ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 239-243
doi: 10.18178/wcse.2017.06.041

A Domain Specific Language for Scripting ETL Process

Sunisa Junsawang and Yachai Limpiyakorn [+]

Department of Computer Engineering, Chulalongkorn University, Bangkok 10330, Thailand

Abstract. Extract-Transform-Load, ETL, is considered as general preprocessing for data preparation of heterogeneous sources such as data warehouse. The model-based ETL is user-friendly but hardly scales up with large complex systems. This research thus presents an approach of code-based ETL of which the commands are stated in a domain specific language, rather than programming or query languages. The DSL has developed to support domain experts for scripting ETL processes without the requirement of programming skill. This would result in less defects and resource consumption for the organization.

Keywords: domain specific language, ETL, relational algebra, query, data warehouse.

1. Introduction

Extract-Transform-Load (ETL) is the process of gathering data from different sources, then transforming the data by rules, and loading to the destination repository such as data warehouse. There are several ETL tools in the market. In general, ETL tools can be categorized into two types: model-based, and code-based. The model-based is user-friendly providing graphical user interface commands, while the code-based requires the ETL commands written in computer languages. Despite of the easy of use of the model-based ETL tools, they hardly scale up with the large complex systems.

In literature, Oliveira and Belo [1] proposed the use of Business Process Modelling Notation (BPMN) for building the simulator of ETL process. The DSL grammars were defined and translated to Java language by Xtext tool. Deneke [2] developed the DSL by C# for building ETL workflows. The Object-Relational Mapper (ORM) was used for data gathering instead of data query with SQL. Lasya and Tanuku [3] proposed an approach to optimizing the executable code generated from a certain SQL query. Several alternative query trees are constructed based on the relational algebra expressions underlying the input SQL query. The Query Execution Plan associated with the query tree that provides the minimum cost will be selected for executable code generation. Santos and Belo [4] proposed modelling ETL conciliation tasks using relational algebra operators. Several scenarios of ETL were illustrated in relational algebra trees for integrity awareness.

This paper presents an approach of code-based ETL commands written in a domain specific language. The DSL commands contain the vocabulary or terminology used in the domain that the experts are familiar with. The syntax of parts of data query commands is developed based on relational algebra. The proposed method would facilitate the domain experts for scripting ETL processes without the requirement of programming skill. This would result in less defects and resource consumption for the organization.

2. Domain Specific Language– DSL

A domain specific language [5] is developed for use in a particular problem domain. The language is designed for ease of use and understanding, as the commands contain the common vocabularies familiar to the domain experts. In general, DSLs come in two main forms: internal and external. Internal or Embedded

[+] Corresponding author. Tel.: + 668 2218 6959; fax: +668 2218 6955.
E-mail address: *Yachai.L@chula.ac.th*.

DSLs are a particular form of API in a host general purpose language. An external DSL is a language that is parsed independently of the host general purpose language.

3. Extract-Transform-Load (ETL)

ETL [6] is the process of gathering data from different sources for loading to data warehouse. It consists of three steps: 1) Extract– identify and collect data from the different sources that contain required data to be stored in the destination repository, 2) Transform– change data such as calculating dimension and measures, cleaning data, filtering data, splitting a column, joining and transposing rows etc., and 3) Load– transfer data to data mart or data warehouse.

4. Relational Algebra and SQL

SQL is a well-known and extensively used query language. The design of SQL is based on relational algebra [7] that explains the theory of set. Some of the SQL clauses/ commands defined on the underlying relational algebra are shown in Table 1.

Table 1: Example SQL and associated Relational Algebra [7].

Relational Algebra	SQL
π a,b	SELECT a,b
σ (d > e) ^ (f = g)	WHERE d > e AND f = g
p x q	FROM p,q
p ∪ q	SELECT * FROM p UNION SELECT * FROM q
p − q	SELECT * FROM p EXCEPT SELECT * FROM q
p ∩ q	SELECT * FROM p INTERSECT SELECT * FROM q

5. Research Methodology

This paper presents an approach to generating the ETL script in Java from the input DSL script. The system is implemented with the Model-View-Controller (MVC) architecture design as illustrated in Fig. 1. The View component, or interface layer includes DSL grammar and output log which are input and output. The Controller component consists of ETL controller, File controller, Transformation Operation controller, and Database Operation controller. The Model component, including DSL parse and Java model, is responsible for the translation of DSL grammar and the coordination between the controller and view components.

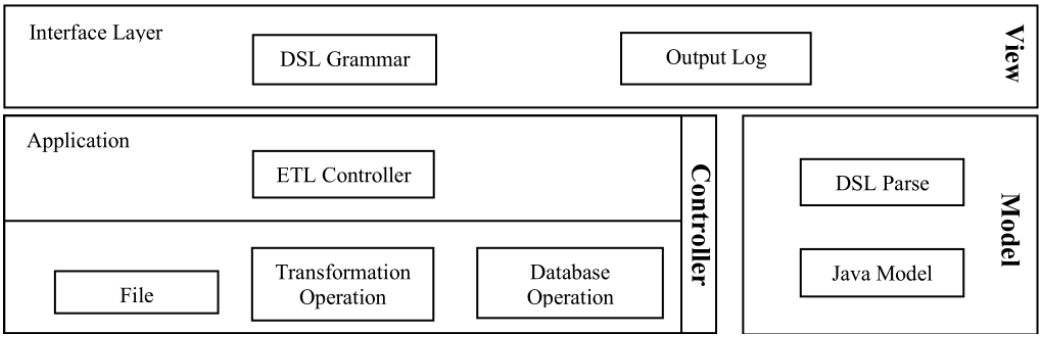

Fig. 1: System design with MVC architecture.

5.1. DSL Design

The developed DSL is expressed in extended Backus–Naur form (EBNF) as shown in Fig. 2. The identifiers come from the vocabularies in the ETL domain. The syntax is defined based on relational algebra [3, 4] for the part of data query. Xtext, an open-source framework for developing programming languages and domain-specific languages, is used as the parser generator. The procedure of translation of the input DSL script to Java code is illustrated in Fig. 3.

```
grammar th.ac.chula.eng.cp.dsl.Etl with
org.eclipse.xtext.common.Terminals
generate etl "DSL_ETL"
ETLMain:
        "jobname:" jobname=ID
        "author:" author=AuthorName
        "version:" version=Version
        "description:" descript=Description
        (element+=FullStepElement)*;
FullStepElement:
        name = (Delete | Load  | LoadFile | Update);
Update:
        "step:" step=ID
        "type:" type=StepType
        "source:" source=(SouceTable|"fix")
        "set:" set=UpdateColumn
        "target:" target=ATableName
        "condition:" condition=Condition
        "end";
LoadFile:
        "step:" step=ID
        "type:" type=StepType
        "source:" source=SourceFile
        "format:" format=FileFormat
        "target:" target=ATableName
        ('('loadType=LoadType')')?
        "end";
SourceFile:
        ID ('/' ID)* ('.'ID);
Load:
        "step:" step=ID
        "type:" type=StepType
        "source:" source=SouceTable
        "condition:" condition=Condition
        "target:" target=ATableName
        ('('loadType=LoadType')')?
        "end";
Delete:
        "step:" step=ID
        "type:" type=StepType
        "target:" target=ATableName
        "condition:" condition=Condition
        "end";
UpdateColumn:
        (ID ('.'ID)? "=" (Value | ID ('.'ID)?)) (',' ID ('.'ID)?
        "=" (Value | ID ('.'ID)?))*;
LoadType:
        "truncate" | "append" | "update&insert";
SouceTable:
        (("column")? ColumnName "from")? TableName;
TableName:
        ATableName ("(+)")?
        ((TableJoin ATableName) ("(+)")?)?;
TableJoin:
        "join" |","|"except"|"union";
Condition:
        (ColumnName Operation (ColumnName | Value))
        ( ("and" | "or")
        (ColumnName Operation (ColumnName | Value)))* |
        "none";
Operation:
        "=" | "!=" | ">" | ">=" | "<" | "<=" | "in" | "not" | "like";
ColumnName:
        ((ID ('.'ID)?) | (ID"("(ID ('.'ID)?)|Value")"))
        (',' ((ID ('.'ID)?)  )| (ID"("(ID ('.'ID)?)")") )?  ;
Value:
        STRING | INT | "NULL" | "null" | '*';
ATableName:
        ID ('.'ID)?;
StepType:
        "load" | "delete" | "loadfile" | "update";
AuthorName:
        ID ID;
Version:
        INT ('.' INT)*;
Description:
        ID (ID)*;
FileFormat:
        "header="("no"|"yes") (',')
        ("footer="("no"|"yes"))(',')
        ("delimiter="("pipe"|"comma"))
        (',') ("quote="("no"|"single"|"double"));
```

Fig. 2: Excerpt of DSL syntax.

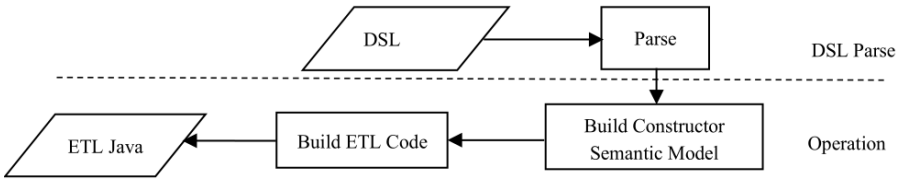

Fig. 3: Translation of DSL script to ETL Java code.

Fig. 4 illustrates the ETL process starts with reading the DSL script. The DSL commands will then be parsed and transformed to Java code via the semantic model as illustrated in Fig. 3. The ETL script in Java will call back end of ETL that consists of the commands for ETL process. The transferred data are reported in the output log.

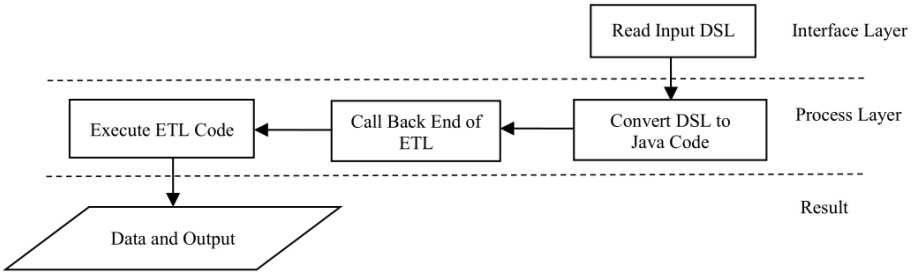

Fig. 4: ETL process with DSL commands.

5.2. Demonstration

Example DSL script is shown in Fig. 5 and the associated semantic model is illustrated with the object model in Fig. 6. The example DSL script is parsed and transformed to the ETL Java code as shown in Fig. 7. The execution of Java code will generate the output log reporting the data transferred to the target repository as shown in Fig. 8.

```
jobname:loadFileandDataDraft
author: Sunisa Junsawang
version:1.0
description:loadFileandDataDraft
step: FileLoadtoTableA
type: loadfile
source:  input_file/test.file
format: header=yes , footer=no , delimiter=pipe , quote=no
target:  tableA (append)
end
step: deletetoTableC
type: delete
target: tableC
condition: att3 > '2000'
end
step: UpdatetoTableA
type: update
source: column cola,colb from tableA(+) join tableB  //optional
set: cola = colb , colc='xx'
target:  tableA
condition: none
end
step: UpdateandInserttoTableCCase1
type: load
source: tableA(+) join tableB
condition: att3 > '2000'
target: tableC  (update&insert)
end
step: UpdateandInserttoTableCCase2
type: load
source: column att1,att2 from tableA(+) join tableB
condition: att3 > '2000'
target: tableC  (update&insert)
end
```

Fig. 5: Example DSL Script.

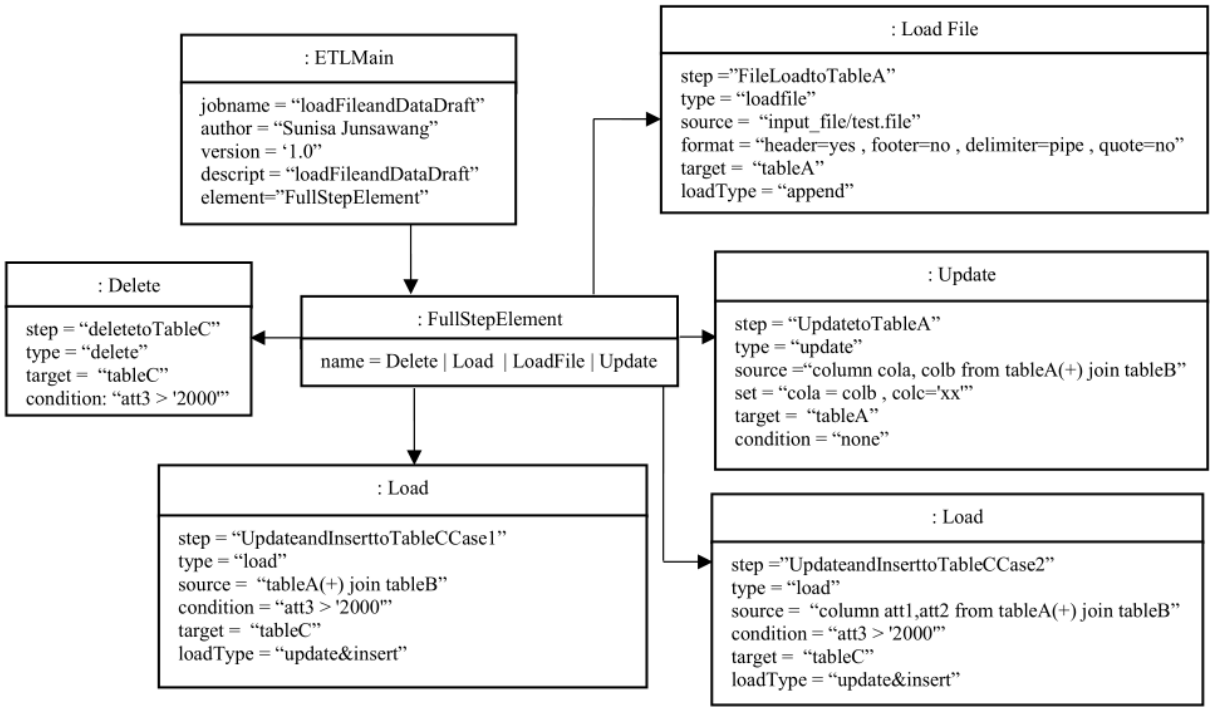

Fig. 6: Semantic Model of Example DSL Script.

```
import th.ac.chula.eng.cp.dslforetl.java.controller.ETLTools;
public class loadFileandDataDraft {
    public static void main(String[] args) {
        ETLTools etl = new ETLTools();
        System.out.println("Job name:loadFileandDataDraft");
        System.out.println("Develop by:Sunisa Junsawang");
        System.out.println("Version:1.0");
        System.out.println("Description:loadFileandDataDraft");
        System.out.println("==========================================");
        System.out.println("Execute loadfile step :FileLoadtoTableA===========>");
        System.out.println("Result " + etl.loadFile("input_file/test.file", "tableA","header=yes , footer=no , delimiter=pipe , quote=no","append"));
        System.out.println("Execute delete step :deletetoTableC===========>");
        System.out.println("Result " + etl.deleteData("tableC","att3 > '2000'"));
        System.out.println("Execute update step :UpdatetoTableA===========>");
        System.out.println("Result " + etl.updateData("column cola,colb from tableA(+) join tableB","cola = colb , colc='xx'","tableA","none"));
        System.out.println("Execute load step :UpdateandInserttoTableCCase1===========>");
        System.out.println("Result " + etl.loadData("tableC", "tableA(+) join tableB","att3 > '2000'","update&insert"));
        System.out.println("Execute load step :UpdateandInserttoTableCCase2===========>");
        System.out.println("Result " + etl.loadData("tableC", "column att1,att2 from tableA(+) join tableB","att3 > '2000'","update&insert"));
        System.out.println("==========================================");
    }
}
```

Fig. 7: ETL Java code generated from example DSL script.

```
Develop by:Sunisa Junsawang
Version:1.0
Description:loadFileandDataDraft
==========================================
Execute loadfile step :FileLoadtoTableA===========>
Result Completed: Load append from test.file to tableA
Execute delete step :deletetoTableC===========>
Result Completed: Delete data from tableC
Execute update step :UpdatetoTableA===========>
Result Completed: updated data on tableA
Execute load step :UpdateandInserttoTableCCase1===========>
Result Completed update&insert into tableC
Execute load step :UpdateandInserttoTableCCase2===========>
Result Completed update&insert into tableC
==========================================
```

Fig. 8: Output log of ETL process.

6. Conclusion

In several organizations, the specification for ETL process is written with natural language by the person who does not have programming skill, and then pass to the programmer to develop the ETL scripts either with the model-based or code-based ETL tools. This work presents an approach of execution the ETL process with the input DSL script containing ETL commands that will be parsed and transformed to Java code for execution. As a result, this would lead to the improvement of the ETL process, as well as reduce the cost and the labor spent on the process.

7. References

[1] B. Oliveira and O. Belo, "On the specification of extract, transform, and load patterns behavior: A domain - specific language approach," 2016.

[2] W. Deneke, "A Domain Specific Model for Generating ETL Workflows from Business Intents," Doctor of Philosophy in Computer Science, University of Arkansas, Fayetteville, University of Arkansas, Fayetteville ScholarWorks@UARK, 2012.

[3] S. Lasya and S. Tanuku, "A Study of Library Databases by Translating Those SQL Queries Into Relational Algebra and Generating Query Trees," 2011.

[4] V. Santos and O. Belo, "Modelling ETL Conciliation Tasks Using Relational Algebra Operators," presented at the 2014 UKSim-AMSS 8th European Modelling Symposium, 2014.

[5] D. Ghosh, *DSLs in Action*: Manning Publications, 2011.

[6] N. Mali and S. Bojewar, "A Survey of ETL Tools," presented at the International Journal of Computer Techniques, 2015.

[7] J. C. Franchitti. *Relational Algebra, Relational Calculus, and SQL*.