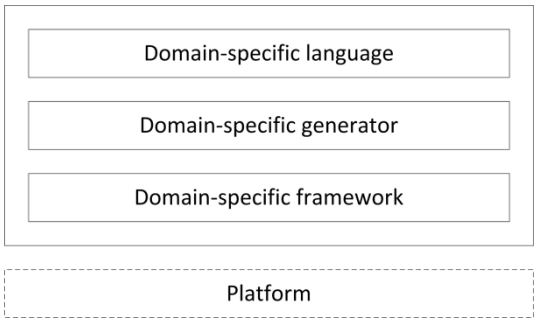
U skladu sa OMG MDA standardom, na najnižem nivou apstrakcije se nalaze instance resursa koji su deo sadržaja konkretnih informacionih sistema. Ovaj nivo apstrakcije se naziva nivo instanci ili M0. Apstraktni nivo iznad se naziva nivo modela ili M1. Na ovom nivou se nalaze modeli kojima se opisuju resursi sa M0 nivoa. Sledeći apstraktni nivo se naziva nivo metamodela ili M2. Osnovna svrha ovog nivoa je da definiše jezik modelovanja pomoću koga se iskazuju modeli sa M1 nivoa. Drugačije govoreci, M2 modeli su metamodeli M1 modela, što implicira da su M1 modeli instance M2 modela. Jedan od najpoznatijih metamodela u MDA je UML metamodel. Najviši apstraktni nivo se naziva nivo meta-metamodela ili M3. Osnovna svrha ovog nivoa je da definiše jezik pomoću koga se iskazuju modeli na nižem apstraktnom nivou, tj. metamodeli. Prema MDA standardu na ovom nivou je definisan jedan jezik koji se naziva MOF (Meta Object Facility). M3 nivo je uveden sa ciljem da se izbegne postojanje većeg broja nekompatibilnih metamodela koji se definišu i razvijaju nezavisno *(Bezivin, 2004)*.

2.5.6. Domensko-specifično modelovanje (DSM)

Kao i MDA, domensko-specifično modelovanje (eng. *domain-specific modeling*, u daljem tekstu DSM) predstavlja pristup razvoju softvera vođen modelima. DSM pristupom se želi postići u velikoj meri automatizacija softvera što neposredno utiče na povećanje produktivnosti i fleksibilnosti razvoja *(Kelly & Tolvanen, 2008; DSM Forum, 2013)*. U osnovi ovog pristupa je izdizanje nivoa apstrakcije i korišćenje domensko-specifičnih jezika modelovanja (eng. *domain-specific language*, u daljem tekstu DSL) *(Van Deursen, Klint, & Visser, 2000; Fowler, 2010; Ghosh, 2010; Voelter, i drugi, 2013)*, umesto jednog opšteg jezika modelovanja kao što je UML koji MDA standard koristi kao osnovni jezik modelovanja, ali dozvoljava i definisanje domensko-specifičnih jezika pomoću ekstenzija i kastomizacija osnovnog UML jezika *(France & Rumpe, 2007)*. Domensko-specifični jezici obezbeđuju minimalan skup semantički bogatih domensko-specifičnih koncepata koji su bliski domenu problema i čije je korišćenje semantički kontrolisano. Uvođenjem domensko-specifičnih jezika omogućava se modelovanje koje se zasniva na direktnom korišćenju koncepata domena *(Kelly & Tolvanen, 2008)*, odnosno korišćenju semantički bogatih apstrakcija što dovodi do povećanja produktivnosti i efikasnosti *(Greenfield, Short, Cook, & Kent, 2004; Mernik, Heering, & Sloane, 2005; Voelter, i drugi, 2013)*. Pored toga, činjenica da sadrže samo relevantne koncepte, odnosno koncepte bliske konceptima domena problema, kao i da je njihovo korišćenje semantički kontrolisano, čini domensko-specifične jezike jednostavnim za razumevanje domenskim korisnicima *(Mernik, Heering, & Sloane, 2005; France & Rumpe, 2005; Kosar, Oliveira, Mernik, Pereira, Črepinšek, & Henriques, 2010; Fowler, 2010)*

Implementacija domensko-specifičnih jezika ostvarena je korišćenjem domensko-specifičnih generatora (eng. *domain-specific generator*) i domensko-specifičnog aplikacionog okvira (eng. *domain-specific framework*) koji tehnološki podržavaju dati domensko-specifični jezik. MDA standard preporučuje još jedan način implementacije koji se postiže pravilima prevođenja apstraktnijeg domensko-specifičnog jezika u konkretniji. Za razliku od njega u okviru DSM pristupa, uvođenjem aplikacionog okvira tehnološka platforma se zapravo „približava“

domensko-specifičnom jeziku *(Voelter, i drugi, 2013)*. Na slici (slika 14) je prikazana DSM arhitektura.



Slika 14. DSM Arhitektura *(Kelly & Tolvanen, 2008)*

Primena DSM pristupa podrazumeva da se korišćenjem domensko-specifičnih jezika formiraju odgovarajući modeli koji se zatim automatski transformišu u izvršni kôd aplikacionog okvira *(Kelly & Tolvanen, 2008; DSM Forum, 2013)*. Drugim rečima, DSM pristup omogućava da se implementacija automatski generiše na osnovu specifikacije sistema korišćenjem odgovarajućih generatora.

Generatori su takođe domensko-specifični. Njima se ekstrahuju informacije iz modela i transformišu u izvršni kod, odnosno njima je omogućeno mapiranje modela, formiranih korišćenjem domensko-specifičnih jezika, u izvršni kôd. Generisanje kôda je najčešće podržano odgovarajućim aplikacionim okvirom *(Kelly & Tolvanen, 2008)* kojim su obezbeđeni implementacioni koncepti bliski konceptima domena problema. Aplikacionim okvirom se izdiše nivo apstrakcije ciljne platforme *(Bettin, 2004; Voelter, i drugi, 2013)*. Njegovim uvođenjem značajno se povećava semantički nivo koji je implementaciono podržan čime se pojednostavljuje generisani kôd i smanjuje broj koraka u razvoju, koji postaje više automatizovan i samim tim mnogo produktivniji.

Prema *(Kelly & Tolvanen, 2008)*, DSM pristupom je omogućeno generisanje potpunog kôda (eng. *full code generation*). Generiše se kôd koji nije potrebno dodatno modifikovati i koji se može direktno kompajlirati. Uzimajući ovo u obzir modeli se mogu smatrati izvršnim.

3. KLJUČNE KARAKTERISTIKE ETL PROCESA

U cilju dobijanja visoko kvalitetnih strateških informacija u obliku koji je pogodan za donošenje odluka, potrebno je sprovesti veći broj različitih tipova funkcija nad podacima u odgovarajućem redosledu. Tipovi funkcija i redosled u kome se one izvršavaju definisan je softverskim procesom koji je u literaturi poznat pod nazivom ETL proces (eng. *extract-transform-load process*). ETL proces je opšte prihvaćen naziv za procese u domenu sistema skladišta podataka, kojima se podaci ekstrahuju iz izvornih sistema, zatim transformišu u oblik koji je pogodan za analizu i donošenje strateških odluka, i na kraju učitavaju u odgovarajuće skladište podataka.

ETL procesi se smatraju najsloženijim, ali i najznačajnijim u čitavom procesu razvoja sistema skladišta podataka. Način na koji je projektovan i implementiran ovaj proces u velikoj meri utiče na kvalitet dobijenih strateških informacija, a samim tim i na upotrebljivost, odnosno uspeh sistema skladišta podataka.

Funkcije koje je potrebno sprovesti nad podacima, kao i redosled njihovog izvršavanja uslovljen je većim brojem faktora, a pre svega kvalitetom podataka u izvornim sistemima, brojem izvornih sistema, zatim poslovnim pravilima i ograničenjima koja su definisana sistemom skladišta podataka, ali i izabranom arhitekturom skladišta podataka.

Kvalitet izvornih podataka u velikoj meri utiče na aktivnosti koje je potrebno sprovesti. Naime, ukoliko poslovna pravila transakcionih sistema odstupaju od poslovnih pravila koja su definisana u sistemu skladišta podataka, potrebno je sprovesti veći broj funkcija kako bi se podaci korigovali, odnosno prilagodili zahtevima skladišta podataka.

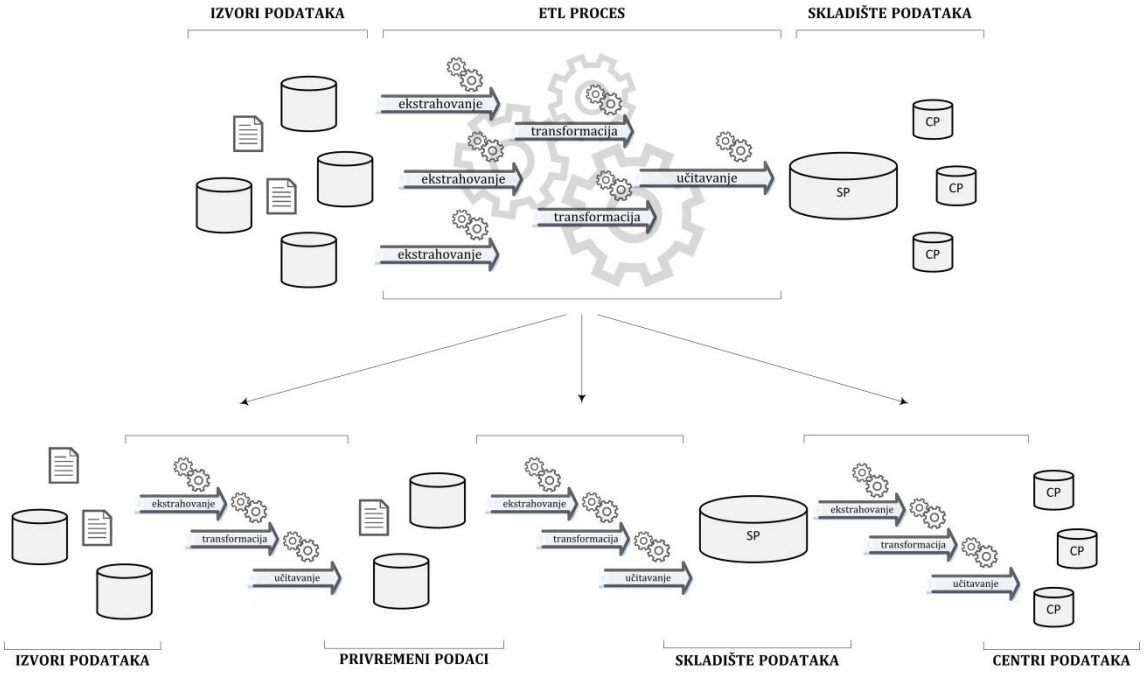
Pored toga, u sistemima skladišta podataka se koristi veći broj različitih izvornih sistema. To mogu biti relacione ili objektne baze podataka, mrežne baze, ali i

različiti sistemi datoteka u kojima su smešteni podaci *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*. Ovi heterogeni sistemi su implementirani na različitim platformama, korišćenjem različitih tehnologija i modela podataka što dodatno povećava složenost aktivnosti koje je potrebno sprovesti.

Isto tako, podaci na svom putu od izvornih sistema do skladišta podataka, odnosno krajnjih korisnika prolaze kroz određene slojeve. Ovi slojevi definisani su izabranom arhitekturom sistema skladišta podataka. Konfiguracija ovih različitih slojeva direktno utiče na strukturu procesa. Na primer, ukoliko je arhitekturom predviđen sloj usaglašenih podataka, ETL procesom se moraju obuhvatiti aktivnosti kojima se ovaj sloj popunjava, ali i aktivnosti kojima se podaci iz ovog sloja dalje transformišu i smeštaju u sloj prezentacionih podataka. Isto tako, na svakom sloju se definiše odgovarajući model podataka koji od arhitekture do arhitekture varira. Na primer, *Kimball* predlaže korišćenje dimenzionalnog modela *(Kimball & Ross, 2002)*, *Inmon* korišćenje relacionog i dimenzionalnog modela *(Inmon W. H., 2005)*, dok *Linsted* predlaže korišćenje data vault i dimenzionalnog modela *(Linstedt & Graziano, 2011)*.

Pored toga, na strukturu procesa utiču i različita pravila transformacije koja proističu iz primene određenog pristupa razvoju sistema skladišta podataka. Naime, noviji pristupi kao što je *Data Vault* pristup, propisuju da se u skladište podataka smeštaju poslovni podaci koji prethodno nisu prečišćeni i transformisani *(Linstedt & Graziano, 2011)*. Ovo podrazumeva da se poslovni podaci ekstrahuju i učitaju u skladište podataka, a da se tek nakon toga transformišu u oblik koji je pogodan za sprovođenje poslovnih analiza (eng. *extract-load-transform ELT*). Na ovaj način su omogućene brojne prednosti, ali su u isto vreme uvedena i različita ograničenja u pogledu izvršavanja procesa transformacije. Na primer, na svom putu do skladišta podataka nad poslovnim podacima se predlažu samo reverzibilne operacije transformacije, kako bi se omogućila rekonstrukcija izvora podataka na osnovu podataka iz skladišta podataka za bilo koji vremenski trenutak. Isto tako, činjenica da se u skladištu podataka smeštaju neizmenjeni poslovni podaci nalaže odloženo sprovođenje njihovih transformacija, čime je

omogućeno sprovođenje različitih poslovnih pravila transformacija bez gubitka poslovnih podataka.



Slika 15. Struktura ETL procesa

Kao što je i prikazano (slika 15), ETL proces se može posmatrati na višem nivou apstrakcije, kao skup aktivnosti kojima se podaci iz jednog sloja podataka ekstrahuju i koriguju, zatim transformišu i konsoliduju i na kraju učitavaju u sledeći sloj podataka *(Adamson, 2010)*. Ova tri funkcionalna koraka se nazivaju ekstrakcija, transformacija i učitavanje i izvršavaju se sekvencijalno.

Pod ekstrakcijom i korekcijom podataka se podrazumeva proces kojim se iz različitih izvora podataka izvlače relevantni podaci i koriguju u skladu sa definisanim ograničenjima. Tokom ovog procesa ekstrahovani podaci se prečišćavaju (eng. *data cleansing*) kako bi se poboljšao njihov kvalitet *(Golfarelli & Rizzi, 2009)*. Razlikuje se statičko ekstrahovanje i inkrementalno ekstrahovanje (Golfarelli & Rizzi, 2009). Statičko ekstrahovanje (inicijalno ekstrahovanje, eng. *initial extraction*) je slučaj kada se skladište podataka popunjava prvi put, dok se pod inkrementalnim ekstrahovanjem (eng. *incremental extraction*;*changed data capture*) podrazumeva ažuriranje skladišta podataka na osnovu promena u podacima izvora. Inkrementalno ekstrahovanje (eng. *changed data capture*) se

može obaviti preko formiranja istorijata promena na osnovu koga se vrši ažuriranje, ili u situacijama kada je moguće izvršiti adaptaciju izvora tako da se odgovarajuće promene prenose u skladište podataka *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*.

Pod transformacijom i konsolidacijom podataka podrazumeva se priprema ekstrahovanih podataka za učitavanje u skladište podataka. Ona podrazumeva konvertovanje ekstrahovanih podataka iz formata izvora u odgovarajući format skladišta podataka *(Golfarelli & Rizzi, 2009)*. Zahteva uspostavljanje mapiranja između sloja izvora podataka i sloja prezentacionih podataka što je veoma složena aktivnost uzimajući u obzir postojanje većeg broja heterogenih izvora *(Golfarelli & Rizzi, 2009)*. Ovaj proces dodatno se komplikuje ako je potrebna integracija više različitih izvora. Rezultat ovog procesa su kolekcije integrisanih podataka koje su spremne za učitavanje u skladište podataka *(Ponniah, 2011)*.

Pod učitavanjem podataka u skladište podataka podrazumevaju se procesi punjenja i osvežavanja skladišta podataka *(Lazarević, Marjanović, Aničić, & Babarogić, 2010)*. Pod procesom punjenja skladišta podataka podrazumeva se inicijalno učitavanje koje se izvršava jednom kada se skladište podataka popunjava prvi put, dok se proces osvežavanja izvršava u planiranim intervalima na osnovu promena u podacima izvora. Proces punjenja se koristi uz tehniku statičkog ekstrahovanja, dok se proces osvežavanja koristi uz tehniku inkrementalnog ekstrahovanja. Osvežavanje se obično sprovodi bez brisanja ili modifikovanja postojećih podataka *(Golfarelli & Rizzi, 2009)*.

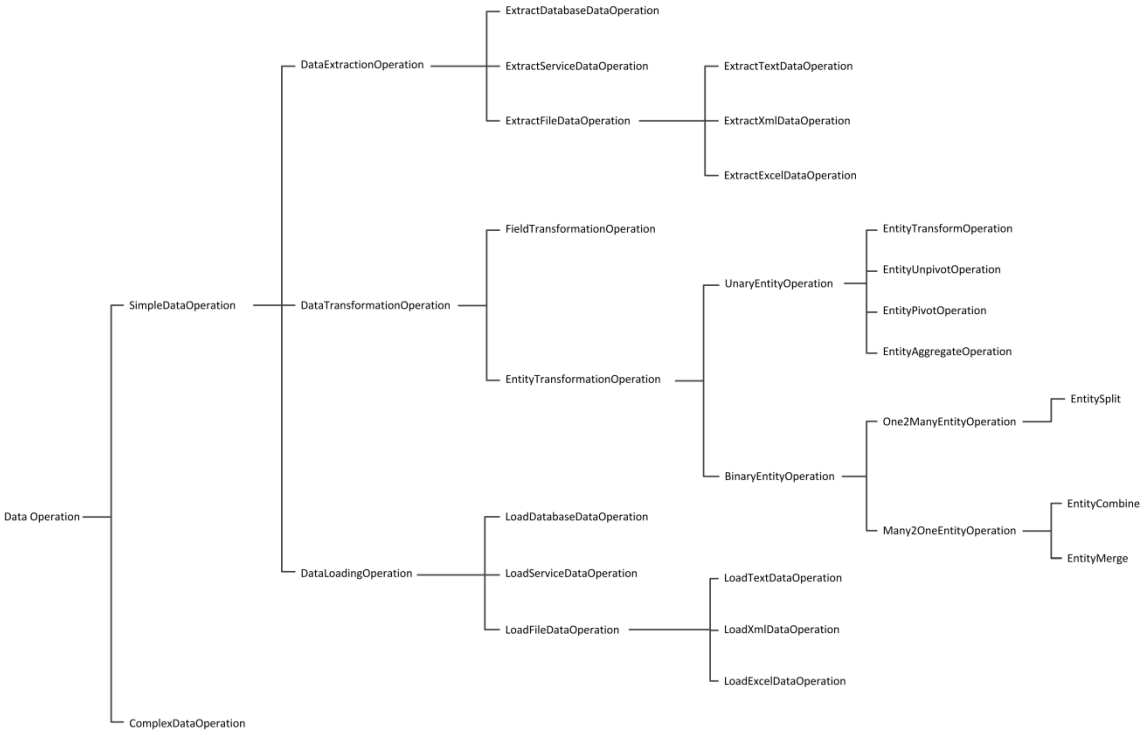
Taksonomija operacija transformacija

Osnovni elementi ETL procesa su odgovarajuće operacije kojima se vrši transformacija podataka. One se mogu podeliti na proste i složene, pri čemu se složene operacije dobijaju kompozicijom prostih operacija transformacija. Ovo jasno govori, da se na najnižem nivou granularnosti nalaze samo proste operacije transformacije. One predstavljaju fundamentalne operacije, odnosno operacije od kojih se gradi ETL proces i kao takve su od posebnog značaja. U skladu sa tim predstavljena je taksonomija operacija transformacija (slika 16). Taksonomijom je

predstavljena hijerarhija ovih operacija. Obuhvaćen je podskup skupa operacija transformacija značajnih za ovu fazu istraživanja.

Kao što je i prikazano (slika 16), operacije transformacije podataka (*DataOperation*) se mogu podeliti na proste (*SimpleDataOperation*) i složene (*ComplexDataOperation*) operacije transformacije, pri čemu složene operacije nastaju kompozicijom prostih operacija transformacija.

Proste operacije transformacije se dalje mogu klasifikovati na: (1) operacije ekstrahovanja podataka, zatim (2) operacije transformisanja podataka i (3) operacije učitavanja podataka. Ove tri identifikovane operacije predstavljaju osnovne operacije kojima se mogu predstaviti sve ostale primitivne operacije ETL procesa.



Slika 16. Taksonomija operacija transformacija

◊ *Operacije ekstrahovanja podataka* (*DataExtractionOperation*) su fundamentalne operacije kojima je omogućena ekstrakcija podataka iz heterogenih izvora podataka. One se dalje, u odnosu na vrstu izvora podataka iz koga se podaci preuzimaju, mogu klasifikovati na: (1) operacije kojima je omogućeno preuzimanje podataka iz baza podataka (*ExtractDatabaseDataOperation*), zatim (2) operacije

kojima je omogućeno preuzimanje podataka iz različitih servisa (*ExtractServiceDataOperation*) i (3) operacije kojima je omogućeno preuzimanje podataka iz datoteka (*ExtractFileDataOperation*). Isto tako, i navedene operacije se dalje mogu klasifikovati, a kao primer se navodi klasifikacija operacija za preuzimanje podataka iz različitih datoteka i to na: operacije za preuzimanje podataka iz tekstualnih datoteka (*ExtractTextDataOperation*), zatim operacije za preuzimanje podataka iz *xml* datoteka (*ExtractXmlDataOperation*) i mnoge druge u zavisnosti tipa datoteka, odnosno formata u kome se podaci čuvaju.

◊ *Operacije transformisanja podataka* (*DataTransformationOperation*) su fundamentalne operacije kojima se realizuje suština ETL procesa. Njima se vrše različite transformacije ekstrahovanih podataka. Podaci koji učestvuju u transformacijama se predstavljaju entitetima, pri čemu se svaki entitet sastoji od više atributa, odnosno polja. Ovo znači da se prilikom transformacije entiteta sprovode i odgovarajuće transformacije polja kao njegovih sastavnih delova. Uzimajući ovo u obzir, operacije transformisanja podataka se mogu klasifikovati na: (1) operacije transformisanja entiteta (*EntityTransformationOperation*) i (2) operacije transformisanja polja entiteta (*FieldTransformationOperation*), pri čemu jedna operacija transformisanja entiteta sadrži više operacija transformisanja polja entiteta.

Dalja klasifikacija operacija transformisanja entiteta se može izvršiti na osnovu strukture ulaza i izlaza operacije. Naime, razlikuju se unarne operacije transformacije (*UnaryEntityOperation*) i binarne operacije transformacije (*BinaryEntityOperation*), uz napomenu da se n-arne operacije mogu predstaviti preko više binarnih operacija *(Vassiliadis, Simitsis, & Baikousi, 2009)*.

Unarnim operacijama se predstavljaju operacije kod kojih se podrazumeva jedan ulaz i jedan izlaz, odnosno jedan ulazni entitet i jedan izlazni entitet, dok se binarnim operacijama predstavljaju operacije kod kojih se podrazumeva dva ulaza i jedan izlaz i obrnuto. Uopšte, binarnim operacijama se vrše različita kombinovanja entiteta, odnosno spajanja i razdvajanja, pa se shodno tome mogu dalje kategorisati na: (1) operacije kojima se vrši transformacija više ulaznih

entiteta u jedan izlazni entitet (*Many2OneEntityOperation*), kao što su operacije *EntityMerge* i *EntityCombine* i (2) operacije kojima se vrši transformacija jednog ulaznog entiteta na više izlaznih entiteta (*One2ManyEntityOperation*) kao što je *EntitySplit*.

Unarne operacije transformacije se dalje mogu analizirati u odnosu na instance entiteta, odnosno kardinalnosti *(Vassiliadis, Simitsis, & Baikousi, 2009)*. Moguće su različite kombinacije: (1) tačno jedna instanca ulaznog entiteta se transformiše u tačno jednu instancu izlaznog entiteta (*EntityTransformOperation*), (2) tačno jedna instanca ulaznog entiteta se transformiše u više instanci izlaznog entiteta (*EntityUnpivotOperation*), (3) više instanci ulaznog entiteta se transformiše u tačno jednu instancu izlaznog entiteta (*EntityPivotOperation*, *EntityAggregateOperation*), itd.

◊ *Operacije učitavanja podataka* su fundamentalne operacije kojima je omogućeno učitavanje transformisanih podataka u različita skladišta podataka. Slično kao i operacije ekstrahovanja podataka, i ove operacije se mogu klasifikovati u odnosu na vrstu izvora podataka u koji se podaci učitavaju, i to na: (1) operacije kojima je omogućeno učitavanje podataka u baze podataka (*LoadDatabaseDataOperation*), zatim (2) operacije kojima je omogućeno učitavanje podataka u servise (*LoadServiceDataOperation*) i (3) operacije kojima je omogućeno učitavanje podataka u datoteke (*LoadFileDataOperation*).

◊ *Složene operacije transformacije* (*ComplexDataOperation*) su operacije kojima se definiše proces obrade podataka, odnosno niz sukcesivnih transformacija podataka koje su neophodne kako bi se odgovarajući podaci izvora podataka ekstrahovali, transformisali i učitali u odgovarajuće skladište podataka. One predstavljaju operacije veće granularnosti koje se definišu kompozicijom prostih operacija transformacija.

4. PREGLED REALIZOVANIH ISTRAŽIVANJA

U ovom poglavlju su razmatrana postojeća rešenja problema formalizacije i automatizacije razvoja ETL procesa. Dat je pregled relevantnih istraživanja sa ciljem da se prikažu dosadašnji rezultati i praksa u oblasti razvoja ETL procesa, a zatim je predstavljena analiza rezultata koji su ovim istraživanjima postignuta. Analizom su razmatrana rešenja problema konceptualizacije i automatizacije ETL procesa i shodno tome navedeni su nedostaci postojećih pristupa i opisani konkretni predlozi za poboljšanja.

4.1. Pregled postojećih pristupa razvoju ETL procesa

Identifikacija postojećih radova u oblasti izvršena je uz pomoć dostupnih servisa i pretraživača, kao što su: Web of Science (*http://apps.webofknowledge.com/*), Scopus (*http://www.scopus.com/*), SpringerLink (*http://link.springer.com/*), ScienceDirect (*http://www.sciencedirect.com/*), ACM digital library (*http://dl.acm.org/*), IEEE digital library (*http://search3.computer.org/*), Google Scholar (*http://scholar.google.com/*), CiteSeer (*http://citeseerx.ist.psu.edu/*). Pretraga radova je izvršena prema ključnim rečima: „*Business Inteligence*“, „*Data Warehouse*“, „*ETL*“, „*ETL Process*“, „*Conceptual model*“, „*Logical model*“, „*Conceptual design*“, „*Logical design*“.

U nastavku je dat hronološki prikaz radova kojima su predstavljena istraživanja za koja se smatra da imaju konkretan doprinos u oblasti razvoja ETL procesa:

◊ (Vassiliadis, Simitsis, & Skiadopoulos, 2002, Novembar): U radu je razmatran problem definisanja ETL aktivnosti i u skladu sa tim je predložena formalna osnova za njihovo konceptualno predstavljanje. Predložen je metamodel kojim je omogućeno konceptualno modelovanje ETL procesa u ranim fazama projekta.

Autori navode da je potrebno obezbediti podršku za modelovanje ETL procesa u ranim fazama projekta. Tokom ranih faza projekta projektanti se bave prikupljanjem zahteva, a paralelno sa tim aktivnostima vrše analizu strukture i sadržaja postojećih izvora podataka i njihovim mapiranjem u opšti model skladišta podataka. Najvažniji rezultat modelovanja ETL procesa jesu mapiranja atributa izvora podataka i atributa tabela skladišta podataka.

U radu je predložen metamodel kojim su definisani koncepti neophodni tokom kreiranja konceptualnih modela ETL procesa i obezbeđena je grafička notacija za njihovo predstavljanje. Autori navode da nisu koristili standardnu UML notaciju zbog činjenice da su atributi uključeni u definiciju elementa kome pripadaju. Na taj način onemogućeno je uspostavljanje veza između atributa što je u ovom pristupu neophodno. Autori navode da je potrebno da se atributi predstave kao elementi modelovanja prvo reda (eng. *first class citizens*).

Predložena je arhitektura koja se sastoji od tri sloja: (1) sloj metamodela (eng. *metamodel layer*), (2) sloj templejta (eng. *template layer*) i (3) sloj šeme (eng. *schema layer*). Prvi sloj, sloj metamodela sadrži minimalan skup konstrukcija (eng. *construct*) koji je prema autorima dovoljan za opisivanje posmatranog problema. Na ovom sloju su definisane sledeće opšte konstrukcije: (1) koncept (eng. *concept*), (2) atribut (eng. *attribute*), (3) transformacija (eng. *transformation*), (4) ETL ograničenje (eng. *ETL constraint*) i (5) veza (eng. *relationship*). Instance ovih konstrukcija se koriste na sloju šema tokom modelovanja konkretnog ETL procesa. Da bi obezbedili specifične konstrukcije, autori su predvideli i sloj templejta na kome se definišu različite konstrukcije u vidu specijalizacija konstrukcija koje su date na sloju metamodela. Na primer, autori uvode konstrukciju *fact table* koja je specijalizacija konstrukcije *concept* sa sloja metamodela. Instanca ove konstrukcije se dalje može koristiti na sloju šema.

Autori navode da im je cilj bio da obezbede konceptualni model kojim će biti moguće predstaviti semantiku ETL procesa, a ne tehničko rešenje za implementaciju ovog procesa. Ovo je po njima veoma važno, jer u ranim fazama projekta treba identifikovati ograničenja i mogućnosti bez ulaženja u detalje

implementacije. Za predloženi konceptualni model smatraju da je: (1) prilagođen za predstavljanje veza između atributa i odgovarajućih ETL aktivnosti u ranim fazama projekta, (2) obogaćen skupom često korišćenih ETL aktivnosti, kao što je dodela surogat ključa i mnoge druge, kao i da je (3) konstruisan na način koji omogućava prilagođavanja i proširenja tako da je projektant u mogućnosti da ga obogati sopstvenim šablonima za ETL aktivnosti.

◊ (Simitsis & Vassiliadis, 2003): U radu se bavi problemom konceptualnog modelovanja ETL procesa. Prikazano istraživanje predstavlja nastavak istraživanja koje su autori ranije predstavili *(Vassiliadis, Simitsis, & Skiadopoulos, 2002, Novembar)*, sa ciljem da se postojeći pristup nadogradi u smislu propisivanja procesa, odnosno redosleda aktivnosti koje je potrebno sprovesti tokom konceptualnog modelovanja ETL procesa.

Autori predlažu sledeće korake u procesu konceptualnog modelovanja ETL aktivnosti: (1) identifikacija relevantnih izvora podataka, (2) određivanje prioriteta identifikovanih izvora podataka u smislu kandidata i aktivnih kandidata, (3) definisanje mapiranja između atributa i (4) označavanje dijagrama sa izvršnim ograničenjima, kao što su praćenje istorije izvršavanja, obrada grešaka, itd.

◊ (Trujillo & Luján-Mora, 2003): U radu je predložen pristup konceptualnom modelovanju ETL procesa koji se bazira na standardnom jeziku modelovanja (UML). Ovim pristupom obezbeđen je mehanizam za specifikaciju opštih operacija ETL procesa, kao što su integracija različitih izvora podataka, transformacija izvornih u ciljne atribute, generisanje surogat ključeva, itd. Kao prednost ovog pristupa navodi se mogućnost jednostavne integracije definisanog ETL procesa sa konceptualnom šemom skladišta podataka.

S obzirom da je UML opšte prihvaćen standard, autori smatraju da njihov pristup zahteva minimum napora kako bi se savladali novi dijagrami za modelovanje ETL procesa. U osnovi, ovim pristupom ETL procesi se modeluju korišćenjem UML dijagrama klasa (eng. *UML class diagram*). ETL proces se sastoji od UML paketa

(eng. *UML package*), koji omogućavaju dekompoziciju ETL procesa na različite logičke nivoe. Svaki ETL mehanizam se predstavlja pomoću UML klase (eng. *UML class*) koja se označava pomoću odgovarajućeg stereotipa (eng. *stereotype*). Za svaki od predstavljanih mehanizama obezbeđena je odgovarajuća grafička reprezentacija. Za povezivanje ovih mehanizama koristi se UML koncept zavisnosti (eng. *dependecy*), koji se predstavlja usmerenom isprekidanom linijom. Koncept UML beleške (eng. *UML note*) se može pridružiti bilo kom ETL mehanizmu sa ciljem da se: (1) objasni funkcionisanje mehanizma i (2) definišu mapiranja izvornih u ciljne atribute. Povezivanje je omogućeno isprekidanom linijom koja nije usmerena. Po pitanju sadržaja ovih elemenata ne postoje restrikcije, već samo preporuke kako bi se omogućio veći stepen fleksibilnosti. Na primer, može se koristiti prirodni jezik kako bi se obezbedio opšti opis, ili čak određeni programski jezik. Izbor je na projektantu.

Da bi omogućili lakše savladavanje složenosti, autori su ovim pristupom predvideli mogućnost dekompozicije složenog ETL procesa na skup jednostavnih procesa. Na ovaj način omogućeno je lakše projektovanje i održavanje ETL procesa. Pored toga, sa ciljem da se smanji kompleksnost predloženog pristupa, autori su predložili manji broj ETL mehanizama, kao što su *agregation* kojim se podaci agregiraju prema zadatom kriterijumu, *filter* kojim se vrši selekcija i verifikacija podataka, *surrogate* kojim se generišu surogat ključevi, itd. Jedan od predloženih mehanizama je i *wrapper*, kojim se definiše transformacija originalne strukture podataka u formu zapisa (eng. *record*) koji se sastoje od atributa. Ova transformacija je neophodna, jer predstavljeni mehanizmi obrađuju podatke u formi zapisa.

◊ (Luján-Mora, Vassiliadis, & Trujillo, 2004): U radu je predstavljen pristup projektovanju ETL procesa, koji se zasniva na modelovanju veza, odnosno mapiranja između izvora i cilja na različitom nivou granularnosti. Ovo mapiranje obuhvata mapiranje na nivou baza podataka, zatim na nivou tabela i na kraju na najnižem nivou granularnosti između atributa. Predloženi pristup se zasniva na korišćenju jedinstvenog jezika modelovanja (*UML*) koji su autori proširili

uvođenjem novih koncepata koji su specifični za posmatrani problem mapiranja. Naime, za problem mapiranja autori su predvideli novu vrstu dijagrama koji su nazvali *dijagram mapiranja podataka* (eng. *data mapping diagram*), a koja predstavlja proširenje osnovnog dijagrama klasa (eng. *UML class diagram*).

Jedan od problema koji je u radu identifikovan je nemogućnost formalnog predstavljanja mapiranja između atributa korišćenjem koncepata koji su ponuđeni standardnim jezikom modelovanja. Atributi su uključeni u definiciju elementa kome pripadaju čime je onemogućeno uspostavljanje veza između njih. Da bi rešili navedeni problem, autori navode da je potrebno da se atributi predstave kao elementi modelovanja prvog reda (eng. *first-class modeling element*; *first-class citizen*), kao što su klase. U tu svrhu, izvršeno je proširenje izabranog jezika modelovanja kroz definisanje stereotipova *Attribute* i *Contains*. Ovi stereotipovi se koriste na dijagramima mapiranja podataka, pri čemu se stereotipom *Contains* označavaju veze između klasa i njenih atributa sa ciljem da se predstavi pripadnost atributa određenoj klasi, dok se stereotipom *Attribute* označavaju klase kojima se predstavljaju atributi.

S obzirom da dijagrami mapiranja podataka mogu biti veoma kompleksni, pristupom je predviđena mogućnost njihovog organizovanja na različitim nivoima uz korišćenje UML paketa (eng. *UML package*). Pristupom su predviđena četiri nivoa: nivo baza podataka (eng. *database level – level 0*), nivo toka podataka (eng. *dataflow level – level 1*), zatim nivo tabela (eng. *table level – level 2*) i na kraju nivo atributa (eng. *attribute level – level 3*). Najopštiji nivo je nivo baza podataka, dok svaki sledeći predstavlja viši nivo detalja. Za modelovanje na nivou baza podataka i tokova podataka koriste se osnovni UML koncepti, dok se na nivou tabela i nivou atributa koriste koncepti kojima je UML proširen.

◊ (Vassiliadis, Simitsis, Georgantas, Terrovitis, & Skiadopoulos, 2005): U radu je obrađen problem logičkog modelovanja ETL procesa. Predložen je frejmvork (eng. *framework*) koji se bazira na posebno razvijenom metamodelu, sa ciljem da se projektantima pruži podrška tokom logičkog modelovanja ETL procesa. Pored frejmvorka dat je i prikaz grafičkog alata „Arktos II“ u kome su autori

implementirali predloženi frejmvork *(Vassiliadis, Vagena, Skiadopoulos, Karayannidis, & Sellis, 2001)*. Rad predstavlja nastavak istraživanja koje je inicijalno predstavljeno u radu *(Vassiliadis, Simitsis, Georgantas, & Terrovitis, 2003)*.

U radu je predstavljen metamodel koji je prilagođen za definisanje ETL aktivnosti. Pored toga definisan je i deklarativan programski jezik (eng. *database programing language LDL*) kojim je omogućeno predstavljanje semantike aktivnosti. Autori navode da je definisani metamodel dovoljno opšt tako da je moguće predstaviti bilo koju ETL aktivnost. Međutim, kako bi omogućili veću produktivnost i fleksibilnost, pristupom su predviđeni i šabloni (eng *templates*) kojima se osnovne konstrukcije predloženog metamodela specijalizuju. Na ovaj način definisane su tipizirane konstrukcije kojima se predstavljaju najčešće korišćene ETL aktivnosti.

Autori navode da se semantika ETL procesa predstavlja kao kombinacija sekvence izvršavanja aktivnosti i tokova podataka. Tokovima podataka se definiše šta svaka aktivnost radi, dok se sekvencama izvršavanja navodi redosled u kome se aktivnosti izvršavaju. U radu su razmatrani samo tokovi podataka, za čije modelovanje autori predlažu korišćenje grafa (eng. *architecture graph*). Predloženi graf omogućava objedinjavanje svih aktivnosti i različitih skladišta podataka koji su neophodni za opisivanje ETL scenarija *(Vassiliadis, Simitsis, & Skiadopoulos, 2002. Maj)*. Autori navode da graf može biti veoma složen što otežava rad projektanta, pa definišu različite nivoe na kome se ETL proces može posmatrati i to: (1) nivo aktivnosti, (2) nivo šema i (3) nivo atributa *(Simitsis, Vassiliadis, Terrovitis, & Skiadopoulos, 2005)*.

Predložena je arhitektura koja se sastoji od tri sloja: (1) sloj metamodela (eng. *metamodel layer*), (2) sloj templejta (eng. *template layer*) i (3) sloj šeme (eng. *schema layer*). Prvi sloj, sloj metamodela sadrži minimalan skup konstrukcija (eng. *construct*) koji je prema autorima dovoljan za opisivanje posmatranog problema. Na ovom sloju su definisane sledeće opšte konstrukcije: (1) tip podatka (eng. *datatype*), (2) funkcija (eng. *function*), (3) elementarna aktivnost (eng. *elementary activity*), (4) skup rekorda (eng. *recordset*) i (5) veza (eng. *relationship*). Instance ovih konstrukcija se koriste na sloju šema tokom modelovanja konkretnog ETL

procesa. Da bi obezbedili specifične konstrukcije, autori su predvideli i sloj templejta na kome se definišu različite konstrukcije u vidu specijalizacija konstrukcija koje su date na sloju metamodela. Na primer, autori uvode konstrukciju *SK assignment* koja je specijalizacija konstrukcije *elementary activity* sa sloja metamodela. Instanca ove konstrukcije se dalje može koristiti na sloju šema.

◊ (Simitsis & Vassiliadis, 2008; Simitsis, 2005): U radovima se bavi problemom mapiranja konceptualnih i logičkih modela sa ciljem da se omogući automatizacija procesa transformacije konceptualnih u logičke modele ETL procesa. Predstavljeno istraživanje se oslanja na rezultate prethodnih istraživanja autora koja se tiču konceptualnog i logičkog modelovanja ETL procesa *(Vassiliadis, Simitsis, & Skiadopoulos, 2002. Maj; Simitsis & Vassiliadis, 2003; Vassiliadis, Simitsis, Georgantas, Terrovitis, & Skiadopoulos, 2005)*. Autori navode da se navedenim problemom niko pre njih nije bavio i da shodno tome predloženi pristup predstavlja prvo rešenje u oblasti. Isto tako, navode da je predloženom metodom omogućena poluautomatska transformacija.

Autori kao prvi korak u rešavanju navedenog problema navode identifikaciju načina na koji se entiteti konceptualnog modela mapiraju u entitete logičkog modela, a zatim kao drugi korak određivanje redosleda izvršavanja aktivnosti korišćenjem informacija koje su dobijene iz konceptualnog modela. U skladu sa ovim informacija predložen je metod prelaska (poluautomatska transformacije) iz konceptualnog u logički model.

Mapiranjem konceptualnog i logičkog modela zapravo se uspostavlja odnos između entiteta konceptualnog i entiteta logičkog modela. Za svaki entitet konceptualnog modela obezbeđen je odgovarajući entitet logičkog modela i definisana je njihova transformacija (na primer, entitet *transformation* se transformiše u entitet *activity*). Za određivanje redosleda izvršavanja aktivnosti, autori predlažu grupisanje transformacija, koje su definisane u konceptualnom modelu, u odgovarajuće faze. Svakom fazom su obuhvaćene one transformacije koje su ekvivalentne u pogledu redosleda izvršavanja.

◊ (Muñoz, Mazón, Pardillo, & Trujillo, 2008): Autori navode da ETL procesi igraju važnu ulogu u sistemima skladišta podataka, ali i da njihov razvoj generiše najveći deo troškova u celokupnom razvoju sistema skladišta podataka. Kao osnovni razlog navode da se projektovanje ovih procesa ne uzima u obzir tokom ranih faza razvoja sistema skladišta podataka. Da bi prevazišli ove probleme predlažu kreiranje konceptualnih modela ovih procesa na samim počecima projekta. Autori predlažu da se ovim modelima opisuje dinamički aspekt ETL procesa, odnosno sekvence aktivnosti koje su obuhvaćene ovim procesima. Za njihovo predstavljanje propisuju jedinstveni jezik modelovanja (*UML*), odnosno korišćenje dijagrama aktivnosti (eng. *activity diagrams*).

Autori navode da dijagrami aktivnosti omogućavaju modelovanje dinamike sistema i da se kao takvi mogu koristiti za modelovanje ETL procesa. U skladu sa tim, predlažu skup parametrizovanih elemenata, u vidu templejta (eng. *template*), koji se mogu koristiti da bi se opisalo ponašanje procesa i omogućila jednostavna integracija sa konceptualnom šemom skladišta podataka.

Kao osnovne prednosti njihovog pristupa, autori navode sledeće: (1) vizuelno modelovanje kompleksnih procesa kao i interoperabilnost sa drugim dijagramima, (2) obezbeđivanje skupa templejta koji omogućavaju opis i ponovno korišćenje opštih koncepata ETL procesa i (3) omogućavanje integracije u globalni i integrisani pristup razvoju skladišta podataka.

◊ (Muñoz, Mazón, & Trujillo, 2009): Autori navode da je u literaturi predloženo nekoliko pristupa konceptualnom modelovanju ETL procesa, ali da ovi pristupi ne predlažu mehanizme koji će se koristiti u fazi automatskog generisanja koda kojim će se izvršavati ETL proces na određenoj platformi. Autori navode da se uvođenjem ovih automatskih mehanizama smanjuje mogućnost nastajanja grešaka, ali i da se smanjuje potrebno vreme za implementaciju kompleksnih ETL procesa.

Za projektovanje ETL procesa, autori predlažu pristup vođen modelima koji se bazira na MDA standardima. Konceptualni model ETL procesa se definiše kao

platformski nezavisan model (eng. *platform independent model PIM*), odnosno kao model koji ne zavisi od korišćene implementacione tehnologije. Za definisanje konceptualnih modela autori predlažu pristup koji se bazira na jedinstvenom jeziku modelovanja (UML), odnosno korišćenju dijagrama aktivnosti *(Muñoz, Mazón, Pardillo, & Trujillo, 2008)*. Ovako definisani modeli se mogu transformisati u različite platformsko specifične modele (neg. *platform specific model PSM*), odnosno u modele kojima se ETL proces predstavlja u određenom tehnološkom okruženju. Transformacije između ovih modela se formalno uspostavljaju korišćenjem transformacionog jezika (eng. *query/view/transformation - QVT*) koji za definisanje transformacija nudi tekstualnu i grafičku notaciju.

Autori navode da, u domenu skladišta podataka, konceptualno modelovanje vodi ka platformski nezavisnim modelima, dok modelovanje platformski zavisnih modela vodi ka logičkim modelima. U skladi sa tim, svaki logički model se bazira na resursima određene tehnologije.

Kao prednost predloženog pristupa navodi se automatsko dobijanje logičkog modela ETL procesa na osnovu konceptualnog modela, čime se povećava stepen produktivnosti i smanjuje potrebno vreme i troškovi razvoja. Autori su, kao primer, u radu dali mapiranje konceptualnog modela ETL procesa u logički model baziran na metamodelu za *Oracle Warehouse Builder* (Griesemer, 2009).

◊ (El Akkaoui, Zimányi, Mazón, & Trujillo, 2011): Autori navode da je jedan od osnovnih problema u projektovanju ETL procesa razmatranje određene tehnologije još od samog početka razvojnog procesa. Na taj način onemogućena je razmena i ponovno korišćenje najboljih praksi na različitim projektima koji su implementirani korišćenjem različitih tehnologija. Autori pokušavaju da harminizuju razvoj ETL procesa kroz predlog opšte i integrisane razvojne strategije, odnosno frejmvorka (eng. *framework*) za modelom vođen razvoj ETL procesa.

Kao osnovne prednosti navode: (1) korišćenje platformski nezavisnih modela za unificirano projektovanje ETL procesa koje se bazira na standardu za modelovanje

poslovnih procesa (eng. *business process model and notation BPMN*) i (2) automatska transformacija ovih modela u specifičan kôd kojim se ETL proces izvršava na konkretnoj platformi.

Za predstavljanje platformski nezavisnih modela autori predlažu BPMN4ETL koji se bazira na BPMN standardu. Na osnovu ovako definisanih modela moguće je dobiti platformski specifične implementacije koje su prilagođene određenim tehnologijama kao što su *Oracle Warehouse Builder* ili *SQL Server Integration Services*. Za kreiranje i upravljanje ETL procesima, kao dodatak grafičkim jezicima, ove tehnologije obezbeđuju i korišćenje odgovarajućih programskih jezika. Za transformaciju platformski nezavisnih modela u platformsko specifičan kôd koriste se model u tekst transformacije (eng. *model-to-text M2T transformation*).

Autori navode da se doprinos njihovog rada ogleda u definisanju platformski nezavisnog metamodela za projektovanje ETL procesa koji omogućava projektantima da se fokusiraju na semantiku ETL procesa umesto na rešavanje implementacionih problema. Korišćenjem ovog metamodela omogućeno je ponovno korišćenje, jer se definisani ETL proces može implementirati na različitim tehnologijama. Pored ovog metamodela, autori kao doprinos navode i skup M2T transformacija kojima se automatski dobija kôd iz ovih modela.

◊ (Vucković, Petrović, Turajlić, & Stanojević, 2012): Autori navode da je projektovanje ETL procesa jedna od najsloženijih i vremenski najzahtevnijih aktivnosti u procesu razvoja sistema skladišta podataka i da je stoga potrebno obezbediti odgovarajuću podršku projektantima. Da bi se omogućilo savladavanje složenosti tokom projektovanja ETL procesa, u radu se predlaže izdizanje nivoa apstrakcije kroz korišćenje odgovarajućih platformski nezavisnih modela kojima se zanemaruju detalji implementacije, kao što je na primer redosled izvršavanja operacija. Autori smatraju da projektanti treba da se fokusiraju na semantiku ETL procesa, a ne na njegovu implementaciju.

Specifikacija ETL procesa vrši se na višem nivou apstrakcije pomoću odgovarajućih apstraktnih operacija. Ovim apstraktnim operacijama predstavlja se semantika

korespondencija između elemenata modela koji učestvuju u transformaciji. Predloženi pristup baziran je na MDD principima i shodno tome predložen je odgovarajući metamodel mapiranja kojim su definisani različiti tipovi apstraktnih operacija koje su karakteristične za ETL proces. Definisani metamodel dobija se proširenjem postojećeg weaving metamodela *(Didonet, Fabro, Bézivin, & Valduriez, 2006)* na način koji je ranije predložen *(Aničić, Nešković, Vučković, & Cvetković, 2012)*.

Autori navode da se osnovni doprinos istraživanja ogleda u uvođenju formalne specifikacije semantike transformacija čime se obezbeđuje statički pogled na proces transformacije, a koji je proširiv i lako razumljiv od strane projektanata. Takođe, navode da je predloženi pristup platformski nezavisan pa se može koristiti u kombinaciji sa drugim pristupima koji se bave specifikacijom dinamičkog aspekta transformacija.

◊ (El Akkaoui, Mazón, Vaisman, & Zimányi, 2012): Autori navode da su za razvoj sistema skladišta podataka neophodni ETL alati koji će omogućiti prilagođavanje konstantnim promenama, odnosno jednostavno kreiranje i modifikaciju izvršnog kôda. Prema autorima, ovo je moguće postići jedino ako se ETL procesi dovedu u vezu sa poslovnim procesima organizacije, čime bi se omogućila jednostavna identifikacija potrebnih podataka, kao i način i vreme kada ti podaci treba da se učitaju u skladište podataka. U tu svrhu, autori propisuju korišćenje de facto standardnog jezika za modelovanje poslovnih procesa (eng. *business process model and notation BPMN*) koji je predložio OMG. U radu je predloženo proširenje ovog standarda, odnosno BPMN metamodela, kako bi se omogućilo konceptualno modelovanje ETL procesa.

Da bi omogućili povezivanje koncepata ETL procesa i koncepata poslovnog procesa, autori su prvo izvršili klasifikaciju ETL procesa, odnosno identifikaciju relevantnih koncepata, a zatim su u skladu sa datom klasifikacijom, izvršili proširivanje BPMN metamodela. Klasifikacijom su obuhvaćena dva pogleda. Prvi pogled se tiče orkestracije procesa (eng. *control process view*), odnosno omogućava grananje i sinhronizaciju tokova kao i obradu grešaka. Drugi pogled se bavi

transformacijom podataka (eng. *data process view*), odnosno predstavljanjem ulaznih i izlaznih informacija za svaku aktivnost procesa.

Autori smatraju da se navedenim pristupom omogućava modelovanje ETL procesa analogno poslovnim procesima, čime je omogućeno povezivanje ETL procesa i poslovnih procesa organizacije. Pored toga, navode da je predloženim pristupom omogućeno fokusiranje na semantiku procesa umesto na njegovu implementaciju, kao i da se dobijeni konceptualni modeli mogu jednostavno prevesti u odgovarajuće logičke modele.

4.2. Analiza postojećih pristupa razvoju ETL procesa

Formalizacija i automatizacija razvoja ETL procesa je predmet velikog broja istraživanja. Trenutno vodeći pristup razvoju softvera je razvoj vođen modelima (MDD). Osnovni cilj koji se želi postići ovim pristupom je automatizacija razvoja softvera kako bi se povećala produktivnost i efikasnost, zatim smanjili troškovi i vreme potrebno za razvoj i održavanje, ali i povećao kvalitet i fleksibilnost dobijenog softverskog rešenja. Ovim pristupom promoviše se korišćenje apstrakcije kao osnovne tehnike u savladavanju složenosti. MDD se zasniva na premisi da najvažniji artefakti koji se kreiraju tokom razvoja softvera nije softverski kôd, već modeli kojima se predstavlja znanje o sistemu koji se razvija. Drugim rečima, u MDD, modeli su primarni softverski artefakti i proces razvoja je automatizovan korišćenjem odgovarajućih transformacija modela koje rezultuju konkretnom implementacijom, tj. izvršnim kôdom. Potrebno je naglasiti da, u cilju omogućavanja automatskih transformacija modela, modeli moraju biti formalno iskazani. Uzimajući u obzir kompleksnost ETL procesa i problema u vezi njihovog razvoja, može se zaključiti da softverska rešenja koja se bave problemom ETL procesa treba da se razvijaju u skladu sa MDD pristupom. S toga je u nastavku izvršena analiza najrelevantnijih pristupa predloženih za formalnu specifikaciju ETL procesa kao i analiza različitih pristupa za automatizovanu implementaciju takvih specifikacija. Predstavljena analiza je publikovana u *(Turajlić, Petrović, & Vučković, 2014)*.

Specifikacija ETL procesa

Specifikacija ETL procesa predstavlja prvi korak u procesu razvoja ETL sistema. Osnovni cilj je da se steknu neophodna znanja o posmatranom sistemu i da se identifikuje šta buduće softversko rešenje treba da omogući, odnosno koje su sve funkcionalnosti u celokupnom procesu neophodne. Modelom se predstavlja znanje, tj. semantika sistema koji se razvija. S obzirom da domenski eksperti imaju neophodno iskustvo i znanje o posmatranom domenu, njihovo aktivno učešće je od presudnog značaja *(Reeves, 2009)*. Nerazumevanje problema koji se rešava i loša komunikacija su najčešći uzročnici neuspešnih projekata *(Fowler, 2010)*. Shodno tome, jezici modelovanja koji se koriste za specifikaciju ETL procesa treba da budu razumljivi domenskim ekspertima. Pre svega, treba da se koriste koncepti koji su specifični za određeni domen. Dodatno, jezici modelovanja treba da budu što je moguće jednostavniji, (tj. da sadrže minimalan broj koncepata - samo neophodne koncepte), ali u isto vreme i semantički bogati kako bi se omogućila specifikacija različitih aspekata ETL procesa na odgovarajućem nivou apstrakcije. Konačno, ukoliko je cilj koji se želi postići jezicima modelovanja ispravno predstavljanje domenskih koncepata, uključujući i njihovu semantiku, oni takođe treba da uključuju i pravila domena kako bi se obezbedilo ispravno korišćenje koncepata. Ova pravila treba da su uključena u jezike modelovanja kako bi se sprečile strukturne i semantičke greške.

Dva karakteristična pristupa za realizaciju MDD principa se predlažu. Primarno, razlika između ova dva pristupa se ogleda u jezicima koji se koriste za formiranje modela. Dok se jednim predlaže korišćenje opštih jezika modelovanja (kao što je UML) i njihovo proširivanje, drugim se predlaže korišćenje specifično razvijenih jezika, odnosno domensko-specifičnih jezika *(France & Rumpe, 2007)*. Uopšte, dosadašnja istraživanja u oblasti razvoja ETL procesa se mogu klasifikovati u odnosu na korišćene jezike modelovanja.

Bazirajući se na premisi da se ETL proces može posmatrati kao specifična vrsta poslovnog procesa i naglašavajući potrebu za standardizacijom, za modelovanje ETL procesa se predlaže korišćenje opštih jezika modelovanja kao što su UML i

BPMN. Ovi jezici su proširivani kako bi se uključili koncepti specifični za domen ETL procesa. Preciznije rečeno, proširenje UML-a je predloženo u *(Trujillo & Luján-Mora, 2003; Luján-Mora, Vassiliadis, & Trujillo, 2004; Muñoz, Mazón, Pardillo, & Trujillo, 2008; Muñoz, Mazón, & Trujillo, 2009)*, dok je proširenje BPMN-a predloženo u *(El Akkaoui & Zimányi, 2009; El Akkaoui, Zimányi, Mazón, & Trujillo, 2011; El Akkaoui, Mazón, Vaisman, & Zimányi, 2012)*. Međutim, s obzirom da su opšti jezici modelovanja predviđeni za opisivanje različitih aspekata bilo kog poslovnog procesa u bilo kom domenu (kako bi obezbedili standardizaciju), oni uključuju veliki broj domenski-neutralnih koncepata koji su definisani na niskom nivou apstrakcije. Prema *(Kelly & Tolvanen, 2008)*, opšti jezici modelovanja ne izdižu nivo apstrakcije iznad nivoa kôda. Kompleksnost ovih jezika modelovanja zajedno sa činjenicom da su isuviše tehnički orijentisani kako bi ih usvojili domenski eksperti vodi ka brojnim problemima u vezi sa njihovim prihvatanjem i korišćenjem. Pored toga, s obzirom da ne sadrže znanje o određenom domenu, različiti koncepti se mogu koristiti i povezivati bez obzira na pravila koja važe u posmatranom domenu *(Kelly & Tolvanen, 2008)*. Drugim rečima, odgovornost je na projektantu. Projektant mora znati semantička pravila i mora da obezbedi da su ona ispunjena. Pored toga, proširenja ovih jezika modelovanja doprinose povećanju njihove kompleksnosti pri čemu većina navedenih nedostataka i dalje ostaje. Samo proširenje ovih jezika modelovanja zahteva velika znanja i poznavanje osnovnih koncepata kako bi se pravilno identifikovali koncepti koje je potrebno specijalizovati *(France & Rumpe, 2007)*.

Korišćenje domensko-specifičnih jezika je predloženo u (Vassiliadis, Simitsis, & Skiadopoulos, 2002, Maj; Vassiliadis, Simitsis, & Skiadopoulos, 2002, Novembar; Vassiliadis, Simitsis, Georgantas, & Terrovitis, 2003; Simitsis & Vassiliadis, 2003; Vassiliadis, Simitsis, Georgantas, Terrovitis, & Skiadopoulos, 2005; Simitsis, Vassiliadis, Terrovitis, & Skiadopoulos, 2005; Simitsis, 2005; Simitsis & Vassiliadis, 2008). Osnovna prednost domensko-specifičnih jezika u odnosu na opšte jezike modelovanja, se ogleda u činjenici da omogućavaju izdizanje nivoa apstrakcije iznad programskih jezika i njihovih apstrakcija. Specifikacija rešenja je data jezicima koji direktno koriste koncepte i pravila koja važe u određenom domenu

problema *(Kelly & Tolvanen, 2008)*. Uopšte, cilj koji se želi postići domensko-specifičnim jezicima je da se obezbedi minimalni skup domensko-specifičnih koncepata sa jasnom i precizno definisanom semantikom, zajedno sa skupom pravila kojima se kontroliše njihovo korišćenje. S obzirom da dozvoljavaju uključivanje domenskih pravila (u formi ograničenja) i sintaksa i semantika koncepata se može kontrolisati čime se nekorektne i nepotpune specifikacije mogu sprečiti. U poređenju sa opštim jezicima modelovanja, domensko-specifični jezici omogućavaju preciznu i nedvosmislenu specifikaciju problema, a u isto vreme su jasniji i jednostavniji za korišćenje domenskim ekspertima *(France & Rumpe, 2005)* s obzirom da ne uključuju nepotrebne koncepte opšte namene. Dodatno, upotreba ovakvih jezika olakšava komunikaciju čime se promoviše timski rad koji je jedan od osnovnih principa agilnih pristupa razvoju softvera.

S obzirom da domensko-specifični jezici omogućavaju formalizaciju semantički bogatih apstrakcija (koje obuhvataju postojeće znanje i iskustvo u domenu ELT procesa) oni su prikladniji za formalnu specifikaciju ETL procesa.

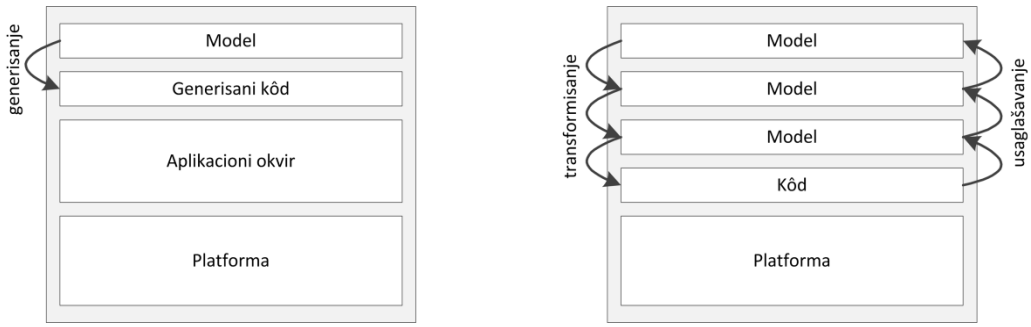
Sa druge strane, uzimajući u obzir kompleksnost ETL procesa, očigledno je da različiti aspekti ETL procesa treba da se modeluju posebno, jer bi u suprotnom specifikacija bila veoma složena i teška za razumevanje. Iako su takvi pristupi predloženi, potrebno je naglasiti da je u većini ovih pristupa fokus na određenom aspektu ETL procesa. Naime u *(Vassiliadis, Simitsis, & Skiadopoulos, 2002, Novembar; Simitsis & Vassiliadis, 2003)* predložen je pristup modelovanju ETL procesa (tj. pristup za specifikaciju mapiranja i transformacija neophodnih u ETL procesu). Međutim nije omogućena specifikacija redosleda izvršavanja transformacija. Dalje, u *(Vassiliadis, Simitsis, & Skiadopoulos, 2002. Maj; Vassiliadis, Simitsis, Georgantas, & Terrovitis, 2003; Vassiliadis, Simitsis, Georgantas, Terrovitis, & Skiadopoulos, 2005; Simitsis, Vassiliadis, Terrovitis, & Skiadopoulos, 2005)* autori se bave logičkim modelovanjem ETL procesa kako bi se opisali tokovi podataka (eng. *data flow*). Autori navode da se semantika ETL procesa definiše preko toka podataka (tj. šta svaka aktivnost radi) i sekvence izvršavanja (tj. redosled i kombinacija aktivnosti). Kao rezultat, u *(Simitsis, 2005; Simitsis & Vassiliadis, 2008)* se predlaže poluautomatska tranzicija iz konceptualnog u logički model, zajedno sa

metodom za određivanje ispravnog redosleda izvršavanja aktivnosti (koristeći informacije koje su preuzete iz konceptualnog modela). Pristup predložen u *(Trujillo & Luján-Mora, 2003)* takođe se bavi statičkim aspektom ETL procesa. Pristup je dalje proširen *(Luján-Mora, Vassiliadis, & Trujillo, 2004)* kako bi se omogućilo praćenje toka podataka na različitim nivoima detalja. U *(Muñoz, Mazón, Pardillo, & Trujillo, 2008; Muñoz, Mazón, & Trujillo, 2009)* autori su prepoznali potrebu za modelovanjem dinamičkog aspekta ETL procesa i stoga predlažu pristup u kome se elementi ETL procesa specificiraju na najvišem nivou apstrakcije dok se sekvenca aktivnosti ETL procesa (eng. *control flow*) specificira na nižim nivoima apstrakcije. Konačno u *(El Akkaoui & Zimányi, 2009; El Akkaoui, Zimányi, Mazón, & Trujillo, 2011; El Akkaoui, Mazón, Vaisman, & Zimányi, 2012)* autori predlažu pristup kojim je omogućeno modelovanje i toka podataka i sekvence izvršavanja. Međutim, korišćenje jednog jezika modelovanja (bilo da je prošireni GPML ili DSL) nije odgovarajuće, s obzirom da se na taj način uključuje veliki broj različitih koncepata. Iz tog razloga se predlaže da se svaki aspekt ETL procesa modeluje posebnim jezikom, koji bi uključio samo one koncepte koji su relevantni za određeni aspekt, čime bi se omogućili jednostavni jezici koje se lako uče i primenjuju. Ovi jezici bi bili osnova konceptualnog okvira ETL procesa.

Na kraju treba naglasiti da neki predloženi pristupi ne obezbeđuju eksplicitne koncepte kojima se omogućava formalna definicija semantike transformacije podataka. Na primer, u *(Vassiliadis, Simitsis, & Skiadopoulos, 2002, Novembar; Simitsis & Vassiliadis, 2003; Luján-Mora, Vassiliadis, & Trujillo, 2004)* se koriste anotacije u kojima se daje objašnjenje semantike transformacija (npr. tipovi, ograničenja, izrazi, uslovi, itd.), dok se u *(Trujillo & Luján-Mora, 2003)* čak i mapiranja atributa definišu u anotacijama. S obzirom da se u ovim pristupima dozvoljava korišćenje prirodnog jezika (obično bez ograničenja u pogledu njihovog sadržaja) oni ne reprezentuju formalnu specifikaciju. Međutim, da bi se obezbedio automatizovani razvoj, MDD zahteva da modeli budu formalno iskazani. Stoga se može zaključiti da je neophodno da se obezbedi sredstvo za formalnu specifikaciju semantike transformacija podataka.

Implementacija ETL procesa

Sledeći korak u razvoju ETL procesa je implementacija specifikacije ETL procesa. Potrebno je naglasiti da je predloženo samo nekoliko pristupa kojima je omogućen automatizovani razvoj ETL procesa u kontekstu MDD-a. Uopšte, da bi se omogućila automatizovan razvoj ETL procesa u skladu sa MDD pristupom, neophodno je da se prvo mapiraju domenski koncepti u implementacione koncepte, a zatim u koncepte konkretnog programskog jezika. Način na koji je postignuta automatizacija razvoja softvera (MDA ili DSM) je još jedna razlika u odnosu na koju će se analizirati predloženi pristupi.



Slika 17. MDD pristupi: DSM pristup (levo) i MDA pristup (desno) *(Iseger, 2010)*

U MDA pristupu, razvoj softvera se može delimično ili u potpunosti automatizovati kroz primenu sukcesivnih transformacija modela. Počinje se od modela kojim se predstavlja specifikacija sistema, a završava modelom koji predstavlja detaljan opis fizičke realizacije na osnovu koga se na kraju može generisati izvršni kôd. Razvoj ETL procesa u skladu sa MDA pristupom je predložen u *(Muñoz, Mazón, Pardillo, & Trujillo, 2008; Mazón & Trujillo, 2008)*. Model ETL procesa je definisan kao platformski nezavisan model (PIM) koji se preko skupa formalno definisanih transformacija automatski transformiše u platformski specifičan model (PSM) na osnovu koga se na kraju generiše izvršni kôd. Međutim, platformski specifični modeli se formiraju za određenu platformu i stoga se predloženim pristupom propisuje da se za svaku platformu ručno razvija odgovarajući metamodel kako bi se omogućilo kreiranje transformacija iz predloženog modela u izabranu platformu. S druge strane, MDA pristup uopšte, zasniva se na profinjavanju (eng. *refinement*) modela preko uzastopnih transformacija modela. Ipak, ovaj proces najčešće zahteva da se generisani modeli ručno proširuju sa dodatnim detaljima.

Ova ručna proširenja mogu dovesti do neslaganja, odnosno razlike između originalnih i generisanih modela (tj. originalni modeli postaju zastareli). Razlike posebno dolaze do izražaja kada se modifikuju modeli koji su delimično generisani. Budući da i dalje nije razrešeno pitanje na koji način omogućiti ispravnu modifikaciju ovih modela, MDA promoviše korišćenje jednog opšteg jezika modelovanja (tj. UML) na svim apstraktnim nivoima *(Fowler, 2010; Cook, 2004)*. Međutim, ovakav pristup, ne samo da povlači sve diskutovane nedostatke u pogledu korišćenja opštih jezika modelovanja, već i prema *(Fowler, 2010)* uvodi dodatnu složenost u razvoj samih transformacija modela. Poboljšanje predloženog pristupa, dato u *(El Akkaoui, Zimányi, Mazón, & Trujillo, 2011)*, se zasniva na direktnom generisanju kôda za specifičnu platformu, čime je eleminisana potreba za definisanje metamodela za ciljnu platformu. Na ovaj način konceptualni model se može automatski transformisati u specifičan kôd kojim se ETL proces izvršava na konkretnoj platformi.

Sa druge strane, DSM pristup omogućava da se implementacija generiše na osnovu specifikacije sistema korišćenjem odgovarajućih generatora kojima se informacije ekstrahuju iz modela i transformišu u izvršni kôd. Drugim rečima, generatori su domensko-specifični, oni mapiraju modele (koji su u skladu sa metamodelom jezika) u izvršni kôd. Prema *(Kelly & Tolvanen, 2008)* ovo je jedini način da se omogući generisanje potpunog kôda (eng. *full code generation*), tj. generisanje kôda koji se ne mora dodatno modifikovati. Generisanje kôda je najčešće podržano odgovarajućim aplikacionim okvirom (eng. *domain framework*) kojim su obezbeđeni implementacioni koncepti, bliski konceptima domena problema *(Cook, 2004)*. Na ovaj način je smanjen jaz između domena problema i domena rešenja koji bi se u suprotnom razrešavao generatorima kôda. Osnovna prednost DSM pristupa, prema *(Kelly & Tolvanen, 2008)*, se ogleda u činjenici da omogućava direktno mapiranje u niži apstraktni nivo (tj. nije neophodno mapiranje domenskih koncepata u implementacione koncepte i dalje u koncepte konkretnog programskog jezika, koje je podložno greškama) čime je obezbeđeno generisanje potpunog kôda, umesto parcijalne implementacije. S obzirom da se generisani kôd

može direktno kompajlirati bez dodatnih intervencija, modeli se smatraju izvršnim.

Iz navedenog se može zaključiti da, ukoliko je cilj formalizacija i automatizacija razvoja ETL procesa, DSM pristup treba usvojiti. Osnovni razlog je što omogućava formalizaciju semantički bogatih apstrakcija u obliku koji omogućava njihovo ponovno korišćenje, ali i generisanje izvršnog kôda na osnovu modela koji pretstavljaju specifikaciju sistema. Sa druge strane, gledajući iz ugla modelovanja, koncepti opštih jezika modelovanja se ne odnose ni na jedan specifičan domen problema, dok iz ugla implementacije oni ne odgovaraju ni jednoj određenoj softverskoj platformi, aplikacionom okviru ili biblioteci komponenti *(Kelly & Tolvanen, 2008)*. Isto tako, MDA pretpostavlja postojanje nekoliko modela na različitim nivoima apstrakcije koji su dobijeni postepenim profinjavanjem (koja može biti automatska i ručna) tako da je automatizacija obično delimično postignuta. Kao dodatna prednost DSM pristupa može se navesti mogućnost veoma jednostavnog prilagođavanja i modela i generatora kôda što čini proces razvoja agilnim. Nakon izvršenih prilagođavanja potrebno je jednostavno regenerisati izvršni kôd.

Potrebno je naglasiti da aplikacionim okvirom (kojim je obezbeđena podrška implementaciji specifikacije ETL procesa u skladu sa DSM pristupom) treba definisati specifične implementacione koncepte koji su bliski konceptima realnog domena *(Cook, 2004)*. Ukoliko i specifikacija i aplikacioni okvir koriste formalne koncepte bliske konceptima ETL domena, transformacija između njih se može u potpunosti automatizovati. Ovim se značajno povećava produktivnost i efikasnost, a ujedno se smanjuju troškovi i vreme potrebni za razvoj i održavanje. Drugim rečima, izdizanjem semantičkog nivoa koji je tehnološki podržan, razvoj se može automatizovati i manji broj koraka je potreban za implementaciju apstraktnih specifikacija.

Konačno, potrebno je naglasiti da je pristup, koji u najvećoj meri zadovoljava postavljene zahteve, predložen u *(El Akkaoui & Zimányi, 2009; El Akkaoui, Zimányi, Mazón, & Trujillo, 2011; El Akkaoui, Mazón, Vaisman, & Zimányi, 2012)*. Autori su

obezbedili čak i ugrađene mehanizme za validaciju sintaksne i semantičke ispravnosti kreiranih modela. Međutim, ovaj pristup je zapravo zasnovan na korišćenju jednog jedinstvenog jezika modelovanja koji predstavlja proširenje opšteg jezika modelovanja – BPMN *(Object Management Group, 2011, Januar)*.

Zaključna razmatranja

Može se zaključiti da razvoj ETL procesa treba da se zasniva na korišćenju apstrakcija s obzirom da su one osnovno metodološko sredstvo za savladavanje složenosti. Pored toga, za specifikaciju ETL procesa treba koristiti semantički bogate apstrakcije, jer se njima obuhvata veće znanje što dovodi do povećanja produktivnosti i efikasnosti *(Greenfield, Short, Cook, & Kent, 2004)*. Isto tako, s obzirom da se traži viši nivo automatizacije, neophodno je formalizovati postojeće znanje i iskustvo u obliku koji omogućava njegovo ponovno korišćenje. Mogućnost ponovnog korišćenja dodatno doprinosi povećanju produktivnosti i efikasnosti, pri čemu se znatno smanjuju troškovi i vreme potrebni za razvoj i održavanje ETL procesa.

Domenski eksperti imaju veoma važnu ulogu tokom modelovanja ETL procesa. Oni poseduju iskustvo i ključna znanja iz posmatranog domena, tj. razumeju semantiku podataka koje je potrebno transformisati. Iz tog razloga modeli treba da budu iskazani konceptima koji su specifični za određeni domen (tj. konceptima koje koriste domenski eksperti). Dodatno, jezici modelovanja treba da budu jednostavni, ali u isto vreme i semantički bogati kako bi se omogućila specifikacija različitih aspekata domena problema na odgovarajućem nivou apstrakcije (tj. kao što je rečeno u *(Greenfield, Short, Cook, & Kent, 2004)*, što su apstrakcije opštije to je i mogućnost njihove primene veća, ali je istovremeno i manji doprinos). Sa druge strane, ovi jezici moraju biti formalni kako bi se omogućile automatske transformacije modela. Prema tome, za formalnu specifikaciju ETL procesa prednost imaju domensko-specifični jezici (u odnosu na opšte jezike modelovanja) s obzirom da obezbeđuju minimalni skup semantički bogatih domensko-specifičnih koncepata koji su bliski domenu problema i samim tim lakši za razumevanje i korišćenje domenskim ekspertima. Pored toga, različite aspekte ETL

procesa treba modelovati posebno. Razlog uvođenju različitih jezika, umesto korišćenja samo jednog, ogleda se u smanjenju složenosti modelovanja. Različiti aspekti se predstavljaju različitim modelima, ali su i jezici jednostavniji i lakši za korišćenje.

Implementacija specifikacija je takođe veoma bitan aspekt. Izdizanjem semantičkog nivoa i obezbeđivanjem odgovarajuće tehnološke podrške, razvoj se može automatizovati i manji broj koraka je neophodan za implementaciju specifikacije. Drugim rečima, ukoliko i specifikacija i aplikacioni okvir koriste formalne koncepte koji su bliski konceptima domena transformacija između njih se može u potpunosti automatizovati.

5. OSNOVNI KONCEPTI PREDLOŽENOG REŠENJA

Problemi koji se rešavaju sistemima skladišta podataka su veoma kompleksni *(Inmon, Strauss, & Neushloss, 2010)*. Složenost se pre svega ogleda u potrebi integracije podataka koji potiču iz heterogenih izvora. Potrebno je izvršiti različite transformacije kako bi se razrešile brojne strukturne i semantičke neusaglašenosti i kako bi se tako dobijeni podaci predstavili u obliku koji je pogodan za efikasno sprovođenje analiza. Promene su takođe veoma bitan faktor koji dodatno utiče na kompleksnost ovih sistema. Potrebno je apsorbovati česte promene u izvorima podataka, kako u pogledu promene njihovog stanja i strukture, tako i u pogledu uključivanja novih izvora podataka. Isto tako, kao posledica potrebe za prilogađavanjem sve oštrijim tržišnim uslovima, česte su i promene u pogledu zahteva, odnosno potreba poslovanja u smislu neophodnih analiza. Još jedan problem sa kojim se suočavaju sistemi skladišta podataka jeste velika količina podataka koju je potrebno prikupiti, obraditi, skladištiti i isporučiti, čime se postavljaju strogi zahtevi u pogledu strukturiranja podataka, ali i performansi i skalabilnosti sistema.

Dosadašnja istraživanja i praksa u domenu skladišta podataka su dala značajne rezultate. Akumulirano je veliko znanje i razumevanje posmatranog domena i postignut je napredak u pogledu formalizacije i automatizacije razvoja skladišta podataka. Razmatran je i statički i dinamički aspekt skladišta podataka, predložene su različite referentne arhitekture *(Golfarelli & Rizzi, 2009; Kimball, Ross, Thornthwaite, Mundy, & Becker, 2010; Linstedt & Graziano, 2011)*, kao i različiti pristupi razvoju skladišta podataka *(Golfarelli & Rizzi, 2009; Kimball, Ross, Thornthwaite, Mundy, & Becker, 2010; Linstedt & Graziano, 2011; Corr & Stagnitto, 2011; Damhof, 2011)*. Predloženim pristupima predstavljene su najbolje prakse i date preporuke u pogledu korišćenja specifičnih arhitektura podataka, zatim neophodnih aktivnosti, njihovog redosleda izvođenja kao i artefakata koje je

potrebno realizovati. Za modelovanje statičkog aspekta skladišta podataka predloženi su različiti modeli podataka, kao što je Anchor Model *(Regardt, Rönnbäck, Bergholtz, Johannesson, & Wohed, 2009; Rönnbäck, Regardt, Bergholtz, Johannesson, & Wohed, 2010)*, Data Vault Model *(Linstedt D. E., 2002; Linstedt D. E., 2003, Januar; Linstedt D. E., 2003, April; Linstedt D. E., 2004; Linstedt D. E., 2005; Linstedt & Graziano, 2011; Jovanovic & Bojicic, 2012; Jovanovic, Bojicic, Knowles, & Pavlic, 2012)* i dobro poznati Dimensional Model *(Kimball & Ross, 2002; Kimball, Ross, Thornthwaite, Mundy, & Becker, 2010; Adamson, 2010)*. Predloženi modeli podataka su razvijeni specifično za domen skladišta podataka kako bi se omogućilo zadovoljenje svih specifičnih zahteva. Za modelovanje dinamičkog aspekta skladišta podataka, odnosno za modelovanje ETL procesa predloženo je više pristupa (pogledati poglavlje 4, strana 68).

Međutim, razvijeni metodološki pristupi u poslednjih nekoliko decenija i pored značajnog napretka, ipak nisu doveli do očekivanih rezultata u rešavanju brojnih teškoća i problema kao što su visoki troškovi razvoja i održavanja, slaba produktivnost, neodgovarajuće ispunjene korisničkih zahteva itd., u razvoju skladišta podataka. Navedeni problemi uzrokovani su pre svega velikom složenošću savremenih poslovnih sistema, čestim promenama u organizacionom i tehnološkom okruženju i sve većom potrebom za integracijom sa okruženjem, naročito sa naglim razvojem Interneta i Web-a i prelaskom na elektronsko poslovanje. Nameće se zaključak da brojni problemi koji se javljaju u procesu razvoja skladišta podataka nisu na odgovarajući način rešeni u postojećim metodološkim pristupima. U skladu sa ovim zaključkom, ovaj rad ima kao osnovnu tezu da je za rešavanje problema u razvoju skladišta podataka izuzetno važna upotreba apstrakcija kao jedinog metodološkog načina za savladavanje složenosti. Pored savladavanja složenosti, apstrakcije se koriste i za uočavanje razlika i sličnosti kako bi se delovi sistema sa istim funkcionalnostima mogli izdvojiti, specificirati i implementirati nezavisno od specifikacije i višestruko koristiti. Drugim rečima, pošto je potreban veći stepen automatizacije, potrebno je formalizovati znanja i stečena iskustva u obliku koji dozvoljava njihovo ponovno korišćenje. Ponovnim korišćenjem se dodatno utiče na produktivnost i efikasnost,

čime se ujedno i smanjuju troškovi uvođenja ovakvih sistema. Jedan od predmeta istraživanja u ovom radu je i implementacija apstraktnih specifikacija koja treba značajno da podigne semantički nivo koji je tehnološki podržan i koji se može automatizovati. Na taj način razvoj postaje više automatizovan i potreban je manji broj koraka u postupku realizacije apstraktnih specifikacija.

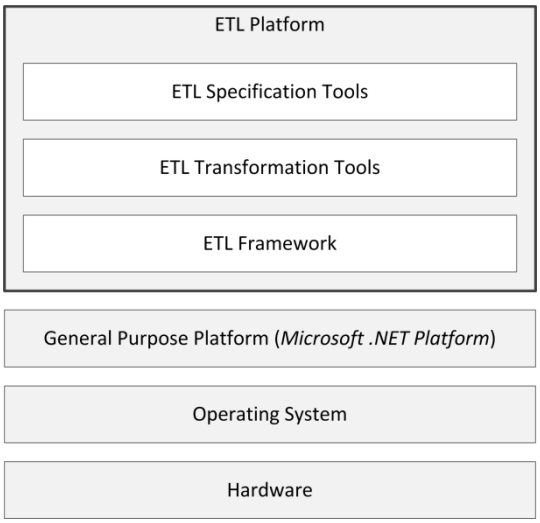
Problem koji se rešava u ovom radu je razvoj ETL procesa i rešenje koje se predlaže ima za cilj da učini takav razvoj u velikoj meri automatizovanim. Predloženo rešenje razvoja zasniva se na *Domain Specific Modeling* (DSM) pristupu i u skladu sa tim korišćene su formalne specifikacije ETL procesa i njihova automatska transformacija u specifičan aplikacioni okvir.

Za specifikaciju ETL procesa, predlaže se korišćenje odgovarajućih domensko-specifičnih jezika (eng. *domain specific language*, u daljem tekstu DSL) kojima su definisani koncepti relevantni za dati domen i koji nisu ekstenzije postojećih opštih jezika modelovanja, kao što su UML *(Object Management Group, 2011, Avgust; Pilone & Pitman, 2005)* i BPMN *(Object Management Group, 2011, Januar; Silver, 2011)*, već su razvijeni kao novi jezici. Svaki od predloženih DSL jezika omogućava modelovanje određenog aspekta ETL procesa. Na ovaj način omogućeno je uvođenje veoma jednostavnih specifičnih jezika modelovanja koji se lako uče i primenjuju i koji sadrže samo one koncepte koji su relevantni za dati aspekt ETL procesa. Pored toga, dodatno se utiče na smanjenje složenosti modelovanja, jer se različiti aspekti ETL procesa nezavisno modeluju. Na ovom mestu je potrebno naglasiti da se u ovom radu ne razmatraju svi aspekti ETL procesa već samo oni najvažniji kao što je redosled izvršavanja aktivnosti ETL procesa i semantika aktivnosti. Specifikacije ostalih aspekata ETL procesa mogu se podržati uvođenjem novih domensko specifičnih jezika.

Implementacija ETL procesa ostvarena je uvođenjem specifičnog aplikacionog okvira da bi se predložena specifikacija tehnološki podržala i omogućila automatizacija razvoja u skladu sa usvojenim DSM pristupom. Uvođenjem aplikacionog okvira značajno se podiže semantički nivo koji je implementaciono podržan, definisanjem specifičnih implementacionih koncepata bliskih konceptima

realnog domena specificiranim u predloženim domensko-specifičnim jezicima. Pre svega, predložena je odgovarajuća softverska arhitektura za rešenja iz domena ETL procesa kojom su identifikovane osnovne komponente i veze između njih, a zatim je data njihova implementacija u izabranom tehnološkom okruženju u vidu generičkih softverskih komponenti. Implementacija konkretnog ETL procesa omogućena je kroz proces kastomizacije ovih generičkih komponenti. Ovaj proces kastomizacije je automatizovan uvođenjem specifičnih generatora kojima se date specifikacije ETL procesa automatski prevode u odgovarajuće softverske komponente koje predstavljaju specijalizacije prethodno realizovanih generičkih komponenti. Bazirajući se na automatskim transformacijama između specifikacija ETL procesa i aplikacionog okvira povećava se i produktivnost i efikasnost razvoja ETL procesa.

U skladu sa usvojenim pristupom razvoju ETL procesa, predlaže se specifična ETL platforma (eng. *ETL Platform*) kojom je omogućen razvoj, izvršavanje i upravljanje ETL procesima. ETL platforma je uvedena (kao proširenje *Microsoft .NET platforme*) da bi se predložene specifikacije ETL procesa tehnološki podržale i omogućila automatizacija razvoja ETL procesa u skladu sa DSM pristupom.



Slika 18. Arhitektura predložene ETL platforme

Kao što je i prikazano (slika 18), arhitektura predložene ETL platforme je organizovana u odgovarajuće slojeve:

- Najniži sloj (*ETL Framework*) reprezentuje izvršno okruženje i sadrži skup servisa kojima se omogućava izvršavanje i upravljanje ETL procesima. Uvođenjem specifičnog aplikacionog okvira značajno se podiže semantički nivo koji je implementaciono podržan i koji je moguće automatizovati. Izvršni kôd aplikacionog okvira se automatski generiše iz formiranih modela.
- Sledeći sloj (*ETL Transformation Tools*) je poseban sloj ETL platforme koji omogućava automatizaciju postupka transformacije modela. Automatske transformacije modela, formiranih u skladu sa definisanim domensko-specifičnim jezicima, u izvršni kôd aplikacionog okvira podržane su specifično razvijenim generatorima.
- U okviru poslednjeg sloja (*ETL Specification Tools*) predložene ETL platforme, koji predstavlja razvojno okruženje, razvijeni su različiti softverski alati kao podrška modelovanju ETL procesa. Definisanje apstraktne i konkretne sintakse specificiranih domensko-specifičnih jezika tehnološki je podržano pomoću *Microsoft DSL Tools* alata. Za formiranje modela u skladu sa definisanim domensko-specifičnim jezicima razvijeni su odgovarajući softverski alati (pre svega grafički editori).

6. FORMALNA SPECIFIKACIJA DOMENSKO-SPECIFIČNIH JEZIKA ZA MODELOVANJE ETL PROCESA

Mogu se identifikovati dva osnovna zahteva u pogledu obezbeđivanja uspeha tokom modelovanja ETL procesa. Prvi zahtev se odnosi na potrebu korišćenja jezika modelovanja koji su jednostavni, odnosno koji sadrže minimalan broj koncepata, ali koji su u isto vreme i semantički bogati tako da omogućavaju specifikaciju različitih aspekata posmatranog ETL procesa. Na ovaj način omogućava se sprovođenje kvalitetnog procesa analize posmatranog problema i u skladu sa tim kvalitetna specifikacija posmatranog problema što je i osnovni cilj modelovanja. Pored ovog osnovnog zahteva nameće se još jedan bitan zahtev koji se ogleda u potrebi da predloženi jezik modelovanja bude formalan. Naime, korišćenje modela u svrhu dokumentovanja i omogućavanja komunikacije između članova tima ima određenu vrednost, ali to ima manju vrednost u odnosu na aktivno učešće modela u procesu razvoja. Savremenim pristupima razvoju softvera, kao što je DSM pristup, propisuju se automatske transformacije modela čime se značajno utiče na povećanje efikasnosti u procesu razvoja. Međutim, da bi se omogućile automatske transformacije modela, potrebno je da modeli budu formalno opisani. Ovo podrazumeva korišćenje formalnih jezika modelovanja, odnosno jezika za koje je definisana apstraktna sintaksa, zatim konkretna sintaksa i na kraju semantika. Apstraktnom sintaksom (metamodelom) se definišu koncepti koji se mogu koristiti tokom modelovanja i za svaki koncept se moraju precizno i jasno definisati semantička pravila kojima se definišu ispravna korišćenja takvog koncepta. Konkretnom sintaksom omogućeno je definisanje vizuelne reprezentacije jezika modelovanja, koja može biti tekstualna ili grafička.

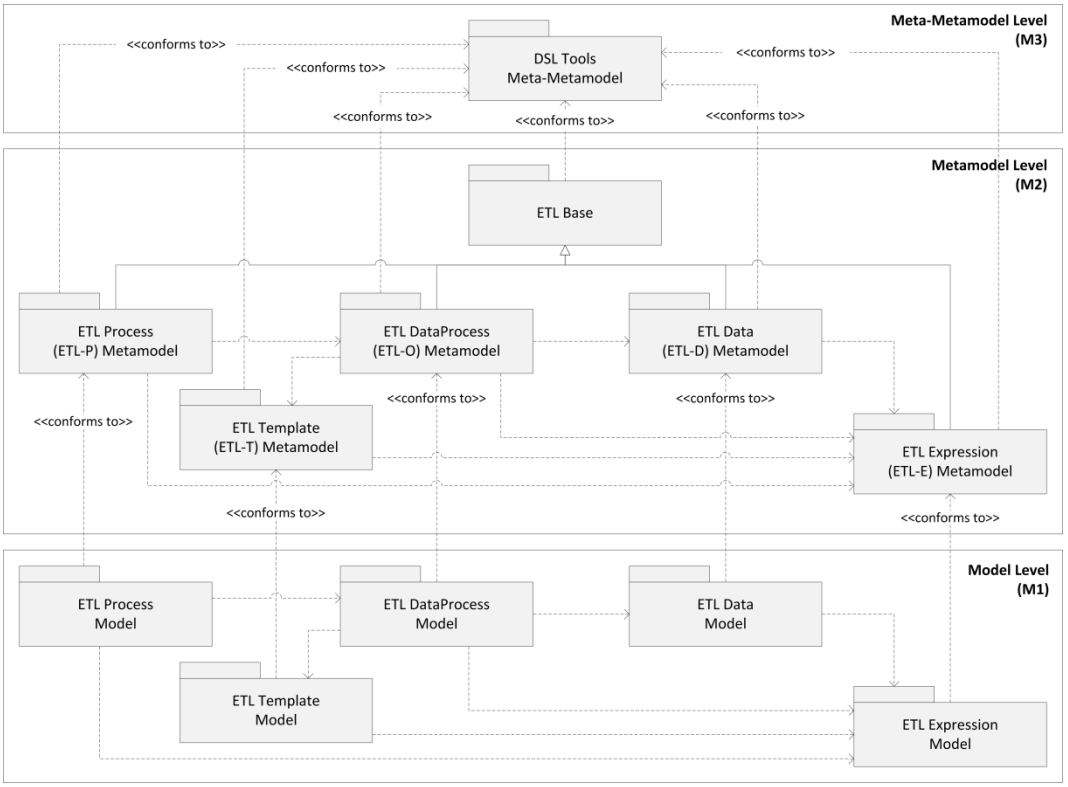
U skladu sa DSM pristupom za formalnu specifikaciju predloženo je i definisano nekoliko novih domensko-specifičnih jezika:

(1) jezik za specifikaciju toka izvršavanja ETL-P,
(2) jezik za specifikaciju operacija transformacija ETL-O,
(3) jezik za specifikaciju izraza ETL-E i
(4) jezik za specifikaciju šablona operacija transformacija podataka ETL-T.

Predloženim jezicima omogućeno je modelovanje različitih aspekata ETL procesa. ETL-P jezik omogućava specifikaciju semantike izvršavanja ETL procesa, odnosno specifikaciju redosleda izvršavanja aktivnosti ETL procesa, dok ETL-O jezik omogućava specifikaciju semantike transformacija podataka, odnosno operacija transformacija i njihovih međuzavisnosti. ETL-E jezik je uveden sa ciljem da se omogući formalna specifikacija semantike transformacija podataka, kao i različitih ograničenja i uslova u vezi sa izvršavanjem ETL procesa. ETL-T jezik je uveden kako bi se omogućilo definisanje različitih šablona operacija transformacija koji se mogu višestruko koristiti.

U nastavku je dat konceptualni okvir kojim se definišu fundamentalni koncepti ETL procesa i njihovi međusobni odnosi. Konceptualni okvir je reprezentovan preko više paketa čija je međuzavisnost data na slici (slika 19).

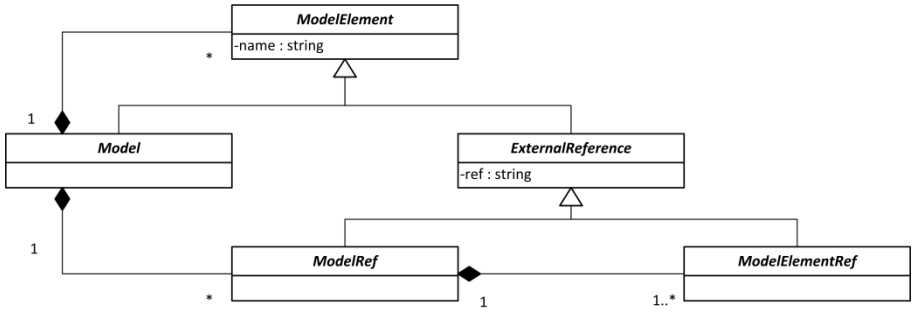
Prikazani paketi odnose se na metamodele jezika koji se kasnije objašnjavaju (paket *ETL Process* odnosi se na metamodel ETL-P jezika; paket *ETL Data Process* odnosi se na metamodel ETL-O jezika; paket *ETL Expression* odnosi se na metamodel ETL-E jezika; paket *ETL Template* odnosi se na metamodel ETL-T jezika). Treba naglasiti da se metamodeli predloženih domensko-specifičnih jezika opisuju konceptima meta-jezika koji je definisan od strane *Microsoft*-a (u okviru *DSL Tools* alata) i kao takav je fiksiran u implementaciju ETL platforme. Stoga se provera validnosti metamodela predloženih domensko-specifičnih jezika obavlja u skladu sa meta-jezikom. Sa druge strane, metamodeli domensko-specifičnih jezika koji opisuju konkretne modele, služe za validaciju tako formiranih modela.



Slika 19. Konceptualni okvir ETL procesa

Paket *ETL Data* se odnosi na metamodel ETL-D jezika kojim bi se omogućilo predstavljanje različitih vrsta modela podataka na uniformni način, koji se koriste u operacijama transformacija (bilo kao njihovi ulazi ili kao rezultati njihovog izvršavanja). ETL-D jezik nije razmatran u ovom radu, jer se problem integracije podataka, tj. projektovanje integracionog pogleda (konsolidovanog modela) čija je namena skrivanje heterogenosti različitih izvora podataka rešava u fazi analize razvoja skladišta podataka. U kontekstu razvoja ETL procesa pretpostavlja se da su različite lokalne šeme koje opisuju heterogene izvore podataka već transformisane u ekvivalentne šeme, opisane na uniformni način konceptima konsolidovanog modela. ETL-D jezik je naveden zbog sveobuhvatnog prikaza predloženog rešenja i zbog potrebe realizacije i implementacije predloženog rešenja. Korišćen je pojednostavljeni metamodel ETL-D jezika (koncepti *Entity* i *Field*) i podrazumeva se da se svi konkretni modeli podataka zasnivaju na datom metamodelu. Treba naglasiti da specificirani jezici ETL procesa sadrže koncepte koji ukazuju na integrisane šeme sadržane u konsolidovanom modelu. Problem integracije i projektovanja konsolidovanog modela razmatran je u *(Bojičić, 2014)*.

Specifikacija apstraktne sintakse kao jezičkog modela zasniva se na metamodelu kojim se definišu specifični koncepti za modelovanje određenog domensko-specifičnog jezika. Osnovni koncepti neophodni za definisanje predloženih jezika predstavljeni su na slici (slika 20).



Slika 20. Osnovni koncepti za definisanje predloženih jezika

Koncept *Model* generalizuje sve predložene domensko-specifične jezike, a njegov sadržaj čine svi model elementi (*ModelElement*) kao i reference (*ExternalReference*) na druge vrste modela (na primer, modeli podataka koji se koriste u operacijama transformacija podataka). Koncepti *ModelRef* i *ModelElementRef* su uvedeni kako bi se obezbedio mehanizam za referenciranje različitih modela i njihovih elemenata.

Prikazani apstraktni koncepti su osnovni koncepti kojima se specificiraju predloženi jezici. Njihovom specijalizacijom uvode se ostali specifični koncepti predloženih jezika.

U nastavku se detaljno razmatraju predloženi jezici za specifikaciju ETL procesa. Za svaki od predloženih jezika predstavljeni su konkretni koncepti i ograničenja u vezi njihovog korišćenja, kao i predlog odgovarajuće konkretne sintakse.

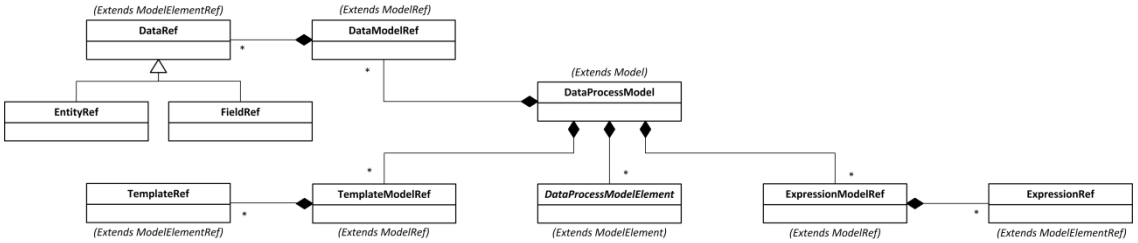
6.1. Jezik za specifikaciju operacija transformacija podataka (ETL-O)

Jezik za specifikaciju operacija transformacija podataka (ETL-O) je uveden sa ciljem da se omogući specifikacija semantike transformacija podataka, odnosno operacija transformacija podataka i njihovih međuzavisnosti. Specifikacijom ovih operacija definiše se tok podataka (eng. *dataflow*), ili drugim rečima proces obrade podataka. Ovim procesom podaci se ekstrahuju iz odgovarajućih izvora, zatim se vrši njihova transformacija i na kraju se tako transformisani podaci učitavaju u odgovarajuće ciljno skladište.

6.1.1. Apstraktna sintaksa

Specifikacija apstraktne sintakse kao jezičkog modela ovde se zasniva na metamodelu kojim se opisuju specifični koncepti ETL-O jezika. Imajući u vidu proširivost i fleksibilnost arhitekture ETL procesa, specifični koncepti uvedeni su proširenjem osnovnih koncepata (slika 20, strana 97).

Model procesa obrade podataka (*DataProcessModel*) i njegovi elementi (*DataProcessModelElement*) modelovani su kao podklase osnovnih koncepata *Model* i *ModelElement*, tim redom. Kao proširenja osnovnih koncepata (*ModelRef* i *ModelElementRef*), uvedeni su i koncepti *DataModelRef* i *DataRef*, zatim *TemplateModelRef* i *TemplateRef*, i na kraju *ExpressionModelRef* i *ExpressionRef*. Ovim konceptima omogućeno je referenciranje specifičnih modela i njihovih elemenata.



Slika 21. ETL-O metamodel (proširenja osnovnih koncepata)

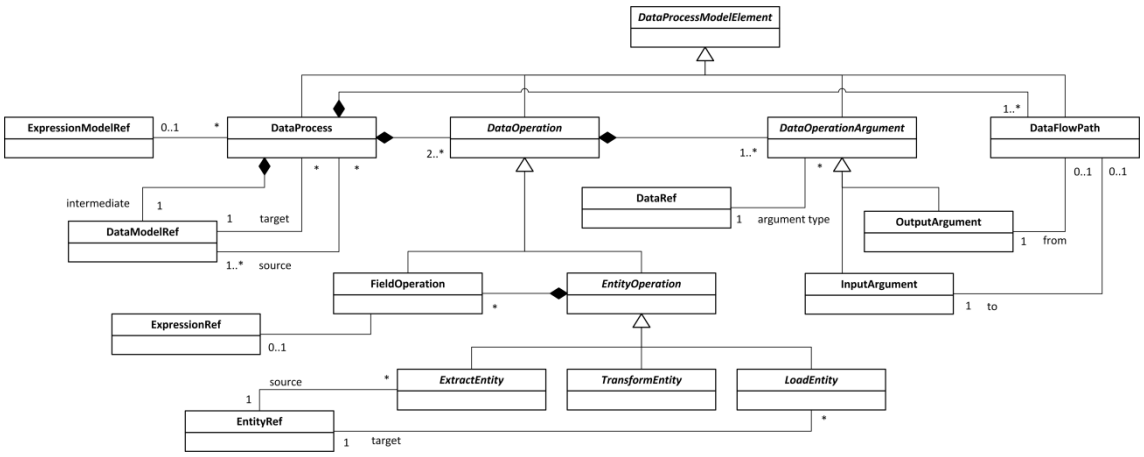
Uvođenjem koncepata *DataModelRef* omogućeno je referenciranje različitih modela podataka koji učestvuju u transformacijama. Referencirani modeli mogu predstavljati modele izvora podataka (*source*), zatim modele ciljnih podataka (*target*), kao i modele međurezultata toka podataka (*intermediate*). Konceptom

DataRef referenciraju se različiti elementi ovih modela podataka (entiteti i njihova polja) i shodno tome uvedeni su specifični koncepti *EntityRef* i *FieldRef*.

Konceptima *ExpressionModelRef* i *ExpressionRef* referenciraju se modeli izraza i njihovi elementi. Ovim modelima specificiraju se različiti tipovi izraza sa namerom da se omogući formalan opis pravila transformacija u operacijama definisanim ETL-O modelom.

Korišćenje različitih šablona operacija transformacija omogućeno je konceptima *TemplateModelRef* i *TemplateRef*, pri čemu se prvim referencira odgovarajući model šablona, a drugim njegovi elementi tj. konkretni šabloni operacija transformacija.

Pored navedenih, uvedeni su, proširenjem osnovnog koncepta *DataProcessModelElement,* i sledeći koncepti za specifikaciju toka podataka: *DataProcess*, *DataOperation*, *DataOperationArgument* i *DataFlowPath*.



Slika 22. ETL-O metamodel

Konceptom *DataProcess* predstavlja se proces obrade podataka. Proces obrade podataka se sastoji od operacija transformacija podataka (*DataOperation*) i odgovarajućih tokova (*DataFlowPath*) kojima se ove operacije povezuju. Isto tako, za svaki tok podataka potrebno je opisati i podatke koji učestvuju u transformacijama. S obzirom da je za specifikaciju podataka koriste posebni modeli, između koncepata *DataProcess* i *DataModelRef* definisane su odgovarajuće veze koje određuju uloge referenciranih modela (*source*, *target*, *intermediate*). Pre

svega, omogućeno je razlikovanje modela izvornih podataka (*source*), zatim modela ciljnih podataka (*target*), kao i modela međurezultata (*intermediate*), odnosno modela podataka koji nastaju kao rezultat izvršavanja operacija transformacija, a koji ne predstavljaju krajnji rezultat procesa obrade podataka toka podataka. Bitno je naglasiti da se za svaki proces obrade definiše tačno jedan model međurezultata. Pored modela podataka, za svaki proces obrade podataka moguće je definisati i odgovarajući model izraza što je i predstavljano vezom između koncepata *DataProcess* i *ExpressionModelRef*.

Specifikacija operacija transformacija podataka omogućena je apstraktnim konceptom *DataOperation*. Ovaj koncept je dalje specijalizovan kako bi se podržala specifikacija operacija transformacija na različitom nivou granularnosti, tj. specifikacija operacija transformacija na nivou entiteta (*EntityOperation*) i specifikacija operacija transformacija na nivou polja entiteta (*FieldOperation*). Kako bi se bliže odredila semantika ovih operacija, koncepti *ExtractEntity* (operacija ekstrahovanja), *TransformEntity* (operacija transformacije) i *LoadEntity* (operacija učitavanja) modelovani su kao podklase koncepta *EntityOperation*. Za koncept *FieldOperation* definiše se logika transformacije pomoću odgovarajućeg izraza (*ExpressionRef*). Za koncepte *ExtractEntity* i *LoadEntity* definisane su odgovarajuće veze (*source*, odnosno *target* respektivno) sa konceptom *EntityRef* kako bi se referencirali entiteti koji se ekstrahuju, odnosno učitavaju.

Kao veoma bitan element specifikacije, razmatrani su i argumenti, odnosno ulazi i izlazi operacija transformacija. Zbog toga su uvedeni koncepti *InputArgument* i *OutputArgument* kao specijalizacije apstraktnog koncepta *DataOperationArgument*. Konceptom *InputArgument* specificiraju se ulazni argumenti operacije, dok se konceptom *OutputArgument* specificiraju izlazni argumenti operacije.

Za svaku operaciju transformacije je moguće specificirati više ulaza i/ili izlaza, pri čemu važi ograničenje da jedna operacija mora imati bar jedan argument (bilo ulazni bilo izlazni). Specifično, za operaciju ekstrahovanja podataka (*ExtractEntity*) navodi se samo izlazni argument, zatim za operaciju učitavanja (*LoadEntity*) samo ulazni argument, dok se za operaciju transformacije entiteta (TransformEntity)

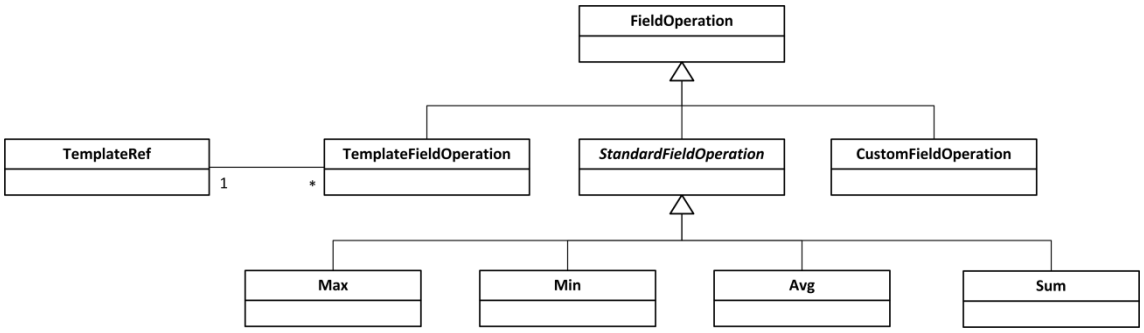
navodi minimalno jedan ulazni i jedan izlazni argument. Isto tako, za operacije transformacije polja entiteta (*FieldOperation*) obavezno je navođenje izlaznog argumenta.

Specifikacija argumenata je upotpunjena vezom sa konceptom *DataRef* čime je omogućeno definisanje tipa argumenta (*argument type*). Važi ograničenje da se za specifikaciju tipova argumenata operacija transformacija entiteta (*EntityOperation*) koristi koncept *EntityRef*, a za operacije transformacija polja entiteta (*FieldOperation*) koncept *FieldRef*.

Međuzavisnost, odnosno redosled izvršavanja operacija transformacija se specificira konceptom *DataFlowPath*. Ovim konceptom predstavljaju se tokovi podataka između operacija tranformacija. Kao što je i predstavljeno metamodelom, izlazni argument jedne operacije (*OutputArgument*) se povezuje sa ulaznim argumentom (*InputArgument*) druge operacije, pri čemu važi ograničenje da takvi argumenti moraju biti istog tipa (što znači da referenciraju isti element modela podataka).

Navedeni koncepti predstaljaju osnov predloženog jezika. Njihovim korišćenjem moguće je opisati bilo koji tok podataka. Međutim, specifikacija iskazana korišćenjem samo ovih koncepata ne bi bila dovoljno precizna. Pre svega, semantika operacija nije precizno definisana. Ovo dalje nalaže identifikovanje specifičnih operacija transformacija.

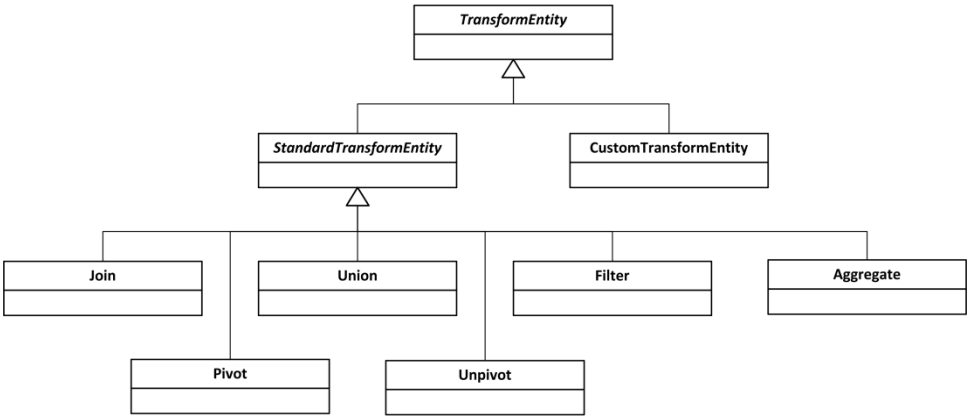
Kao specijalizacije koncepta *FieldOperation* uvedena su tri koncepta: *CustomFieldOperation*, *StandardFieldOperation* i *TemplateFieldOperation*.



Slika 23. ETL-O metamodel (specijalizacije koncepta FieldOperation)

Konceptom *CustomFieldOperation* omogućena je specifikacija proizvoljnih transformacija, dok je apstraktnim konceptom *StandardFieldOperation* omogućeno uvođenje standardnih (predefinisanih) operacija transformacija polja entiteta, kao što su na primer navedene operacije agregiranja podataka *min*, *max*, avg, *sum*, itd. Svaka standardna operacija je predstavljena posebnim konceptom kao specijalizacija koncepta *StandardFieldOperation*. Podrška korišćenju šablona operacija transformacija obezbeđena je konceptom *TemplateFieldOperation*. S obzirom da se šabloni operacija definišu u posebnim modelima, koncept *TemplateRef* se koristi za njihovo referenciranje.

Koncepti *CustomTransformEntity* i *StandardTransformEntity* uvedeni su kao ekstenzija koncepta *TransformEntity*.



Slika 24. ETL-O metamodel (specijalizacije koncepta TransformEntity)

Konceptom *CustomTransformEntity* omogućena je specifikacija proizvoljnih transformacija, dok je apstraktnim konceptom *StandardTransformEntity* omogućeno uvođenje standardnih operacija transformacija entiteta, kao što su operacije *join*, *union*, *filter*, *aggregate, pivot, unpivot*, itd. Svaka standardna operacija je predstavljena posebnim konceptom kao specijalizacija koncepta *StandardTransformEntity*.

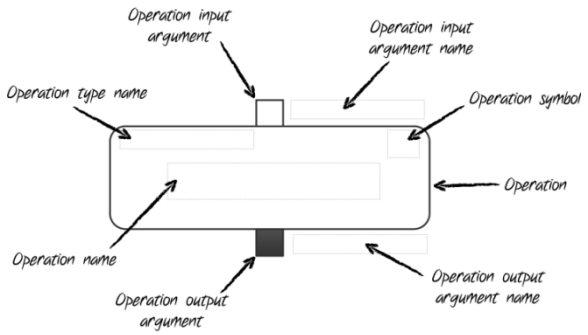
Za ove operacije specificiraju se i dodatni uslovi i ograničenja. Na primer, za operaciju spajanja (*join*) navodi se uslov spajanja (*join condition*), za operaciju filtriranja (*filter*) uslov filtriranja (*filter condition*), dok se za operaciju agregiranja (*aggregate*) navodi uslov grupisanja (*group condition*), itd. S obzirom da se za

specifikaciju uslova spajanja i uslova filtriranja koriste odgovarajući logički izrazi definisane su i odgovarajuće veze između koncepta *Join* i koncepta *ExpressionRef*, kao i koncepta *Filter* i koncepta *ExpressionRef*. Za specifikaciju uslova grupisanja uspostavljena je veza između koncepta *Aggregate* i koncepta *FieldRef*, pri čemu važi ograničenje da se mogu referencirati samo polja ulaznog entiteta (dat preko *EntityRef* ulaznog argumenta) i da se mora referencirati najmanje jedno polje.

6.1.2. Konkretna sintaksa

Kao što je i opisano u drugom poglavlju ovog rada, za definisanje jezika modelovanja pored apstraktne sintakse potrebo je definisati i odgovarajuću konkretnu sintaksu. U skladi sa tim, u nastavku je dat predlog grafičke notacije za identifikovane koncepte predloženog jezika ETL-O.

Predstavljanje različitih operacija transformacija (specijalizacije koncepta *DataOperation*), omogućeno je odgovarajućim grafičkim elementom (slika 25). U centralnom delu ovog elementa prikazuje se naziv operacije (eng. *operation name*), dok se u gornjem levom uglu prikazuje naziv tipa operacije (eng. *operation type name*), odnosno naziv koncepta koji se grafičkim elementom predstavlja. Gornji desni ugao grafičkog elementa je rezervisan za odgovarajući grafički simbol (eng. *operation symbol*) kojim se bliže određuje tip operacije.

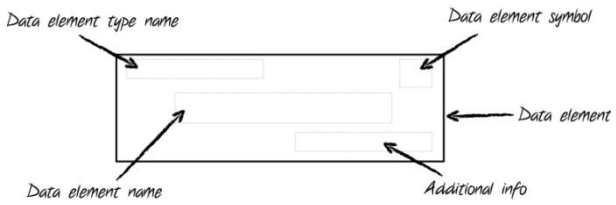


Slika 25. Grafički element za predstavljanje operacija transformacija

Kao sastavni deo ovog grafičkog elementa, uvedeni su i odgovarajući elementi za prikazivanje ulaznih i izlaznih argumenata operacije (koncepti *InputArgument* i *OutputArgument*). U pitanju su mali kvadrati na spoljnoj ivici, pri čemu se izlazni argument (eng. *operation output argument*) predstavlja crnim kvadratom, a ulazni

argument (eng. *operation input argument*) belim kvadratom. Prikazivanje naziva ovih argumenata (eng. *operation input argument name* i *operation output argument name*) nije obavezno.

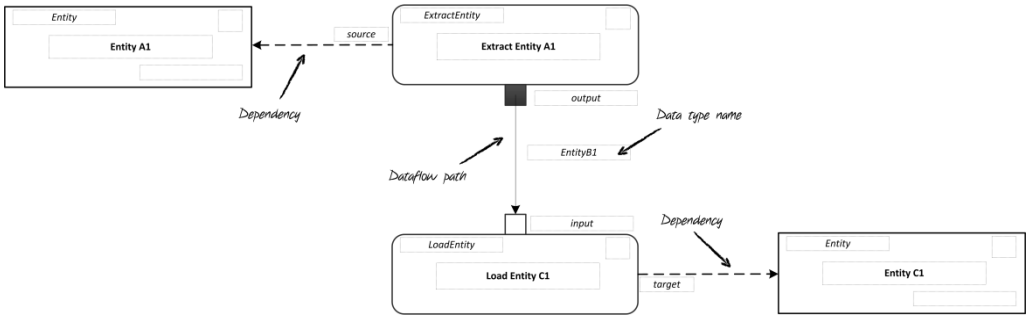
Za predstavljanje podataka koji učestvuju u transformacijama, tj. za predstavljanje referenciranih modela podataka (koncept *DataModelRef*), entiteta (koncept *EntityRef*) i polja entiteta (koncept *FieldRef*) predlaže se grafički element koji je predstavljan na slici (slika 26). U centralnom delu grafičkog elementa prikazuje se naziv (eng. *data element name*), a u gornjem levom uglu tip reference (eng. *data element type name*), pri čemu tip reference može uzeti vrednosti: „*Model Podataka*“ (eng. „*Data Model*“), *Entitet* (eng. „*Entity*“) ili „*Polje Entiteta*“ (eng. „*Field*“). U gornjem desnom uglu prikazuje se odgovarajući grafički simbol (eng. *data element symbol*) kojim se reprezentuje tip reference, dok je u donjem desnom uglu predviđeno prikazivanje dodatnih informacija (eng. *additional info*). Na primer, prilikom predstavljanja polja entiteta (na dijagramu operacije toka podataka) koja učestvuju u uslovu grupisanja (*group condition*) operacija agregiranja entiteta prikazuje se tekst „*grupiše*“ (eng. „*group by*“).



Slika 26. Grafički element za predstavljanje referenciranih podataka

Predstavljanje međuzavisnosti, odnosno tokova podataka (koncept *DataFlowPath*) između operacija transformacija omogućeno je usmerenim linijama (slika 27), pri čemu je smer od izlaznog argumenta jedne operacije ka ulaznom argumentu druge operacije. Opciono, uz liniju je moguće prikazati naziv toka podataka, odnosno naziv entiteta koji se tokom prenosi (eng. *data type name*).

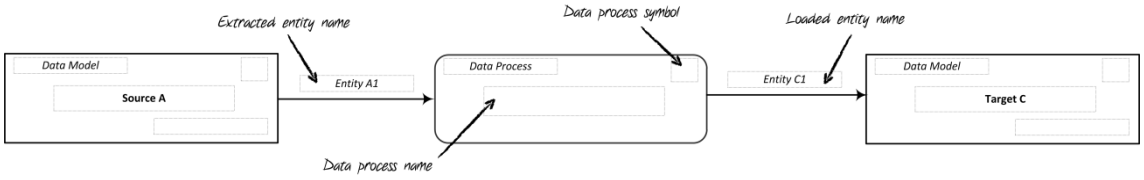
Prikazivanje ostalih relevantnih veza, odnosno zavisnosti (eng. *dependency*) omogućeno je isprekidanom usmerenom linijom (slika 27). Pre svega, misli se na prikazivanje izvornih entiteta operacija ekstrahovanja (*source*) i ciljnih entiteta operacija učitavanja (*target*).



Slika 27. Grafički element za prikazivanje zavisnosti

Za specifikaciju procesa obrade podataka koristi se više dijagrama. Na ovom mestu je dat pregled predložene notacije, dok je detaljan opis i objašnjenje dato u poglavlju 7 (strana 123).

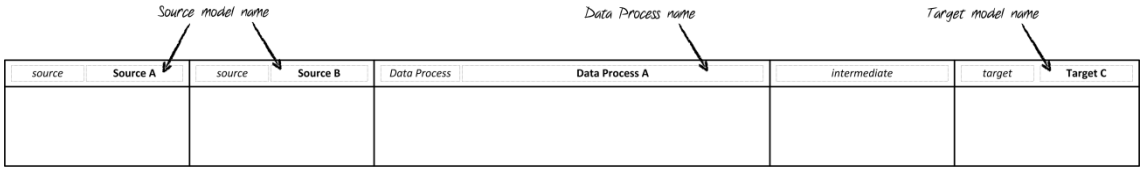
Na *dijagramu složenih operacija transformacija* (eng. *data process diagram*) predstavljaju se identifikovani procesi obrade podataka (koncept *DataProcess*) i različiti modeli podataka (koncept *DataModelRef*). Kao što je i prikazano (slika 28), za predstavljanje ovog koncepta uveden je odgovarajući grafički element kojim se u centralnom delu prikazuje naziv procesa obrade (eng. *data process name*), dok se u gornjem desnom uglu prikazuje odgovarajući grafički simbol (eng. *data process symbol*). Veze sa referenciranim modelima podataka predstavljaju usmerenim linijama, pored kojih se navode nazivi entiteta koji se ekstrahuju (eng. *extracted entity name*), odnosno učitavaju (eng. *loaded entity name*).



Slika 28. Notacija za dijagram složenih operacija transformacija podataka

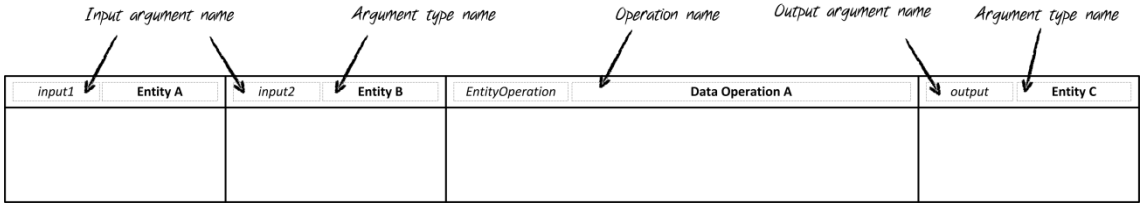
Dijagram složene operacije transformacije (eng. *complext transformation operation diagram*) prikazuje detalje određenog procesa obrade podataka (slika 29). U centralnom delu dijagrama navode se operacije transformacije entiteta (specijalizacije koncepta *EntityOperation*) koje čine dati proces obrade podataka, dok su levi i desni deo dijagrama rezervisani za prikaz referenciranih modela podataka (koncept *DataModelRef*). Za svaki referencirani model podataka prikazuje se uloga (*source*, *target* ili *intermediate*) i naziv (eng. *source model name*,

target model name, *data process name*). Kao ugnježđeni grafički elementi, prikazuju se odgovarajući entiteti (koncept *EntityRef*) ovih modela podataka.



Slika 29. Notacija za dijagram složene operacije transformacije podataka

Dijagram proste operacije transformacije (eng. *simple transformation operation diagram*) je sličan prethodnom dijagramu, s tom razlikom što se umesto operacija transformacija entiteta navode operacije transformacije polja entiteta (*FieldOperation*). Isto tako, na levom delu dijagrama se prikazuju ulazni argumenti operacije (*InputArgument*), dok se na desnom delu dijagrama prikazuju izlazni argumenti dijagrama (*OutputArgument*).

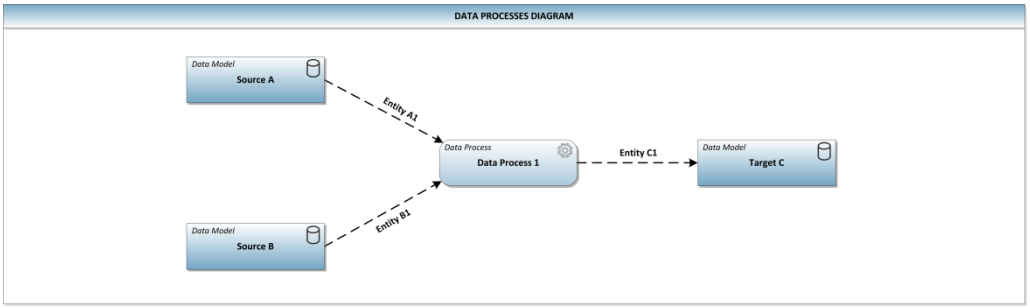


Slika 30. Notacija za dijagram proste operacije transformacije podataka

Za svaki argument operacije se prikazuje njegov naziv (eng. *input argument name*, *output argument name*), zatim tip (eng. *argument type name*) i polja (*FieldRef*) od kojih se sastoji. Polja se prikazuju u vidu ugnježđenih grafičkih elemenata.

6.1.3. Primer

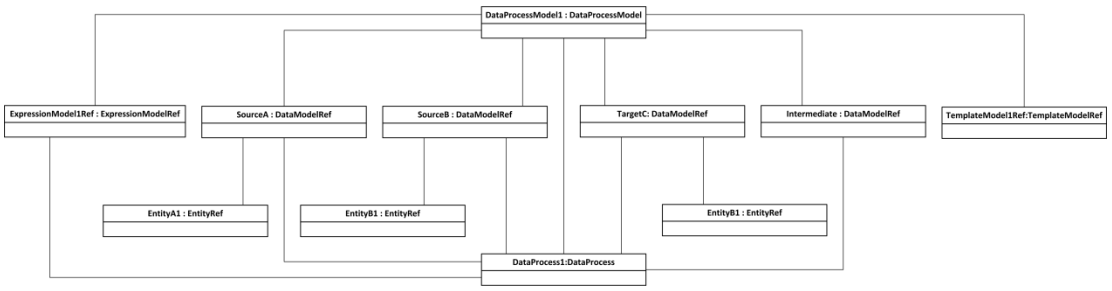
U ovom delu je dat primer jednog procesa obrade podataka (*DataProcess1*).



Slika 31. Dijagram složenih operacija transformacija

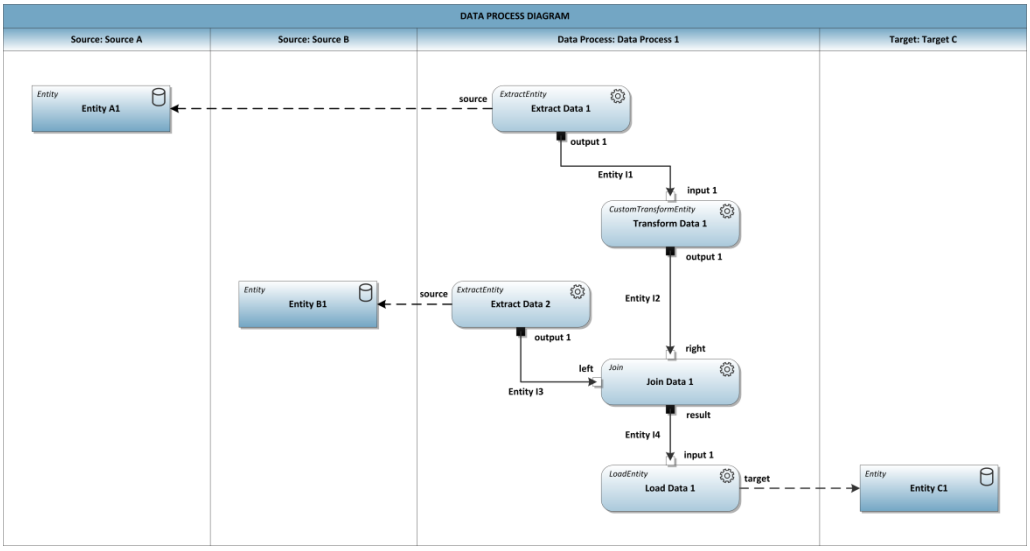
Predstavljene su dva izvorna skladišta podataka *SourceA* i *SourceB*, kao i ciljno skladište *TargetC* (slika 31). Iz prvog izvora podataka (*SourceA*) se ekstrahuju entiteti *EntityA1*, dok se iz drugog izvora (*SourceB*) ekstrahuju entiteti *EntityB1*. Rezultat izvršavanja procesa obrade podataka su entiteti *EntityC1* koji se učitavaju u skladište *TargetC*.

U nastavku je dat konkretan model koji je reprezentovan dijagramom objekata.



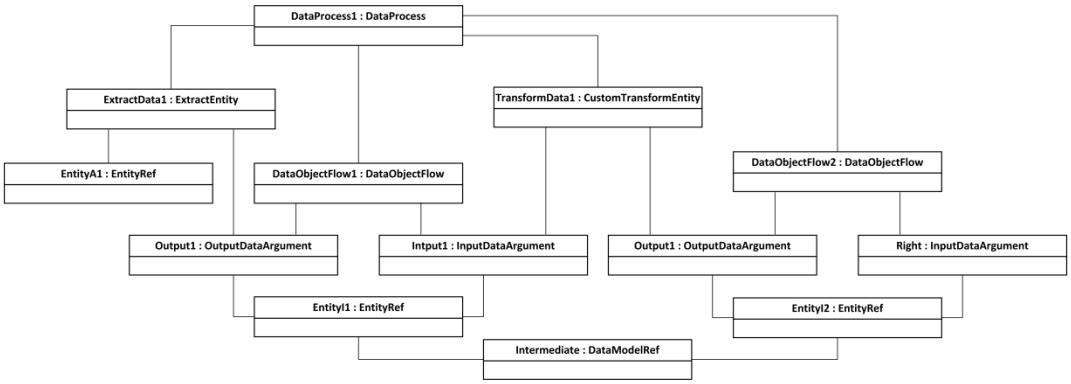
Slika 32. Dijagram objekata složenih operacija transformacija

Na sledećem dijagramu (slika 33) su predstavljene operacije transformacije entiteta koje čine proces obrade podataka *DataProcess1* i definisan je redosled njihovog izvršavanja. Proces obrade podataka započinje operacijama ekstrahovanja *ExtractData1* i *ExtractData2*. Operacija *TransformData1* počinje nakon što je završena operacija *ExtractData1*, dok operacija *JoinData1* počinje sa izvršavanjem tek pošto su završene prethodne operacije. Na kraju se izvršava operacija učitavanja podataka *LoadData1* kojim se proces obrade završava.

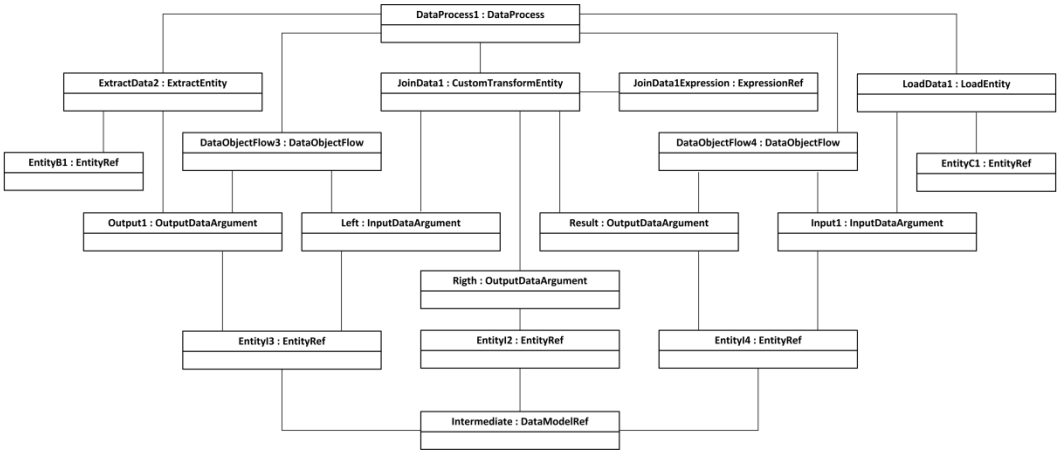


Slika 33. Dijagram složene operacije transformacije podataka za DataProcess1

U nastavku je dat konkretan model za operaciju *DataProcess1* koji je reprezentovan dijagramom objekata (slika 34, slika 35).

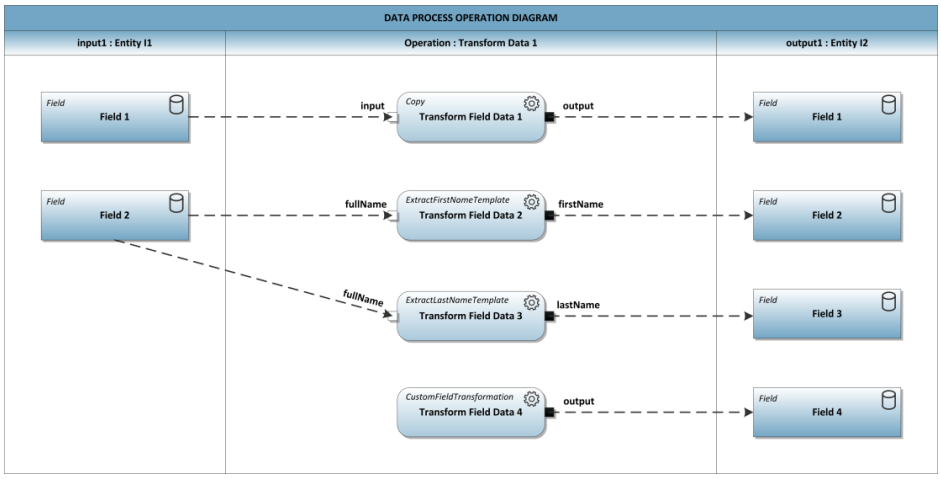


Slika 34. Dijagram objekata za operaciju *DataProcess1*



Slika 35. Dijagram objekata za operaciju DataProcess1 (nastavak)

Za svaku operaciju transformacije entiteta kreira se odgovarajući dijagram proste operacije transformacije. U nastavku je dat primer za *TransformationData1*.



Slika 36. Dijagram proste operacije transformacije podataka za TransformData1

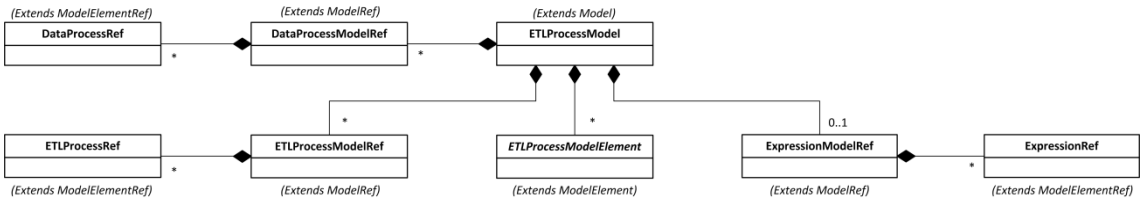
6.2. Jezik za specifikaciju toka izvršavanja ETL procesa (ETL-P)

Pošto je data specifikacija složenih operacija transformacija potrebno je da se definiše redosled njihovog izvršavanja, odnosno semantika izvršavanja ETL procesa. Naime, neke složene operacije transformacije se mogu izvršavati paralelno, dok su neke od njih međusobno zavisne što podrazumeva njihovo sekvencijalno izvršavanje. Isto tako, potrebno je omogućiti i specifikaciju različitih uslova koji utiču na tok izvršavanja ETL procesa.

6.2.1. Apstraktna sintaksa

Specifikacija apstraktne sintakse kao jezičkog modela ovde se zasniva na metamodelu (slika 37 i slika 38) kojim se opisuju specifični koncepti ETL-P jezika. Imajući u vidu proširivost i fleksibilnost arhitekture ETL procesa, specifični koncepti uvedeni su proširenjem osnovnih koncepata (slika 20, strana 97).

Model toka izvršavanja ETL procesa (*ETLProcessModel*) i njegovi elementi (*ETLProcessModelElement*) modelovani su kao podklase osnovnih koncepata *Model* i *ModelElement*, tim redom. Kao proširenja osnovnih koncepata (*ModelRef* i *ModelElementRef*), uvedeni su i koncepti *DataProcessModelRef* i *DataProcessRef*, zatim *ETLProcessModelRef* i *ETLProcessRef*, i na kraju *ExpressionModelRef* i *ExpressionRef*. Ovim konceptima omogućeno je referenciranje specifičnih modela i njihovih elemenata.

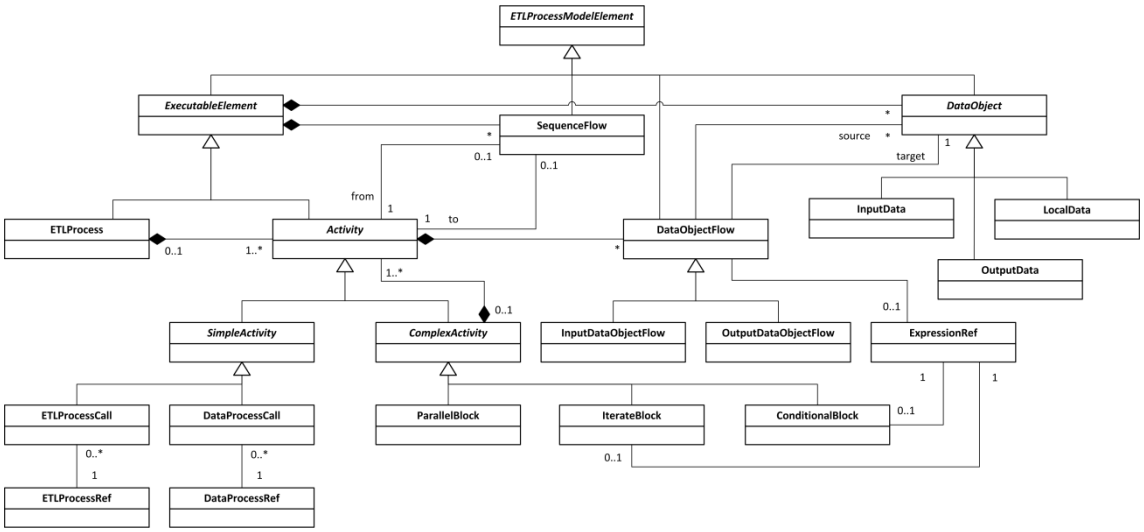


Slika 37. ETL-P metamodel (proširenja osnovnih koncepata)

Uvođenjem koncepta *DataProcessModelRef* omogućeno je referenciranje različitih modela procesa obrade podataka, dok je konceptom *DataProcessRef* omogućeno referenciranje njihovih elemenata (*DataProcess*, slika 22, strana 99). Isto tako, konceptima *ETLProcessModelRef* i *ETLProcessRef* referenciraju se modeli ETL procesa i njihovi elementi (*ETLProcess*, slika 38).

Konceptima *ExpressionModelRef* i *ExpressionRef* referencira se odgovarajući model izraza i njegovi elementi sa ciljem da se omogući specifikacija različitih uslova i ograničenja u vezi sa tokom izvršavanja ETL procesa.

Pored navedenih, uvedeni su, proširenjem osnovnog koncepta *ETLProcessModelElement* i sledeći koncepti: *ExecutableElement*, *DataObject*, *SequenceFlow* i *DataObjectFlow*. Navedeni koncepti predstavljaju specijalizacije koncepta *ETLProcessModelElement*.



Slika 38. ETL-P metamodel

Za specifikaciju funkcionalnosti ETL procesa uvedene su klase *ETLProcess* i *Activity* koje su modelovane kao podklase klase *ExecutableElement*. Apstraktna klasa *Activity* je uvedena kako bi se predstavile aktivnosti od kojih se ETL proces (*ETLProcess*) sastoji. Ove aktivnosti se mogu klasifikovati na proste i kompleksne što je modelovano apstraktnim klasama *SimpleActivity* i *ComplexActivity*. Njihovom specijalizacijom uvode se specifične aktivnosti kojima se preciznije opisuju aktivnosti iz domena ETL procesa.

Osnovna aktivnost koja se sprovodi tokom ETL procesa je transformacija podataka, odnosno proces obrade podataka. Predstavljanje ovih aktivnosti je omogućeno klasom *DataProcessCall* koja je modelovana kao podklasa klase *SimpleActivity*. Koncept *DataProcessRef* se koristi za referenciranje procesa obrade podataka (*DataProcess*) koji su definisani u ETL-O (slika 22, strana 99). Sa druge

strane, referenciranje ETL procesa (*ETLProcess*, slika 38) omogućeno je uvođenjem koncepta *ETLProcessRef*.

Kako bi se omogućila specifikacija redosleda izvršavanja aktivnosti ETL procesa potrebno je da se obezbede odgovarajući koncepti kojima se mogu predstaviti tri osnovne upravljačke strukture: (1) sekvenca, (2) selekcija i (3) iteracija. Pored toga, kako bi se omogućila optimizacija izvršavanja, potrebno je uvesti koncept konkurentnog izvršavanja aktivnosti. U skladu sa ovim, uveden je koncept *SequenceFlow*, ali i skup specifičnih kompleksnih aktivnosti kao što su *ParallelBlock*, *IterateBlock*, *ConditionalBlock*. Koncept *SequenceFlow* je uveden sa ciljem da se omogući predstavljanje osnovne upravljačke strukture tj. sekvenca aktivnosti. Uzimajući u obzir postavljeni cilj da se omogući jednostavno predstavljanje i lako razumevanje ETL procesa (prilagođeno domenskim ekspertima) uvedene su i odgovarajuće blok upravljačke strukture. Ove strukture su modelovane kao podklase apstraktne klase *ComplexActivity* i kao takve mogu da sadrže druge aktivnosti, bilo da su one proste ili složene. Njihovom hijerarhijskom kompozicijom omogućeno je predstavljanje semantike toka izvršavanja celokupnog ETL procesa. Korišćenjem ovih specifičnih upravljačkih struktura omogućena je specifikacija različitih ograničenja u pogledu izvršavanja ETL procesa. Na primer, semantika koncepta *ParallelBlock* je da se sve sadržane aktivnosti izvršavaju konkurentno, dok je semantika *IterateBlock* koncepta da se sve sadržane aktivnosti izvršavaju dokle god je ispunjen uslov iteriranja. Za specifikaciju uslova (logičkih izraza) u ovim različitim aktivnostima (npr. *IterateBlock* i *ConditionalBlock*) predlaže se korišćenje formalnog jezika ETL-E (*ExpressionRef*).

Predstavljanje ulaza i izlaza aktivnosti, kao i stanja procesa i kompleksnih aktivnosti omogućena je apstraktnom klasom *DataObject*, odnosno odgovarajućim podklasama *InputData*, *OutputData* i *LocalData*.

Tokovi podataka između različitih aktivnosti ETL procesa predstavljaju se apstraktnom klasom *DataObjectFlow*. Odgovarajuće podklase su uvedene sa ciljem da se jasno razdvoje ulazni i izlazni tokovi podataka određene aktivnosti. Svaka

aktivnost može imati više ulaznih i izlaznih argumenata, pri čemu za svaki mora postojati odgovarajući ulazni ili izlazni tok. Polazište (*source*) i odredište (*target*) toka podataka modelovano je odgovarajućim vezama sa klasom *DataObject*. Navedenim kardinalnostima su uvedena ograničenja kojima se propisuje da mora postojati najmanje jedno polazište, kao i da mora postojati tačno jedno odredište. Ograničenje u pogledu odredišta je uvedeno kako bi se kompleksnost specifikacije održala na niskom nivou.

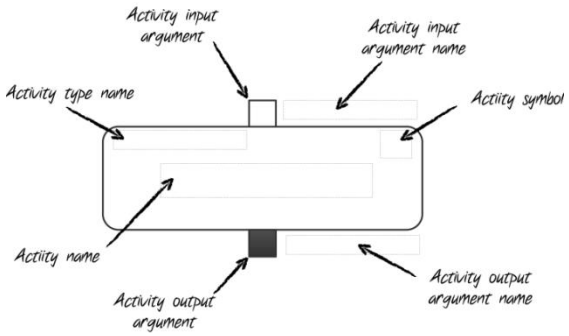
Opis toka podataka se može dalje obogatiti i odgovarajućim transformacijama. Naime, u određenim situacijama neophodno je izvršiti i odgovarajuće transformacije podataka na putu od izvora do odredišta. Potrebno je naglasiti da je navođenje transformacije obavezno u slučaju da tok podataka ima višestruke izvore. Za specifikaciju ovih transformacija predviđeno je korišćenje predloženog jezika ETL-E (*ExpressionRef*).

6.2.2. Konkretna sintaksa

Za predstavljanje koncepata ETL-P jezika je usvojena grafička notacija. Shodno tome, u nastavku su predstavljeni odgovarajući grafički elementi i date su napomene u vezi njihovog korišćenja.

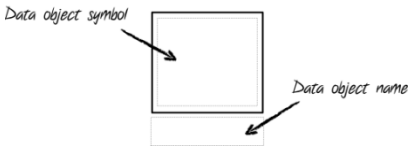
Predstavljanje prostih aktivnosti ETL procesa (*SimpleActivity*), omogućeno je odgovarajućim grafičkim elementom (slika 39). U centralnom delu se prikazuje naziv aktivnosti (eng. *activity name*), dok se u gornjem levom uglu prikazuje naziv tipa aktivnosti (eng. *activity type name*). Gornji desni ugao grafičkog elementa je rezervisan za odgovarajući grafički simbol (eng. *activity symbol*) kojim se bliže određuje tip proste aktivnosti.

Kao sastavni deo ovog grafičkog elementa, uvedeni su i elementi za predstavljanje ulaznih i izlaznih argumenata operacije (koncepti *InputData* i *OutputData*). U pitanju su mali kvadrati na spoljnoj ivici, pri čemu se izlazni argument (eng. *activity output argument*) predstavlja crnim kvadratom, a ulazni argument (eng. *activity input argument*) belim kvadratom. Prikazivanje naziva ovih argumenata (eng. *activity input argument name* i *activity output argument name*) nije obavezno.



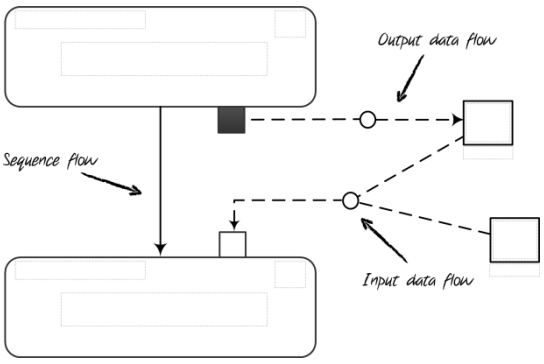
Slika 39. Grafički element za predstavljanje prostih aktivnosti

Predstavljenim grafičkim elementom (slika 40) omogućeno je prikazivanje podataka procesa (*DataObject*). U centralnom delu ovog grafičkog elementa prikazuje se odgovarajući grafički simbol (eng. *data object symbol*), dok se u donjem delu prikazuje naziv (eng. *data object name*).



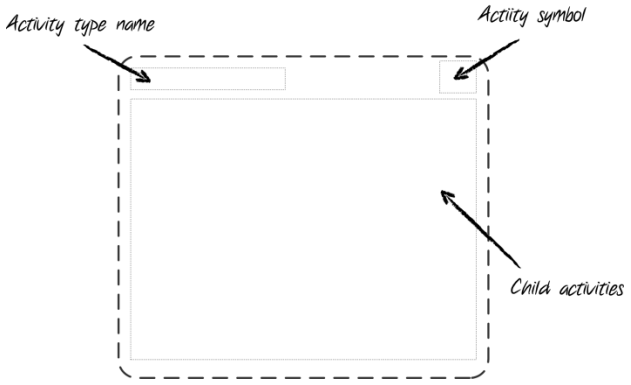
Slika 40. Grafički element za predstavljanje podataka procesa

Za predstavljanje koncepta *SequenceFlow* predviđeno je korišćenje usmerene linije (eng. *sequence flow*), dok je za predstavljanje koncepata *InputDataObjectFlow* (eng. *input data flow*) i *OutputDataObjectFlow* (eng. *output data flow*) predviđeno korišćenje usmerenih isprekidanih linija (slika 41) sa odgovarajućim krugom u sredini (ispunjen krug znači da je zadata transformacija).



Slika 41. Notacija za predstavljanje sekvence aktivnosti i toka podataka

U nastavku je opisan grafički element (slika 42) za predstavljanje kompleksnih aktivnosti ETL procesa (*ComplexActivity*).

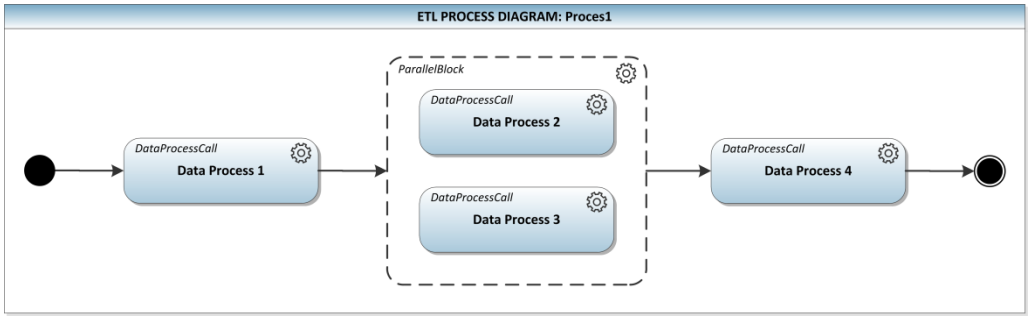


Slika 42. Grafički element za predstavljanje kompleksnih aktivnosti

U gornjem levom uglu prikazuje se naziv tipa aktivnosti (eng. *activity type name*), odnosno naziv koncepta koji se grafičkim elementom predstavlja, dok se u gornjem desnom uglu prikazuje odgovarajući grafički simbol (eng. *activity symbol*). Centralni deo grafičkog elementa (eng. *child activities*) je rezervisan za prikaz ugnježdenih grafičkih elemenata kojima se prikazuju pripadajuće aktivnosti.

6.2.3. Primer

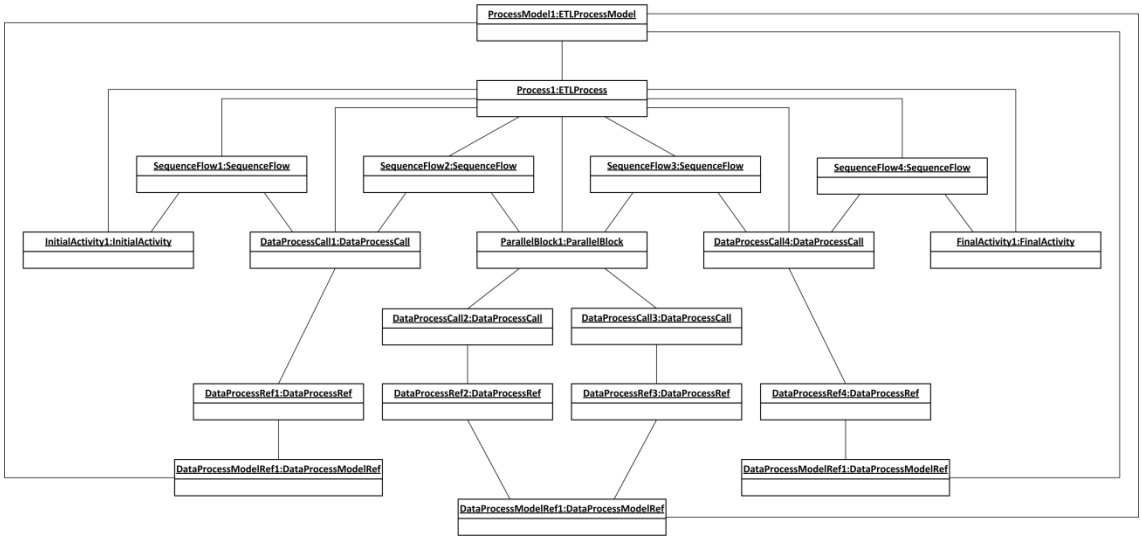
U ovom delu rada je ilustrovano korišćenje jezika ETL-P. Predstavljen je primer kojim je opisan tok izvršavanja za ETL proces *Proces1*.



Slika 43. Dijagram toka izvršavanja za Proces1

Predstavljeni ETL proces sastoji se od četiri aktivnosti kojima se reprezentuju odgovarajući procesi obrade podataka. Tok izvršavanja započinje sa aktivnošću *DataProcess1*, zatim se nastavlja konkurentnim izvršavanjem aktivnosti *DataProcess2* i *DataProcess3*, a završava se izvršavanjem aktivnosti *DataProcess4*.

U nastavku je dat model toka izvršavanja koji je reprezentovan dijagramom objekata.



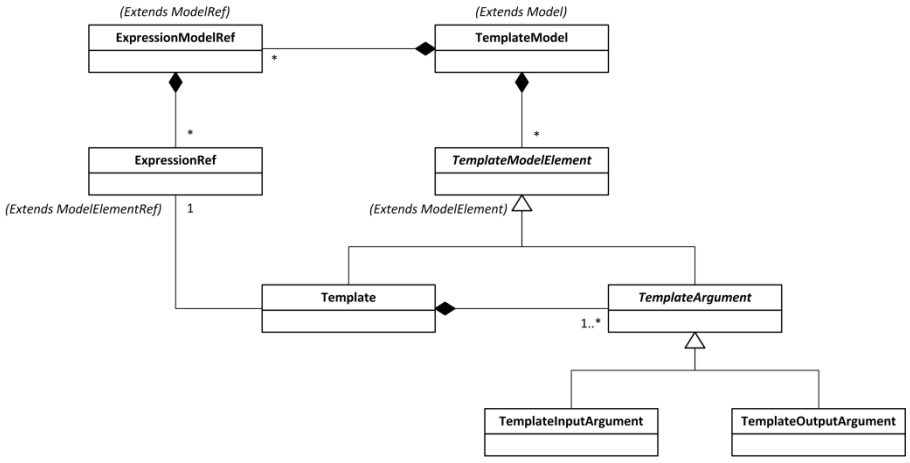
Slika 44. Dijagram objekata za tok izvršavanja Proces1

6.3. Jezik za specifikaciju šablona operacija transformacija (ETL-T)

U ovom delu rada je predstavljen jezik za specifikaciju šablona (eng. *template*) operacija transformacija. Osnovni cilj koji se želi postići ovim jezikom je da se omogući definisanje novih operacija transformacija polja entiteta koje se mogu višestruko koristiti.

6.3.1. Apstraktna sintaksa

Apstraktna sintaksa, odnosno koncepti predloženog jezika ETL-T i njihovi međusobni odnosi predstavljeni su metamodelom koji je reprezentovan korišćenjem UML dijagrama klasa (slika 45).



Slika 45. ETL-T metamodel

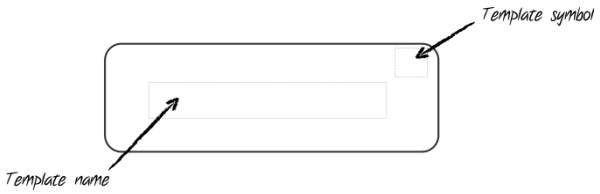
Model šablona operacija transformacija predstavlja sa klasom *TemplateModel*, dok se koncepti modela koji čine njegov sadržaj predstavljaju kao podklase klase *TemplateModelElement*. Za referenciranje modela izraza i njegovih elemenata uvedene su klase *ExpressionModelRef* i *ExpressionRef*. Navedene klase su modelovane kao podklase baznih klasa ETL procesa (slika 20, strana 97).

Šabloni operacija transformacija predstavljaju se klasom *Template*, dok se odgovarajući argumenti operacije predstavljaju klasom *TemplateArgument*. Razlikovanje ulaznih i izlaznih argumenata operacije omogućeno je klasama *InputTemplateArgument* i *OutputTemplateArgument*.

6.3.2. Konkretna sintaksa

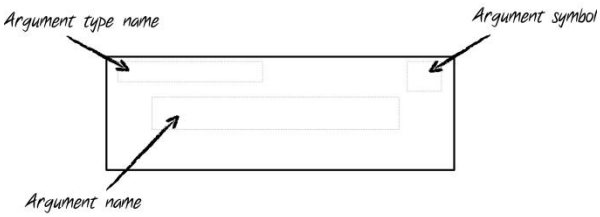
Za predstavljanje koncepata ETL-T jezika je usvojena grafička notacija. Shodno tome, u nastavku su predstavljeni odgovarajući grafički elementi i date su napomene u vezi njihovog korišćenja.

Šabloni operacija transformacija predstavljaju se grafičkim elementom koji je prikazan na slici (slika 46). U centralnom delu ovog grafičkog elementa navodi se naziv šablona (eng. *template name*), dok se u gornjem desnom uglu prikazuje odgovarajući grafički simbol (eng. *template symbol*).



Slika 46. Grafički element za predstavljanje šablona operacija transformacija

Za predstavljanje argumenata operacija transformacija (koncepti *TemplateInputArgument* i *TemplateOutputArgument*) obezbeđen je poseban grafički element (slika 47). Naziv argumenta se navodi u centralnom delu (eng. *argument name*), dok se u gornjem levom uglu navodi tip argumenta (eng. *argument type name*) kako bi se razlikovali ulazni i izlazni argumenti operacije. Gornji desni ugao grafičkog elementa je rezervisan za prikazivanje odgovarajućeg grafičkog simbola (eng. *argument symbol*).

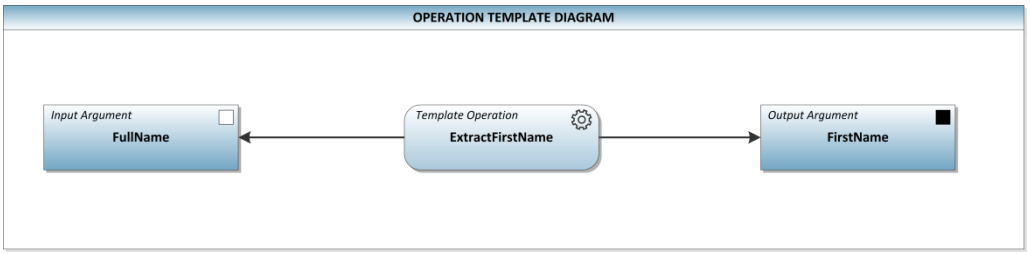


Slika 47. Grafički element za predstavljanje argumenata operacija transformacija

Povezivanje šablona operacije transformacije i njenih argumenata omogućeno je usmerenim linijama, pri čemu je smer od grafičkog elementa kojim se predstavlja šablon operacije ka elementu kojim se predstavlja argument (bilo ulazni bilo izlazni).

6.3.3. Primer

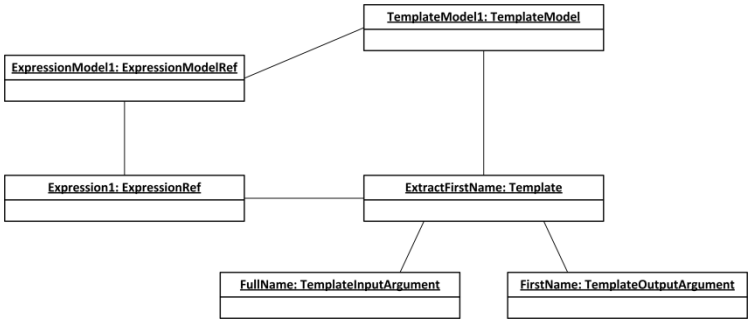
U ovom delu rada je ilustrovano korišćenje jezika ETL-T. Definisan je šablon operacije transformacije *ExtractFirstName* kojim je omogućena ekstrakcija imena iz punog naziva.



Slika 48. Šablon operacije ExtractFirstName

Šablonom su predviđena dva argumenta *FullName* i *FirstName*, pri čemu je prvi ulazni, a drugi izlazni argument. Semantika operacije je data odgovarajućim izrazom koji je iskazan korišćenjem jezika ETL-E.

U nastavku je dat konkretan model šablona reprezentovan dijagramom objekata.



Slika 49. Dijagram objekata za šablon operacije ExtractFirstName

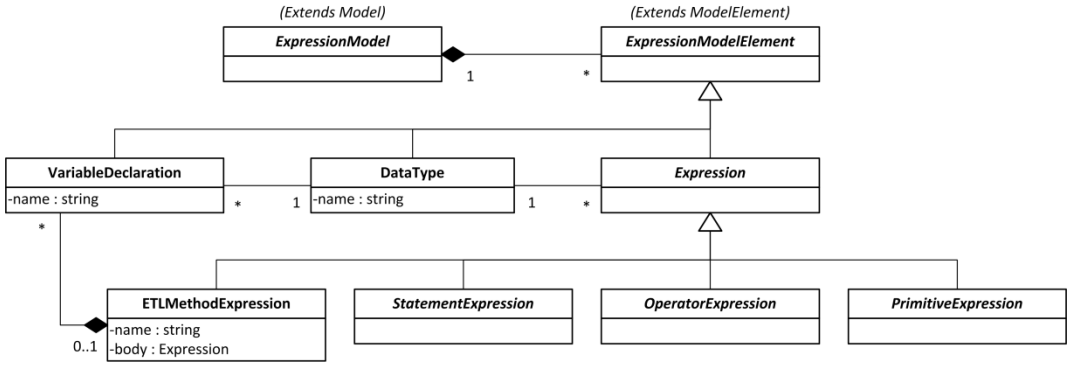
6.4. Jezik za specifikaciju izraza (ETL-E)

ETL-E jezik je uveden sa ciljem da se omogući formalna specifikacija semantike transformacija podataka, kao i različitih ograničenja i uslova u vezi sa izvršavanjem ETL procesa.

6.4.1. Apstraktna sintaksa

Apstraktna sintaksa ETL-E jezika definisana je kao metamodel u formi UML dijagrama klasa (slika 50). Model izraza predstavlja se klasom *ExpressionModel*, dok se koncepti modela koji čine njegov sadržaj predstavljaju kao podklase klase *ExpressionModelElement*. Navedene klase su modelovane kao podklase baznih klasa ETL procesa (slika 20, strana 97).

Prikazani metamodel ETL-E jezika definiše različite vrste izraza: *ETLMethodExpression*, *StatementExpression*, *OperatorExpression* i *PrimitiveExpression*. Nadklasa za sve izraze je apstraktna klasa *Expression*. Klasa *VariableDeclaration* reprezentuje povezivanje imena promenljive sa tipom (*DataType*) kojim se reprezentuju predefinisani tipovi podataka. Rezultat evaluacije svakog izraza mora biti određenog tipa (*DataType*).

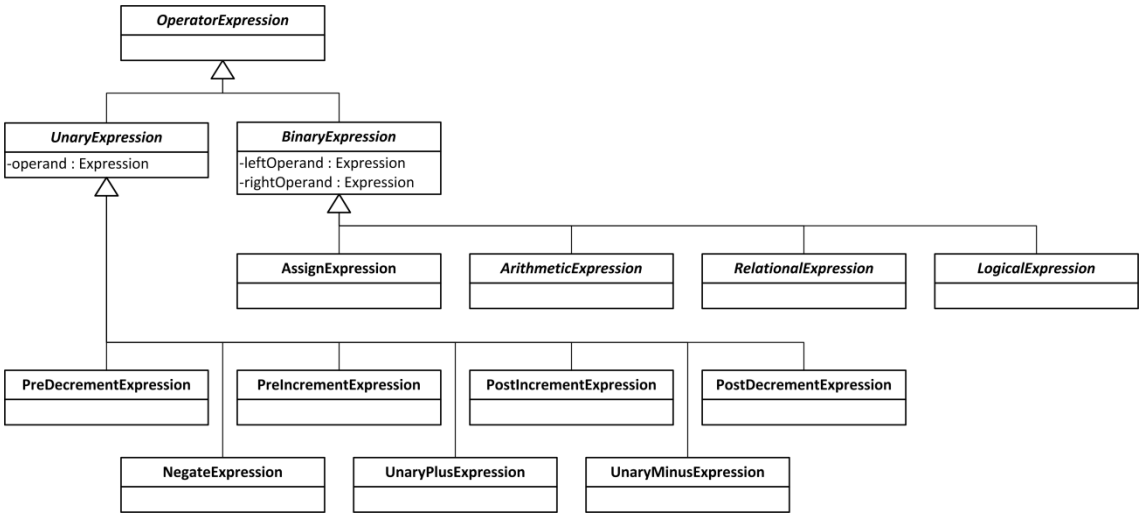


Slika 50. ETL-E metamodel

Klasa *ETLMethodExpression* reprezentuje metode definisane u ETL-E jeziku i na koje ukazuju koncepti specificiranih jezika ETL procesa.

Klasa *OperatorExpression* reprezentuje složeni izraz koji se može formirati od prostih (*PrimitiveExpression*) i drugih složenih izraza. S obzirom na to da je jedna od fundamentalnih karakteristika izraza mogućnost da sadrže izraze kao

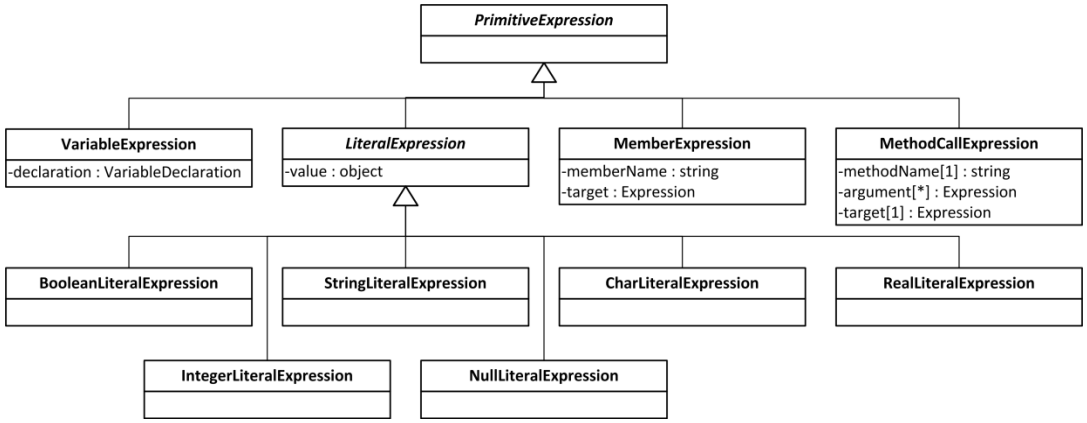
operande, metamodelom su predviđene dve klase izraza sa operatorima: izrazi koji sadrže jedan operand (*UnaryExpression*) i izrazi koji sadrže dva operanda (*BinaryExpression*), a na koje ukazuju atributi *leftOperand* i *rightOperand*.



Slika 51. Specijalizacije klase OperatorExpression

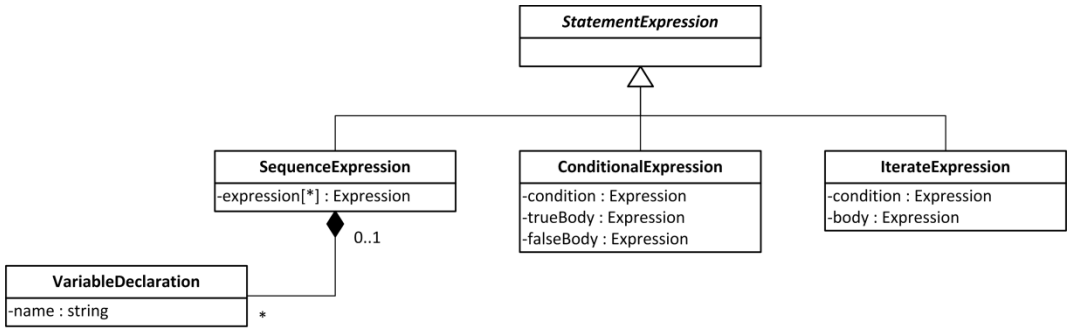
Kriterijum specijalizacije klase *BinaryExpression* u podklase *AssignExpression, ArithmeticExpression, RelationalExpression i LogicalExpression,* je vrsta operatora (operator dodeljivanja, aritmetički operatori, relacioni operatori ili logički operatori) koji se primenjuju u odgovarajućim izrazima kao instancama navedenih podklasa. Isti kriterijum, tj. vrsta unarnog operatora, je korišćen za specijalizaciju klase *UnaryExpression* u podklase: *NegateExpression, PreDecrementExpression*, *PostDecrementExpression*, *PreDecrementExpression*, *PreIncrementExpression*, *UnaryPlusExpression* i *UnaryMinusExpression*.

Klasa *PrimitiveExpression* specijalizuje se u sledeće podklase:*VariableExpression*, *LiteralExpression, MethodCallExpression i MemberExpression.* Instance klasa *VariableExpression* referenciraju deklaracije promenljive (Variable*Declaration*). Klasom *MethodCallExpression* reprezentuju se pozivi metoda, a klasom *MemberExpression* pristup atributima i svojstvima objekata.



Slika 52. Specijalizacije klase PrimitiveExpression

Klasa *LiteralExpression* reprezentuje konstante (numeričke, znakovne, itd.).



Slika 53. Specijalizacije klase StatementExpression

Predstavljanje osnovnih upravljačkih struktura (sekvenca, selekcija i iteracija) omogućeno je apstraktnom klasom *StatementExpression*, odnosno njenim podklasama *SequenceExpression*, *ConditionalExpression* i *IterateExpression*.

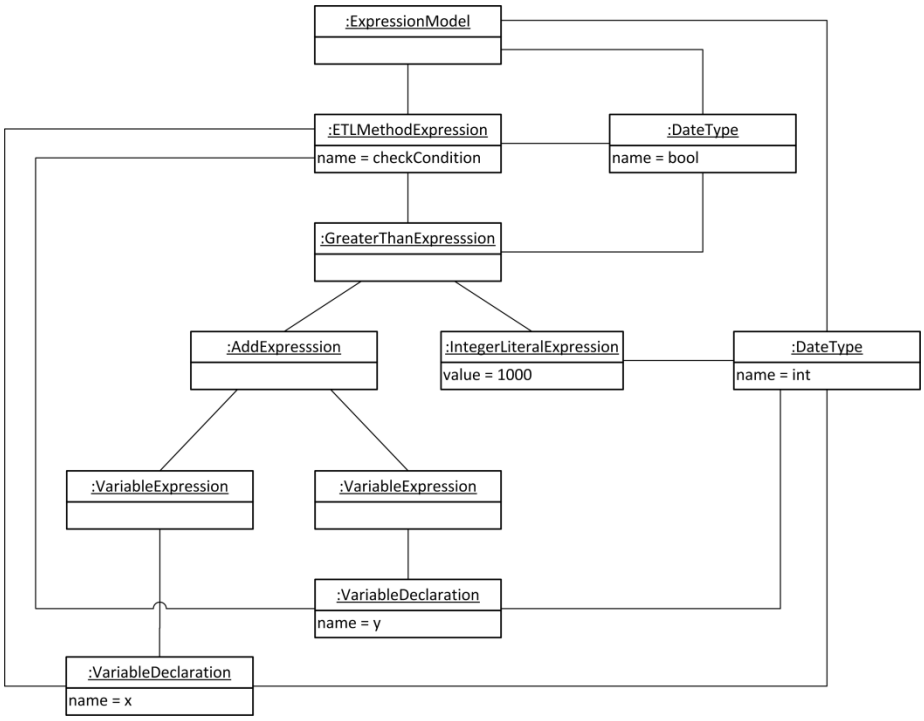
6.4.2. Konkretna sintaksa

Konkretna sintaksa jezika je u tekstualnom obliku kao što je i prikazano u primeru na slici (slika 54). Prikazana je metoda *checkCondition* čiji je rezultat tipa *bool*. Definisana su dva ulazna parametra *x* i *y*, oba tipa *int*. Telo metode je predstavljeno odgovarajućim izrazom i navedeno je između vitičastih zagrada.

```
ETLMethod checkCondition : bool
      input x : int
      input y : int
{
      x + y > 1000
}
```

Slika 54. Primer konkretne sintakse ETL-E jezika

Konkretna sintaksa data u primeru, reprezentovana je preko dijagrama objekata.



Slika 55. Dijagram objekata za primer konkretne sintakse ETL jezika

7. MODELOVANJE ETL PROCESA KORIŠĆENJEM DEFINISANIH DOMENSKO-SPECIFIČNIH JEZIKA

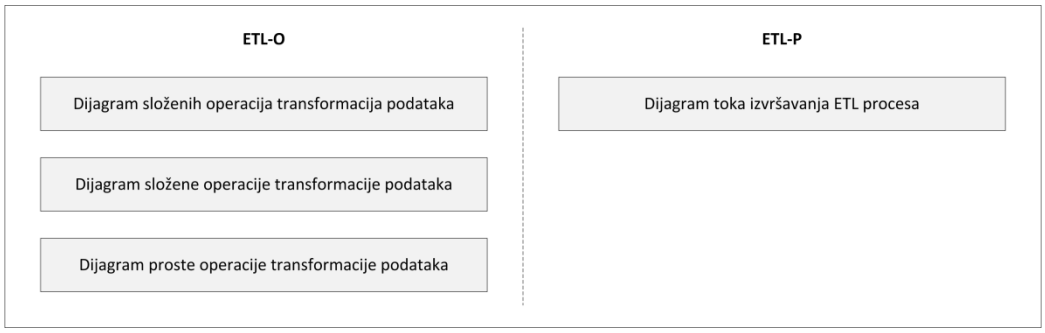
Predloženi pristup modelovanju ETL procesa podrazumeva kreiranje posebnih modela za svaki aspekt ETL procesa. Razdvajanjem različitih aspekata u posebne modele značajno se smanjuje složenost modelovanja ETL procesa. Modeli se formiraju u skladu sa definisanim specifikacijama ETL procesa, tj. korišćenjem koncepata odgovarajućeg domensko specifičnog jezika (na primer, ETL-O za modelovanje operacija transformacija podataka, kao osnovnih aktivnosti ETL procesa, kao i ETL-P za modelovanje redosleda izvršavanja ovih aktivnosti). Redosled formiranja ovih modela nije unapred zadat i u velikoj meri zavisi od iskustva projektanta i njegovog poznavanja realnog sistema.

Međutim, modeli koji se formiraju u skladu sa predloženom ETL-O specifikacijom mogu biti veoma složeni (što svakako zavisi od kompleksnosti realnog sistema koji se modeluje, tj. broja neophodnih transformacija podataka), te je potrebno omogućiti kreiranje modela na različitim apstraktnim nivoima kroz postepeno uvođenje detalja. Hijerarhijski opis procesa obrade podataka je postignut uvođenjem skupa dijagrama kao grafičke reprezentacije modela. Na vrhu hijerarhije nalazi se dijagram složenih operacija transformacija podataka, a na sledećim nižim nivoima uvode se detaljniji dijagrami: dijagram složene operacije transformacije i dijagram proste operacije transformacije.

Shodno navedenom, predloženim pristupom modelovanju ETL procesa uvedena su četiri tipa dijagrama: (1) dijagram složenih operacija transformacija podataka, (2) dijagram složene operacije transformacije podataka, (3) dijagram proste operacije transformacije podataka i (4) dijagram izvršavanja ETL procesa.

Svaki dijagram sadrži grafičke elemente koji reprezentuju koncepte definisanih domensko specifičnih jezika (prva tri reprezentuju koncepte ETL-O jezika, dok se četvrtim reprezentuju koncepti ETL-P jezika). U stvari za svaki predloženi DSL

definisana je odgovarajuća konkretna sintaksa (poglavlje 6) koja se koristi za jasnu grafičku predstavu modela i kao takva je izuzetno pogodna za modelovanje ETL sistema.



Slika 56. Dijagrami predloženog pristupa modelovanju ETL procesa

U nastavku ovog poglavlja dat je primer kojim se ilustruje formiranje modela u grafičkoj notaciji korišćenjem predloženih dijagrama. Razmatran je razvoj ETL procesa u kontekstu sistema skladišta podataka Fakulteta organizacionih nauka kojim se integrišu podaci različitih službi fakulteta.

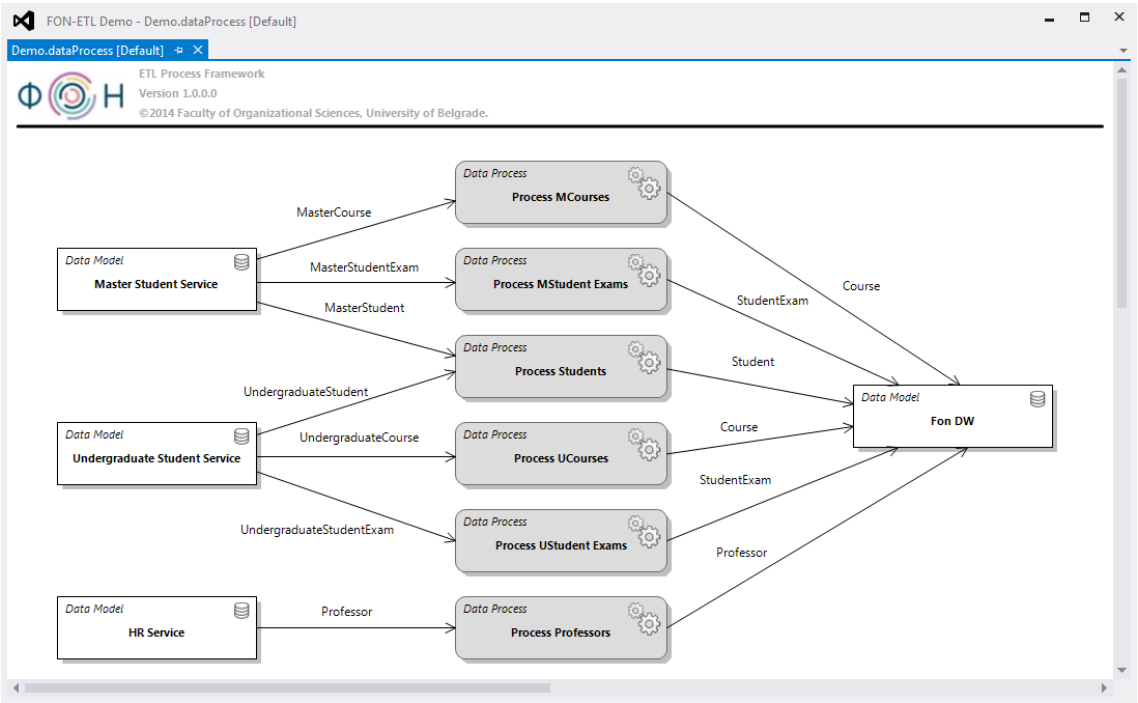
Dijagram složenih operacija transformacija podataka

Dijagramom složenih operacija transformacija podataka predstavljaju se složene operacije transformacije podataka, tj. procesi obrade podataka od kojih se ETL procesi sastoje. Za svaki proces obrade podataka se predstavljaju odgovarajuća skladišta podataka (izvorna i ciljna) iz kojih se podaci ekstrahuju, odnosno u koja se podaci učitavaju.

Skladišta podataka se predstavljaju pravougaonikom, dok se procesi obrade podataka predstavljaju pravougaonikom sa zaobljenim ivicama. Ulazi i izlazi procesa obrade podataka predstavljaju se usmerenim linijama, pri čemu linije koje predstavljaju ulaz u proces obrade podataka imaju smer od skladišta podataka ka procesu obrade podataka, dok linije kojima se predstavljaju izlazi imaju smer od procesa obrade podataka ka skladištu podataka. Iznad svake linije se navode nazivi entiteta kako bi se bliže odredili ulazi i izlazi procesa obrade podataka.

Primer dijagrama složenih operacija transformacija podataka dat je na slici (slika 57). Na dijagramu je predstavljen reprezentativan skup procesa obrade podataka

koji su neophodni u kontekstu razvoja posmatranog sistema skladišta podataka. Na primer, predstavljen je proces *ProcessStudents* kojim se obrađuju podaci o studentima sa različitih nivoa studija. Ovim procesom ekstrahuju se podaci o studentima (predstavljeni entitetima *UndergraduateStudent* i *MasterStudent*) iz izvornih skladišta *UndergraduateStudentService* i *MasterStudentService*, zatim transformišu u odgovarajuće entitete *Student* i na kraju učitavaju u ciljno skladište *FonDW*. Slično se mogu opisati i ostali navedeni procesi obrade podataka.



Slika 57. Dijagram složenih operacija transformacija podataka

Svakako, u složenim sistemima skladišta podataka moguće je identifikovati veći broj procesa obrade podataka. Predstavljanje svih identifikovanih procesa obrade podataka na jednom dijagramu bi bilo nepraktično, jer bi dijagram bio nepregledan i veoma težak za razumevanje. Shodno tome, omogućeno je kreiranje većeg broja ovih dijagrama, a na projektantu je da odredi kriterijum grupisanja identifikovanih procesa obrade podataka.

Dijagram složene operacije transformacije podataka

Dijagramom složene operacije transformacije podataka (eng. *complex transformation operation diagram*) daje se semantika procesa obrade podataka. Na ovom nivou apstrakcije se definiše način na koji se podaci iz izvornih skladišta

podataka ekstrahuju, zatim transformišu i na kraju učitavaju u ciljno skladište podataka. Definišu se neophodne proste operacije transformacije podataka (uopšte operacije ekstrahovanja, operacije transformisanja i operacije učitavanja podataka) kao i redosled njihovog izvršavanja.

Za svaku prostu operaciju transformacije podataka definišu se odgovarajući argumenti, pri čemu se za operacije ekstrahovanja definiše samo izlazni, za operacije učitavanja samo ulazni, dok se za operacije transformacije definišu i ulazni i izlazni argumenti. Navedenim argumentima se definišu ulazi i izlazi operacije, odnosno entiteti koji učestvuju u transformaciji. Shodno tome, ove operacije se nazivaju *operacije transformacije entiteta* (eng. *entity transformation operation*).

Međuzavisnost, odnosno redosled izvršavanja navedenih prostih operacija transformacija podataka definiše se odgovarajućim tokovima podataka. Ovim tokovima podataka povezuje se izlazni argument jedne operacije sa ulaznim argumentom druge operacije (izlaz jedne operacije predstavlja ulaz u drugu operaciju).

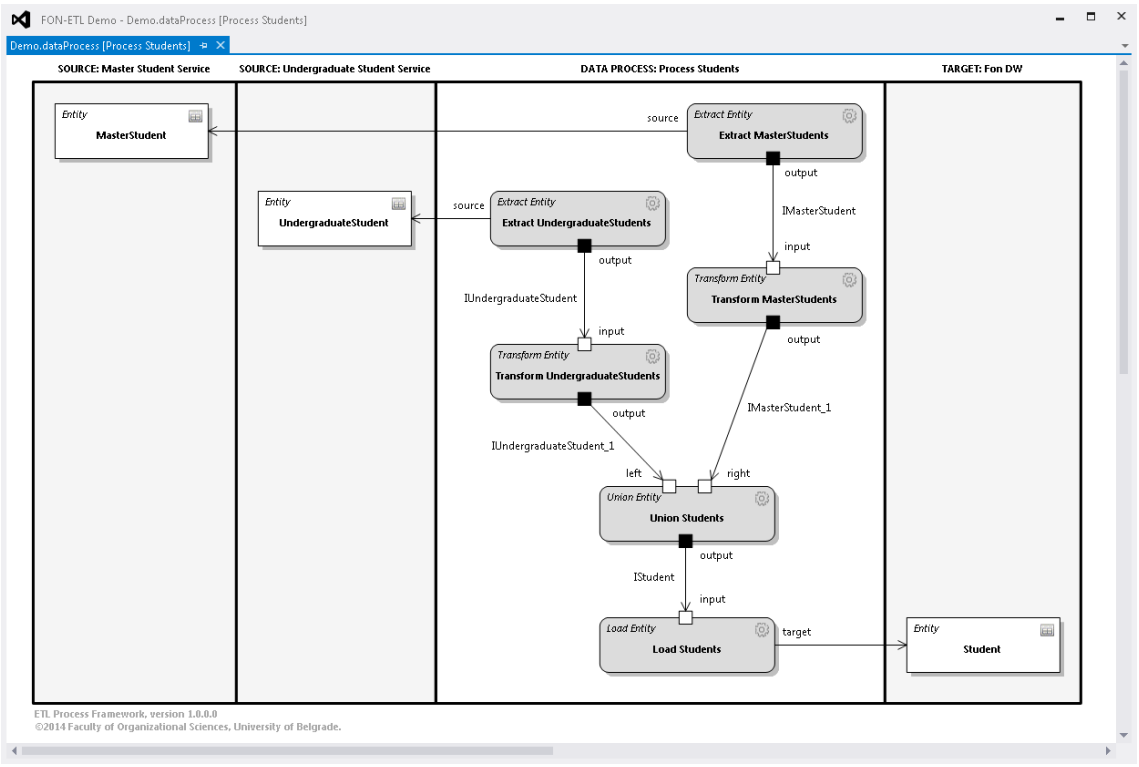
Pored prostih operacija transformacija podataka, na dijagramu se predstavljaju i odgovarajuća skladišta podataka (izvorna i ciljna skladišta podataka posmatranog procesa obrade podataka predstavljena na dijagramu složenih operacija transformacija podataka) zajedno sa njihovim elementima, tj. entitetima koji učestvuju u transformacijama.

Dijagram je organizovan na sledeći način: u levom delu dijagrama predstavljaju se izvorna skladišta podataka (označena sa *SOURCE*), u desnom ciljna skladišta podataka (označena sa *TARGET*), dok se u centralnom delu dijagrama (označen sa *DATA PROCESS*) predstavlja složena operacija transformacije podataka, tj. proste opercije transformacije podataka zajedno sa tokovima podataka koji ih povezuju.

Elementi izvornih i ciljnih skladišta podataka (entiteti koji učestvuju u transformacijama) se predstavljaju pravougaonicima, dok se proste operacije transformacije podataka predstavljaju pravougaonicima sa zaobljenim ivicama.

Argumenti operacija transformacija se predstavljaju kvadratima (beli za ulazne argumente i crni za izlazne argumente), a tokovi podataka usmerenim linijama. Isto tako, usmerene linije (označene sa *source* ili *tagret*) se koriste za predstavljanje izvornih entiteta operacija ekstrahovanja, kao i ciljnih entiteta operacija učitavanja.

Na slici (slika 58) je prikazan dijagram složene operacije transformacije podataka kojim je data semantika procesa obrade podataka *ProcessStudents*. Izvršavanje predstavljenog procesa obrade podataka počinje operacijama *ExtractUndergraduateStudents* i *ExtractMasterStudents*, kojima se ekstrahuju podaci o studentima (predstavljeni entitetima *UndergraduateStudent* i *MasterStudent*) iz skladišta *UndergraduateStudentService* i *MasterStudentService*.



Slika 58. Dijagram složene operacije transformacije podataka za ProcessStudents

Rezultati izvršavanja ovih operacija predstavljeni su entitetima *IMasterStudent* i *IUndergraduateStudent*. Proces se nastavlja operacijama *TransformMasterStudents* i *TransformUndergraduateStudents*. Ovim operacijama se sprovode neophodne transformacije nad entitetima *IMasterStudent* i *IUndergraduateStudent* i kao rezultat generišu se entiteti *IMasterStudent_1* i *IUndergraduateStudent_1* koji

zajedno predstavljaju ulaz u operaciju *UnionStudents*. Rezultat izvršavanja ove operacije je predstavljen entitetima *IStudent* koji se dalje operacijom *LoadStudent* učitavaju u ciljno skladište *FonDW*. Potrebno je naglasiti da entiteti *IMasterStudent*, *IUndergraduateStudent, IMasterStudent_1*, *IUndergraduateStudent_1* i *IStudent* predstavljaju međurezultate posmatranog procesa obrade podataka.

Dijagram proste operacije transformacije podataka

Na najnižem nivou apstrakcije za opis procesa obrade podataka koristi se dijagram proste operacije transformacije podataka (eng. *simple operation transformation diagram*). Ovim dijagramom daje se semantika operacija transformacija entiteta, odnosno način na koji se ulazni podaci operacije transformišu u izlazne podatke.

Na ovom nivou apstrakcije definišu se *operacije transformacije polja entiteta* (eng. *entity field transformation operation*) pri čemu se svakom operacijom definiše transformacija jednog ili više polja ulaznih entiteta u odgovarajuće polje izlaznog entiteta. Slično kao i za operacije transformacije entiteta, za svaku operaciju se definišu odgovarajući ulazni i izlazni argumenti, pri čemu se umesto entiteta navode njihova polja.

Semantika ovih operacija je od posebnog interesa. Potrebno je opisati na koji način se vrši transformacija, odnosno koje je sve funkcije potrebno sprovesti nad podacima i u kom redosledu. S obzirom da se daje algoritam obrade na veoma niskom nivou apstrakcije, koristi se predloženi ETL-E jezik za koji je definisana odgovarajuća tekstualna notacija.

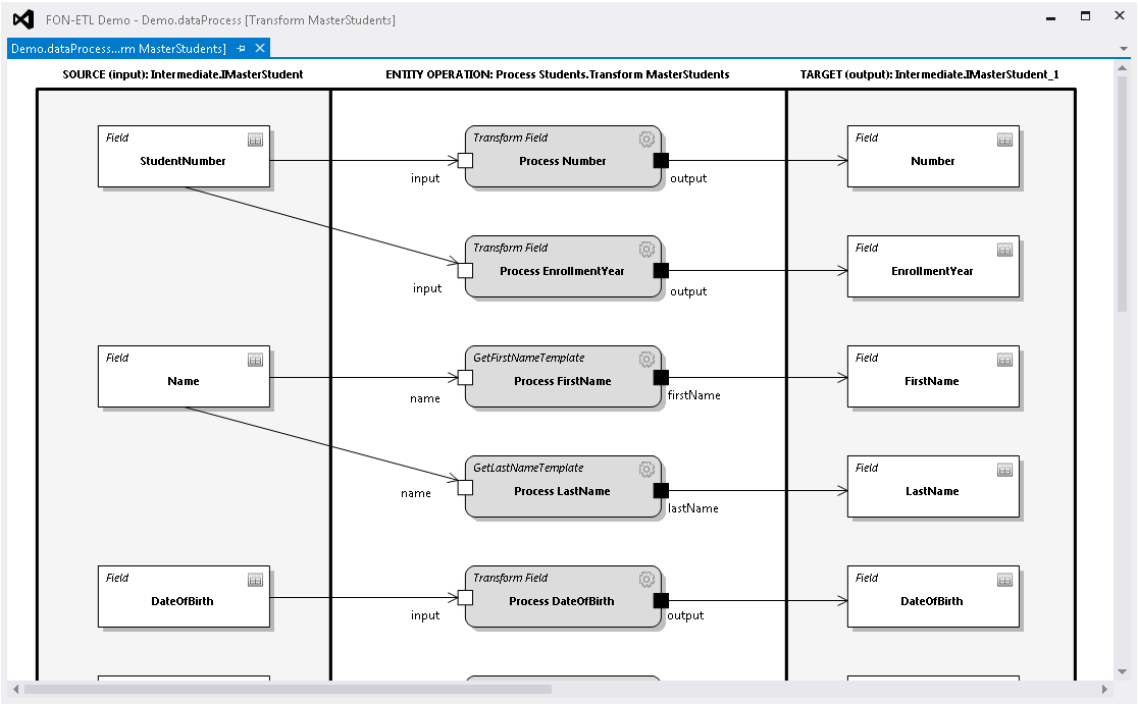
Na dijagramu se, pored operacija transformacija polja entiteta, predstavljaju i entiteti koji su prethodno identifikovani kao ulazi i izlazi posmatrane operacije, a za svaki identifikovani entitet se predstavljaju njegovi elementi, odnosno polja (eng. *field*) od kojih je sačinjen.

Dijagram je organizovan na sledeći način: u levom delu dijagrama predstavljaju se ulazni argumenti (odnosno ulazni entiteti, označeni sa *SOURCE*), u desnom izlazni argumenti (odnosno izlazni entiteti, označeni sa *TARGET*), dok se u centralnom

delu dijagrama (označen sa *ENTITY OPERATION*) predstavljaju operacije transformacije polja entiteta.

Polja entiteta se predstavljaju pravougaonicima, dok se operacije transformacije polja entiteta predstavljaju pravougaonicima sa zaobljenim ivicama. Ulazi i izlazi ovih operacija predstavljaju se usmerenim linijama, pri čemu linije kojima se predstavljaju ulazi imaju smer od polja ulaznog entiteta ka ulaznom argumentu operacije, dok linije kojima se predstavljaju izlazi operacije imaju smer od izlaznog argumenta operacije ka polju izlaznog entiteta.

Na slici (slika 59) je prikazan dijagram proste operacije transformacije za operaciju transformacije entiteta *TransformMasterStudents*.



Slika 59. Dijagram proste operacije transformacije za TransformMasterStudents

Na dijagramu (slika 59) su predstavljeni entiteti *IMasterStudent* i *IMasterStudent_1* sa pripadajućim poljima. Pored toga, za svako polje izlaznog entiteta (*IMasterStudent_1*) definisane su odgovarajuće operacije transformacije koje za ulaz mogu imati više polja ulaznog entiteta (*IMasterStudent*). Na primer, za operaciju *ProcessNumber* kao ulaz je definisano polje *Number*, dok je za operacije *ProcessFirstName* i *ProcessLastName* definisano polje *Name.*

Kao što je i prikazano, jedno polje ulaznog entiteta može učestvovati u više transformacija, dok jedno polje izlaznog entiteta može učestvovati u samo jednoj transformaciji. Bitno je naglasiti da se za svako izlazno polje mora definisati tačno jedna operacija transformacije. U slučaju da za izlazno polje ne postoji odgovarajuće ulazno polje definiše se operacija transformacije koja nema ulaz.

Za svaku navedenu operaciju transformacije polja entiteta se definiše odgovarajući izraz (eng. *expression*) kojim se opisuje neophodna transformacija. Na primer, za operaciju *ProcessNumber* je definisan izraz *Trim(Split(input, '/', 0))* što znači da se vrednost izlaznog parametra (*output*) dobija tako što se na vrednost ulaznog parametra (*input*) sukcesivno primenjuju funkcije *Split* i *Trim*. Semantika operacija *ProcessFirstName* i *ProcessLastName* je data referenciranim šablonima operacija *GetFirstNameTemplate* i *GetLastNameTemplate*, respektivno.

Dijagram toka izvršavanja ETL procesa

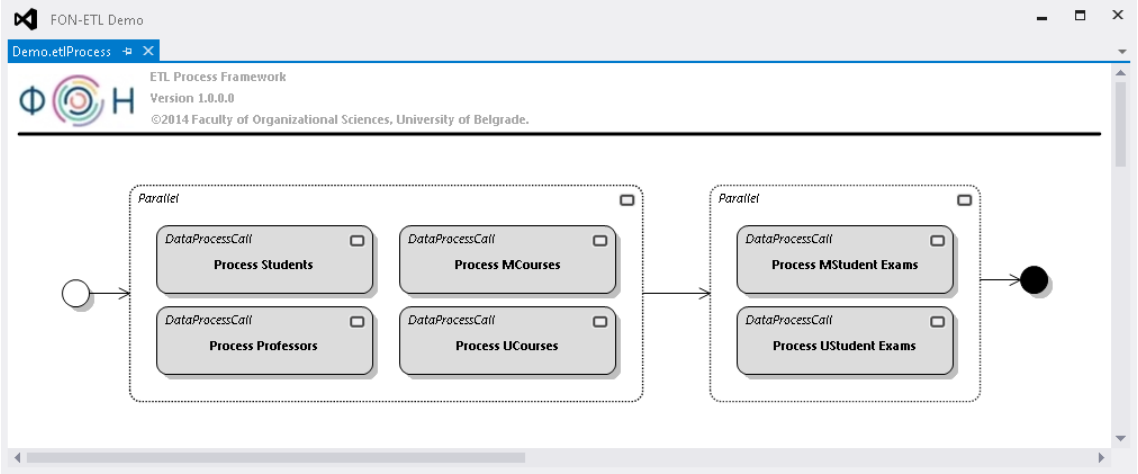
Dijagramom toka izvršavanja ETL procesa (eng. *ETL process execution diagram*) definišu se aktivnosti ETL procesa kao i redosled njihovog izvršavanja. Osnovna aktivnost koja se sprovodi tokom ETL procesa je složena operacija transformacije podataka, tj. proces obrade podataka. Isto tako, kao aktivnost ETL procesa se može predstaviti i ETL proces.

Za definisanje redosleda izvršavanja ovih aktivnosti koriste se osnovne upravljačke strukture: *sekvenca* (eng. *sequence*), *selekcija* (eng. *selection*) i *iteracija* (eng. *iteration*). Pored navedenih, uvedena je još jedna kako bi se omogućilo definisanje konkurentnog izvršavanja aktivnosti (eng. *parallel*).

Aktivnosti ETL procesa se predstavljaju pravougaonicima sa zaobljenim ivicama. Početak i kraj izvršavanja ETL procesa predstavlja se krugom, pri čemu se početak predstavlja belim, a kraj crnim krugom. Sekvenca se predstavlja usmerenom linijom, dok se ostale upravljačke strukture predstavljaju pravougaonicima sa zaobljenim ivicama (koristi se isprekidana linija).

Na slici (slika 60) je prikazan dijagram toka izvršavanja ETL procesa. Izvršavanje predstavljenog ETL procesa započinje paralelnim izvršavanjem aktivnosti

ProcessStudents, *ProcessProfessors*, *ProcessMCourses* i *ProcessUCourses*, a završava takođe, paralelnim izvršavanjem aktivnosti *ProcessMStudentExams* i *ProcessUStudentExams.*



Slika 60. Dijagram toka izvršavanja ETL procesa

8. IMPLEMENTACIJA ETL PLATFORME

Osnov uspešne realizacije softverskog rešenja predstavlja definisanje stabilne softverske arhitekture. Naime, realizacija softverskog sistema je veoma kompleksan i vremenski zahtevan zadatak. Da bi se omogućilo savladavanje pomenute složenosti predlaže se izdizanje nivoa apstrakcije i posmatranje softverskog sistema preko skupa komponenti. Ove komponente se dalje grupišu u odgovarajuća funkcionalna područja, odnosno slojeve čime se prikaz softverskog sistema značajno pojednostavljuje. Pored identifikacije strukturnih elemenata softverskog rešenja i njihovog grupisanja, softverskom arhitekturom se definiše i ponašenje, odnosno kolaboracija između ovih strukturnih elemenata. Za svaku komponentu se definiše odgovarajući interfejs preko koga se ostvaruje komunikacija *(Microsoft Patterns & Practices Team, 2009)*.

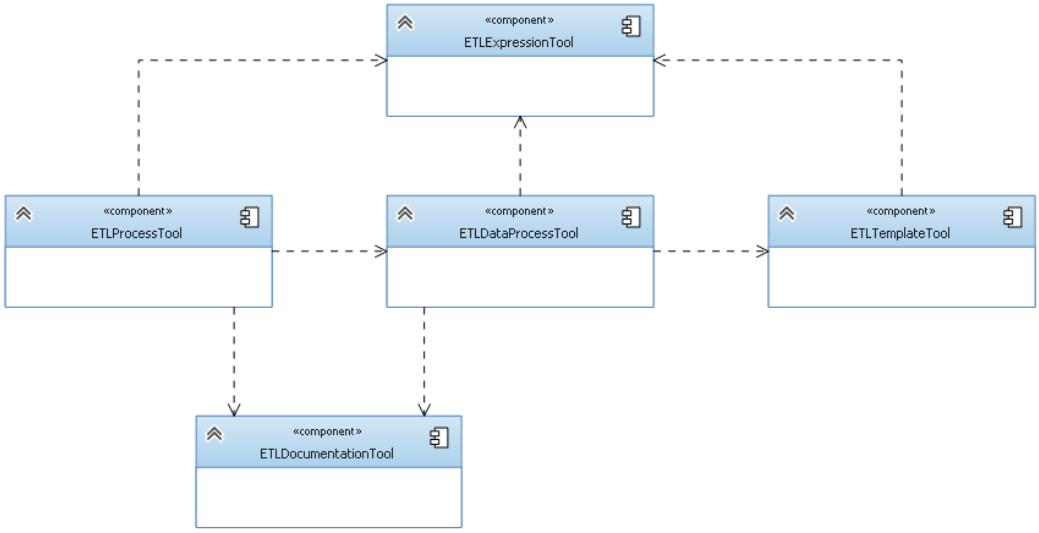
Uopšte, definisanje softverske arhitekture predstavlja proces strukturiranja softverskog rešenja tako da zadovolji različite tehničke i operativne zahteve. Proces definisanja softverske arhitekture karakteriše niz značajnih odluka kojima se utiče na kvalitet, performanse, održavanje i uopšte na celokupan uspeh softverskog rešenja.

Uzimajući u obzir prirodu problema koji se rešavaju ETL sistemima potrebno je definisati stabilnu i fleksibilnu arhitekturu kojom se sa jedne strane omogućavaju skalabilna softverska rešenja, odnosno rešenja visokih performansi, a sa druge strane softverska rešenja koja se mogu lako proširiti i nadograditi tako da odgovaraju novim zahtevima i promenama u okruženju.

U skladu sa navedenim zahtevima predložena je opšta fizička softverska arhitektura neophodna za realizaciju ETL platforme. Predloženom softverskom arhitekturom data je specifikacija softverskog rešenja kojim su definisane osnovne komponente ETL platforme, njihove odgovornosti i uloge, kao i ograničenja u pogledu interakcija koje se između njih mogu ostvariti. U nastavku sledi kratak opis i prikaz predložene softverske arhitekture (slika 61).

Slika 61. Opšta fizička softverska arhitektura ETL platforme

Osnovne komponente su podeljene u dva sloja: (1) razvojno okruženje (*ETLDevelopment*) i (2) izvršno okruženje (*ETLExecution*). Komponente razvojnog okruženja predstavljaju alate kojima je omogućeno formiranje modela u skladu sa odgovarajućim domensko-specifičnim jezikom. Izvršnom okruženju pripadaju komponente koje predstavljaju generatore za automatsko generisanje kôda iz formiranih modela, kao i komponente pomoću kojih je realizovan izvršni aplikacioni okvir.

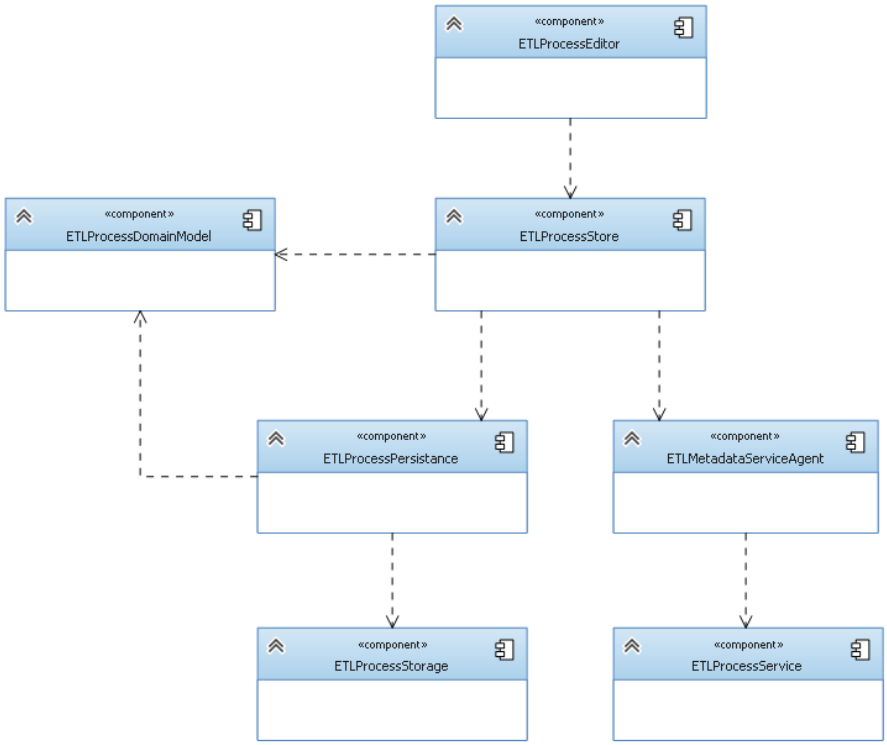


Slika 62. Dijagram komponenti za razvojno okruženje

Razvojno okruženje čine sledeće komponente (slika 62): *ETLDataProcessTool, ETLProcessTool*, *ETLExpressionTool* i *ETLTemplateTool*. Ovim komponentama predstavljeni su alati kojima se realizuju specificirani domensko-specifični jezici ETL-P, ETL-O, ETL-E i ETL-T (pogledati poglavlje 6). Pored navedenih, definisana je i *ETLDocumentationTool* komponenta kojom se predstavlja alat za izradu dokumentacije ETL procesa.

Kao što će i biti objašnjeno u poglavlju 9, navedeni alati su realizovani korišćenjem *Microsoft DSL Tools* alata. U nastavku je dat dijagram komponenti za

ETLProcessTool. Komponentom *ETLProcessEditor* realizuje se konkretna sintaksa ETL-P jezika (grafički elementi), dok se komponentom *ETLProcessDomainModel* realizuje apstratna sintaksa ETL-P jezika (koncepti jezika). Komponentom *ETLProcessStore* realizuju se osnovne funkcionalnosti u vezi sa kreiranjem konkretnog opisa toka izvršavanja ETL procesa. *ETLProcessEditor*, *ETLProcessStore*, *ETLProcessDomainMododel*, *ETLProcessPersistance* predstavljaju ekstenzije odgovarajućih komponenti koje su date u *Microsoft DSL Tools*. *ETLMetadataServiceAgent* je komponenta kojom se apstrahuje komunikacija sa rečnikom ETL resursa, dok je komponentom *ETLProcessPersistance* omogućeno snimanje i učitavanje modela iz lokalnog (privremenog) skladišta *ETLProcessStorage*.

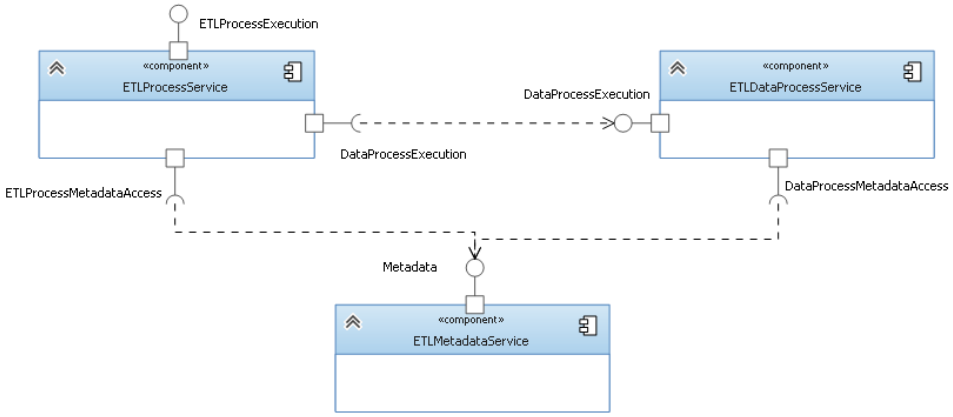


Slika 63. Dijagram komponenti za ETLProcessTool

Dijagrami komponenti za *ETLDataProcessTool*, *ETLExpressionTool* i *ETLTemplateTool* imaju strukturu kao dati dijagram komponenti (slika 63).

Izvršno okruženje čine sledeće komponente (slika 64): *ETLProcessService*, *ETLDataProcessService* i *ETLMetadataService*. Navedenim komponentama omogućeno je izvršavanje ETL procesa, pri čemu je izvršavanje procesa obrade

podataka realizovano komponentom *ETLDataProcessService*, dok je izvršavanje toka ETL procesa realizovano komponentom *ETLProcessService*. Komponenta *ETLMetadataService* je uvedena kako bi se obezbedila realizacija rečnika ETL resursa.



Slika 64. Dijagram komponenti za izvršno okruženje

9. ETL RAZVOJNO OKRUŽENJE

Razvojno okruženje je veoma bitan element predložene ETL platforme. Razvojnim okruženjem je potrebno obezbediti odgovarajuću infrastrukturu kojom će se podržati sprovođenje različitih aktivnosti tokom procesa razvoja softverskih rešenja i to na način koji obezbeđuje postizanje visokog stepena produktivnosti i efikasnosti. U osnovi, razvojnim okruženjem se predstavlja integrisani skup različitih alata koji se koriste tokom razvoja softverskog rešenja.

Razvoj jednog ovakvog okruženja zahteva veliko iskustvo i odgovarajuća znanja, ali isto tako i značajne resurse koji se ogledaju u potrebnom vremenu i finansijskim sredstvima. Stoga se kao rešenje predlaže proširivanje postojećih razvojnih okruženja koja nude savremene platforme za razvoj softverskih rešenja. Na ovaj način bi se iskoristila postojeća znanja i iskustva koja su u njima implementirana i omogućile bi se značajne uštede tokom procesa razvoja predloženog ETL razvojnog okruženja. Postojeća infrastruktura kao i različiti alati koji su u njima implementirani predstavljaju osnov predložene ETL razvojne platforme kojom se dalje uvode specifični alati neophodni tokom razvoja ETL procesa.

S obzirom da je kao opšta razvojna platforma usvojena *.NET platforma*, u ovom poglavlju se razmatra implementacija specifičnih ETL alata kao proširenja alata *Microsoft Visual Studio*[3] (u daljem tekstu *MSVS*). *MSVS* je osnovni alat koji se koristi u *.NET platformi*. Predstavlja savremeni i veoma moćan alat kojim je omogućena podrška različitim aktivnostima tokom procesa razvoja softverskih rešenja. Kao jedna od ključnih karakteristika ovog alata navodi se upravo njegova fleksibilnost. Definisani su različiti mehanizmi kojima je omogućeno dodavanje novih alata i prilagođavanje konkretnim potrebama.

[3] Korišćena je specifična verzija „*Microsoft Visual Studio Ultimate 2012*“

Kao osnovni alati predloženog ETL razvojnog okruženja implementirani su: *ETLProcessTool*, *ETLDataProcessTool*, *ETLExpressionTool*, *ETLTemplateTool* i *ETLDocumentationTool*. Navedenim alatima omogućeno je formiranje modela u skladu sa predloženim domensko-specifičnim jezicima, kao i automatsko generisanje dokumentacije ETL procesa.

9.1. Alati za modelovanje ETL procesa

Osnovni skup alata kojima je omogućeno modelovanje ETL procesa čine sledeći alati: *ETLProcessTool*, *ETLDataProcessTool*, *ETLExpressionTool* i *ETLTemplateTool*. Navedenim alatima omogućeno je formiranje modela ETL procesa u skladu sa definisanim domensko-specifičnim jezicima ETL-P, ETL-O, ETL-E i ETL-T (pogledati poglavlje 6, strana 94) i za njihovu implementaciju je korišćen alat *Domain-Specific Language Tools* (u daljem tekstu *DSL Tools*) *(Microsoft, 2012)*. Pored DSL Tools alata, za implementaciju parsera ETL-E jezika korišćen je alat Irony *(Ivantsov, 2009)*.

9.1.1. Alati za jezike sa grafičkom sintaksom (ETL-P, ETL-O i ETL-T)

DSL Tools je specifičan alat razvijen u okviru .NET platforme sa ciljem da se obezbedi podrška razvoju softvera vođen modelima. Obezbeđeno je okruženje za razvoj grafičkih domensko-specifičnih jezika (eng. *design phase*), ali isto tako i okruženje kojim je podržano njihovo korišćenje (eng. *runtime phase*) u vidu kreiranja odgovarajućih modela i njihove transformacije u različite softverske artefakte *(Cook, Jones, Kent, & Wills, 2007)*. Može se reći da je cilj koji se želi postići ovim alatom obezbeđivanje kompletne podrške razvoju i korišćenju grafičkih domensko-specifičnih jezika.

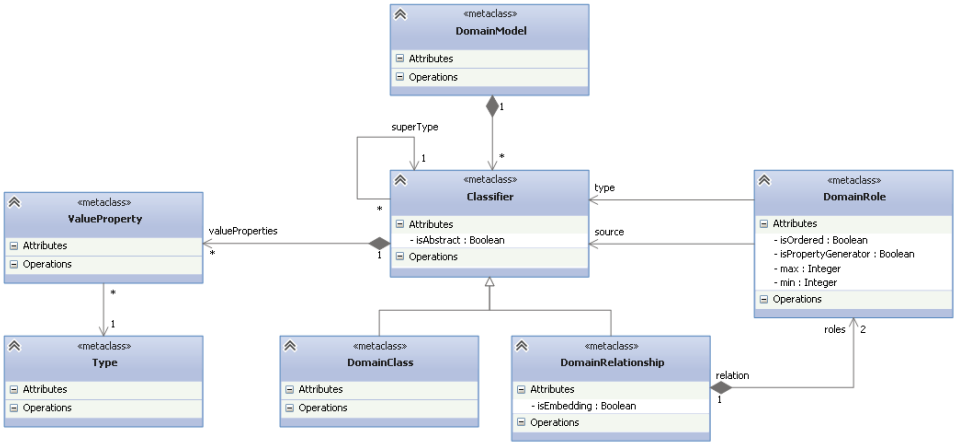
Osnovna prednost ovog alata se ogleda u činjenici da predstavlja integralni deo postojećeg razvojnog okruženja *.NET* platforme. Dodatno, kao prednost ističe se i činjenica da se razvijeni domensko-specifični jezici (reprezentovani različitim specifičnim alatima) automatski integrišu u okviru istog razvojnog okruženja čime je značajno olakšano njihovo korišćenje.

Razvoj domensko-specifičnog jezika korišćenjem alata *DSL Tools* se sastoji iz dva osnovna koraka. Prvi korak podrazumeva definisanje domensko-specifičnog jezika, dok se u drugom koraku generiše implementacija odgovarajućeg sintaksnog editora.

Definisanje domensko-specifičnog jezika

Razvoj domensko-specifičnog jezika započinje definisanjem odgovarajućeg domenskog modela (eng. *domain model*) *(Cook, Jones, Kent, & Wills, 2007)*. Ovim modelom se definišu osnovni koncepti jezika i veze između njih, kao i različita ograničenja u vezi njihovog korišćenja. Pored apstraktne sintakse, definiše se i konkretna sintaksa jezika u vidu grafičkih elemenata kojima se koncepti jezika reprezentuju.

Domensko-specifični jezici (ETL-P, ETL-O, ETL-E i ETL-T) opisuju se konceptima meta-jezika koji je definisan od strane *Microsoft*-a u okviru *DSL Tools* alata i kao takav je fiksiran u implementaciji ETL platforme. Na slici (slika 65) je prikazana pojednostavljena verzija meta-jezika koji je prikazan kao meta-metamodel u notaciji UML dijagrama klasa *(Bézivin, Hillairet, Jouault, Kurtev, & Piers, 2005)*.



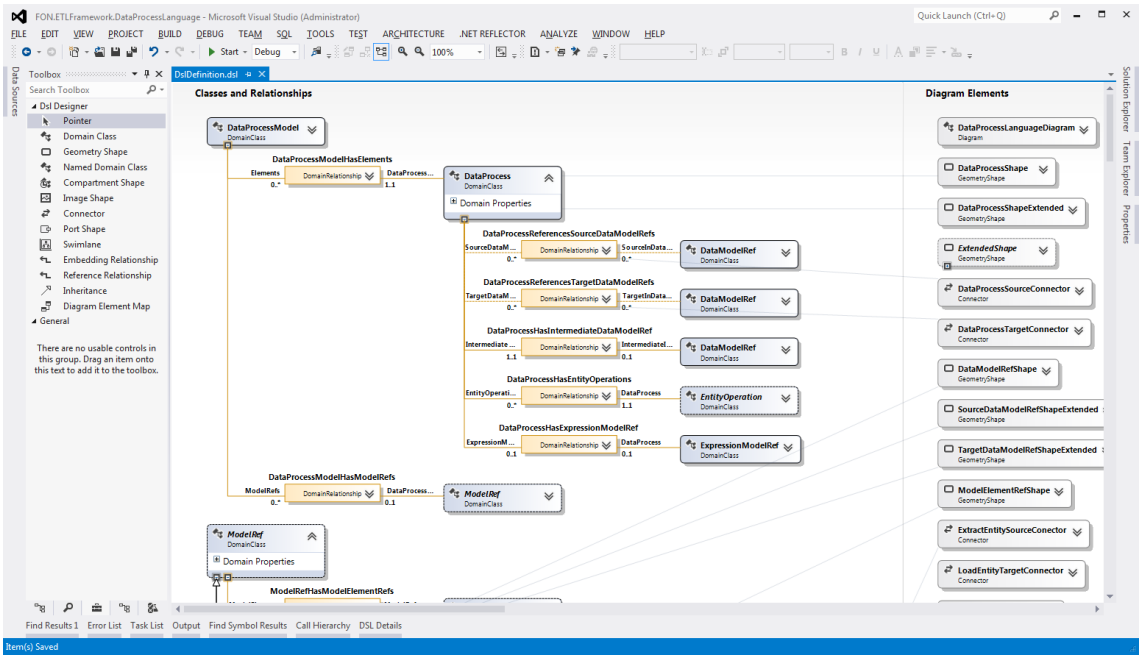
Slika 65. Pojednostavljena verzija DSL meta metamodela (Bézivin et al., 2005)

Meta-konceptom *DomainClass* predstavljaju se koncepti domensko-specifičnih jezika, dok se veze između ovih koncepata predstavljaju meta-konceptom *DomainRelationship* (ukoliko je veza kompozitna *isEmbeding* ima vrednost *true*). Za definisanje uloga u vezama koristi se meta-koncept *DomainRole*, dok se za

predstavljanje atributa (svojstava) koncepata i njihovih veza koristi meta-koncept *ValueProperty*.

Definisanje domenskog modela je podržano odgovarajućim meta-editorom. Za svaki meta-koncept (*DomainClass*, *DomainRelationship*, itd.) obezbeđen je odgovarajući grafički element. Prevlačenjem grafičkog elementa na dijagram kreira se instanca odgovarajućeg meta-koncepta. Na primer, prevlačenjem grafičkog elementa kojim se predstavlja meta-koncept *DomainClass* kreira se njegova instanca, odnosno odgovarajući koncept domensko-specifičnog jezika.

U nastavku je dat prikaz meta-editora (slika 66) na kome je predstavljen domenski model ETL-O jezika. Apstraktna sintaksa, odnosno koncepti ETL-O jezika i veze između ovih koncepata su predstavljeni u levom delu dijagrama (deo označen sa „*Classes and Relationships*“), dok je konkretna sintaksa definisana u desnom delu dijagrama (deo označen sa „*Diagram Elements*“).



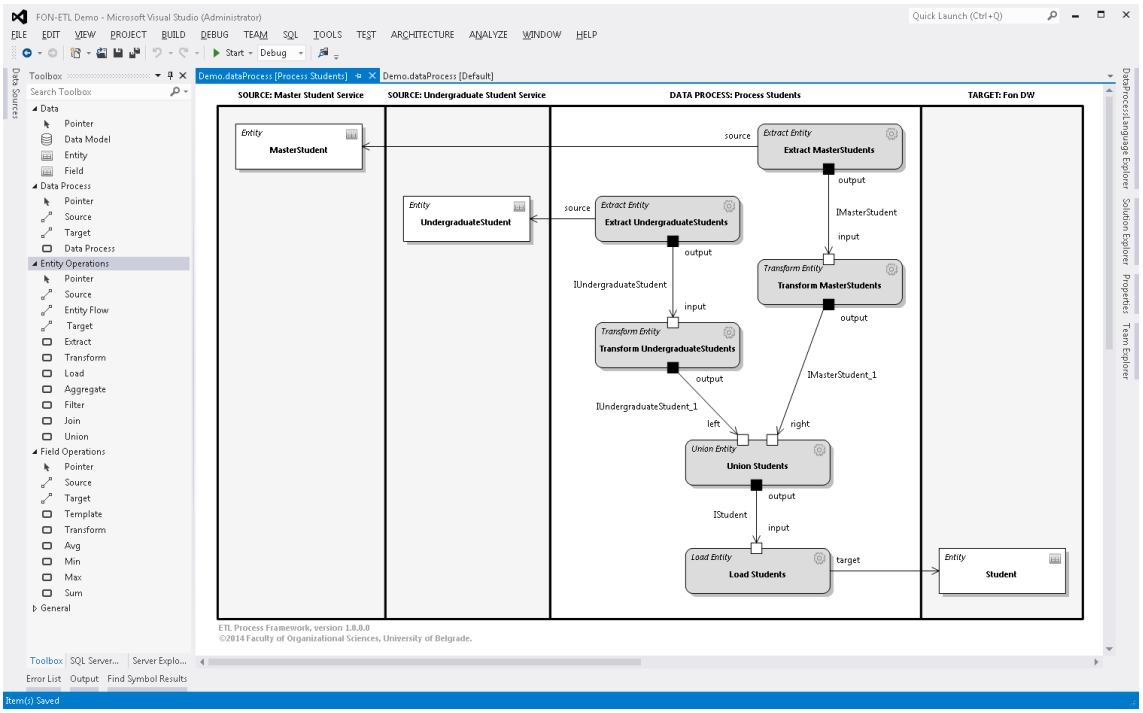
Slika 66. Prikaz meta-editora za kreiranje domensko-specifičnih jezika (DSL Tools)

Pored apstraktne i konkretne sintakse domensko-specifičnog jezika, definišu se i različita ograničenja kako bi se obezbedilo kreiranje validnih modela. Implementacija različitih ograničenja specificiranih domensko-specifičnih jezika data je u programskom jeziku *C#*.

Generisanje sintaksnog editora

Nakon definisanja domensko-specifičnog jezika, vrši se automatsko generisanje implementacije specifičnog sintaksnog editora. *DSL Tools* alatom je definisan veći broj *T4 šablona* (*Text Template Transformation Toolkit – T4,* strana 147) kako bi se omogućile ove automatske transformacije domenskog modela. Krajnji rezultat su specifični softverski moduli (*VSPackage*) kojima je omogućena instalacija implementiranog sintaksnog editora u konkretno razvojno okruženje (konkretna instanca alata *MSVS*).

Bitno je istaći i fleksibilnost *DSL Tools* alata koja se pre svega ogleda u mogućnosti prilagođavanja procesa transformacije domenskog modela. Naime, postojeći skup *T4 šablona* se može prilagoditi (bilo promenom postojećih šablona, bilo dodavanjem novih) čime je moguće uticati na implementaciju sintaksnih editora. Na primer, izvršeno je prilagođavanje ovih šablona *(Recchia & Guerot, 2009)* kako bi se omogućilo korišćenje višestrukih dijagrama. Na ovaj način omogućeno je formiranje modela na različitim apstraktnim nivoima kroz postepeno uvođenje detalja (pogledati poglavlje 7, strana 123).



Slika 67. Prikaz razvijenog alata ETLDataProcessTool

Na slici (slika 67) je predstavljen razvijeni alat *ETLDataProcessTool*. Prikazan je razvijeni ETL-O editor i primer dijagrama složene operacije transformacije podataka koji je u njemu kreiran. Kreiranje dijagrama je omogućeno posebno razvijenim komponentama koje su prikazane u levom delu slike (*Toolbox*). Komponentama su implementirani grafički elementi kojima se reprezentuju koncepti ETL-O jezika i grupisane su u četiri kategorije (*data process*, *data*, *entity operations* i *field operations*) kako bi se omogućilo lakše korišćenje.

Slično se može pokazati i za preostale domensko-specifične jezike ETL-P i ETL-T koji imaju grafičku sintaksu. Međutim, za domensko-specifične jezike koji imaju tekstualnu sintaksu, kao što je ETL-E, *DSL Tools* nije dovoljan. Potrebno je implementirati specifičan parser ETL-E jezika.

9.1.2. Alati za jezike sa tekstualnom sintaksom (ETL-E)

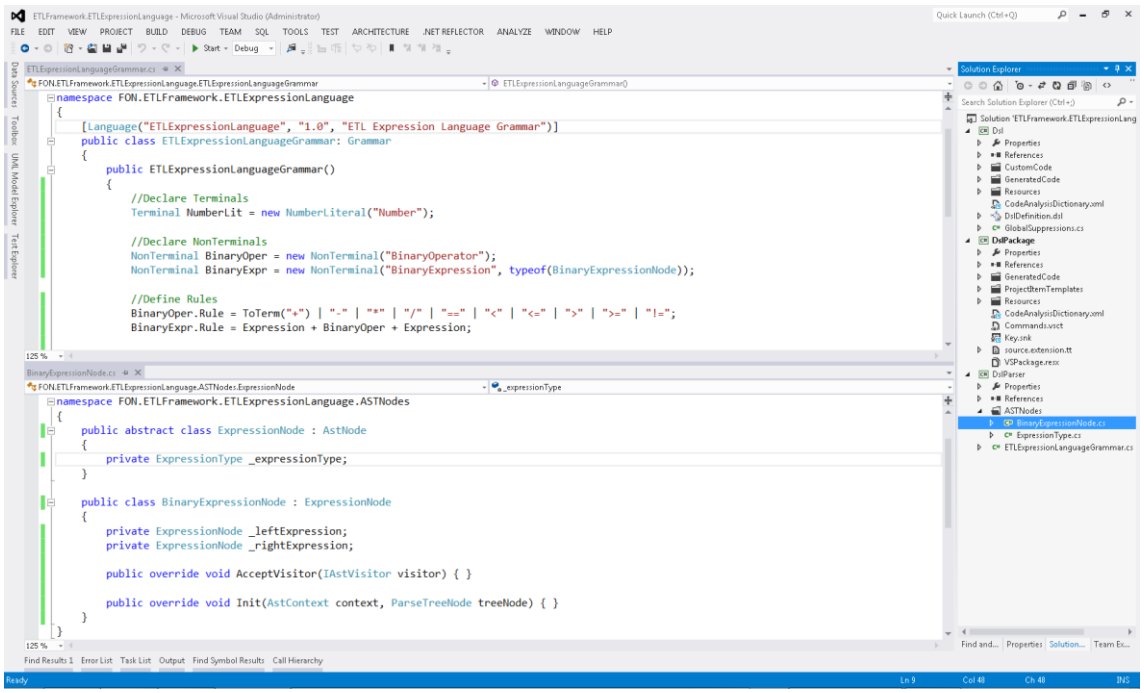
Za implementaciju parsera ETL-E jezika korišćen je alat *Irony* (autor *Roman Ivantsov*) čiji je osnovni cilj podrška razvoju domensko-specifičnih jezika sa tekstualnom sintaksom za *.NET* platformu. Alat je implementiran korišćenjem programskog jezika C# i spada u kategoriju softvera otvorenog koda[4].

Specifičnost ovog alata ogleda se u pristupu kojim se ne zahteva korišćenje posebnog metajezika za specifikaciju gramatike domensko-specifičnog jezika *(Ivantsov, 2009)*. Gramatika jezika se definiše korišćenjem programskog jezika C#, odnosno korišćenjem specifičnih C# klasa, kao što su *Terminal*, *NonTerminal*, *KeyTerm* (formiraju se izrazi bazirani na produkcionim pravilima *Backus-Naur Form* - BNF). Pored toga, pristupom se ne zahteva generisanje skenera (eng. *scanner*) i parsera (eng. *parser*), već se koristi univerzalni parser *(Ivantsov, 2009)*. Ovaj univerzalni LALR parser (eng. *Look Ahead Left to Right*) se izvršava u skladu sa gramatikom jezika koja je predstavljena specifičnim objektnim modelom. Rezultat parsiranja je odgovarajuće stablo apstraktne sintakse (eng. *Abstract Syntax Tree - AST*). Za predstavljanje čvorova ovog stabla obezbeđena je bazna

[4] http://irony.codeplex.com/

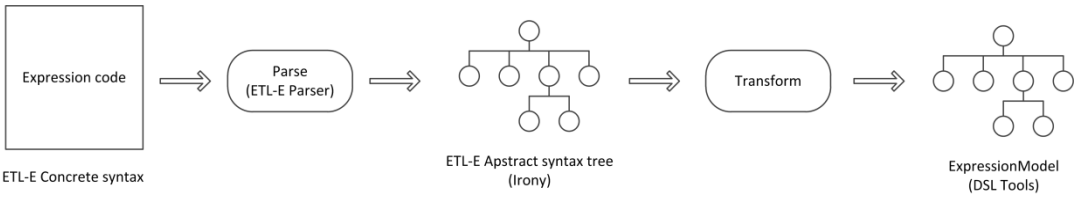
klasa *ASTNode* iz koje se izvode konkretne podklase specifične za posmatrani domensko-specifičan jezik.

Na slici (slika 68) su prikazani ključni elementi implementacije ETL-E parsera.



Slika 68. Ključni elementi implementacije ETL-E parsera

Implementacija ETL-E parsera je prvi korak u implementaciji domensko-specifičnog jezika ETL-E. Sledeći korak podrazumeva implementaciju domenskog modela korišćenjem alata *DSL Tools*. Postupak implementacije je isti kao i za ostale domensko-specifične jezike ETL-P, ETL-O i ETL-T s tim što se ne definišu grafički elementi, tj. konkretna sintaksa.



Slika 69. Uloga alata *Irony* i *DSL Tools* u implementaciji jezika ETL-E

Na slici (slika 69) je predstavljena uloga korišćenih alata *Irony* i *DSL Tools* u implementaciji domensko-specifičnog jezika ETL-E. Konkretna sintaksa jezika je implementirana korišćenjem alata *Irony*, dok je apstraktna sintaksa implementirana alatom *DSL Tools*.

9.2. Alat za generisanje dokumentacije ETL procesa

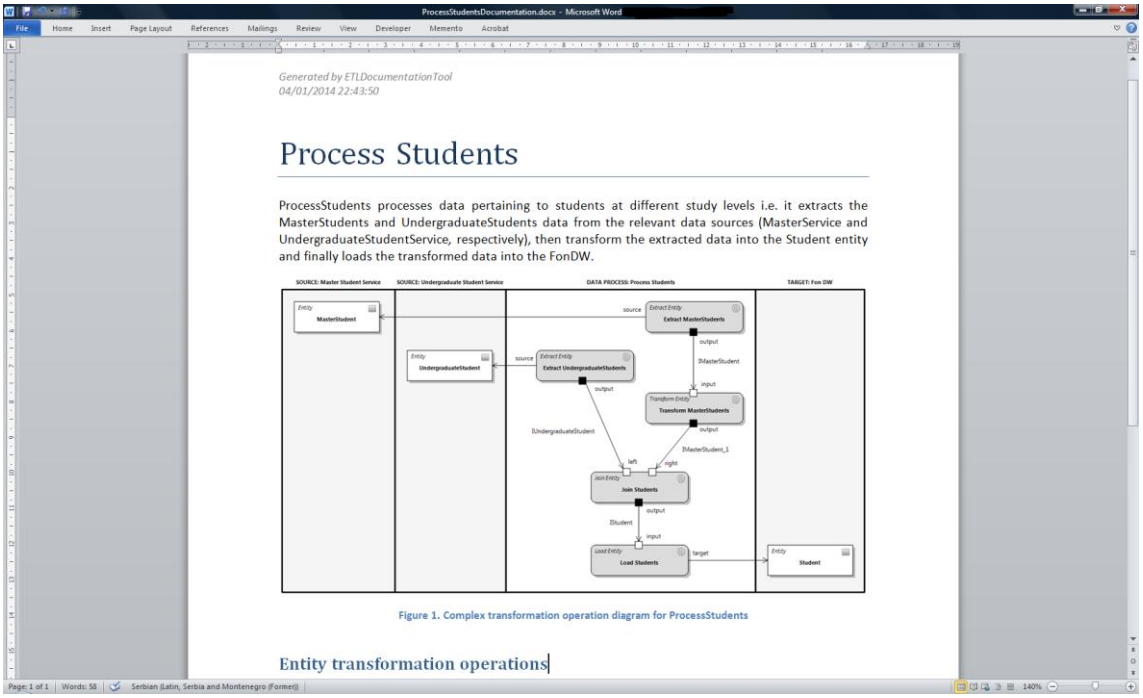
Dokumentacija predstavlja veoma bitan artefakt svakog projekta, a posebno projekata koji se bave razvojem softverskih rešenja. Značaj dokumentacije se posebno ističe u fazi održavanja softverskog rešenja. Potrebno je dokumentovati različite odluke koje su donete tokom projektovanja i implementacije softverskog rešenja što čini osnov za donošenje odluka tokom njegove nadogradnje.

Od izuzetne važnosti je da dokumentacija bude u skladu sa implementiranim softverskim rešenjem, odnosno da bude validna. Ukoliko dokumentacija ne odgovara stvarnom stanju onda je njena upotrebna vrednost veoma mala i često vodi ka lošim odlukama tokom održavanja softverskog rešenja. Izrada dokumentacije je aktivnost koja se često zanemaruje i kojoj se ne pridaje velika pažnja. Potrebno je dosta vremena za njenu izradu i redovno ažuriranje kako bi odražavala stvarno stanje. Upravo zbog ovoga neophodno je obezbediti odgovarajuću podršku kroz automatizaciju ovih aktivnosti.

Kao jedan od osnovnih alata predloženog ETL razvojnog okruženja razvijen je alat *ETLDocumentationTool*. Ovim alatom omogućeno je automatsko generisanje dokumentacije implementiranog ETL procesa. Za implementaciju je korišćen *Text Template Transformation Tool* alat (pogledati 10.1.1, strana 147) i shodno tome, razvijen je skup specifičnih *T4 šablona*. Ovim šablonima je omogućena automatska transformacija modela ETL procesa (iskazanih specificiranim domensko-specifičnim jezicima ETL-P, ETL-O, ETL-T i ETL-E) u odgovarajući *.docx* format (*OpenXML standard*[5]).

Na slici (slika 70) je prikazan dokument koji je generisan alatom *ETLDocumentationTool*. Dokumentovan je proces obrade podataka *ProcessStudents* koji je razmatran u poglavlju 7, strana 123.

[5] *http://officeopenxml.com/*



Slika 70. Primer generisane dokumentacije za proces obrade podataka ProcessStudents

10. ETL IZVRŠNO OKRUŽENJE

Uvođenje specifičnog izvršnog okruženja u okviru predložene ETL platforme ima za cilj da se omogući izvršavanje softverskih rešenja koja se bave problemom ETL procesa na način koji obezbeđuje visoke performanse, kao i da se obezbedi podrška za fleksibilna softverska rešenja kojima je moguće u veoma kratkom roku odgovoriti na promene. Pored implementacije jasno definisanih funkcionalnih zahteva, posebno se ističu i nefunkcionalni zahtevi kao što su visoke performanse, skalabilnost i fleksibilnost.

Kao osnovni elementi ETL izvršnog okruženja predloženi su odgovarajući servisi *ETLProcessService*, *ETLDataProcessService* i *ETLMetadataService*. Navedeni servisi predstavljaju minimalni skup servisa kojima je omogućeno izvršavanje i upravljanje ETL procesima.

Servisi *ETLProcessService* i *ETLDataProcessService* su uvedeni sa ciljem da se razdvoji izvršavanje upravljačkog toka i toka podataka, odnosno da se omogući konkurentno izvršavanje različitih procesa obrade podataka koji čine ETL proces.

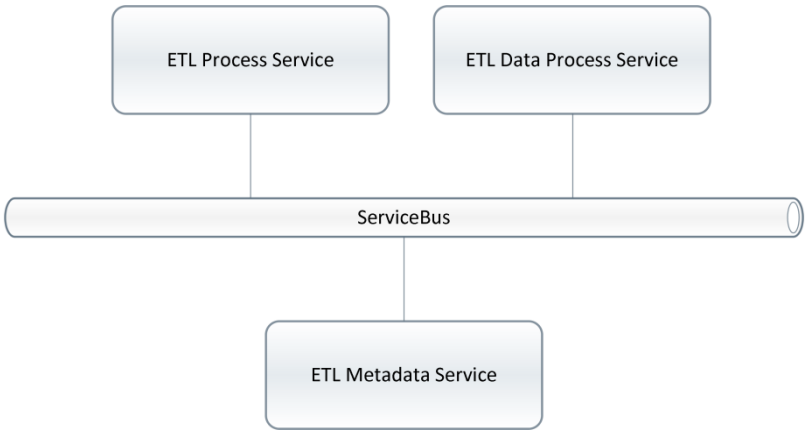
Konkurentno izvršavanje funkcija softverskog rešenja je osnovni mehanizam kojim je moguće uticati na povećanje performansi. Paralelizacijom izvršavanja omogućeno je bolje iskorišćenje hardverske infrastrukture, što kao rezultat ima brže izvršavanje softverskog rešenja. Pored toga, ukoliko se omogući i distribuirano izvršavanje, realizacijom horizontalne skalabilnosti, performanse će se drastično povećati.

Implementacijom distribuiranog izvršavanja nastaju problemi tokom instalacije i održavanja softverskog rešenja. Povećava se složenost softverskog rešenja čime se utiče na povećanje troškova realizacije. Takođe, ukoliko se uzme u obzir činjenica da su promene u zahtevima česte, kao i da se na ove promene mora odgovoriti u veoma kratkom roku, jasno je da se ovi problemi moraju na adekvatan način rešiti.

Kao rešenje se predlaže dinamičko izvršavanje ETL procesa, odnosno automatsko generisanje, kompajliranje i izvršavanje ETL procesa u vreme izvršavanja softverskog rešenja. Uveden je poseban servis *ETLMetadataService* kojim su obezbeđeni metapodaci, odnosno odgovarajući opisi ETL procesa neophodni tokom dinamičkog generisanja i kompajliranja izvršnog kôda. Ovo nije ni malo jednostavan inženjerski zadatak. Potrebno je veliko iskustvo i znanje, kao i odgovarajuća tehnološka podrška (pogledati 10.1, strana 147). Međutim, rezultati koji se na ovaj način mogu postići daleko prevazilaze troškove i uložen rad.

Navedeni servisi su implementirani korišćenjem *Windows Communication Foundation* (u daljem tekstu WCF) tehnologije koja je namenjena za kreiranje servisno-orijentisanih aplikacija. Pre svega, omogućena je komunikacija koja se bazira na asinhronoj razmeni poruka. Podržani su različiti formati poruka, kao i različiti transportni protokoli što ovu tehnologiju čini veoma fleksibilnom i pogodnom za korišćenje u različitim scenarijima (bilo da je cilj postizanje interoperabilnosti ili postizanje efikasne komunikacije).

Povezivanje navedenih servisa, u skladu sa servisno-orijentisanom arhitekturom, zahteva postojanje posebne komponente *ServiceBus*. Ovom komponentom se smanjuje zavisnost između servisa i utiče se na skalabilnost i fleksibilnost softverskog rešenja (uvođenje novih servisa je veoma jednostavno i nema uticaj na postojeće servise).



Slika 71. Servisno-orijentisana arhitektura predloženog ETL izvršnog okruženja

10.1. Tehnike transformacije

Podrška automatskim transformacijama modela i generisanje izvornog koda je veoma bitan element platformi za razvoj softvera. Savremene platforme za razvoj softvera, kao što je *Microsoft .NET* platforma nude veći broj tehnika kako bi tehnološki podržale različite scenarije u kojima moguće primeniti generisanje.

Podrška je neophodna kako tokom razvoja, tako i tokom izvršavanja softverskih rešenja. Automatskim transformacijama tokom razvoja utiče se produktivnost i kvalitet softverskih rešenja, dok se podrškom tokom izvršavanja značajno doprinosi fleksibilnosti softverskih rešenja. Uz odgovarajuću podršku moguće je realizovati softverska rešenja koja se dinamički, tokom izvršavanja prilagođavaju potrebama i zahtevima iz okruženja.

.NET platforma nudi veći broj tehnologija kojima su omogućene automatske transformacije *(Hazzard & Jason, 2013)*: (1) *Text Template Transformation Toolkit*, (2) *Code Document Object Model*, (3) *Reflection.Emit* i (4) *Lambda Expressions*. Veći broj tehnologija je razvijen kako bi se podržali različiti scenariji bilo tokom razvoja, bilo tokom izvršavanja softverskog rešenja. U nastavku je dat kratak opis ovih tehnologija i identifikovani su scenariji u kojima su primenljive u kontekstu razvoja i izvršavanja ETL procesa.

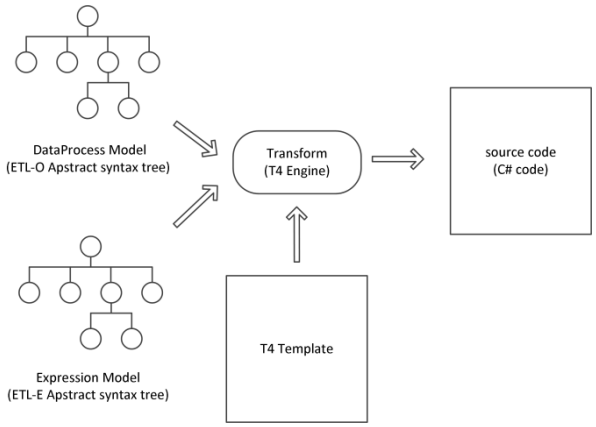
10.1.1. Text Template Transformation Toolkit

Text Template Transformation Toolkit (u daljm tekstu *T4*) je veoma jednostavna i fleksibilna tehnologija kojom je omogućeno generisanje različitih softverskih artefakata. Pristup koji je omogućen ovom tehnologijom zasniva se na kreiranju odgovarajućih tekstualnih šablona (eng. *template*) kojima se opisuje neophodna transformacija. Izvršavanje ovih transformacija je automatizovano i podržano je razvojnim okruženjem.

Kreiranje šablona je veoma jednostavno. Navode se odgovarajuće direktive kojima se definišu parametri transformacije, kao i odgovarajući upravljački i tekstualni blokovi. Za definisanje upravljačkih blokova koriste se *.NET* programski jezici *C#* ili *VB.NET.*

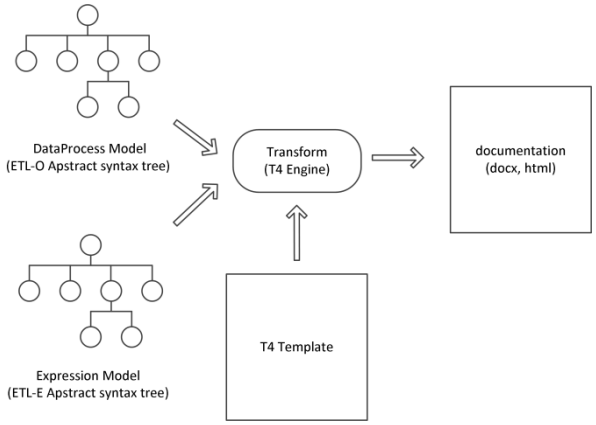
Rezultat izvršavanja transformacija je tekst, pa se *T4* smatra veoma fleksibilnom tehnologijom. Pored izvornog kôda mogu se generisati i drugi artefakti neophodni tokom razvoja softverskog rešenja (npr, *XML* datoteke, *Word* datoteke, itd.).

U nastavku su prikazana dva moguća scenarija korišćenja tehnike *T4*. Prvim je predstavljeno generisanje izvornog kôda (slika 72), dok je drugim predstavljen proces kreiranja datoteke (npr. *docx* ili *html*) kojom se dokumentuje određeni proces obrade podataka (slika 73).



Slika 72. Proces generisanja izvornog koda za proces obrade podataka (T4 tehnika)

Proces transformacije (u oba scenarija) započinje učitavanjem odgovarajućeg *T4* šablona u specifično izvršno okruženje (*T4 Engine*) kojim je omogućeno automatsko izvršavanje transformacija. Pored šablona, kao ulaz u proces transformacije obezbeđeni su model procesa obrade podataka i referencirani model izraza. Rezultat izvršavanja transformacija je tekst čiji je format određen učitanim šablonom.



Slika 73. Proces generisanja dokumentacije za proces obrade podataka (T4 tehnika)

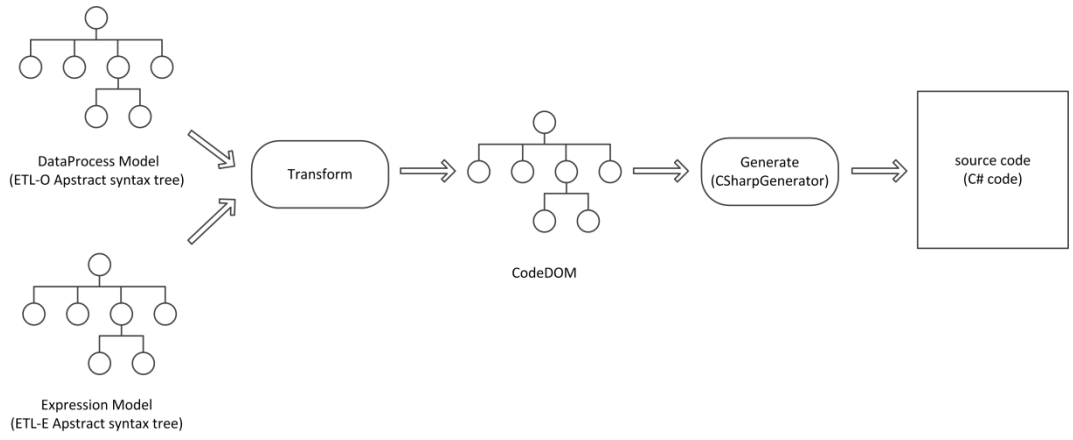
10.1.2. Code Document Object Model

Code Document Object Model (u daljm tekstu *CodeDOM*) je veoma moćna tehnologija kojom je omogućeno parsiranje, generisanje i kompajliranje izvornog koda. Zasniva se na korišćenju specifičnog objektnog modela, kojim je moguće predstaviti izvorni kod na način koji je nezavisan od korišćenog programskog jezika.

Za predstavljanje izvornog koda implementirana je složena hijerarhija klasa u čijoj osnovi je klasa *CodeObject*. Kao podklase ove klase uvedene su *CodeNamespace*, *CodeNamespaceImport*, *CodeStatement*, *CodeExpression* i mnoge druge sa ciljem da se omogući precizan opis različitih komponenti izvornog koda, kao što su tipovi (interfejs, klasa, zapis, itd.), njihovi članovi (polja, svojstva, metode, itd.), parametri itd.

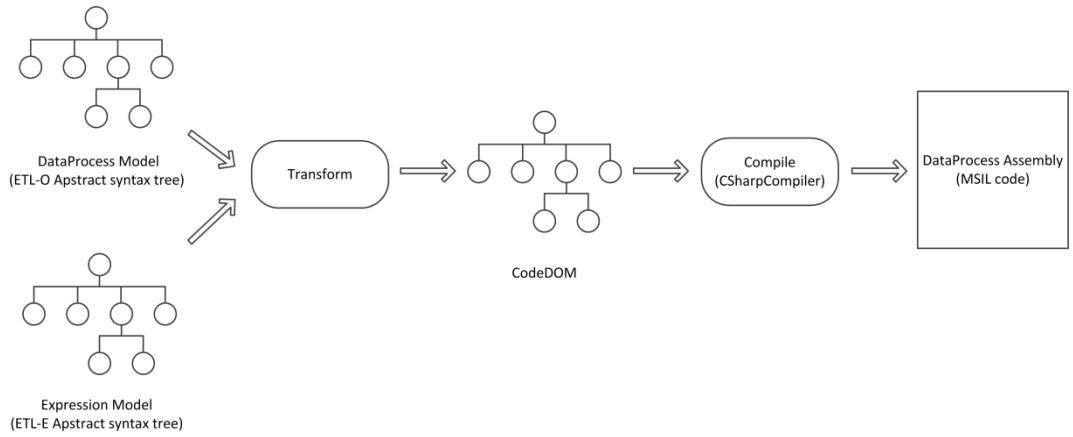
Objektni model se može kreirati automatski, parsiranjem odgovarajućeg izvornog koda, ali se isto tako može kreirati i programski. Pored toga, na osnovu objektnog modela moguće je automatski generisati izvorni kod u izabranom programskom jeziku. Izvorni kod predstavljen objektnim modelom se može kompajlirati čime se dobija odgovarajući sklop (eng. *assembly*), odnosno *Microsoft intermediate language* kôd (u daljem tekstu *MSIL*).

U nastavku su prikazana dva moguća scenarija korišćenja *CodeDOM* tehnike. Prvim je predstavljen proces generisanja izvornog kôda (slika 74), dok je drugim predstavljen proces kompajliranja modela procesa obrade podataka (slika 75).



Slika 74. Proces generisanja izvornog koda za proces obrade podataka (CodeDOM tehnika)

Oba scenarija započinju učitavanjem odgovarajućih modela, tj. modela procesa obrade podataka i referenciranih modela izraza. Kao sledeći korak, sprovodi se transformacija učitanih modela (eng. *transform*) u odgovarajući objektni model (*CodeDOM*), a zatim se u slučaju generisanja izvršava odgovarajući generator koda (*CSharpGenerator*), odnosno u slučaju kompajliranja odgovarajući kompajler (*CSharpCompiler*).



Slika 75. Proces kompajliranja za proces obrade podataka (CodeDOM tehnika)

Komponente *CSharpGenerator* i *CSharpCompiler* su obezbeđene *.NET* platformom, dok se specifična komponenta kojom se vrši transformacija modela procesa obrade i modela izraza u odgovarajući objektni model (*CodeDOM*) mora obezbediti kroz ETL platformu.

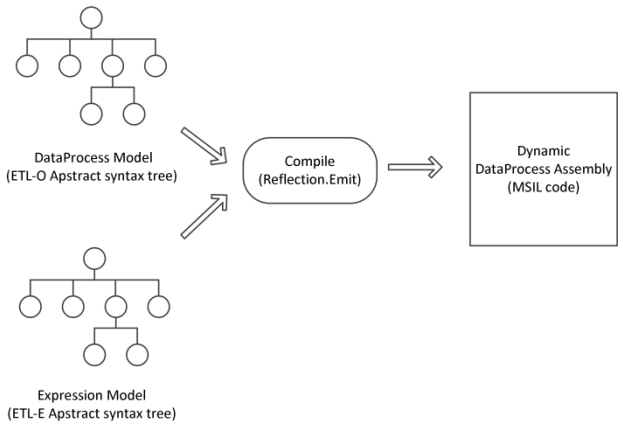
10.1.3. Reflection.Emit

Tehnologija *Reflection.Emit* je uvedena sa ciljem da se omogući podrška generisanju tokom izvršavanja softverskog rešenja. Omogućeno je generisanje dinamičkih sklopova (eng. *dynamic assembly*), odnosno različitih tipova koji su neophodni tokom izvršavanja softverskog rešenja. Rezultat transformacije je *MSIL* kod koji se direktno izvršava u okviru izvršnog okruženja *.NET* platforme.

Proces generisanja podrazumeva da se prvo generiše sklop (eng. *assembly*) i u okviru njega jedan ili više modula, a zatim i odgovarajući tipovi unutar ovih modula, kao što su interfejsi, klase i njihovi članovi. Kao podrška ovom procesu implementirane su specifične komponente *AssemblyBuilder*, *ModuleBuilder*,

TypeBuilder, *MethodBuilder* itd. Generisanje metoda, odnosno *MSIL* instrukcija, podržano je komponentom *ILGenerator*.

U nastavku je prikazan scenario korišćenja *Reflection.Emit* tehnologije (slika 76). Prikazan je proces kompajliranja odgovarajućeg modela procesa obrade podataka čiji je rezultat dinamički sklop. Specifična komponenta kojom se vrši kompajliranje se mora obezbediti kroz predloženu ETL platformu.



Slika 76. Proces kompajliranja za proces obrade podataka (Reflection.Emit tehnika)

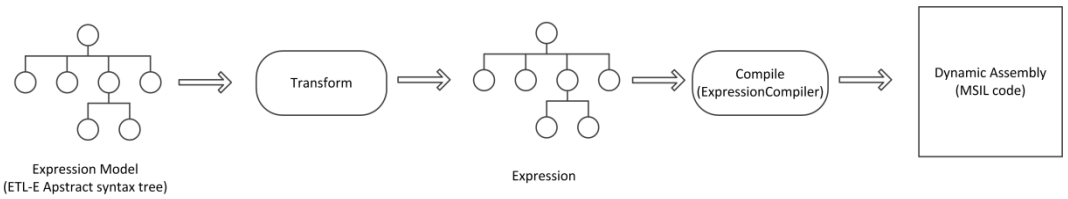
Iako veoma moćna tehnologija, kojom su obezbeđene izuzetne performanse, mogu se navesti i odgovarajući nedostaci. Pre svega, zahteva se rad na veoma niskom nivou apstrakcije koji podrazumeva poznavanje *MSIL* jezika. Pored toga, generisani kôd je učitan u memoriju i tamo ostaje dokle god se izvršava softversko rešenje[6].

10.1.4. Lambda Expression

Slično kao i *CodeDOM* tehnologija, uvodi specifičan objektni model kojim je moguće predstaviti izvorni kod. Na osnovu ovako predstavljanog izvornog kôda moguće je automatski, u vreme izvršavanja softverskog rešenja, generisati odgovarajuće dinamičke sklopove. Pored korišćenog objektnog modela, razlika između ove dve tehnologije je što se *Lambda Expression* tehnologijom predstavljaju samo izrazi (eng. *expressions*).

[6] *Garbage collector* (Microsoft, 2014) nema kontrolu nad ovim kôdom.

Za predstavljanje izraza implementirana je složena hijerarhija klasa u čijoj osnovi je klasa *Expression*. Kao podklase ove klase uvedene su *UnaryExpression*, *BinaryExpression*, *ParameterExpression* i mnoge druge sa ciljem da se omogući precizan opis potencijalno složenih izraza. Pored ovih klasa uvedene su i *BlockExpression*, *ConditionalExpression*, *LoopExpression*, *ForExpression*, itd. kako bi se omogućilo predstavljanje osnovnih upravljačkih struktura.



Slika 77. Proces kompajliranja za proces obrade podataka (Expression tehnika)

Osnovna prednost ove tehnologije ogleda se u činjenici da izdiže nivo apstrakcije u odnosu na *Reflection.Emit* tehnologiju. *MSIL* kôd se automatski generiše na osnovu kreiranog objektnog modela izraza.

10.2. Izvršavanje procesa obrade podataka

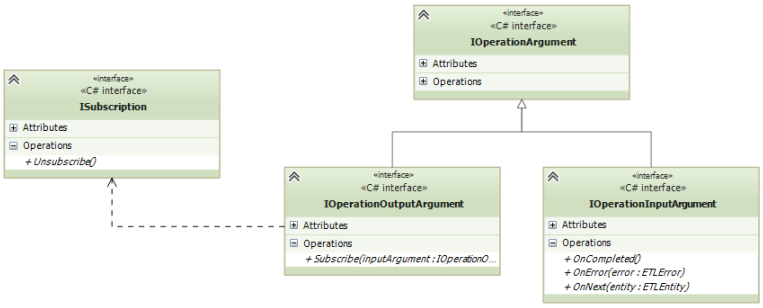
Proces obrade podataka se sastoji od većeg broja prostih operacija transformacija koje su međusobno povezane, formirajući na taj način odgovarajući tok podataka.

Proste operacije transformacije se mogu klasifikovati na operacije ekstrahovanja kojima se podaci ekstrahuju iz odgovarajućih izvora podataka, zatim operacije transformacije kojima se vrši transformacija ekstrahovanih podataka i na operacije učitavanja kojima se vrši učitavanje transformisanih podataka u ciljno skladište podataka. Proces obrade podataka počinje operacijama ekstrahovanja, a završava operacijama učitavanja podataka.

Tok izvršavanja procesa obrade podataka određen je međuzavisnostima operacija transformacija, dok je vreme izvršavanja ovih operacija uslovljeno dostupnošću podataka. Ovo znači da operacije transformacije mogu početi sa izvršavanjem tek kada su odgovarajući podaci dostupni. Efikasna razmena podataka između operacija transformacija je ključni zahtev u pogledu implementacije procesa obrade podataka.

Predloženim aplikacionim okvirom usvojen je *push* mehanizam, što znači da svaka operacija transformacije, po završenoj obradi, prosleđuje podatke sledećoj operaciji transformacije. Razmena podataka između povezanih operacija transformacija obavlja se preko njihovih izlaznih i ulaznih argumenata.

Implementacija usvojenog mehanizma zasniva se na *Observer Pattern*-u pri čemu izlazni argument jedne operacije transformacije igra ulogu *Subject*, a ulazni argument druge operacije transformacije igra ulogu *Observer (Microsoft, 2014)*.



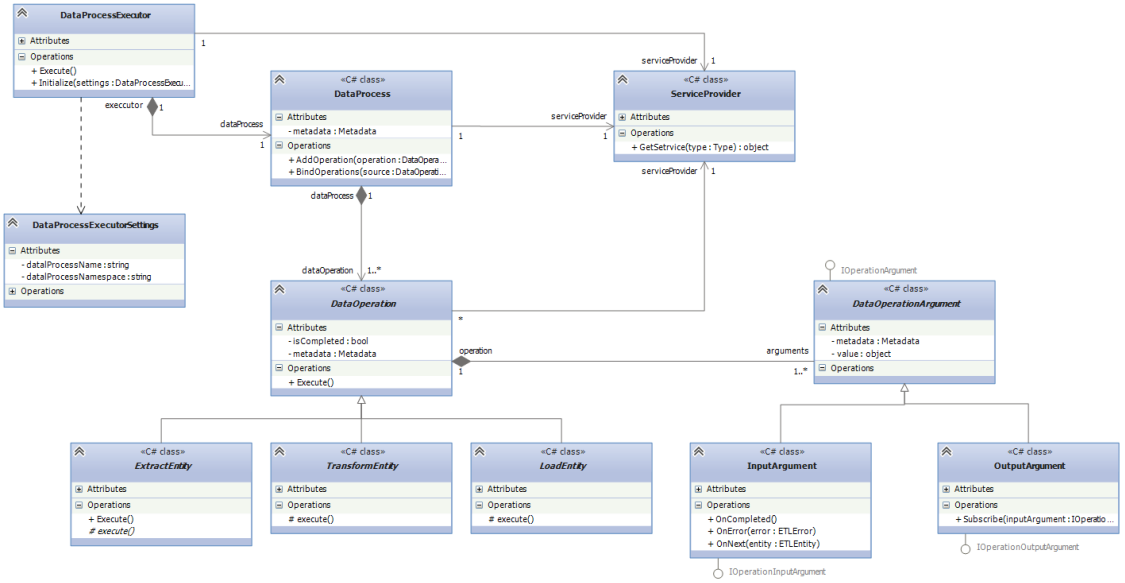
Slika 78. Implementacija push mehanizma

Kao što je i predstavljeno (slika 78), interfejsom *IOperationInputArgument* definisane su tri metode, pri čemu je metodom *OnNext* omogućeno prosleđivanje podataka, dok je metodama *OnCompleted* i *OnError* omogućeno obaveštavanje da je operacija završila sa obradom podataka, odnosno da je nastala greška tokom obrade podataka. Interfejsom *IOperationOutputArgument* definisana je metoda *Subscribe* kojom se realizuje povezivanje argumenata operacija. Rezultat izvršavanja ove metode je predstavljen interfejsom *ISubscription*, kako bi se omogućilo raskidanje prethodno uspostavljene veze između argumenata operacija.

Osnovne klase predloženog rešenja predstavljene su dijagramom klasa (slika 79). Klasa *DataProcessExecutor* je uvedena kako bi se obezbedila podrška izvršavanju procesa obrade podataka. Metodom *Initialize* vrši se podešavanje izvršnog okruženja, dok se metodom *Execute* inicira izvršavanje procesa obrade podataka. Parametri metode *Initialize* su predstavljeni klasom *DataProcessExecutorSettings*.

Upravljanje različitim komponentama, koje su neophodne tokom izvršavanja procesa obrade podataka (*ETLMetadataServiceAgent*, *DataOperationManager*, *EntityManager*, itd.), omogućeno je klasom *ServiceProvider*.

Klasom *DataProcess* predstavljen je proces obrade podataka, dok su operacije transformacije podataka predstavljene apstraktnom klasom *DataOperation*.



Slika 79. Dijagram klasa za izvršavanje procesa obrade podataka

Formiranje procesa obrade podataka podržano je metodama *AddOperation* i *BindOperations*. Metoda *AddOperation* omogućava dodavanje operacija transformacija podataka, dok *BindOperations* omogućava njihovo povezivanje. Pozivom metode *Execute* inicira se izvršavanje procesa obrade podataka.

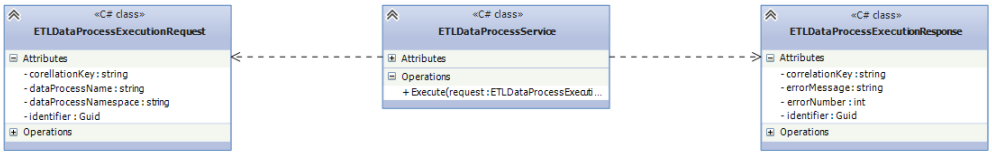
Standardna klasifikacija operacija transformacija (operacije ekstrahovanja, transformisanja i učitavanja podataka) realizovana je apstraktnim klasama *ExtractEntity*, *TransformEntity* i *LoadEntity*. Ove klase su implementirane kao podklase klase *DataOperation*. Ulazi i izlazi operacija transformacija podataka predstavljaju se apstraktnom klasom *DataOperationArgument*, odnosno njenim podklasama *InputArgument* i *OutputArgument*. Ovim podklasama implementirani su odgovarajući interfejsi (*IOperationInputArgument*, *IOperationOutputArgument*) kako bi se obezbedila implementacija u skladu sa usvojenim mehanizmom razmene podataka (slika 78).

Predloženim aplikacionim okvirom implementiran je veći broj konkretnih operacija transformacija podataka. Ove operacije su implementirane kao podklase

klase *DataOperation*, odnosno preciznije rečeno kao podklase klase *ExtractEntity*, *TransformEntity* ili *LoadEntity*.

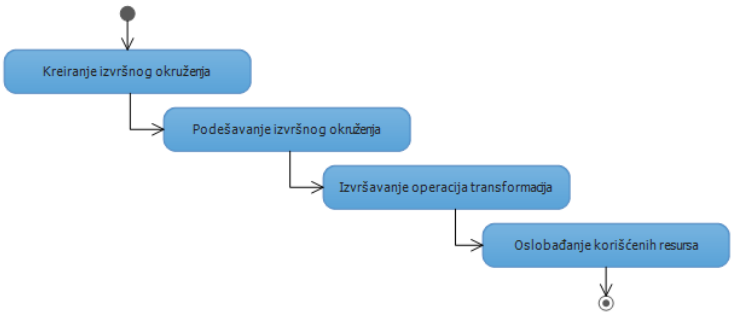
10.2.1. Osnovni scenario izvršavanja procesa obrade podataka

Izvršavanje procesa obrade podataka omogućeno je specifičnim servisom *ETLDataProcessService*. Izvršavanje započinje pozivom metode *Execute*, odnosno prijemom poruke *ETLDataProcessExecutionRequest*, a završava se formiranjem i slanjem poruke *ETLDataProcessExecutionResponse*.



Slika 80. ETLDataProcess Request/Response

U nastavku su prikazani osnovni koraci tokom izvršavanja procesa obrade podataka (slika 81), a zatim je dat njihov opis i odgovarajući dijagrami sekvenci.

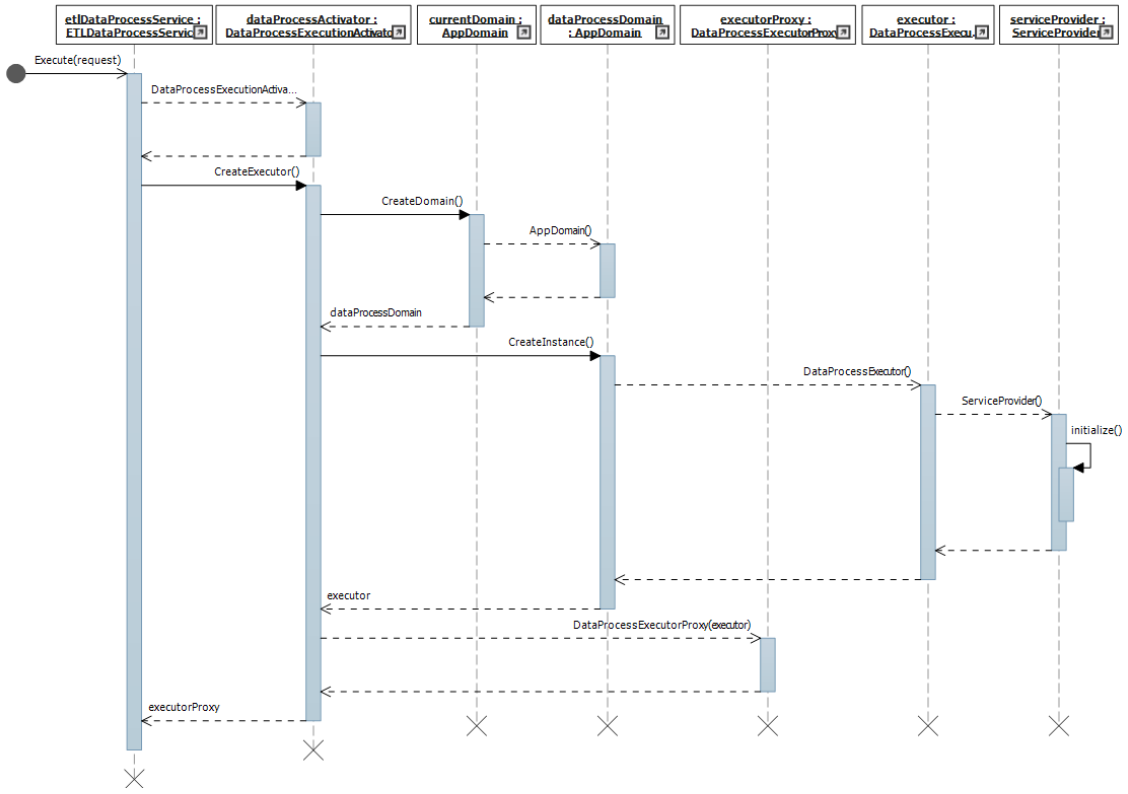


Slika 81. Osnovni koraci tokom izvršavanja procesa obrade podataka

Kreiranje izvršnog okruženja

Tokom ovog koraka kreira se specifično izvršno okruženje kojim će se omogućiti izvršavanje zahtevanog procesa obrade podataka. Ovo podrazumeva kreiranje posebnog aplikacionog domena i instanciranje odgovarajućih objekata u njemu. Potreba za posebnim aplikacionim domenom proizilazi iz činjenice da se tokom izvršavanja procesa obrade podataka generiše i kompajlira izvršni kod. Kreiraju se dinamički sklopovi i učitavaju u *.NET* izvršno okruženje.

Implementacija je data klasom *DataProcessExecutionActivator* koja izlaže metodu *CreateExecutor*. Rezultat izvršavanja navedene metode je predstavljen klasom *DataProcessExecutorProxy* koja je uvedena sa ciljem da se apstrahuje komunikacija sa objektima iz novokreiranog izvršnog okruženja.



Slika 82. Dijagram sekvenci za kreiranje izvršnog okruženja procesa obrade podataka

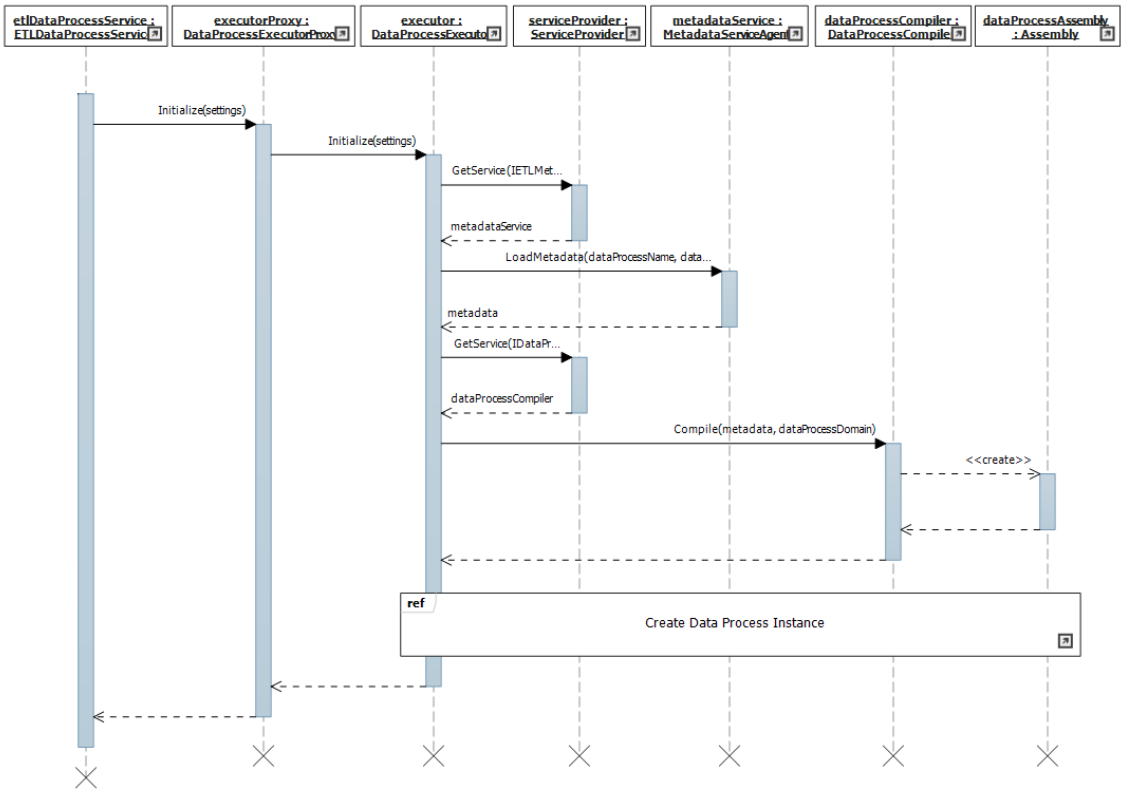
Podešavanje izvršnog okruženja

Podešavanje izvršnog okruženja podrazumeva učitavanje metapodataka koji su neophodni za izvršavanje određenog procesa obrade podataka (model procesa obrade podataka i referencirani modeli izraza), zatim generisanje i kompajliranje izvršnog koda u skladu sa učitanim metapodacima i na kraju instanciranje procesa obrade podataka i njegovo podešavanje. Ovaj korak započinje pozivom metode *Initialize* klase *DataProcessExecutorProxy*.

Učitavanje metapodataka omogućeno je klasom *MetadataServiceAgent* i realizuje se pozivom metode *LoadMetadata*. Ova klasa je uvedena sa ciljem da se apstrahuje komunikacija sa servisom *ETLMetadataService* kojim se izlažu metapodaci.

Generisanje i kompajliranje izvršnog koda podržano je posebno razvijenom klasom *DataProcessCompiler* koja izlaže metodu *Compile*. Kao ulazni parametri ove metode, prosleđuju se prethodno učitani metapodaci i aplikacioni domen u koji će se rezultati kompajliranja učitati. Rezultat kompajliranja je dinamički sklop u kome su definisani svi tipovi neophodni za izvršavanje procesa obrade podataka.

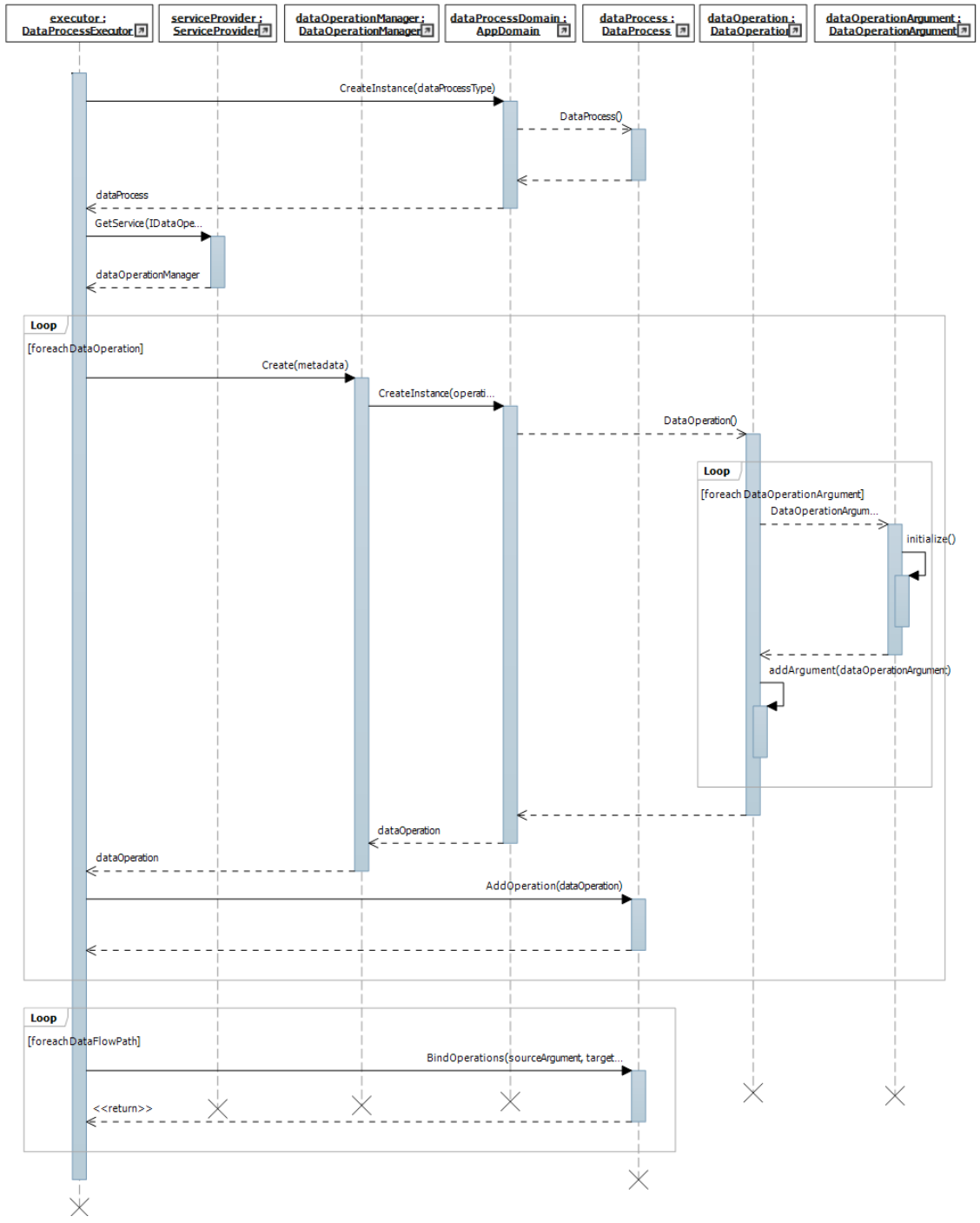
Nakon kompajliranja vrši se instanciranje procesa obrade podataka i njegovo podešavanje. Pod podešavanjem se podrazumeva instanciranje svih prethodno generisanih operacija transformacija podataka i njihovo učitavanje korišćenjem metode *AddOperation*. Pored toga, vrši se i povezivanje ovih operacija kako bi se formirali tokovi podataka. Povezivanje operacija transformacija se realizuje pozivanjem operacije *BindOperations* za svaki *DataFlowPath* koji je definisan metapodacima. Instanciranje operacija transformacija podataka omogućeno je klasom *DataOperationManager* i realizuje se pozivom metode *Create*.



Slika 83. Dijagram sekvenci za podešavanje izvršnog okruženja procesa obrade podataka

Na sledećoj slici (slika 84) je predstavljeno instanciranje i podešavanje procesa obrade podataka (eng. *Create Data Process Instance*). Proces obrade podataka,

operacija transformacije i argument operacije su predstavljeni objektima *dataProcess*, *dataOperation* i *dataOperationArgument*, respektivno, uz napomenu da se kreiraju instance specifičnih klasa.

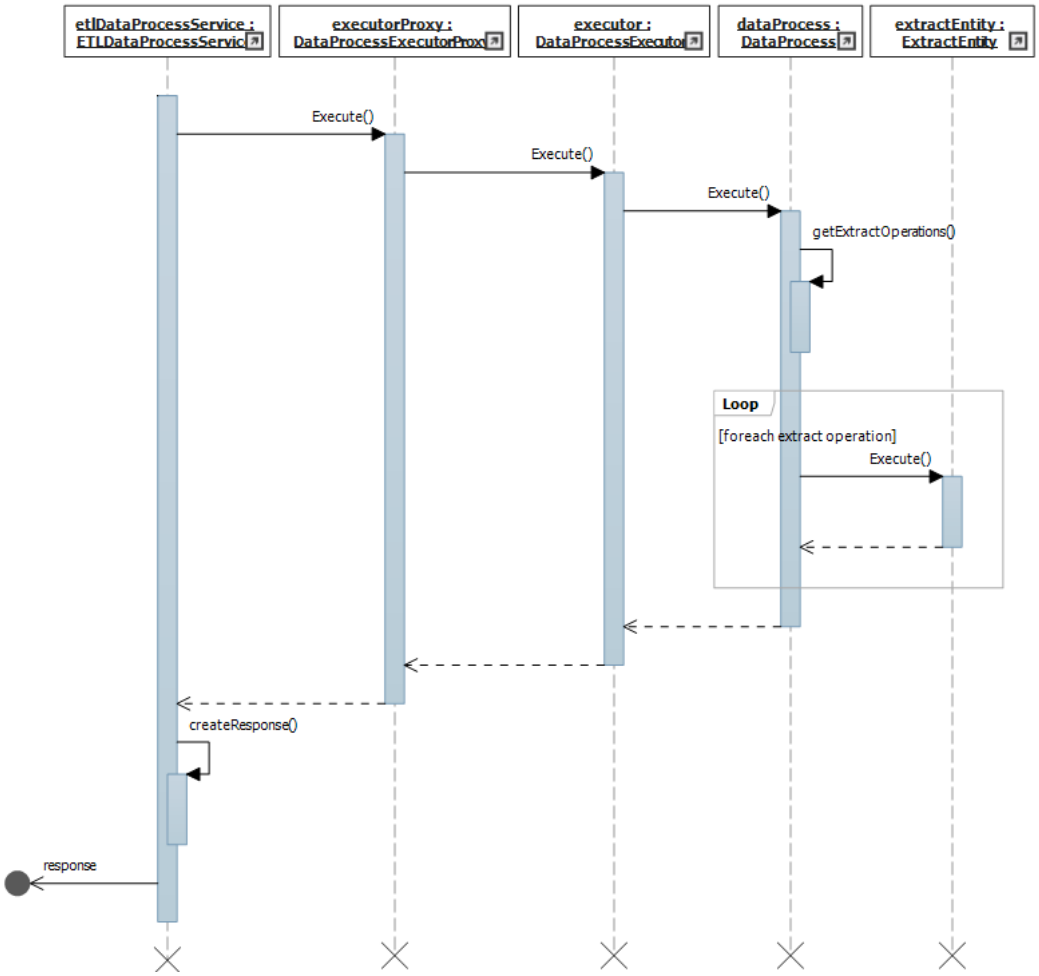


Slika 84. Dijagram sekvenci za instanciranje i podešavanje procesa obrade podataka

Izvršavanje operacija transformacija podataka

Izvršavanje operacija transformacija podataka započinje pozivom metode *Execute* klase *DataProcessExecutorProxy*.

S obzirom da je u prethodnom koraku izvršeno instanciranje i podešavanje procesa obrade podataka, u ovom koraku se formira kolekcija operacija ekstrahovanja podataka, a zatim se inicira njihovo izvršavanje pozivanjem metode *Execute*. Ostale operacije transformacije se automatski izvršavaju po pristizanju podataka (posledica implementiranog mehanizma razmene podataka, slika 78).



Slika 85. Dijagram sekvenci za izvršavanje operacija transformacija podataka

Proces obrade podataka je završen kada su sve operacije transformacije završile sa obradom podataka (atribut *isComplete* ima vrednost *true* na svim operacijama).

Oslobađanje korišćenjih resursa

Kao poslednji korak tokom izvršavanja procesa obrade podataka, sprovode se aktivnosti kojima se oslobađaju svi korišćeni resursi. Ovo se pre svega odnosi na oslobađanje aplikacionog domena koji je korišćen za izvršavanje procesa obrade podataka. Oslobađanje resursa se inicira pozivom metode *Dispose* klase *ETLDataProcessExecutorProxy*.

10.3. Izvršavanje ETL procesa

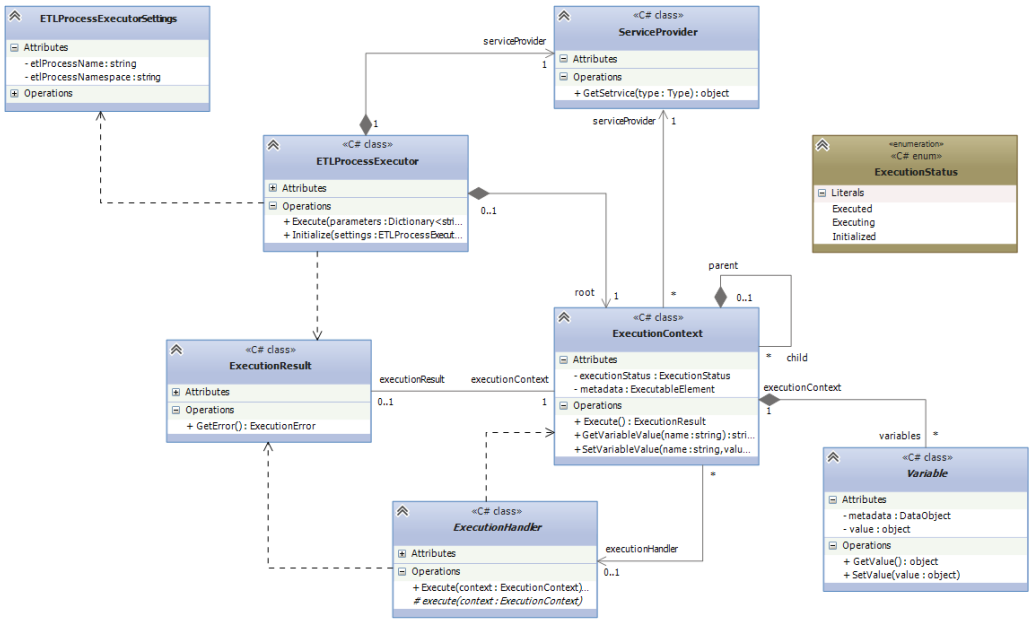
ETL proces se sastoji od većeg broja aktivnosti kojima se vrši transformacija poslovnih podataka u strateške informacije. Ovim aktivnostima predstavljaju se osnovne funkcionalnosti ETL procesa. Definisanje toka izvršavanja, odnosno redosleda izvršavanja ovih aktivnosti omogućeno je odgovarajućim skupom upravljačkih struktura (strana 109).

Implementacija ETL procesa podrazumeva implementaciju osnovnih funkcionalnosti ETL procesa, kao i implementaciju različitih mehanizama kojima je omogućeno upravljanje njihovim izvršavanjem. S obzirom da je implementacija osnovnih funkcionalnosti ETL procesa podržana posebnim okruženjem (strana 152), u ovom delu rada se razmatra implementacija toka izvršavanja ETL procesa. Specifično, razmatra se implementacija različitih mehanizama kojima su realizovane upravljačke strukture, ali i različite aktivnosti kojima je omogućeno izvršavanje osnovnih funkcionalnosti ETL procesa.

Osnovne klase predloženog rešenja predstavljene su dijagramom klasa (slika 86). Klasa *ETLProcessExecutor* je uvedena kako bi se obezbedila podrška izvršavanju ETL procesa. Ovom klasom su izložene dve metode: metoda *Initialize* kojom se vrši podešavanje izvršnog okruženja i metoda *Execute* kojom se inicira izvršavanje ETL procesa. Parametri metode *Execute* služe za prenos vrednosti ulaznih argumenata ETL procesa.

Upravljanje različitim komponentama koje su neophodne tokom izvršavanja ETL procesa (*ETLMetadataServiceAgent*, *ExecutionContextManager*, itd) omogućeno je klasom *ServiceProvider*.

Klasom *ExecutionContext* se predstavlja kontekst izvršavanja ETL procesa, odnosno kontekst izvršavanja aktivnosti ETL procesa. Ovom klasom se implementira stanje određene aktivnosti ETL procesa, ali se isto tako obezbeđuje i pristup različitim servisima koji su neophodni tokom njenog izvršavanja.

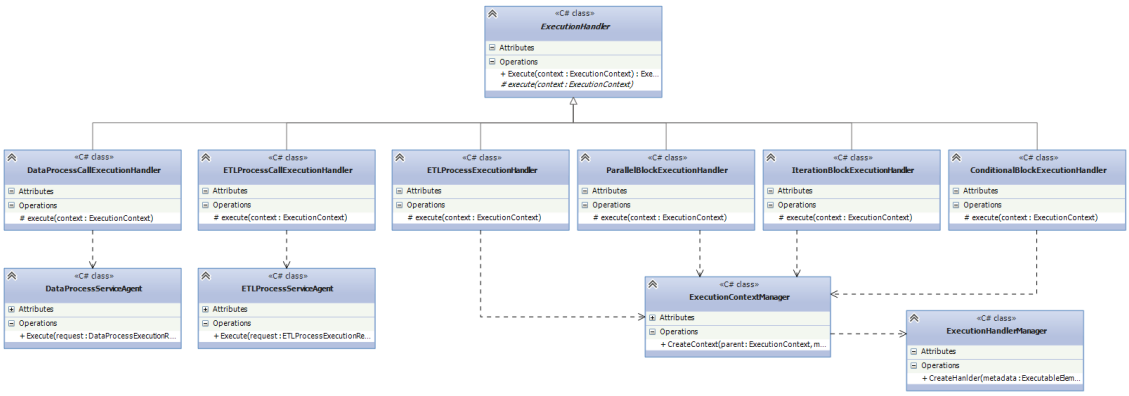


Slika 86. Dijagram klasa za izvršavanje ETL procesa

Svako pojedinačno izvršavanje aktivnosti ETL procesa podrazumeva kreiranje novog konteksta izvršavanja. Novi kontekst se kreira u okviru tekućeg konteksta, čime se formira odgovarajuća hijerarhija (*parent-child* veza) u čijem vrhu se nalazi kontekst izvršavanja ETL procesa (koreni kontekst izvršavanja). Nadređeni kontekst može biti kontekst izvršavanja ETL procesa ili kontekst izvršavanja složene aktivnosti.

Stanje aktivnosti ETL procesa definisano je skupom ulaznih i izlaznih argumenata, kao i skupom lokalnih promenljivih. Argumenti i lokalne promenljive predstavljaju se apstraktnom klasom *Variable*, odnosno njenim podklasama *InputArgument*, *OutputArgument* i *LocalVariable*. Uvođenjem ovih specifičnih klasa, omogućena je implementacija različitih ograničenja u vezi sa korišćenjem argumenata i lokalnih promenljivih. Na primer, vrednost izlaznog argumenta je dostupna tek nakon što je aktivnost izvršena (*executionStatus* ima vrednost *Executed*).

Izvršavanje aktivnosti ETL procesa podržano je apstraktnom klasom *ExecutionHandler*. Implementacija je data preko dve metode, pri čemu se prvom (*Execute*) implementira opšte ponašanje, svojstveno svim aktivnostima ETL procesa, dok se drugom (*execute*) implementiraju specifičnosti pojedinačnih aktivnosti. Obe metode kao ulazni argument očekuju *ExecutionContext*. Rezultat izvršavanja aktivnosti predstavljen je klasom *ExecutionResult*.

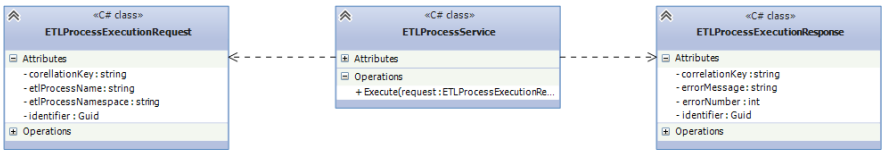


Slika 87. Hijerarhija klasa ExecutionHandler

Podrška različitim aktivnostima ETL procesa obezbeđena je implementacijom većeg broja podklasa klase *ExecutionHandler* (slika 87). Svakom podklasom je reimplementirana metoda *execute* kako bi se obezbedilo specifično ponašanje.

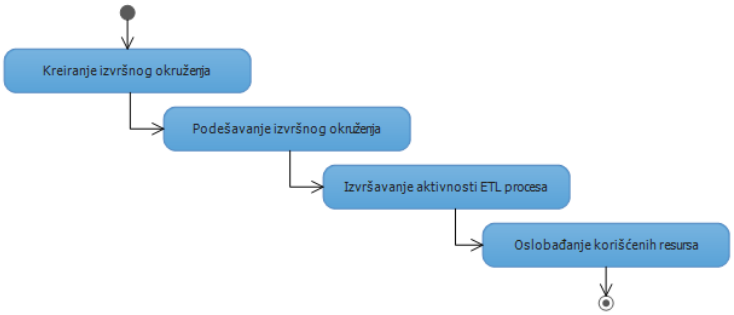
10.3.1. Osnovni scenario izvršavanja ETL procesa

Izvršavanje ETL procesa omogućeno je specifičnim servisom *ETLProcessService*. Izvršavanje se inicira pozivom metode *Execute*, odnosno prijemom poruke *ETLProcessExecutionRequest*, a završava se formiranjem i slanjem poruke *ETLProcessExecutionResponse*.



Slika 88. ETLProcessService Request/Response

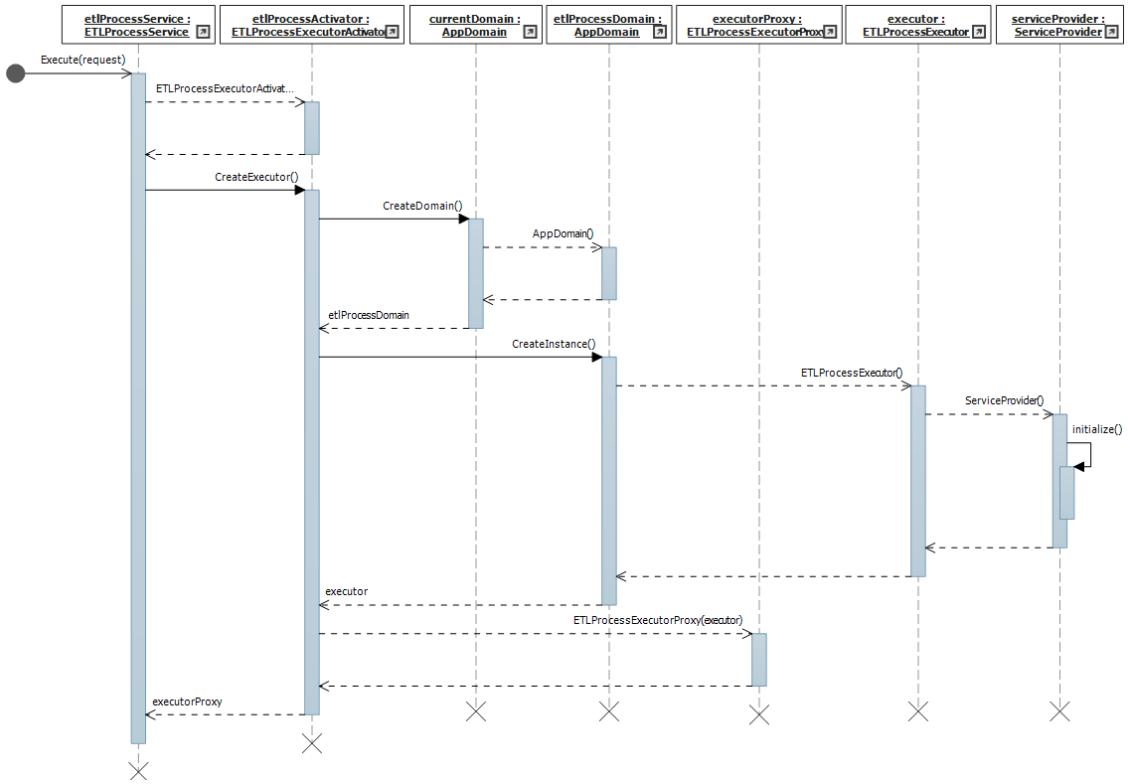
U nastavku su prikazani osnovni koraci tokom izvršavanja ETL procesa (slika 89), a zatim je dat njihov opis i odgovarajući dijagrami sekvenci.



Slika 89. Osnovni koraci tokom izvršavanja ETL procesa

Kreiranje izvršnog okruženja

Tokom ovog koraka kreira se specifično izvršno okruženje kojim će se omogućiti izvršavanje zahtevanog ETL procesa. Ovo podrazumeva kreiranje posebnog aplikacionog domena i instanciranje odgovarajućih objekata u njemu. Potreba za posebnim aplikacionim domenom proizilazi iz činjenice da se tokom izvršavanja ETL procesa generiše i kompajlira izvršni kod. Kreiraju se dinamički sklopovi i učitavaju u *.NET* izvršno okruženje.



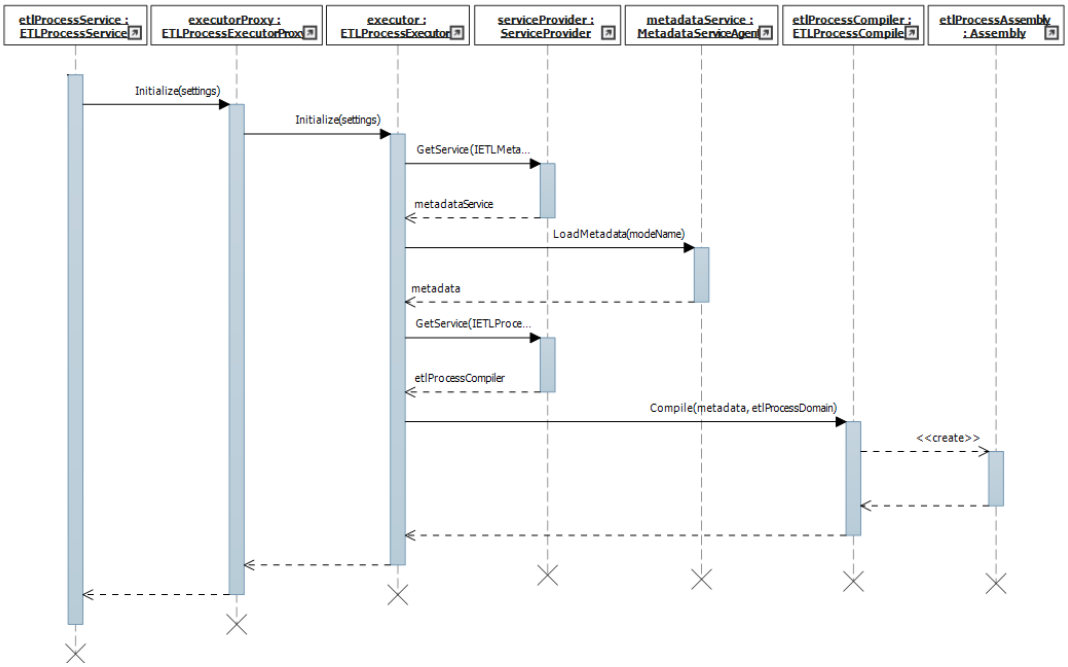
Slika 90. Dijagram sekvenci za kreiranje izvršnog okruženja ETL procesa

Navedena funkcionalnost je implementirana klasom *ETLProcessExecutionActivator* koja izlaže metodu *CreateExecutor*. Rezultat izvršavanja ove metode predstavljen je klasom *ETLProcessExecutorProxy* koja je uvedena sa ciljem da se apstrahuje komunikacija sa objektima iz novokreiranog izvršnog okruženja.

Podešavanje izvršnog okruženja

Podešavanje izvršnog okruženja podrazumeva učitavanje svih metapodataka koji su neophodni za izvršavanje određenog ETL procesa (model ETL procesa i referencirani modeli izraza), a zatim generisanje i kompajliranje izvršnog koda u skladu sa ovim metapodacima. Ovaj korak započinje pozivom metode *Initialize* klase *ETLProcessExecutorProxy*.

Učitavanje metapodataka omogućeno je klasom *MetadataServiceAgent* i realizuje se pozivom metode *LoadMetadata*. Ova klasa je uvedena sa ciljem da se apstrahuje komunikacija sa servisom *ETLMetadataService* kojim se izlažu metapodaci.



Slika 91. Dijagram sekvenci za podešavanje izvršnog okruženja ETL procesa

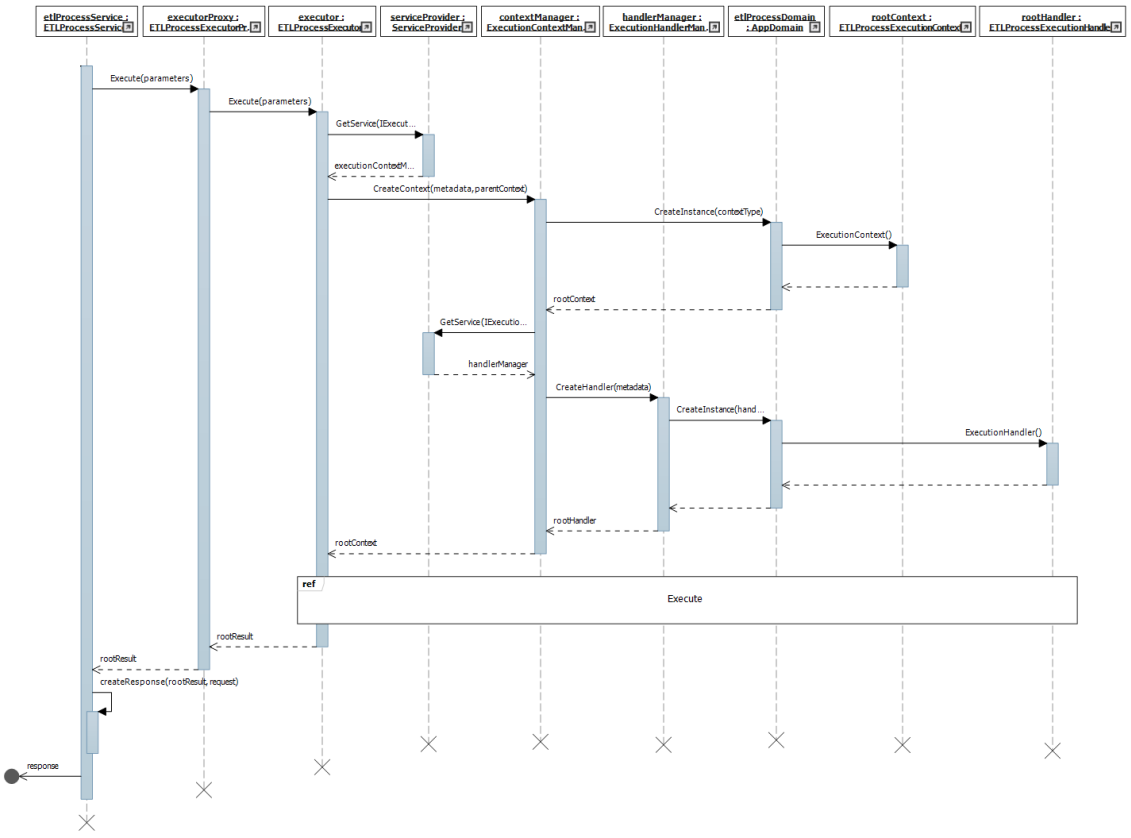
Generisanje i kompajliranje izvršnog koda podržano je posebno razvijenom klasom *ETLProcessCompiler* koja izlaže metodu *Compile*. Kao ulazni parametri ove metode, prosleđuju se prethodno učitani metapodaci i aplikacioni domen u koji će se

rezultati kompajliranja učitati. Rezultat kompajliranja je dinamički sklop u kome su definisani svi tipovi neophodni za izvršavanje ETL procesa.

Izvršavanje aktivnosti ETL procesa

Izvršavanje aktivnosti ETL procesa je vođeno metapodacima i podrazumeva instanciranje odgovarajućeg konteksta izvršavanja za svako pojedinačno izvršavanje aktivnosti. Svakom kontekstu izvršavanja se dodeljuje odgovarajuća instanca klase *ExecutionHandler* kojom je data konkretna implementacija aktivnosti.

Izvršavanje aktivnosti ETL procesa započinje pozivom metode *Execute* klase *ETLProcessExecutorProxy*. Za svaki ETL proces se kreira koreni kontekst izvršavanja (instanca klase *ETLProcessExecutionContext*) i poziva se metoda *Execute* čime se započinje izvršavanje njegovih aktivnosti.

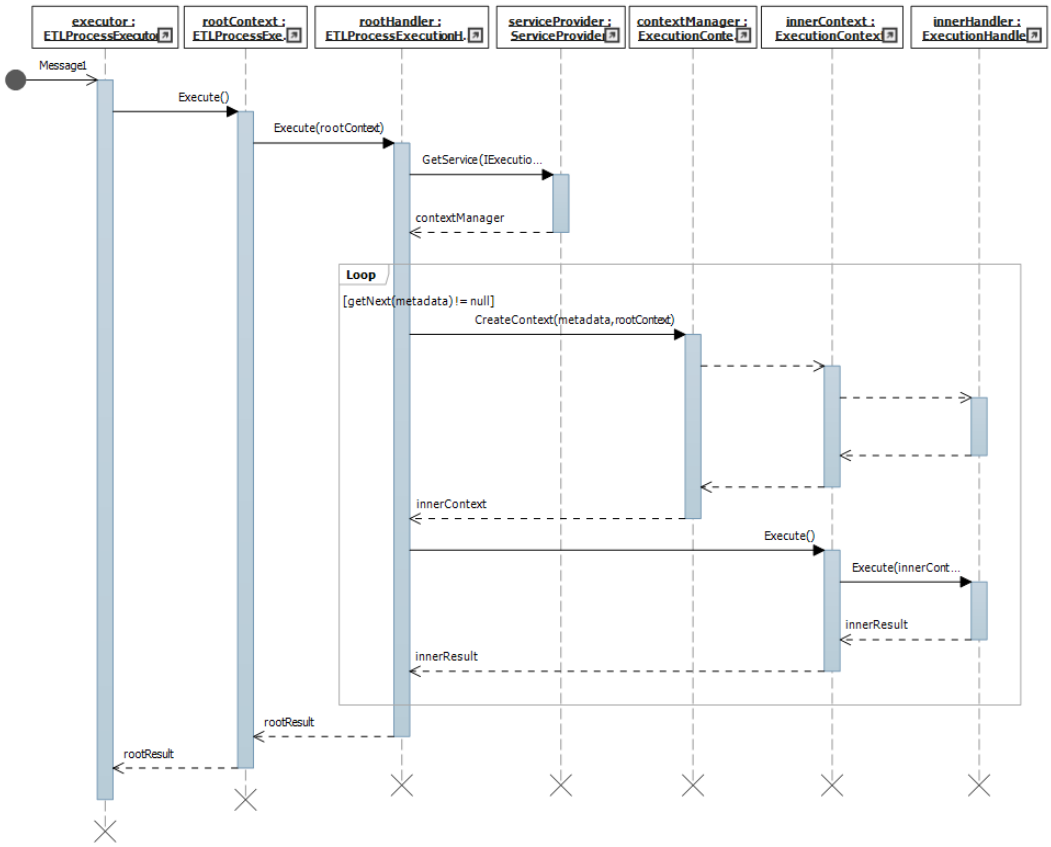


Slika 92. Dijagram sekvenci za izvršavanje aktivnosti ETL procesa

Kreiranje konteksta izvršavanja omogućeno je klasom *ExecutionContextManager*. Poziva se metoda *CreateContext* i prosleđuju se metapodaci i tekući kontekst

izvršavanja. Na osnovu metapodataka se dinamički instancira odgovarajući kontekst izvršavanja i automatski povezuje sa nadređenim kontekstom izvršavanja. Nakon toga, kreira se instanca odgovarajuće klase *ExecutionHandler* i dodeljuje se kontekstu izvršavanja. Instanciranje ove klase je takođe dinamičko na osnovu metapodataka i omogućeno je klasom *ExecutionHandlerManager*.

U nastavku je prikazano (slika 93) izvršavanje korenog konteksta (*rootContext*). Izvršavanje aktivnosti ETL procesa predstavljeno je objektima *innerContext* i *innerHandler* uz napomenu da se za svako pojedinačno izvršavanje aktivnosti kreiraju instance specifičnih klasa.



Slika 93. Dijagram sekvenci za izvršavanje korenog konteksta ETL procesa

Izvršavanjem poslednje aktivnosti ETL procesa završava se izvršavanje ETL procesa.

Oslobađanje korišćenih resursa

Kao poslednji korak tokom izvršavanja ETL procesa, sprovode se aktivnosti kojima se oslobađaju svi resursi koji su korišćeni tokom izvršavanja ETL procesa. Ovo se pre svega odnosi na oslobađanje aplikacionog domena koji je korišćen za izvršavanje ETL procesa. Oslobađanje resursa se inicira pozivom metode *Dispose* klase *ETLProcessExecutorProxy*.

11. ZAKLJUČAK

U tezi je dato originalno rešenje problema konceptualizacije i automatizacije ETL procesa. Predloženo rešenje se zasniva na formalnoj specifikaciji ETL procesa i njenoj automatizaciji uz pomoć specifičnog aplikacionog okvira.

U skladu sa DSM (Domain-Specific Modeling) pristupom, za formalnu specifikaciju predloženo je i razvijeno nekoliko novih domensko-specifičnih jezika: jezik za specifikaciju operacija transformacija podataka (ETL-O), jezik za specifikaciju toka izvršavanja ETL procesa (ETL-P), jezik za specifikaciju izraza (ETL-E) i jezik za specifikaciju šablona operacija transformacija (ETL-T). Svaki od ovih jezika definiše koncepte koji su relevantni za specifični aspekt ETL procesa. Stoga su predloženi domensko-specifični jezici veoma pogodni za rešavanje problema u domenu ETL sistema. Modelovanje ETL procesa zapravo se svodi na modelovanje određenog aspekta ETL procesa pomoću odgovarajućeg domensko-specifičnog jezika i na ovaj način se značajno smanjuje složenost modelovanja.

Za svaki uvedeni domensko-specifični jezik definisana je apstraktna i konkretna sintaksa i semantika. Jezički model apstraktne sintakse za svaki od uvedenih domensko-specifičnih jezika definisan je metamodelom kojim se opisuju specifični koncepti određenog aspekta ETL procesa. Pored toga, svakom definisanom konceptu pridružena su semantička pravila što implicira da je korišćenje takvog koncepta u modelovanju semantički kontrolisano.

Pored toga, u tezi je data i konkretna sintaksa za predložene domensko-specifične jezike. Konkretna sintaksa za predložene jezike ETL-O, ETL-P i ETL-T obezbeđuje grafičke elemente, koji reprezentuju koncepte definisane odgovarajućim jezičkim modelom i koriste se za formiranje dijagrama kao grafičke reprezentacije modela. Zapravo, razvijeni grafički editori omogućavaju sintaksno vođeno kreiranje dijagrama na osnovu definisanog odgovarajućeg domensko-specifičnog jezika. Pored grafičkih editora, razvijen je i odgovarajući tekstualni editor pomoću koga se formiraju modeli izraza u skladu sa definisanim ETL-E jezikom.

Obično velika složenost realnih ETL procesa utiče i na složenost njihovog modelovanja. Posebno mogu biti složeni modeli koji se kreiraju u skladu sa definisanim ETL-O jezikom. U cilju smanjenja složenosti modelovanja ETL procesa, u tezi je uvođenjem tri vrste dijagrama, kao grafičke reprezentacije modela zasnovanim na ETL-O jeziku, ostvaren hijerhijski opis ETL procesa. Na najvišem apstraktnom nivou kreira se dijagram složenih operacija transformacija, a na sledećim nižim nivoima formiraju se, kroz postepeno uvođenje detaljnijih opisa, sledeća dva detaljnija dijagrama: dijagram složene operacije transformacije i diagram proste operacije transformacije.

Implementacija specificiranih domensko-specifičnih jezika ostvarena je uvođenjem specifičnog aplikacionog okvira kao tehnološke podrške predloženoj formalnoj specifikaciji. Uvođenjem aplikacionog okvira značajno je podignut semantički nivo koji je implementaciono podržan i koji se može automatizovati. Implemetacija se zapravo zasniva na automatskoj transformaciji modela, formiranim u skladu sa odgovarajućim domensko-specifičnim jezikom, u izvrsni kôd aplikacionog okvira. Visok semantički nivo koji je implementaciono podržan aplikacionim okvirom, utiče na smanjenje broja koraka u razvoju ETL procesa, koji postaje više automatizovan i samim tim mnogo produktivniji.

Uvedena je i specifična ETL platforma (kao proširenje *Microsoft .NET platforme*) kojom je specificirana tehnološka podrška razvoju i izvršavanju modela ETL procesa. Zapravo, ETL platformom se opisuje tehnološka podrška modelovanju (tj. formiranju modela zasnovanim na odgovarajućim definisanim DSL) i implementaciji domensko-specifičnih jezika. Tehnološka podrška modelovanju ostvarena je razvojem odgovarajućih softverskih alata (pre svega sintaksnih grafičkih editora i tekstuelnog editora). Automatske transformacije modela, formiranih korišćenjem sintaksnih editora i samim tim zasnovanim na odgovarajućim DSL, u izvršni kôd aplikacionog okvira podržane su sa specifično razvijenim generatorima. Aplikacioni okvir podržan je skupom servisa koji omogućavaju izvršavanje i upravljanje izvršnim kôdom, generisanim iz formiranih modela.

U tezi je predstavljena opšta fizička softverska arhitektura ETL platforme, kao osnova za njenu implementaciju. Opšta fizička softverska arhitektura daje specifikaciju softverskog rešenja koje definiše osnovne komponente ETL platforme. Sve komponente su podeljene u dva osnovna sloja: razvojno okruženje i izvršno okruženje. Komponente razvojnog okruženja predstavljaju alate pomoću kojih se definiše apstraktna i konkretna sintaksa (u grafičkoj i tekstuelnoj notaciji) specificiranih jezika ETL procesa, kao i razvijene alate (sintaksne editore) koji se koriste za kreiranje modela u skladu sa odgovarajućim domensko-specifičnim jezikom. Izvršnom okruženju pripadaju komponente koje predstavljaju generatore za automatsko generisanje kôda iz formiranih modela, kao i komponente pomoću kojih je realizovan izvršni aplikacioni okvir. Fizička implementacija opšte fizičke arhitekture realizovana je u *Microsoft .NET* implementacionom okruženju uz korišćenje softverskih komponenti za realizaciju komponenti iz opšte fizičke arhitekture.

U tezi je data detaljna analiza postojećih rešenja problema konceptualizacije i automatizacije ETL procesa, razlike u odnosu na njih i prednost ovde predloženog rešenja.

U cilju validacije rešenja koje se predlaže u ovom radu i koje se zasniva na eksplicitnom korišćenju definisanih domensko-specifičnih jezika za modelovanje ETL procesa i automatskim transformacijama formiranih modela u izvršno okruženje, sprovedeno je više eksperimentalnih testiranja. Jedan od realizovanih testova (prikazan u tezi) primenjen je, uz podršku razvijenih softverskih komponenti, na razvoj ETL procesa u izabranom poslovnom domenu. Takođe, planiraju se i testiranja ovog rešenja u praksi u različitim poslovnim domenima.

11.1. Ostvareni doprinosi

Doprinos ove disertacije se može definisati kao originalno rešenje za konceptualizaciju i automatizaciju ETL procesa. Pojedinačno, ostvaren je veći broj doprinosa među kojima se izdvajaju:

Naučni doprinosi

- Pregled i detaljna analiza postojećih rešenja problema konceptualizacije i automatizacije ETL procesa, razlike u odnosu na njih i prednosti ovde predloženog rešenja.
- Predloženo rešenje zasniva se na formalnoj specifikaciji ETL procesa za koju je predloženo nekoliko novih domensko-specifičnih jezika, gde svaki od njih definiše koncepte koji su relevantni za specifični aspekt ETL procesa. Definisani su sledeći jezici:
    - jezik za specifikaciju operacija transformacija podataka (ETL-O),
    - jezik za specifikaciju toka izvršavanja ETL procesa (ETL-P),
    - jezik za specifikaciju izraza (ETL-E) i
    - jezik za specifikaciju šablona operacija transformacija (ETL-T).

  Svaki od predloženih jezika definiše koncepte koji su relevantni za specifičan aspekt ETL procesa i zbog toga su veoma pogodni za rešavanje problema u domenu ETL sistema. Na ovaj način se značajno smanjuje složenost modelovanja.
- U skladu sa DSM pristupom za svaki od uvedenih jezika definisana je apstraktna i konkretna sintaksa i semantika.
    - Jezički model apstraktne sintakse za svaki DSL definisan je metamodelom kojim se opisuju specifični koncepti određenog aspekta ETL procesa. Svakom definisanom konceptu pridružena su semantička pravila, pa je korišćenje takvog koncepta u modelovanju semantički kontrolisano.
    - Konkretna sintaksa za jezike ETL-O, ETL-P i ETL-T obezbeđuje grafičke elemente koji reprezentuju koncepte definisane odgovarajućim jezičkim modelom.

  - Za jezike ETL-P i ETL-T, uvedeni su odgovarajući sintaksni dijagrami, kao grafička reprezentacija modela formiranih u skladu sa datim jezicima:
    - dijagram toka izvršavanja ETL procesa i
    - dijagram šablona operacije transformacije podataka.
  - Uzimajući u obzir da u modelovanju realnih, složenih ETL procesa posebno mogu biti složeni modeli koji se kreiraju u skladu sa ETL-O jezikom, uvedena su tri tipa dijagrama:
    - dijagram složenih operacija transformacija,
    - dijagram složene operacije transformacije i
    - dijagram proste operacije transformacije,

    koji su grafička reprezentacija modela zasnovani na ETL-O jeziku. Na ovaj način ostvaren je hijerarhijski opis ETL procesa kojim se u velikoj meri smanjuje složenost modelovanja.

  - Konkretna sintaksa ETL-E jezika je data u tekstualnoj notaciji.
- Definisana je specifična ETL platforma koja opisuje slojeve za tehnološku podršku razvoju i izvršavanju modela ETL procesa.
- Definisana je opšta fizička softverska arhitektura predložene ETL platforme kao osnova za njenu implementaciju.

Stručni doprinosi

- Implementacija specificiranih domensko-specifičnih jezika ostvarena je uvođenjem specifičnog aplikacionog okvira kao tehnološke podrške predloženoj formalnoj specifikaciji. Implemetacija se zapravo zasniva na automatskoj transformaciji modela, formiranim u skladu sa odgovarajućim domensko-specifičnim jezikom, u izvrsni kôd aplikacionog okvira. Visok semantički nivo koji je implementaciono podržan aplikacionim okvirom, utiče na smanjenje broja koraka u razvoju ETL procesa, koji postaje više automatizovan i samim tim mnogo produktivniji.

- Realizovana je specifična ETL platforma za tehnološku podršku razvoju i izvršavanju modela ETL procesa, kao fizička implementacija predložene opšte fizičke softverske arhitekture.
    - Tehnološka podrška modelovanju ostvarena je razvojem odgovarajućih softverskih alata (pre svega sintaksnih editora). Razvijeni su grafički editori koji omogućavaju sintaksno vođeno kreiranje dijagrama, kao grafičke reprezentacije modela, na osnovu definisanog odgovarajućeg domensko-specifičnog jezika (ETL-O, ETL-P i ETL-T). Pored grafičkih editora, razvijen je i odgovarajući tekstualni editor pomoću koga se formiraju modeli izraza u skladu sa definisanim ETL-E jezikom. Dodatno, razvijen je i alat kojim je omogućeno automatsko generisanje dokumentacije ETL procesa.
    - Automatske transformacije modela, formiranih korišćenjem sintaksnih editora u izvršni kôd aplikacionog okvira podržane su sa specifično razvijenim generatorima.
    - Aplikacioni okvir podržan je skupom servisa koji omogućavaju izvršavanje i upravljanje izvršnim kôdom, generisanim iz formiranih modela.

11.2. Pravci budućeg istraživanja

Uzimajući u obzir rezultate koji su postignuti ovom disertacijom, u budućim istraživanjima se mogu razmatrati:

- unapređenja predloženog konceptualnog okvira ETL procesa kroz usavršavanje postojećih i uvođenje novih domensko-specifičnih jezika,
- unapređenja predloženog razvojnog okruženja kroz uvođenje novih funkcionalnosti, kao i novih alata (sintaksnih editora) u slučaju novih domensko-specifičnih jezika i
- unapređenja predloženog izvršnog okruženja, kroz usavršavanje rada postojećih servisa i uvođenje novih kojima bi se obezbedile dodatne funkcionalnosti.

Shodno navedenom, predlaže se:

- Proširivanje predloženog konceptualnog okvira ETL procesa kako bi se omogućilo predstavljanje različitih vrsta modela podataka na uniformni način, koji se koriste u operacijama transformacija. Moguće je definisati domensko-specifičan jezik sa grafičkom notacijom za uniformno predstavljanje različitih vrsta modela podataka.
- Proširivanje predloženog konceptualnog okvira ETL procesa kako bi se omogućilo predstavljanje hardverske infrastrukture na kojoj se servisi ETL platforme izvršavaju. Moguće je definisati domensko-specifičan jezik sa grafičkom notacijom za predstavljanje osnovnih koncepata hardverske infrastrukture.
- Proširivanje izvršnog okruženja kako bi se omogućilo optimalno izvršavanje modela ETL procesa u odnosu na raspoloživu hardversku infrastrukturu. Moguće je razviti specifičan servis za predloženu ETL platformu kojom će se, na osnovu modela ETL procesa i modela hardverske infrastrukture, odrediti optimalan plan izvršavanja ETL procesa *(Turajlić, 2014)*.

Pored navedenog, predlaže se i primena razvijenog softverskog rešenja u različitim domenima kako bi se potvrdila validnost rešenja. Pre svega, predlaže se primena na nekoliko realnih projekta na kojima će se paralelno primenjivati predloženo softversko rešenje. Na ovaj način biće moguće dobiti povratne informacije na osnovu kojih će se izvršiti dodatna kvalitativna evaluacija predloženog rešenja.

12. LITERATURA

Adamson, C. (2010). *Star Schema The Complete Reference.* McGraw-Hill Education.

Albin, S. T. (2003). *The art of software architecture: design methods and techniques* (T. 9). John Wiley & Sons.

Ambler, S. W. (2006, Novembar). *A RUP-based approach to developing a data warehouse - Part 1: Setting the stage.* Preuzeto Septembar 2012 sa IBM developerWorks: http://www.ibm.com/developerworks/rational/library/nov06/ambler/index.html

Ambler, S. W. (2006, Novembar). *A RUP-based approach to developing a data warehouse - Part 2: Building the data warehouse, one iteration at a time.* Preuzeto Septembar 2012 sa IBM developerWorks: http://www.ibm.com/developerworks/rational/library/dec06/ambler/

Aničić, N., Nešković, S., Vučković, M., & Cvetković, R. (2012). Specification of data schema mappings using weaving models. *Computer Science and Information Systems, 9*(2), 539-559.

Ariyachandra, T., & Watson, H. (2010). Key organizational factors in data warehouse architecture selection. *Decision Support Systems, 49*(2), 200-212.

Atkinson, C., & Kuhne, T. (2003). Model-Driven Development: A Metamodeling Foundation. *Software, IEEE, 20*(5), 36-41.

Babarogić, S. (2004). *Primena arhitekture bazirane na modelima na razvoj programskih sistema u .NET okruženju.* Magistarska teza, Fakultet organizacionih nauka, Univerzitet u Beogradu, Beograd.

Ballard, C., Farrell, D. M., Gupta, A., Mazuela, C., & Vohnik, S. (2006). *Dimensional Modeling: In a Business Intelligence Environment.* Vervante.

Bettin, J. (2004). Model-Driven Software Development. (D. Frankel, Ur.) *MDA Journal, 13*(4), 2-13.

Bezivin, J. (2004). In Search of a Basic Principle for Model Driven Engineering. *UPGRADE - The European Journal for the Informatics Professional, 5*(2), 21-24.

Bézivin, J., Hillairet, G., Jouault, F., Kurtev, I., & Piers, W. (2005). Bridging the MS/DSL Tools and the Eclipse Modeling Framework. *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*, *5.*

Böhnlein, M., & Ulbrich-vom Ende, A. (2000). Business process oriented development of data warehouse structures. U R. Jung, & R. Winter (Urednici), *Data Warehousing 2000* (str. 3-21). Physica-Verlag HD.

Bojičić, I. (2014). *Modelom vođen razvoj skladišta podataka zasnovan Data Vault pristupu - doktorska disertacija u izradi.* Beograd: Fakultet organizacionih nauka, Univerzitet u Beogradu.

Bezivin, J. (2005). On the unification power of models. *Software & Systems Modeling, 4*(2), 171-188.

Clark, T., Sammut, P., & Willans, J. (2008). *Applied metamodelling: a foundation for language driven development.* London: Middlesex University.

Cook, S. (2004). Domain-Specific Modeling and Model Driven Architecture. (D. Frankel, Ur.) *MDA Journal*, 2-10.

Cook, S., Jones, G., Kent, S., & Wills, A. C. (2007). *Domain Specific Development with Visual Studio DSL Tools.* Addison-Wesley.

Corr, L., & Stagnitto, J. (2011). *Agile Data Warehouse Design: Collaborative Dimensional Modeling, from Whiteboard to Star Schema.* DecisionOne Consulting.

Damhof, R. (2011, Januar). *[Blog post: Data Vault schools].* Preuzeto Septembar 2013 sa Data Management & Decision Support: http://prudenza.typepad.com/dwh/2011/01/data-vault-schools.html

Didonet, M., Fabro, D., Bézivin, J., & Valduriez, P. (2006). Weaving Models with the Eclipse AMW plugin. *In Eclipse Modeling Symposium, Eclipse Summit Europe.*

DSM Forum. (2013). *http://www.dsmforum.org/why.html*. Preuzeto 2013 sa DSM Forum: http://www.dsmforum.org/why.html

El Akkaoui, Z., & Zimányi, E. (2009). Defining ETL worfklows using BPMN and BPEL. *ACM twelfth international workshop on Data warehousing and OLAP* (pp. 41-48). ACM.

El Akkaoui, Z., Mazón, J.-N., Vaisman, A., & Zimányi, E. (2012). BPMN-Based Conceptual Modeling of ETL Processes. *Data Warehousing and a Knownledge Discovery. 7448*, pp. 1-14. Springer Berlin Heidelberg.

El Akkaoui, Zimányi, E., Mazón, J.-N., & Trujillo, J. (2011). A model-driven framework for ETL process development. *In Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP* (pp. 45-52). ACM.

Fowler, M. (2010). *Domain-Specific Languages.* Pearson Education.

France, R., & Rumpe, B. (2005). Domain specific modeling. *Software and Systems Modeling, 4*(1), 1-3.

France, R., & Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. *In 2007 Future of Software Engineering* (str. 37-54). IEEE Computer Society.

Frankel, D. (2002). *Model Driven Architecture: Applying MDA to Enterprise Computing.* John Wiley & Sons, Inc.

Ghosh, D. (2010). *DSLs in action.* Manning Publications Co.

Giordano, A. D. (2010). *Data Integration Blueprint and Modeling: Techniques for a Scalable and Sustainable Architecture.* Pearson Education.

Golfarelli, M. (2009). Data warehouse life-cycle and design. U L. Liu, & M. T. Özsu (Urednici), *Encyclopedia of Database Systems* (str. 658-664). Springer.

Golfarelli, M. (2010). From User Requirements to Conceptual Design in Warehouse Design: A Survey. U L. Bellatreche, *Data Warehousing Design and Advanced Engineering Applications: Methods for Complex Construction* (str. 1-16). IGI Global.

Golfarelli, M., & Rizzi, S. (2009). *Data Warehouse Design: Modern Principles and Methodologies.* McGraw-Hill, Inc.

Greenfield, J., & Short, K. (2003). Software factories: assembling applications with patterns, models, frameworks and tools. *In Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (str. 16-27). Anaheim, California, USA: ACM.

Greenfield, J., Short, K., Cook, S., & Kent, S. (2004). *Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools.* John Wiley & Sons.

Griesemer, B. (2009). *Oracle Warehouse Builder 11g: Getting Started.* Packt Publishing, Ltd.

Hazzard, K., & Jason, B. (2013). *Metaprogramming in. NET.* Manning Pub.

Inmon, W. H. (2005). *Building the Data Warehouse.* John Wiley & Sons.

Inmon, W. H., Strauss, D., & Neushloss, G. (2010). *DW 2.0: The Architecture for the Next Generation of Data Warehousing.* Morgan Kaufmann.

Iseger, M. (2010). *Domain-specific modeling for generative software development.* Preuzeto 2013 sa Developer Fusion: http://www.developerfusion.com/article/84844/domainspecific-modeling-for-generative-software-development/

Ivantsov, R. (2009). *Irony - .NET Language Implementation Kit*. Preuzeto 2012 sa CodePlexProject Hosting for Open Source Software: http://irony.codeplex.com/

Jarke, M., Lenzerini, M., Vassiliou, Y., & Vassiliadis, P. (2003). *Fundamentals of Data Warehouses* (2nd izd.). New York: Springer Berlin Heidelberg.

Jovanovic, V., & Bojicic, I. (2012). Conceptual Data Vault Model. *Proceedings of the SAIS Conference*, *23*, str. 131-136.

Jovanovic, V., Bojicic, I., Knowles, C., & Pavlic, M. (2012). Persistent staging area models for data warehouses. *Issues in Information Systems, 13*, 121-132.

Kelly, S., & Tolvanen, J. P. (2008). *Domain-Specific Modeling: Enabling Full Code Generation.* John Wiley & Sons.

Kimball, R., & Ross, M. (2002). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling.* John Wiley & Sons.

Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., & Becker, B. (2010). *The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence.* John Wiley & Sons.

Kleppe, A. G., Warmer, J., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison-Wesley Professional.

Kosar, T., Oliveira, N., Mernik, M., Pereira, V. J., Črepinšek, M. D., & Henriques, R. P. (2010). Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems, 7*(2), 247-264.

Kuhne, T. (2006). Matters of (meta-) modeling. *Software and Systems Modeling, 5*(4), 369-385.

Kurtev, I., Bezivin, J., Jouault, F., & Valduriez, P. (2006). Model-based DSL frameworks. *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (str. 602-616). Portland, Oregon,USA: ACM.

Lazarević, B., Marjanović, Z., Aničić, N., & Babarogić, S. (2010). *Baze podataka.* Beograd: Fakultet organizacionih nauka.

Linstedt, D. (2010, Avgust). *[Blog post: Data Vault and Staging Area].* Preuzeto Septembar 2013 sa Dan Linstedt: http://danlinstedt.com/datavaultcat/data-vault-and-staging-area/

Linstedt, D. E. (2002, July). *Data Vault Series 1 - Data Vault Overview.* Preuzeto 2012 sa The Data Administration Newsletter: http://www.tdan.com/view-articles/5054/

Linstedt, D. E. (2003, April). *Data Vault Series 3 - End Dates and Basic Joins.* Preuzeto Jun 2012 sa The Data Administration Newsletter: http://www.tdan.com/view-articles/5067/

Linstedt, D. E. (2003, Januar). *Data Vault Series 2 - Data Vault Components.* Preuzeto 2012 sa The Data Administration Newsletter: http://www.tdan.com/view-articles/5155/

Linstedt, D. E. (2004, Januar). *Data Vault Series 4 - Link Tables.* Preuzeto Jun 2012 sa The Data Administration Newsletter: http://www.tdan.com/view-articles/5172/

Linstedt, D. E. (2005, Januar). *Data Vault Series 5 - Loading Practices.* Preuzeto Jun 2012 sa The Data Administration Newsletter: http://www.tdan.com/view-articles/5285/

Linstedt, D., & Graziano, K. (2011). *Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault.* CreateSpace.

List, B., Bruckner, R., Machaczek, K., & Schiefer, J. (2002). A comparison of data warehouse development methodologies case study of the process warehouse. *In Database and Expert Systems Applications. LNCS 2453*, str. 203-215. Springer Berlin Heidelberg.

Luján-Mora, S., & Trujillo, J. (2005). A Data Warehouse Engineering Process. *Advances in Information Systems. LNCS 3261*, str. 14-23. Springer Berlin Heidelberg.

Luján-Mora, S., Vassiliadis, P., & Trujillo, J. (2004). Data Mapping Diagrams for Data Warehouse Design with UML. *In Conceptual Modeling–ER 2004. LNCS 3288*, str. 191-204. Springer Berlin Heidelberg.

Mazón, J.-N., & Trujillo, J. (2008). An MDA approach for the development of data warehouses. *Decision Support Systems, 45*(1), 41-58.

Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR), 37*(4), 316-344.

Microsoft. (2012). *Microsoft Visual Studio 2012 Visualization & Modeling SDK*. Preuzeto 2013 sa Microsoft Download Center: http://www.microsoft.com/en-us/download/details.aspx?id=30680

Microsoft. (2014). *Garbage Collection.* Preuzeto 2014 sa Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/0xy59wtx(v=vs.110).aspx

Microsoft. (2014). *Observer Design Pattern.* Preuzeto 2014 sa Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ee850490(v=vs.110).aspx

Microsoft Patterns & Practices Team. (2009). *Microsoft Application Architecture Guide.* Microsoft Corporation.

Milicev, D. (2009). *Model-Driven Development with Executable UML.* John Wiley & Sons.

Muñoz, L., Mazón, J.-N., & Trujillo, J. (2009). Automatic generation of ETL processes from conceptual models. *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP* (str. 33-40). ACM.

Muñoz, L., Mazón, J.-N., Pardillo, J., & Trujillo, J. (2008). Modelling ETL Processes of Data Warehouses with UML Activity Diagrams. *In On the Move to Meaningful Internet Systems: OTM 2008 Workshops. LNCS 5333*, str. 44-53. Springer Berlin Heidelberg.

Object Management Group. (2003). *Common Warehouse Metamodel 1.1.* Specification, Object Management Group, Inc.

Object Management Group. (2003). *MDA Guide Version 1.0.1.* Specification, Publisher Object Management Group, Inc.

Object Management Group. (2011, Avgust). *Unified Modeling Language (UML) version 2.4.1.* Specification, Object Management Group, Inc.

Object Management Group. (2011, Januar). *Business Process Model and Notation (BPMN) Version 2.0.* Specification, Object Management Group, Inc.

Petrović, M. (2012). *Uporedna analiza izabranih metodologija razvoja softvera.* Seminarski rad - Razvoj informacionih sistema (prof. dr Zoran Marjanović), Fakultet organizacionih nauka, Univerzitet u Beogradu, Beograd.

Petrović, M. (2013). *Prikupljanje i analiza zahteva u procesu razvoja skladišta podataka.* Seminarski rad - Upravljanje podacima (prof. dr Zoran Marjanović), Fakultet organizacionih nauka, Univerzitet u Beogradu.

Pilone, D., & Pitman, N. (2005). *UML 2.0 in a Nutshell.* O'Reilly Media, Inc.

Ponniah, P. (2011). *Data Warehousing Fundamentals for IT Professionals.* John Wiley & Sons.

Poole, J. (2003). *Common Warehouse Metamodel Developers Guide* (T. 24). John Wiley & Sons.

Recchia, P., & Guerot, A. (2009). *[Blog post: Multiply Dsl points of view].* Preuzeto 2011 sa MexEdge: A smart coding blog: http://blog.mexedge.com/2009/01/13/multiply-dsl-points-of-view/

Reeves, L. (2009). *A Managers Guide to Data Warehousing.* John Wiley & Sons.

Regardt, O., Rönnbäck, L., Bergholtz, M., Johannesson, P., & Wohed, P. (2009). Anchor Modeling. *In Conceptual Modeling-ER 2009, LNCS. 5829*, pp. 234-250. Springer Berlin Heidelberg.

Rizzi, S. (2008). Data Warehouse. U B. W. Wah (Ur.), *Wiley Encyclopedia of Computer Science and Engineering* (str. 1-9). John Wiley & Sons, Inc.

Rizzi, S. (2009). Business Intelligence. U L. Liu, & M. T. Özsu (Urednici), *Encyclopedia of Database Systems* (str. 287-288). Springer.

Romero, O., & Abelló, A. (2009). A survey of multidimensional modeling methodologies. *International Journal of Data Warehousing and Mining (IJDWM), 5*(2), 1-23.

Romero, O., & Abello, A. (2010). A framework for multidimensional design of data warehouses from ontologies. *Data & Knowledge Engineering, 69*(11), 1138-1157.

Rönnbäck, L., Regardt, O., Bergholtz, M., Johannesson, P., & Wohed, P. (2010). Anchor modeling - Agile information modeling in evolving data environments. *Data & Knowledge Engineering, 69*(12), 1229-1253.

Silver, B. (2011). *BPMN Method and Style: With BPMN Implementer's Guide.* Cody-Cassidy Press.

Simitsis, A. (2005). Mapping conceptual to logical models for ETL processes. *Proceedings of the 8th ACM international workshop on Data warehousing and OLAP* (str. 67-76). ACM.

Simitsis, A., & Vassiliadis, P. (2003). A methodology for the conceptual modeling of ETL processes. *In CAiSE workshops*, *75.*

Simitsis, A., & Vassiliadis, P. (2008). A method for the mapping of conceptual designs to logical blueprints for ETL processes. *Decision Support Systems, 45*(1), 22-40.

Simitsis, A., Vassiliadis, P., Terrovitis, M., & Skiadopoulos, S. (2005). Graph-based modeling of ETL activities with multi-level transformations and updates. *In Data Warehousing and Knowledge Discovery*, 43-52.

Suknović, M., & Delibašić, B. (2010). *Poslovna inteligencija i sistemi za podršku odlučivanju.* Beograd: Fakultet organizacionih nauka, Univerzitet u Beogradu.

Trujillo, J., & Luján-Mora, S. (2003). A UML Based Approach for Modeling ETL Processes in Data Warehouses. *In Conceptual Modeling-ER 2003. 2813*, str. 307-320. Springer Berlin Heidelberg.

Turajlić, N. (2014). *Novi modeli i metode za selekciju i kompoziciju web servisa na osnovu nefunkcionalnih karakteristika (Doktorska disertacija).* Beograd: Fakultet organizacionih nauka, Univerzitet u Beogradu.

Turajlić, N., Petrović, M., & Vučković, M. (2014). Analysis of ETL Process Development Approaches: Some Open Issues. *Proceedings of the XIV International Symposium SYMORG 2014* (str. 41-45). University of Belgrade, Faculty of Organizational Sciences.

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-Specific Languages: An Annotated Bibliography. *Sigplan Notices, 35*(6), 26-36.

Vassiliadis, P. (2009). A Survey of Extract-Transform-Load Technology. *International Journal of Data Warehousing and Mining (IJDWM), 5*(3), 1-27.

Vassiliadis, P., Simitsis, A., & Baikousi, E. (2009). A taxonomy of ETL activities. *In Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP* (str. 25-32). ACM.

Vassiliadis, P., Simitsis, A., & Skiadopoulos, S. (2002, Novembar). Conceptual modeling for ETL processes. *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP* (str. 14-21). ACM.

Vassiliadis, P., Simitsis, A., & Skiadopoulos, S. (2002. Maj). Modeling ETL activities as graphs. *In DMDW*, *58*, str. 52-61. Toronto, Canada.

Vassiliadis, P., Simitsis, A., Georgantas, P., & Terrovitis, M. (2003). A Framework for the Design of ETL Scenarios. *In Advanced Information Systems Engineering* (str. 520-535). Springer Berlin Heidelberg.

Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., & Skiadopoulos, S. (2005). A generic and customizable framework for the design of ETL scenarios. *Information Systems, 30*(7), 492-525.

Vassiliadis, P., Vagena, Z., Skiadopoulos, S., Karayannidis, N., & Sellis, T. (2001). ARKTOS: towards the modeling, design, control and execution of ETL processes. *Information Systems, 26*(8), 537-561.

Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L., i drugi. (2013). *DSL Engineering-Designing, Implementing and Using Domain-Specific Languages.* dslbook.org.

Vučković, M., Petrović, M., Turajlić, N., & Stanojević, M. (2012, Decembar). The Specification of ETL Transformation Operations based on Weaving Models. *INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL, 7*, 968-975.

Watson, H. J., & Ariyachandra, T. (2005). *Data Warehouse Architectures: Factors in the Selection Decision and the Success of the Architectures.* Preuzeto 2013 sa http://www.terry.uga.edu/~hwatson/DW_Architecture_Report.pdf: Terry College of Business, University of Georgia.

Winter, R., & Strauch, B. (2003). A method for demand-driven information requirements analysis in data warehousing projects. *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference* (pp. 9-pp). IEEE.

BIOGRAFIJA AUTORA

Marko (Vlastimir) Petrović je rođen 07. avgusta 1977. godine u Beogradu. Završio je Treću beogradsku gimnaziju („Sveti Sava“) u Beogradu, prirodno-matematički smer. Nakon završene gimnazije, 1996. godine upisao je Fakultet organizacionih nauka, smer informacioni sistemi. Diplomirao je 2002. godine sa prosečnom ocenom 8,19 a diplomski rad pod naslovom „*Razvoj Web Servisa korišćenjem .NET platforme*“ je odbranio sa ocenom deset. Poslediplomske (magistarske) studije na Fakultetu organizacionih nauka, smer informacioni sistemi upisao je 2002. godine, a zatim 2006. godine upisuje doktorske studije na Fakultetu organizacionih nauka, smer informacioni sistemi. Položio je sve ispite predviđene planom i programom, sa prosečnom ocenom 10.

◊ *Radno iskustvo*

- Od 2001. godine angažovan je kao saradnik na Fakultetu organizacionih nauka
- Od 2003. godine zaposlen na Fakultetu organizacionih nauka u zvanju asistent-pripravnik
- Od 2007. godine zaposlen na Fakultetu organizacionih nauka u zvanju asistent
- Od 2014. godine zaposlen na Fakultetu organizacionih nauka (Centar za razvoj informacionih sistema) u zvanju stručni saradnik

◊ *Nastavne aktivnosti*

Od 2001. do 2003. godine je radio kao saradnik na Fakultetu organizacionih nauka u Beogradu, na predmetima Principi programiranja i Projektovanje programa. Od 2003. godine do 2007. godine je radio kao asistent-pripravnik na Fakultetu organizacionih nauka u Beogradu, na predmetima Principi programiranja, Uvod u informacione sisteme, Arhitektura računara i operativni sistemi, Programski jezici i prevodioci i Projektovanje programa. Od 2007. do 2014. godine radi kao asistent

na Fakultetu organizacionih nauka u Beogradu na predmetima Programski jezici i Osnove informaciono komunikacionih tehnologija.

Prilikom evaluacije od strane studenata njegov pedagoški rad je redovno ocenjivan visokom ocenom.

◊ *Prikaz nastavnih aktivnosti u okviru doktorskih studija*

Doktorske studije je upisao školske 2006/2007 na Fakultetu Organizacionih nauka (smer informacioni sistemi). Položio je sve planom i programom predviđene ispite sa prosečnom ocenom 10 i to:

- Metodologija programiranja 10 ESPB
- Razvoj programa i programski jezici 10 ESPB
- Izabrana poglavlja iz operacionih istraživanja 10 ESPB
- Upravljanje preduzećem – menadžment 10 ESPB
- Projektovanje organizacione strukture 10 ESPB
- Statistika u menadžmentu 10 ESPB
- Metodologija i organizacija naučnih istraživanja 10 ESPB
- Razvoj informacionih sistema 10 ESPB
- Upravljanje podacima 10 ESPB

Odbranio je pristupni rad pod nazivom *„Razvoj ETL procesa zasnovan na MDD pristupu"* i ostvario 30 ESPB bodova. Ukupno je ostvario 120 ESPB bodova sa prosečnom ocenom 10.

◊ *Pregled projekata*

Pregled izabranih projekata i softverskih rešenja u čijoj je izradi učestvovao kao član ili vođa tima:

- „***NCTS***", Ministarstvo finansija, Uprava carina, Srbija, Beograd, 2013-2014.
- „***PopisPoljoprivrede***", Republički zavod za statistiku, Srbija, Beograd, 2012.
- „***ResursniPortal***", Ministarstvo zdravlja, Crna Gora, Podgorica, 2012.
- „***Popis***", Republički zavod za statistiku, Srbija, Beograd, 2011-2012.
- „***Tehnis***", Ministarstvo ekonomije i regionalnog razvoja, Srbija, Beograd, 2010.
- „***BelVille***", Blok 67 Associates d.o.o., Srbija, Beograd, 2008-2010.
- „***DirectDebit***", Udruženje banaka Srbije, Srbija, Beograd, 2008-2009.
- „***TiCat***", Telekom Srbija, Srbija, Beograd, 2007-2009.
- „***TB LAB 2006***", Ministarstvo Zdravlja, Srbija, Beograd, 2006-2007.
- „***Cybersure***", Mantacore Spearhead d.o.o., Beograd, 2006-2007.

- „***TB 2005***“, Ministarstvo Zdravlja, Srbija, Beograd, 2005.

◊ *Pregled publikovanih i objavljenih radova*

Objavio je, u saradnji sa drugim autorima, više naučnih radova u časopisima međunarodnog i nacionalnog značaja, kao i u zbornicima sa domaćih i međunarodnih konferencija.

Rad objavljen u časopisu međunarodnog značaja (SCIe lista)

- M. Vučković, M. Petrović, N. Turajlić, M. Stanojević, „***The Specification of ETL Transformation Operations based on Weaving Models***“, Int J Comput Commun, ISSN 1841-9836, 7(5): 968-975, 2012. IF 2012 – 0.441 (M23)

Rad objavljen na međunarodnoj konferenciji

- N. Turajlić, M. Petrović, M. Vučković, „***Analysis of ETL Process Development Approaches: Some Open Issues***“. In Proc. of SYMORG'14, 45-51. 2014.
- N. Turajlić, M. Petrović, M. Vučković, I. Dragović, „***Groundwork for Presentation Pattern Metamodels***“, INFOTEH, Vol.12, 731-736, Jahorina, 2013.

Rad objavljen u domaćem časopisu

- M. Petrović, N. Turajlić, I. Dragović, „***Pregled i uporedna analiza prezentacionih paterna***“, InfoM 9(34), 35-41, Beograd, 2010.
- S. Nešković, M. Petrović, „***Modelovanje poslovnih procesa korišćenjem OMG BPMN 2.0 standarda***“, InfoM 8(31), 12-18, Beograd, 2009.
- M. Minović, V. Štavljanin, D. Starčević, M. Petrović, „***Primena J2ME tehnologija u mobilnom bankarstvu***“, InfoM 3(10), 33-37, Beograd, 2004.
- S. Lazarević, M. Petrović, „***Korelacija jednakosti, hešinga i nasleđivanja***“, InfoM 2(8), 37-40, Beograd 2003.

Rad objavljen na domaćoj konferenciji

- M. Petrović, M. Minović, S. Lazarević, „***Mobilni uređaji i .NET Web servisi***“, Zbornik radova YU INFO, Kopaonik, 2006.
- M. Mihić, M. Petrović, J. Tomašević, „***Web servisi kao podrška primeni Cost-Benefit analize***“, Zbornik radova YU INFO, Kopaonik, 2006.
- S. Lazarević, M. Petrović, „***Korelacija jednakosti, hešinga i nasleđivanja***“, Zbornik radova SYMOPIS, 247-251, Herceg Novi, 2003.

Izjava o autorstvu

Potpisani: Marko Petrović

Broj indeksa: 14/2006

Izjavljujem

da je doktorska disertacija pod naslovom

RAZVOJ PROCESA EKSTRAKCIJE, TRANSFORMACIJE I PUNJENJA PODATAKA SKLADIŠTA PODATAKA ZASNOVAN NA MODELOM VOĐENOM PRISTUPU

- rezultat sopstvenog istraživačkog rada,
- da predložena disertacija u celini ni u delovima nije bila predložena za dobijanje bilo koje diplome prema studijskim programima drugih visokoškolskih ustanova,
- da su rezultati korektno navedeni i
- da nisam kršio autorska prava i koristio intelektualnu svojinu drugih lica.

U Beogradu,

11.09.2014. godine

Potpis doktoranda

Izjava o istovetnosti štampane i elektronske verzije doktorskog rada

Ime i prezime autora:	Marko Petrović
Broj indeksa:	14/2006
Studijski program:	Informacioni sistemi
Naslov rada:	Razvoj procesa ekstrakcije, transformacije i punjenja podataka skladišta podataka zasnovan na modelom vođenom pristupu
Mentor:	prof. dr Zoran Marjanović

Potpisani Marko Petrović

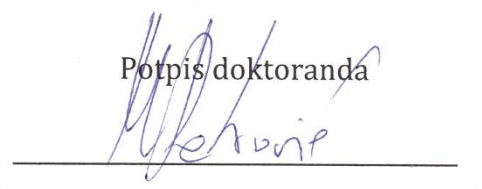
Izjavljujem da je štampana verzija mog doktorskog rada istovetna elektronskoj verziji koju sam predao za objavljivanje na portalu **Digitalnog repozitorijuma Univerziteta u Beogradu.**

Dozvoljavam da se objave moji lični podaci vezani za dobijanje akademskog zvanja doktora nauka, kao što su ime i prezime, godina i mesto rođenja i datum odbrane rada.

Ovi lični podaci mogu se objaviti na mrežnim stranicama digitalne biblioteke, u elektronskom katalogu i u publikacijama Univerziteta u Beogradu.

U Beogradu,
11.09.2014. godine

Potpis doktoranda



Izjava o korišćenju

Ovlašćujem Univerzitetsku biblioteku „Svetozar Marković“ da u Digitalni repozitorijum Univerziteta u Beogradu unese moju doktorsku disertaciju pod naslovom:

RAZVOJ PROCESA EKSTRAKCIJE, TRANSFORMACIJE I PUNJENJA PODATAKA SKLADIŠTA PODATAKA ZASNOVAN NA MODELOM VOĐENOM PRISTUPU

koja je moje autorsko delo.

Disertaciju sa svim prilozima predao sam u elektronskom formatu pogodnom za trajno arhiviranje.

Moju doktorsku disertaciju pohranjenu u Digitalni repozitorijum Univerziteta u Beogradu mogu da koriste svi koji poštuju odredbe sadržane u odabranom tipu licence Kreativne zajednice (Creative Commons) za koju sam se odlučio.

1. Autorstvo
2. Autorstvo – nekomercijalno
3. **Autorstvo – nekomercijalno – bez prerade**
4. Autorstvo – nekomercijalno – deliti pod istim uslovima
5. Autorstvo – bez prerade
6. Autorstvo – deliti pod istim uslovima

U Beogradu,
11.09.2014. godine

Potpis doktoranda