Prijedlog arhitekture za generator opisnika ETL procesa

Matija Novak, Dragutin Kermek, Ivan Magdalenić
Sveučilište u Zagrebu
Fakultet organizacije i informatike
Pavlinska 2, 42000 Varaždin
{matija.novak, dragutin.kermek, ivan.magdalenic}@foi.hr

Sažetak. *U kreiranju skladišta podataka najvažniji proces na koji se troši najviše vremena jest proces ekstrakcije, transformacije i učitavanja podataka (eng. Extract Transform Load - ETL), stoga je razumljivo da se taj proces treba automatizirati. U ovome radu fokus je na automatizaciji generiranja opsinika ETL procesa, koja danas nije dovoljno istražena. Cilj je izgradnja sustava koji koristi semantiku kod modeliranja ETL procesa te koji predlaže potrebne transformacije i mapiranja za automatsko generiranje opisnika ETL procesa. U ovome radu prezentirana je arhitektura koja omogućava automatsko integriranje tradicionalnih ETL alata i sustava koji generira potrebna mapiranja i transformacije korištenjem ontologija. Arhitektura prototipa je bazirana na porukama, što omogućuje paralelno procesiranje.*

Ključne riječi. ETL, skladište podataka, arhitektura, generator, ontologije, bazirano na porukama

1 Uvod

Proces ekstrakcije, transformacije i učitavanja (eng. *Extract Transform Load* - ETL) [1] je najvažniji dio izgradnje skladišta podataka koji iziskuje oko 70% vremena i resursa. ETL se koristi za ekstrakciju i transformaciju podataka iz različitih izvora, što uključuje čišćenje i usklađivanje podataka koji dolaze iz različitih izvora te učitavanje podataka u samo skladište podataka (eng. *data warehouse* - DW). [2] Postoji puno različitih tema istraživanja u području ETL-a, a neke od glavnih tema su: načini efikasnog provođenja ETL-a [3, 4], načini optimiziranja ETL-a[5, 6], načini modeliranja ETL-a na konceptualnoj i logičkoj razini [7, 8], načini automatizacije cijelog procesa [9, 10] i njegova testiranja [11, 12] itd.

U ovome radu fokus je na automatizaciji ETL procesa. Postoje različite vrste automatizacije ETL-a: automatsko prepoznavanje potrebnih mapiranja i transformacija, automatsko testiranje ETL-a, automatsko generiranje opisnika ETL procesa i sl. Naš fokus je automatsko generiranje opisnika ETL procesa (eng. *automatic generation of ETL workflows*), a glavna ideja rada je korištenje sustava koji koristi semantiku kod modeliranja ETL procesa te koji predlaže potrebne transformacije i mapiranja za automatsko generiranje opisnika ETL procesa. Publikacije koje opisuju takve sustave (npr. [13, 14, 15]) fokusirane su na predlaganje potrebnih transformacija i mapiranja, ali samo korištenje navedenog nije dovoljno istraženo. Pregledom literature nisu pronađeni primjeri korištenja tih prijedloga za automatsko generiranje opisnika ETL procesa.

Glavna ideja ovoga rada je automatsko iskorištavanje informacija o potrebnim transformacijama i mapiranjima koje su dobivene od spomenutih sustava u tradicionalnim ETL alatima. Kako bi to bilo moguće, u ovome radu predlaže se arhitektura za generator opisnika ETL procesa i kreiran je prototip alata pomoću Apache Camel [16] biblioteke. Generator opisnika ETL procesa (eng. *ETL workflow generator*) jest alat koji koristi informacije o transformacijama i mapiranjima te generira opisnik ETL procesa za postojeće tradicionalne ETL alate.

Iako se ovo istraživanje fokusira na ETL u domeni skladišta podataka, u budućnosti bismo mogli proširiti tu ideju na korištenje i u drugim operativnim aplikacijama. Poess i ostali kažu [12]: "U posljednje vrijeme ETL je zamijenjen generaliziranim terminom integracija podataka (eng. *data integration* - DI)." Oni definiraju DI kao: "DI opisuje proces ekstrakcije i kombiniranja podatka iz različitih formata izvora, transformiranje podataka u unificirani model podataka za prezentaciju i učitavanje podataka u spremište podataka."[12] Dok se ETL koristi za izgradnju skladišta podatka, DI se može koristiti za izgradnju skladišta podataka i sinkronizaciju podataka između operativnih aplikacija. Također u posljednje vrijeme pojavile su se ideje o ETL procesu koji je vođen sadržajem [10]. Razlog tome je potreba za integracijom velikih količina podataka na webu i sama činjenica o tome da nisu svi podaci strukturirani, no to je trenutno izvan fokusa ovoga rada.

Ovaj rad strukturiran je kroz 4 poglavlja. Poglavlje 2 daje pregled područja ETL procesa. Poglavlje 3 opisuje generator opisnika ETL procesa, prikazuje predloženu arhitekturu sustava s opisom potrebnih uzoraka za implementaciju. Zaključak i ideja o sadržaju budućeg rada dani su u poglavlju 4.

2 Pregled područja ETL procesa

Razna istraživanja bave se optimizacijom, modeliranjem i automatizacijom ETL procesa. Vassiliadis je 2009. godine napravio sistematičan pregledni rad [17] iz područja ETL tehnologija gdje je opisao konceptualne i logičke probleme modeliranja, zatim probleme kod pojedinih faza ETL-a i naposljetku dao pregled nekih prototipa alata. Iz njegovog je rada vidljivo da se ETL proces može promatrati na tri razine:

- Konceptualna - najviša razina abstrakcije; na ovoj razini svi izvori podataka se tretiraju jednako i nisu klasificirani po vrsti izvora, a razlog tome je neutvrđenost vrste izvora, tj. nije poznato je li to datoteka, baza podataka ili nešto drugo. Transformacije su obično unija, presjek, razlika, filter, spajanje itd. Također, za skladišta podataka nije poznat tip skladišta (zvijezda ili pahuljica). Članci poput [18] pokušavaju optimizirati i pojednostaviti konceptualno modeliranje ETL procesa.
- Logička - na ovoj razini poznata je vrsta izvora, tip skladišta, no i dalje je nepoznat podatak o tome koristi li se relacijska baza za skladište podataka ili nešto drugo.
- Fizička - fizička implementacija samog procesa.

Većina znanstvenika fokusirana je na konceptualnu i logičku razinu, dok je fizička razina bila istraživana ranije (npr. Kimball[19]).

Kod modeliranja na konceptualnoj i logičkoj razini koriste se dva pristupa:

- Pristup baziran na unificiranom jeziku modeliranja (eng. *unified modelling language* - UML) [20, 21, 22]
- Semantički pristup [14, 15, 23]

Svaki pristup ima svoje prednosti i nedostatke. UML ima prednost u baziranju na standardnom jeziku koji se koristi za modeliranje i dobro je prihvaćen. Nedostatak UML pristupa su restrikcije koje se moraju pratiti i koje dolaze sa samim jezikom. Drugi pristupi imaju prednost u prikazivanju ETL procesa bez restrikcija koje se javljaju u prethodno definiranim jezicima, ali s druge strane nisu dobro prihvaćeni, što je njihov nedostatak.[15]

Naiqiao i ostali [24] predlažu "generator podataka za ETL proces koji prepoznaje semantiku, gdje se aktivnosti ETL procesa semantički transformiraju u simbole i postavljaju ograničenja nad njima te gdje se konkretni skupovi podataka izvode rješavanjem ograničenja." U ovome radu fokus je na semantičkom pristupu korištenjem ontologija jer takav pristup daje mogućnost rezoniranja nad podacima i proširuje dosadašnja znanja u sustavu. Obično se koristi OWL jezik (eng. *web ontology language* - OWL) za definiranje ontologija. Abello i ostali kažu [25]: "... korištenje OWL ontologije umjesto globalne sheme daje formalan model nad kojim se mogu primijeniti automatski mehanizmi rezoniranja."

Znanstvenici koji se fokusiraju na ETL proces baziran na ontologijama su npr. Skoutas [13, 23], Bansal [14], Chakraborty [26]. Ti radovi međutim ne obrađuju temu implementacije ETL procesa. Implementacija u tim radovima daje samo informaciju o tome što bi se trebalo napraviti, no nije objašnjeno kako se to koristi u samom ETL procesu. Bansal u članku [14] daje ideju o izvođenju SPARQL upita direktno nad podacima koji se obično rade u skladištu podataka. Ta metoda je međutim sporija od tradicionalnih ETL-ova u smislu izvršavanja.

Ovaj rad fokusira se na automatizaciju mapiranja između izvora i odredišta i definiranje transformacija koje treba primijeniti nad podacima korištenjem ontologija. Cilj je iskorištavanje brzina tradicionalnih ETL alata i optimizacija koja je napravljena u njima te iskorištavanje prednosti koje nosi semantičko modeliranje. Generator opisnika ETL procesa trebao bi transformirati semantičko znanje u tradicionalne ETL alate, a ne graditi novi ETL alat ili okvir kao što je to rađeno u prethodnim istraživanjima koja su proveli Nath ili McCusker [27, 28].

3 Generator opisnika ETL procesa

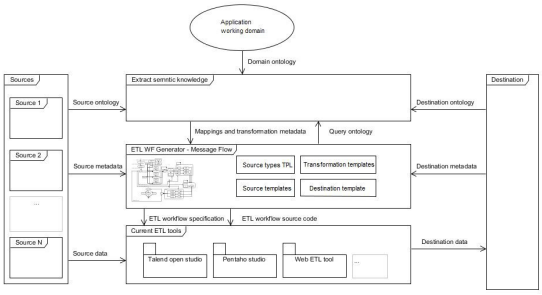
Kao što je već rečeno, postoje brojni Skoutasovi članci [13, 23] koji opisuju način modeliranja ETL proces iz ontologija. Ukratko rečeno, kreira se ontologija za svaki izvor, zatim ontologija domene i ontologija odredišta i tada se postavljaju upiti nad kreiranim ontologijama. Tako se dobivaju mapiranja između odredišta i izvora te transformacije koje se moraju obaviti. S druge strane postoje ETL alati koji su brzi i korisni (poput Talend Open Studio [29] ili Pentaho Data Integration [30]), no sam opisnik ETL procesa potrebno je kreirati ručno. Naš cilj je kreiranje generatora opisnika ETL procesa koji će primijeniti znanje dobiveno iz prvih sustava koji predlažu transformacije i mapiranja, a također i generirati potrebnu specifikaciju za tradicionalni alat.

U sljedećim sekcijama prikazana je: generalna arhitektura, detaljan tijek poruka u generatoru i opis korištenih uzoraka dizajna (eng. *design patterns*).

3.1 Predložena arhitektura

Na slici 1 prikazana je generalna arhitektura iz koje se može vidjeti da generator opisnika ETL procesa komunicira sa sljedećim dijelovima:

- Izvori (eng. *sources*) — predstavlja izvore podatka poput tablica, datoteka, weba i sl.;
- Odredište (eng. *destination*) — predstavlja skladište podataka u koje se podaci trebaju učitati, a ovdje mogu biti različiti tipovi modela (zvijezda ili pahuljica);



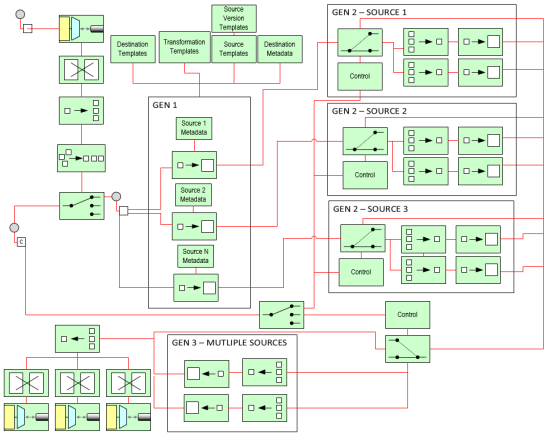
Slika 1: Generalna arhitektura generatora opisnika ETL procesa

- Ekstrakcija semantičkog znanja (eng. *extract semantic knowledge*) — predstavlja postojeći sustav koji određuje mapiranja i transformacije koje je potrebno provesti bazirajući se na ontologijama izvora, odredišta i domene;
- Trenutni ETL alati (eng. *current ETL tools*) — predstavlja postojeće alate poput Talend Open Studia, Pentahoa, Web ETL toola i sl. Svaki od tih alata ima svoju specifikaciju na temelju koje obavlja ETL proces;
- Generator opisnika ETL procesa (eng. *ETL workflow generator*) — generira specifikaciju za postojeće ETL alate bazirajući se na mapiranjima i transformacijama koje dobije. U sljedećoj sekciji dan je detaljan opis generatora. Kao što se može vidjeti iz slike 1, generator ima tri objekta s predlošcima koji sadrže potrebne informacije za: određene tipove izvora (baze podataka, datoteke itd.), izvore (MySQL, PostgreSQL, XML, TXT itd.), transformacije (odvajanje (eng. *split*), spajanje (eng. *merge*), dupliciranje (eng. *duplicate*), formatiranje datuma (eng. *date format*) itd.) i odredišta (zvijezda, pahuljica).

Interna arhitektura generatora opisnika ETL procesa (dalje u tekstu ponekad se naziva ETL generator) se bazira na porukama, a interna struktura je prikazana kao tijek poruka (slika 2). Tijek poruka kreće u trenutku kada generator dobije prvu poruku, a ona je specifikacija mapiranja i transformacija (M&T) za finalnu specifikaciju nekog od postojećih ETL alata. Slika koristi simbole za tijekove poruka koji predstavljaju standardne uzorke poruka.

Korišteni uzorci poruka su jedan tip integracijskih uzoraka (eng. *enterprise integration patterns*) i koriste se kako bi uspješno upravljali porukama. Ti uzorci i njihovi simboli preuzeti su iz knjige Enterprise Integration Patterns autora Hohpea i Woolfa [31] i prikazani su u tablici 1. Kako bi se kreirao dijagram, korišten je alat MS Visio u koji su uključeni predlošci simbola koji su preuzeti s web-stranice Apache Camel[16].

Prednost korištenja poruka je u tome što se inicijalne poruke za M&T mogu podijeliti u manje poruke. Te poruke tada predstavljaju jedinstvene transformacije koje treba napraviti nad jednim, dva ili više atributa i



Slika 2: Generator opisnika ETL procesa - tijek poruka

izvora. Svaka takva odvojena poruka može se procesirati paralelno. I dok se svaka poruka procesira, u nju se dodaje potreban opis koji se šalje dalje u druge dijelove sustava. S takvim sustavom umjesto da se generira specifikacija za postojeći ETL alat, moguće je provoditi sam ETL proces, no fokus ovog rada je generiranje specifikacije.

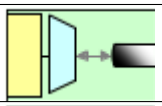
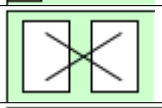
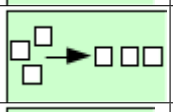
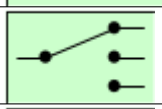
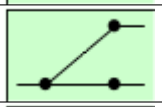
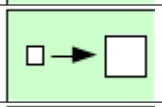
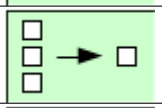
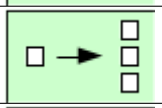
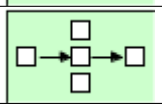
3.2 Opis tijeka poruka

Kako bi bolje razumjeli što se događa u samom sustavu, objasnit ćemo tijek poruka. Prvo, inicijalna poruka s M&T kreirana je na temelju ulaznih podataka. Ulazni podaci se dobivaju iz postojećih sustava koji prepoznaju potrebna mapiranja i transformacije. Inicijalna poruka se pretvara (pomoću Message transformera) u XML format koji će se koristiti za poruke unutar ETL generatora. XML je odabran jer se njime može jednostavno manipulirati (npr. kod dodavanja novog elementa ili dodavanja svojstva kao atributa). U istom trenutku kada je poslana inicijalna poruka šalje se jedna komandna poruka na prvi router kako bi se on konfigurirao.

Sustav se sastoji od tri modula: GEN1, GEN2 i GEN3. GEN1 modul je zadužen za dodavanje informacija u poruke koje sadrže informacije o M&T, a kojima je potreban samo jedan izvor i imaju transformacije koje se sastoje od jednog atributa (npr. transformacija pretvaranja datuma). Jedan Content Enricher postoji za svaki izvor koji dodaje podatke u poruke, a te podatke dobiva iz predloška.

GEN2 modul je zadužen za poruke koje trebaju jedan izvor i imaju transformacije koje se sastoje od dva ili više atributa (npr. transformacija spajanja). Jedan GEN2 modul sastoji se od više Content Enrichera za svaki izvor i dodaje podatke u ulazne poruke bazirajući se na nekom predlošku. Svaki Content Enricher zna što treba dodati da bi kompletno opisao izvor za koji je odgovoran. GEN3 modul radi na sličnom principu kao GEN2 modul, ali se razlikuje u tome što je zadužen za poruke koje trebaju više izvora.

Tablica 1: Korišteni integracijski uzorci

Ime uzorka	**Simbol** [31, 16]	**Kratak opis**
Chanel Adapter		Omogućuje povezivanje jednog sustava u kanal poruka.
Message translator		Prevodi jedan format podatka u drugi te ima istu svrhu kao uzorak Adapter iz knjige GOF [32].
Resequencer		Transformira tijek ulaznih poruka u ispravni redosljed.
Content Based Router		Usmjerava poruke na temelju njihovog sadržaja.
Detour		Neke poruke idu direktno u odredište dok neke druge idu u Content Based Router na procesiranje.
Content Enricher		Proširuje poruke sa informacijama koje nedostaju.
Aggregator		Spaja odvojene poruke koje trebaju ići zajedno u jednu poruku.
Splitter		Dijeli jednu poruku na više poruka.
Composed message processor		On je kombinacija Splitter-a i Router-a jer dijeli ulazne poruke u više poruka koje se trebaju različito procesirati, ali se u nekom trenutku trebaju opet spojiti u jednu poruku.

Kada je inicijalna poruka u ispravnom formatu, ona se dijeli u više poruka gdje svaka predstavlja jednu transformaciju s informacijom o tome koji atribut izvora se obrađuje i u koje odredište se treba učitati.

Jedna poruka može izgledati ovako:

```
<message id="1">
 <source>
  <user_source>Source_1<user_source>
  <name>employees</name>
  <attribute>date of employment
  </attribute>
 </source>
 <destination>
  <name> employees </name>
  <attribute>date</attribute>
 </destination>
 <transformation>
  <type>convert</type>
  <format>European date</format>
 </transformation>
</message>
```

Poruka ima tri dijela: a) izvor — on ima osnovne informacije o izvoru (naziv izvora (eng. *source name*), naziv tablice (eng. *table name*) i naziv atributa (eng. *attribute name*) iz kojeg se podaci trebaju ekstrahirati); b) odredište — ima osnovne informacije o odredištu (ime tablice (eng. *table name*) i atributa (eng. *attribute name*) u koje treba učitati podatke (kako se ETL generator može koristiti samo za rad s jednim skladištem, nema potrebe za informacijom o nazivu odredišta jer sve poruke imaju isto odredište); c) transformacije — osnovne informacije o transformacijama (npr. tip (eng. *type*) transformacije i format transformacije) koje je potrebno provesti.

Postoje dva tipa poruka u sustavu: normalne poruke i komandne poruke. Komande poruke se koriste za konfiguraciju sustava. Normalne poruke su poruke koje sadrže informacije o mapiranjima i transformacijama. Poruke se slažu određenim redoslijedom, s tim da se komandne poruke šalju prve. Poruke se preusmjeravaju ovisno o njihovom tipu. Normalne poruke idu prvo na router i onda se šalju na jedan od Content Enrichera unutar GEN1 modula (ovisno o tome kojem pripadaju). Svaka poruka se procesira i proširuje.

Na primjer, poruka prikazana ranije može se proširiti na sljedeći način (novi dijelovi su podebljani) :

```
<message id="1">
 <source>
  <name>employees</name>
```

```
  <attribute>date of employment
  </attribute>
  <user_source>Source_1
  <user_source>
  <type>MySQL</type>
  <ip>192.168.5.1</ip>
  <username>pero</username>
  <password>123456</password>
 </source>
 <destination>
  <name> employees </name>
  <attribute>date</attribute>
  <type>DIM</type>
  <ip>192.168.5.2</ip>
  <username>dw</username>
  <password>654321</password>
 </destination>
 <transformation>
  <type>convert</type>
  <format>dd.mm.yyyy.</format>
 </transformation>
</message>
```

Content enricher je dodao u izvor i odredište sljedeće elemente: tip (eng. *type*), IP, korisničko ime (eng. *username*) i lozinku (eng. *password*). Dodane informacije moraju se dati kako bi se krajnji opisnik mogao spojiti na određeni izvor ili odredište. Također, transformacije su se promijenile. Format transformacije se promijenio iz European Datea u specifični tip datuma na koji je potrebno prebaciti taj atribut.

Svaka poruka ne mora sadržavati transformacije, tj. može imati samo izvor i odredište. Predlošci za transformacije sadrže sve potrebne informacije za svaku transformaciju. Na primjer poruka koja traži transformaciju dijeljenja nad "Imenom Prezimenom" u "Ime" i "Prezime" generirala bi dvije nove poruke. Komandne poruke koje uključuju dva atributa iz istog ili više različitih izvora šalju se na GEN2 i GEN3 modul.

U GEN2 modulu komandne poruke govore da treba nešto napraviti nad dva različita atributa u istom izvoru, npr. kod spajanja imena i prezimena. U tom slučaju GEN2 modul instancira Aggregator i Content Enricher te dodaje nova pravila u Dynamic Router kako bi on znao da postoji novi kanal. Aggregator ima informaciju o tome koliko poruka treba dobiti (u slučaju imena i prezime mora dobiti dvije poruke) i kada ih dobije, one su kopirane u novu poruku koja se šalje na Content Enricher. Content Enricher u tom slučaju dodaje informaciju kao Content Enricher u GEN1 modulu. Originalna poruka koja je stigla u Aggregator prosljeđuje se dalje, ali zaobilazi Content Enricher. Ako je transformacija spajanje imena i prezimena, tada se dvije poruke nadopunjuju informacijama potrebnim za transformaciju spajanja. Isti postupak vrijedi i za GEN3 komandne poruke. Jedina razlika je u tome da je GEN3 modul korišten za poruke s više izvora. Normalne ulazne poruke koje nisu potrebne u GEN2 ili GEN3 modulu preskaču ih i šalju se direktno na sljedeći Dynamic Router.

Zanimljiva situacija dogodila bi se pri spremanju imena i prezimena u skladište samo zajedno, a ne odvojeno. U tom slučaju poruke za ime i prezime ne bi postojale i u GEN2 modulu ništa se ne bi generiralo kao izlaz. Da bi se to riješilo, za svaki takav slučaj na početku se generira poruka za ime i prezime koja nema informaciju o odredištu. Dakle kada poruke stignu u GEN2 ili GEN3 modul, one se kopiraju u jednu novu poruku i nakon toga se unište originali.

Nakon GEN3 modula dolazi aggregator koji spaja sve poruke u jednu specifikaciju. Na temelju komandnih poruka koje dobije na početku on zna koliko poruka treba primiti. Kada spoji sve poruke, finalna specifikacija se šalje prema routeru koji onda prosljeđuje specifikaciju na adapter koji ju pretvara u specifikaciju koju razumije postojeći ETL alat.

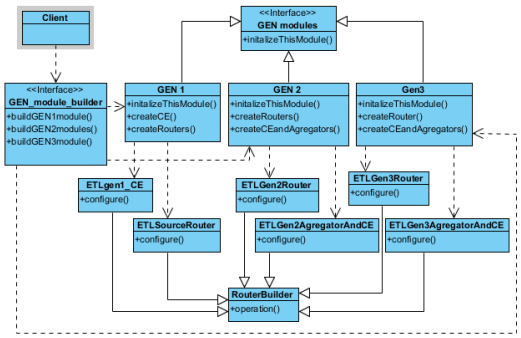
Glavna prednost ovog sustava je obrada poruka neovisno jedna o drugoj, što omogućuje paralelan rad. Također sustav se može podijeliti na više servera, npr. za svaki modul od GEN1 do GEN3 mogao bi biti poseban server, a elementi unutar modula također bi mogli biti na posebnim serverima. Ako bi sustav bio distribuiran na više servera, tada bi prednost bila još veća jer bi se mogao provesti ETL proces, a ne samo generiranje specifikacije.

3.3 Implementacija

Za implementaciju sustava koristio se programski jezik Java. Slika 3 prikazuje korištene uzorke i važne klase. Kako bi se generirala svojstva u GEN2 i GEN3 modulu, koristi se meta programiranje bazirano na predlošcima (eng. *template based meta programming*). Korišteni meta model je opisan u [33]. Predlošci su kreirani za: ETLGen2AgregatorAndCE, ETLGEN2Router, ETLGEN3Router, ETLSourceRouter itd. Svaki predložak sadrži ključne riječi koje se mogu konfigurirati tijekom izvršavanja poput "//ADD RULE" ili "//ADD FROM".

U sljedećem primjeru prikazan je izvorni kod ETLSourceRouter predloška.

```
package templates;
import org.apache.camel.builder.RouteBuilder;
public class ETLSourceRouter
            extends RouteBuilder{
@Override
public void configure() throws Exception {
from("file:target/gen1_router?noop=false").
  choice().
```



Slika 3: Dijagram klasa - generiranje GEN modula

```
  when(xpath("/message/command = 'yes2'")).
  to("file:target/cmd_router_gen2").
  otherwise().
  when(xpath("/message/command = 'yes3'")).
  to("file:target/cmd_router_gen3");
        //ADD RULE
}}
```

Ključna riječ '//ADD RULE' može se vidjeti na kraju prethodnog primjera. Na ovom jednostavnom primjeru objasnit će se princip na kojem je izgrađen cijeli sustav. Kada se primi komandna poruka koja u svom opisu ima novi izvor, dodaje se novi izvorni kod za taj izvor u ETLSourceRouter predložak. Preciznije, sam predložak se kopira i dodaje se novo pravilo na kraj predloška koji se kasnije izvršava.

Kada je nova kopija ETLSourceRoutera spremna, ta klasa se kompilira i uključuje (eng. *import*) u program te je u tom trenutku spremna za korištenje. Taj proces prikazan je u sljedećem primjeru koda:

```
static String compileTemplateClass(File f)
        throws CompilationException {

JavaCompiler jc =
  ToolProvider.getSystemJavaCompiler();
StandardJavaFileManager fileManager =
  jc.getStandardFileManager(null, null, null);
DiagnosticCollector<JavaFileObject> diag =
  new DiagnosticCollector<JavaFileObject>();

List<File> fileList = Arrays.asList(f);
Iterable<? extends JavaFileObject>
  compilationUnits1 = fileManager.
  getJavaFileObjectsFromFiles(fileList);
boolean ok = jc.getTask(null, fileManager,
  diag, null, null, compilationUnits1).call();

try { fileManager.close();}
catch (IOException ex) { ... }

printDiagnostics(diag, !ok);

//MOVE THE CLASS TO BUILD FOLDER
String f = f.getName().
  substring(0, f.getName().lastIndexOf("."));
String file_class_name = f_name + ".class";
String build_dir =
   System.getProperty("user.dir")
   + File.separator + "build"
   + File.separator + "classes";
String file_parent_dir =
  f.getParent().substring(f.getParent().
  indexOf("src") + 4);
build_dir += File.separator + file_parent_dir;

moveCompiledClass(f.getParent(),
   file_class_name, build_dir);

String class_name = file_parent_dir
   + File.separator + f_name;

return class_name.replace(File.separator, ".");
}
```

Problem implementacije prikazane u prethodnom primjeru je kreiranje kompiliranog koda u istom direktoriju gdje se nalazi izvorni kod te je stoga potrebno premjestiti kompilirani kod u direktorij za izvršavanje (eng. *build directory*). Da bi se uključila nova kompilirana klasa u program, koristi se Java refleksija, a u sljedećem primjeru dan je kod za taj dio.

```
Class c = Class.forName(fullClassName);
etlsourcerouter =
        (RoutesBuilder) c.newInstance();
...
//add etlsourcerouter to Apache Camel context
context.addRoutes(etlsourcerouter);
```

Na kraju, kako je sustav kreiran kao sustav poruka, svaki predložak je praktički klasa koja proširuje (eng. *extend*) RouteBuilder klasu iz Apache Camel biblioteke. Posljednji korak koji je potrebno napraviti nakon uključivanja klase jest dodati tu klasu u kontekst koji je trenutno aktivan, što je prikazano u posljednjoj liniji prethodnog primjera. Kada je to napravljeno, RouterBuilder može početi raditi svoj posao, a to je čitanje ulaznih poruka, njihovo procesiranje i generiranje izlaznih poruka. Sav izvorni kod dostupan je na https://github.com/matnovak-foi/ETL_WorkflowGenerator s GPLv3 licencom.

4 Zaključak i budući rad

U ovome radu prikazana je arhitektura koja integrira sustave koji generiraju mapiranja i transformacije pomoću ontologija i tradicionalnih ETL alata. Arhitektura prezentiranog sustava bazira se na porukama, što omogućuje procesiranje više različitih poruka u isto vrijeme.

Sustav koji je ovako izgrađen fleksibilan je i može se implementirati kao distribuirani sustav. Također je vrlo brz jer se većina poruka može procesirati paralelno. Prezentirani sustav generira specifikaciju koja se može koristiti u postojećim ETL alatima, što je ključna razlika u odnosu na postojeće semantičke ETL alate koji se implementiraju od početka. Isto tako, sustav se može jednostavno mijenjati tako da generira izvorni kod ili da odmah sam obrađuje ETL proces. Druga prednost ove arhitekture je mogućnost promjene tipa odredišta, a to je moguće zato jer je odredište opisano predloškom. Različita odredišta bi se mogla opisati drugim predlošcima te bi se mogli učitati podaci u bilo kakvo odredište. U budućnosti planiramo koristiti ovaj ETL sustav za druge svrhe osim skladišta.

Implementacija prototipa prikazanog ETL generatora napravljena je pomoću Apache Camel biblioteke jer ona podržava sve potrebne uzorke poruka koji su potrebni za implementaciju takvog ETL generatora.

Trenutni rad se fokusira na korištenje tehnika generativnog programiranja kako bi sustav bio fleksibilan. Trenutno se za svaki novi slučaj sustav treba ručno konfigurirati. Za svaku novu vrstu izvora potrebno je ručno kreirati novi Content Enricher u GEN1 modulu. U budućnosti bi bila vrlo korisna automatizacija procesa na temelju sadržaja poruka. Druga moguća nadogradnja ETL generatora je uključivanje Subscriber uzorka kako

bi alat mogao promijeniti postojeću specifikaciju na temelju promjena koje se jave u izvorima.

Literatura

[1] R. Kimball and J. Caserta, *The Data Warehouse-ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Wiley, 2004.

[2] M. Novak and K. Rabuzin, "Prototype of a Web ETL Tool," *International Journal of Advanced Computer Science and Applications*, vol. 5, no. 6, pp. 97–103, 2014.

[3] K. Sun and Y. Lan, "SETL: A scalable and high performance ETL system," in *System Science, Engineering Design and Manufacturing Informatization (ICSEM), 2012 3rd International Conference on*, vol. 1, (Software Engineering Institute, Beihang University, Beijing, China), pp. 6–9, 2012.

[4] K. Rabuzin and M. Novak, "WebETL Tool - A Prototype in Action," in *ICCGI 2014, The Ninth International Multi- . . .*, (Sevilja, Spain), pp. 67–71, 2014.

[5] A. Simitsis, P. Vassiliadis, and T. Sellis, "Optimizing ETL processes in data warehouses," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pp. 564–575, 2005.

[6] A. Behrend and T. Jörg, "Optimized incremental ETL jobs for maintaining data warehouses," in *14th International Database Engineering and Applications Symposium, IDEAS '10*, (University of Bonn, Germany), pp. 216–224, 2010.

[7] A. Simitsis, D. Skoutas, and M. Castellanos, "Representation of conceptual ETL designs in natural language using Semantic Web technology," *Data & Knowledge Engineering*, vol. 69, no. 1, pp. 96–115, 2010.

[8] D. Skoutas, A. Simitsis, and T. Sellis, *Ontology-Driven Conceptual Design of ETL Processes Using Graph Transformations*, vol. 5530 of *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[9] X. Zhang, W. Sun, W. Wang, Y. Feng, and B. Shi, "Research on generating incremental ETL processes automatically," *Jisuanji Yanjiu yu Fazhan/Computer Research and Development*, vol. 43, no. 6, pp. 1097–1103, 2006.

[10] A. Berro, I. Megdiche, and O. Teste, "A Content-Driven ETL Processes for Open Data," *18th East European Conference on Advances in Databases and Information Systems and Associated Satellite Events, ADBIS 2014*, vol. 312, pp. 29–40, 2015.

[11] N. Du, X. Ye, and J. Wang, "A schema aware ETL workflow generator," *Information Systems Frontiers*, vol. 16, no. 3, pp. 453–471, 2012.

[12] M. Poess, T. Rabl, H.-A. Jacobsen, and B. Caufield, "TPCDI: The first industry benchmark for data integration," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1367–1378, 2014.

[13] D. Skoutas and A. Simitsis, "Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data," *International Journal on Semantic Web and Information Systems*, vol. 3, no. 4, pp. 1–24, 2007.

[14] S. K. Bansal, "Towards a Semantic Extract-Transform-Load (ETL) Framework for Big Data Integration," in *2014 IEEE International Congress on Big Data*, pp. 522–529, IEEE, 2014.

[15] S. Bergamaschi, F. Guerra, M. Orsini, C. Sartori, and M. Vincini, "A semantic approach to ETL technologies," *Data & Knowledge Engineering*, vol. 70, no. 8, pp. 717–731, 2011.

[16] The Apache Software Foundation, "Apache Chamel: Enterprise Integration Patterns."

[17] P. Vassiliadis, "A Survey of Extract-Transform-Load Technology," *International Journal of Data Warehousing and Mining*, vol. 5, no. 3, pp. 1–27, 2009.

[18] S. Dupor and V. Jovanovic, "An approach to conceptual modelling of ETL processes," in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1485–1490, IEEE, 2014.

[19] R. Kimball, *The Data Warehouse Lifecycle Toolkit*. The Data Warehouse Lifecycle Toolkit, John Wiley & Sons, 2008.

[20] L. Muñoz, J.-N. Mazón, and J. Trujillo, "A family of experiments to validate measures for UML activity diagrams of ETL processes in data warehouses," *Information and Software Technology*, vol. 52, no. 11, pp. 1188–1203, 2010.

[21] J. Trujillo and S. Lujan-Mora, "A UML based approach for modeling ETL processes in data warehouses," in *Conceptual modeling - ER 2003 Proceedings*, vol. 2813 of *Lecture notes in computer science*, (Berlin, Germany), pp. 307–320, Springer-Verlag Berlin, 2003.

[22] S. Luján-Mora, P. Vassiliadis, and J. Trujillo, "Data mapping diagrams for data warehouse design with UML," in *Lecture Notes in Computer*

Science, vol. 3288 of *Lecture notes in computer science*, (University of Alicante, Spain), pp. 191–204, Springer-Verlag Berlin, 2004.

[23] D. Skoutas and A. Simitsis, "Designing ETL processes using semantic web technologies," in *Proceedings of the 9th ACM international workshop on Data warehousing and OLAP - DOLAP '06*, (New York, New York, USA), p. 67, ACM Press, 2006.

[24] N. Du, X. Ye, and J. Wang, "A semantic-aware data generator for ETL workflows," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 4, pp. 1016–1040, 2016.

[25] A. Abello, O. Romero, T. Pedersen, R. Berlanga Llavori, V. Nebot, M. Aramburu, and A. Simitsis, "Using Semantic Web Technologies for Exploratory OLAP: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. PP, no. 99, pp. 1–1, 2014.

[26] J. Chakraborty, A. Padki, and S. K. Bansal, "Semantic etl - state-of-the-art and open research challenges," in *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*, pp. 413–418, 2017.

[27] R. P. Deb Nath, K. Hose, and T. B. Pedersen, "Towards a programmable semantic extract-transform-load framework for semantic data warehouses," in *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP*, DOLAP '15, (New York, NY, USA), pp. 15–24, ACM, 2015.

[28] J. P. McCusker, K. Chastain, S. Rashid, S. Norris, and D. L. McGuinness, "Setlr: the semantic extract, transform, and load-r," *PeerJ Preprints*, vol. 6, p. e26476v1, 2018.

[29] Talend, "Talend Open Studio Integration Software Platform."

[30] Pentaho Corporation, "Data Integration - Pentaho Business Analytics Platform."

[31] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2012.

[32] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, Limited, 2005.

[33] M. Novak, I. Magdalenić, and D. Radošević, "Common Metamodel of Component Diagram and Feature Diagram in Generative Programming," *Journal of Computer Science*, vol. 12, no. 10, pp. 517–526, 2016.