

Master thesis : Creation of a domain specific language for an Extract-Transform-Load system

Auteur : Wauthoz, Julien

Promoteur(s) : Lambert, Marie; Fontaine, Pascal

Faculté : Faculté des Sciences appliquées

Diplôme : Master : ingénieur civil en informatique, à finalité spécialisée en "management"

Année académique : 2021-2022

URI/URL : <http://hdl.handle.net/2268.2/14583>

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.

University of Liège - Applied Science Faculty
Academic year 2021-2022



Creation of a Domain Specific Language for an Extract Transform Load system

Master thesis submitted for the degree of MSc in Computer Science and
Engineering by Julien Wauthoz

Supervisor
Pascal Fontaine

Company
FundProcess

Abstract

Creation of a Domain Specific Language for an Extract - Transform - Load system

Julien WAUTHOZ

Supervisor: Pascal FONTAINE

Master thesis submitted for the degree of MSc in Computer Science and Engineering
Academic year 2021-2022, University of Liège

Organisations use different systems with different structures to run their day to day activities. The number and variety of these systems is increasing and this led to some concerns such as incompatibilities. In response to that, a technology called Extract-Transform-Load (ETL) is used. It allows organisations to interconnect and manage their different systems.

The easiest way to build ETLs is to use ETL tools. ETL tools are mainly used by developers and IT staff because they require some programming knowledge. This work aims at creating a Domain Specific Language (DSL) that makes the creation of ETLs easier. By doing so, the ETL technology could be used by a larger public having little to no programming experience.

The DSL will use ETL.NET framework which allows to build ETLs into any .NET application. A transpiler will translate code from the DSL that we created into ETL.NET code.

This thesis will first give a better overview about ETL and ETL tools. Then, It will detail how the DSL was created, as well as its implementation. Then, it will confront the DSL to the literature to identify the completeness of the language and give some guidelines for further development.

Acknowledgements

I would like to thank Frédéric Duquenne, Stéphane Royer and Moyra Bonjean from FundProcess for giving me the opportunity to work on this project. I would like to express them my deepest gratitude for their time, guidance and advise during this year. I would also thank them for the freedom they gave me on choice of the subject and on its realisation.

I would also like to thank Pascal Fontaine, my promoter, for its guidance and for being available when I needed some recommendations.

Contents

1	Extract-Transform-Load (ETL)	1
1.1	Introduction	1
1.2	FundProcess	2
1.3	Data Warehousing, Business Intelligence and Dimensional Modeling Primer	3
1.4	ETL	5
1.4.1	What is an ETL ?	5
1.4.2	ETL today	5
1.4.3	Challenges	6
1.4.4	ETL tools type	6
1.4.5	ETL tool comparison	7
2	Domain Specific Language (DSL)	10
2.1	DSL Creation	10
2.1.1	Why creating a DSL ?	10
2.1.2	Creating the language	10
2.1.2.1	COBOL Inspiration	11
2.1.2.2	SQL inspiration	12
2.2	DSL Implementation	15
2.2.1	Structure of the transpiler	15
2.2.1.1	Lexer	15
2.2.1.2	Parser	16
2.2.1.3	Tree Builder	17
2.2.1.4	ETL.NET Generation	19
2.3	Possible operations with EtlScript	20
2.3.1	Stream	20
2.3.2	Source	20
2.3.3	Operation	21
2.3.4	Destination	23
2.3.5	More operators?	23
2.4	Integration in FundProcess System	25
2.4.1	Different Modes	25
2.4.2	Unit Testing	27
3	Categorising features	30
3.1	ETL subsystems	30
3.1.1	Extraction subsystems	30
3.1.2	Cleaning and conforming subsystems	31
3.1.3	Delivering subsystems	32

3.1.4	Managing subsystems	33
3.1.5	Summary	35
4	Macro	37
4.1	Creating Macro With Data imported with the DSL	37
5	Conclusion	39
Appendix		41
A.1	LL(1) Grammar	41
A.2	ANTLR Grammar	41
A.3	Subsystems not considered	46
A.4	Scratch	47
A.5	EtlScript Example	47
A.5.1	Data Processor	47
A.5.2	File Retriever	50
A.5.3	File Processor	51

Chapter 1

Extract-Transform-Load (ETL)

1.1 Introduction

Since computers exist, the variety of IT systems and hardware, as well as the computing power, have grown exponentially. Organisations around the world have directly benefited from these cutting edge technologies and put together the most efficient systems to stay ahead of their competition but this has also created complex problems to solve. These systems are not always compatible with each others and data needed to operate them might be stored in different locations. Interconnecting, operating and managing such systems undoubtedly have a significant impact on costs because they require highly skilled staff. The need to integrate data into organisation's business strategy and design choices became crucial.

To answer these problems, a new concept emerged in the 1970's : Extract Transform Load (ETL) systems. They are in charge of extracting data from its original source, transform it (clean it, combine it and put it in the right format) and load it into its target destination. Thanks to them, organisations can combine systems from different vendors with different data structures. As explained in a further section, it is one of the core concept of data warehousing and business intelligence.

At the beginning, ETLs were designed by organisations themselves and it was not an easy task. Later, some ETL tools became available making the task of building ETLs easier. The first ETL tools were designed for large companies and were expensive (the main one was Prism and was originally priced at \$25.000 [4]). Then some well know companies (Microsoft, SAP, Oracle) became interested in the technology and developed their own version. As this technology evolved with time, different types emerged on the market. Around the year 2000, midsize ETL tools were introduced and were much more affordable. And this leads us to the current situation where some of them are free or even open source.

Even if this technology has been around for a long time, it is still mainly used by developers and IT staff to import large volumes of data or to connect systems together. In this work, our goal is to create a Domain Specific Language (DSL) to make the creation of ETLs easier. We want people with little to no programming experience to be able to use ETL tools. Thanks to this DSL, we will drastically reduce the learning curve and provide an access to the technology to a larger public. This will allow to use the technology at a larger scale and make it more attractive.

1.2 FundProcess



This project is realised in collaboration with FundProcess [7]. FundProcess is a fintech company based in Luxembourg who creates software as a service solutions and integrated programming tools for assets management. Their solution is principally meant for investment funds, in particular SICAV (société d'investissement à capital variable). An investment fund is a company where capital is raised from a number of investors and is invested in accordance to a defined investment policy. Those funds use a Portfolio Management Service (PMS) to manage all the different portfolios of their clients. There are 3 important concepts for a fund:

- custodian bank: provides securities services, it safeguards assets.
- transfer agent: maintains an investor's financial records and tracks each investor's account balance. It ensures that investors receive their due interest and dividend payments.
- central administration: computes funds value and performance (NAV).

FundProcess provides an ETL technology that connects to the three members, it allows to load and manage data on their software. With this, customers can transfer their data from their source system to a FundProcess database. Then, FundProcess offers a macro technology which allows to design investment, risk, compliance and private equity functionalities. It is a dedicated programming environment where it is possible to display dashboards, calculations, fees, ... Finally, they provide a report designer where it is possible to design custom marketing, legal and internal reports.

Their solution is principally meant for people working in the financial domains like fund and wealth manager, private bankers or advisory boutiques. Their typical clients have little or no programming experience. Thus, their tools are designed for simplicity of use, e.g., the macro designer has a syntax dedicated to business, making it easier for their client to design macros by themselves. It is also in this context that this project will take place, where the focus will mainly be on the ETL technology.

1.3 Data Warehousing, Business Intelligence and Dimensional Modeling Primer

We hear everywhere that one of the most important assets of any company is information, that data is the new oil. The two main purposes of data and information are operational record keeping and analytical decision making. Those will be split in two categories: operational system and Data Warehouse/Business Intelligence (DW/BI) systems [1].

Operational system are used to run the company on a day to day basis. They allow to perform actions like signing new customers or keeping track of the sells of the day. They are optimised to process transactions as fast as possible, most of the time one at a time. Data produced are raw data, so they are primitive, highly detailed and probably not in the right format. They reflect the current state of the company.

On the other hand, Data Warehouse/Business Intelligence (DW/BI) systems reflect a more global state of the company and are used to analyse the performance (compare this week sells with last week sells), make some strategic decisions, perform machine learning and AI, ... The DW/BI systems contain historical information, they provide summarised and standardised information. These systems almost never deal with one transaction at a time and need lots of them to be meaningful. They need to offer a query system to retrieve data and be optimised for it. So, Data Warehousing are really powerful business intelligence tools and they fit really well in the four steps growth strategy that most companies are using today: planning, data gathering, data analysis, business action.

These systems require completely different needs and thus different implementations. By failing to do so, some problems will more likely be encountered such as: no access to collected data, no meaningful data, impossible to slice data in the right way, ...

To avoid these problems, the DW/BI system must comply with the following requirements:

- make information easily and quickly accessible
- must present consistent data
- must adapt to change
- must present information in a timely way
- must protect information from outside access
- information that comes from it must be trusted by the business community

As we start to understand, Data Warehousing is not just about IT but it also has a really important business component as it may have a significant impact on decisions taken by the strategic instance of a company.

A few DW/BI architectures exists, but there is one that stands out and which is widely used today: the DW/BI Kimball architecture. In this work, the focus will be on this one. It consists of four components with distinct roles to consider.

1) The **Operational Source Systems** (Source transaction on Figure 1.1) that was already detailed before and on which you have little or no control over the content and the format of the data.

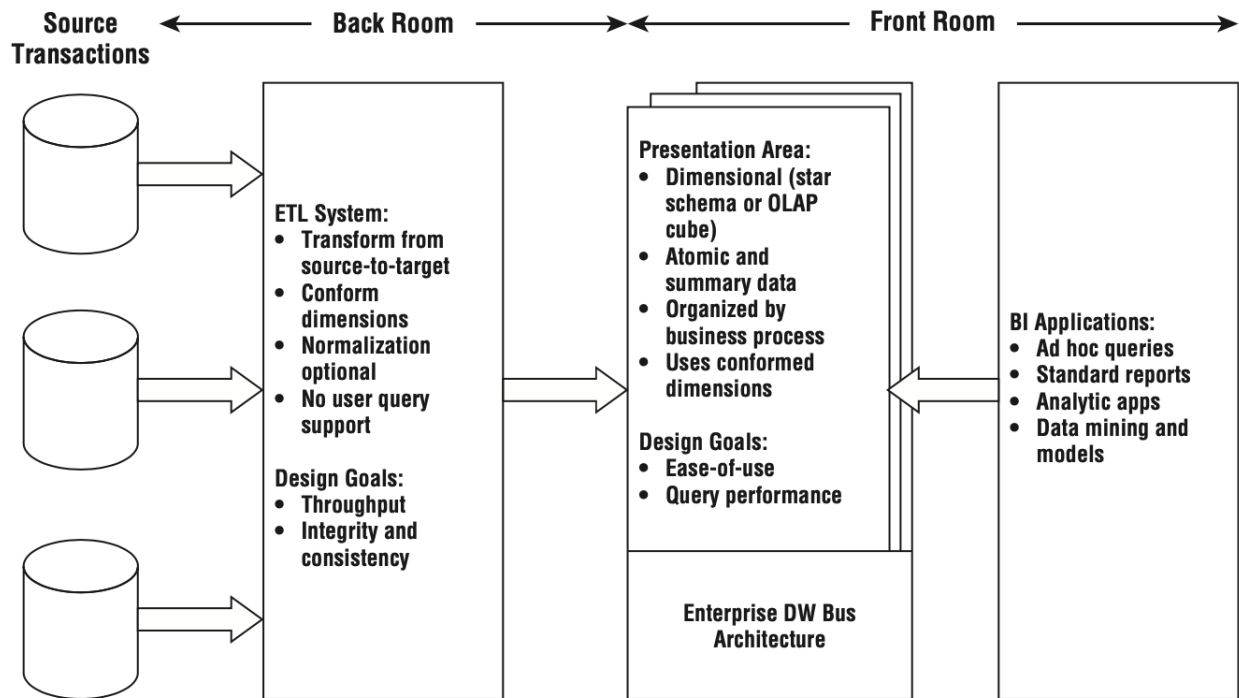


Figure 1.1 – The four components Of Kimball Architecture

2) The **Extract - Transform - Load system** consists of a work area, instantiated data structures, and a set of processes.

3) The **Presentation Area** to support business Intelligence. It is where data are organized, stored and made available for BI applications.

4) The **Business Intelligence Applications**, it is all the capabilities provided to business users for analytic decision making and it should contribute to key business decisions and growth of any company. They use data by performing query in the presentation area.

1.4 ETL

1.4.1 What is an ETL ?

The ETL process will be dissected in order to understand its different phases [1]. ETL stands for Extract - Transform - Load:

- **Extract:** getting data from its original source (file, database, CRM,...) which means reading and understanding the source data with minimal impact on the source system and then copying it in the ETL system for the next step.
- **Transform:** This step has multiple roles:
 - 1) cleaning the data, meaning that it will perform actions like:
 - handling missing and null data
 - removing duplicates
 - removing outliers
 - recoding same data to a common denominator, e.g. 'M' for 'male'
 - converting data types into standardized formats (for instance, for dates)
 - 2) data enriching which consists of actions like:
 - adding new information to the raw data already collected like joining different sources
 - calculating fields with already existing fields
 - 3) combine data
 - 4) put data in the right format for the last stepThe transform step adds value to the data by improving its quality.

- **Load:** load data into the target destination. Physical structuring and loading of data into the presentation area of the target dimensional models. After this step, data should be complete and usable for Business Intelligence tools.

An ETL will combine structured and unstructured data retrieved from source systems and land them in a data warehouse. It allows to have data in a structure optimised for reporting. It improves the quality, standardization and clarity of data. It will allow to work with clear information and disambiguate unclear raw data.

1.4.2 ETL today

In a perfect world, every company would store all the data needed for running their business in the same file system and under the same structure. When they need data for reporting or any other use, it would immediately be available and ready for use.

But this is never the case, companies use different systems with different structures to run their business. They probably have different systems for CRM, accounting, logistics, ... but if a customer decides to change his address, it will probably be modified in the CRM but will it be automatically adapted in their other systems ? Most likely not and this may cause some inconsistencies between them. So this is where an ETL comes handy and solves many present and future problems. Building a good ETL from day one is important because with a good design that needs fewer maintenance, analysts are less dependant of data engineers to have access to the data they need.

Another very common use case of ETLs is during a Merge or Acquisition (M&A) processes. The two companies have a lot of challenges to overcome during this period and one of them will be to decide what systems and tools they keep, or do they buy new ones ? They have to figure out a way to merge data from the two companies into one system everyone will use. Depending of the incompatibility of data, it may be problematic. But in any case, they will need an ETL to create a sustainable new structure to be able to work efficiently.

1.4.3 Challenges

ETLs can bring value by giving the possibility to transfer data from one system to another even with very different structures but it also comes with its own set of challenges. The two most important ones are described below.

The first one is data volume. The need for an ETL is even bigger when large volumes of data must be considered. The time needed to process a fixed amount of data with an ETL highly depends on how the ETL was implemented and on the number of operations that are performed. To give an order of idea, for a cloud ETL and a few sorting operations, processing 100 GB can be done in a few hours [5]. Some companies have to deal with huge amount of data, e.g., Facebook ingests 500+ terabytes of data almost every day in an unstructured format [6]. A powerful ETL system is a must have for those companies in order to take maximum advantage of data. But with great power comes big responsibilities: it implies that there is less room for mistakes, for example if an SQL query is badly designed, it might not have an impact with small datasets but with bigger ones there might be hours of difference in the response time between a good and a bad query. Another consideration to take into account to achieve good performances is the need to implement parallel extraction solutions. Dealing with a high level of parallelism induce more complexity while building and maintaining the ETL.

The second challenge is the diversity of sources which also increase the complexity of the ETL and possibilities for errors. Different sources may have different owners and different ways to access them. While dealing with many different types of databases, all possible errors that can appear from each one of them must be considered and they might be very different from one another.

1.4.4 ETL tools type

In order to build an ETL, an ETL tool should be used. Many ETL tools are available on the market and choosing which one to use largely depends on where the data is stored and business requirements. In this section, the main types of ETL tools are presented [10].

The first aspect on which ETL tools can vary is data refreshment time which can be categorise in three types:

1. **BatchProcessing**: processing large volumes of data takes time and ressources on a computer. The solution given by this type of ETL is to perform data processing in batches during off-business hours. The main advantage is that computing power can be used at time when most of the system is not used. The bad part is that accessing the most recent data is not guaranteed. Companies only have access to data from the day before which can cause serious problems, e.g., in the financial domain, if big transactions are missed and only seen 24h later, it is a big problem for fraud detection.

2. **Real-Time:** for some specific applications where batch processing is not enough like in the financial domain example explained above. It is also required in some cases like internet of things where there are too much data and batch processes wouldn't be able to handle such quantities. A real-time ETL will process the information as soon as it's available. The structure is more complicated than the batch processing structure since it makes use of a stream processing platform.
3. **Near-Real-Time:** It's a solution that falls between the previous two. Data processing is performed every few minutes, typically between 5 and 10 minutes. It's a good alternative for users that are not satisfied with data that are one day old but that can't handle the complex structure of a real-time ETL.

ETLs tools can also vary depending if they are available in the cloud or not:

1. **Cloud-Native:** data is moving into the cloud and so are ETLs, with big data this is a critical requirement since data can't be extracted and loaded from the cloud every time. So, an ETL that can perform those actions directly in the cloud is needed.
2. **On-Premise:** the ETL is hosted at the facility (in-house) and is totally under the control of the technical team. It gives some advantages, especially in terms of security, customisation and cost.

Finally, they can be open or closed source:

1. **Open-Source:** low-cost alternative but sufficient for many small to medium size businesses. Open-source projects bring many advantages and can really add value to a project. One of the aspects that brings the most value to an ETL is the number of connectors that it has, the possibility to connect to lots of different databases, files, SaaS platforms, ... and the fact that it is open source enhance even more this aspect because, e.g., if a connector is available but doesn't output the data in the desired format, there is a possibility to directly modify and customize it. Or, if a connector is missing but is really important for some users, they could implement it by themselves. Last example, if a connector brings some errors, there is no need to go through the customer support team to fix them, they can directly be fixed in the source code.
2. **Closed-Source:** They usually are more powerful ETL tools and they come with a fee. Companies offering this type of ETLs often give a customer support with it to help build them.

1.4.5 ETL tool comparison

Different companies have very different needs in terms of ETL tools. Companies working with big data will require highly efficient ETLs in terms of performances in order to deal with huge volumes of data. Some ETL tools are available for them but come with a fee. Here is a list of some of them with their respective price [6]:

Big data ETL tool	Starting Price
Hevo Data	\$249/month
Talend	\$12,000/year (or \$1170/month)
Informatica	\$2000/month
IBM	\$19.000/month

Table 1.1 – ETL tools prices

However, not everyone needs or can afford such offers. For example, students or small to medium size companies don't need super powerful ETL systems. Some ETL tools are also available for them at a much more affordable price or even free. Obviously, those ETL tools don't reach the same performance levels but some can already build really well performing ETLs.

In this section, in order to compare things at the same level, only free ETL tools will be considered. Only the best ones are listed and pro and cons are provided for each one of them.

SSIS

The first one considered is SSIS. It is developed by Microsoft and is one of the most known ETL tool since it is one of the oldest one (released in 2005).

Its main advantage is that it works very well with SQL server and more generally in a Microsoft environment. It has been around for a long time and made its proof by outperforming many expensive ETL tools. It is also very stable. As a consequence, it has a large user base. Another advantage is that it has a graphical user interface with drag and drop functionalities, so it is easy to use.

SSIS also has some disadvantages. First, as just said, it has been around for a long time, it is a good thing but it also comes with limitations. It gets a bit old and is neglected by Microsoft. New features are slow to be released, e.g., excel or flat file connections are limited by version and type and it gets very frustrating for users. The other concern about SSIS is that it is easy to get trapped in a pure Microsoft environment. It is installed as an extension in Visual Studio and by consequent, it can only be used in VS. It is the same thing with the database management system: it can only be used with SQL server so it's a limiting factor.

As a conclusion, SSIS is good in a full Microsoft environment. It can process big chunks of data, it is very fast, it contains lots of operators but it is a bit outdated and limiting if working with non Microsoft tools.

Kettle

A more modern solution will now be considered. Pentaho is a data integration and business analytics company. Inside the solution, there is a tool called Pentaho Data Integration [2] (also called Kettle) that provides ETL capabilities. Pentaho is not free but Kettle is a free and open source ETL solution.

Kettle solves some of the problems encountered with SSIS. First, it is compatible with many databases such as MySQL or MS SQL Server. It can also connect to some SaaS platforms such as Salesforce or Google Analytics. So it is not as limited as SSIS. Another positive point is that it is open source and as explained earlier it brings many advantages such as the number and customization of connectors.

There are still some problems that Kettle is not able to solve. Users still have to install a software. The user needs to install Pentaho Data Integration and setup the environment variables. So the ETL tool is stored on the user's computer. For most of the functionalities, extra plugins need to be installed. Therefore, it can quickly become time and memory consuming.

ETL.NET

ETL.NET [8] was released in 2021 by FundProcess and has a different approach than the previous two ETLs that were presented. Whereas those two use graphical interface with drag and drop system, ETL.NET is a .NET library.

The main advantage is that it doesn't require any software installation. Users just need to add a reference from a nugget to the core package as a normal library and then they can use the ETL functionality and create an ETL. It's really fast and easy to deploy and build ETLs. A simple ETL can be implemented in a few minutes.

The downside is that it is probably harder to use than a graphical user interface. It requires some coding skills which can be a problem for users having little programming knowledge. The next chapter will try to overcome this problem by reducing the learning curve.

Chapter 2

Domain Specific Language (DSL)

2.1 DSL Creation

2.1.1 Why creating a DSL ?

As explained above, the down side of using ETL.NET was it is more complex to use than a graphical user interface. To overcome that, a Domain Specific Language (DSL) for ETL.NET will be created. The goal of that DSL will be to make the use of ETL.NET accessible to people who have no or little programming skills. Those users must be able to use ETL.NET without having to understand all the complexity of the language. Typically, it uses a stream of events with lambda expressions which should be hidden to the end users.

2.1.2 Creating the language

The first step is to create the language and in this section, a few ideas will be tested and the best ones to achieve our objective will be selected. But first, some more details about ETL.NET will be provided in order to understand how it works : it is a stream based language meaning there is a stream as input, then some operators will be applied on that stream to create a different output stream. The following code shows an example of that (it doesn't work because parameters are not set correctly but it is just for the purpose of making the example simpler) :

```
var outStream = inStream
    .Where(//where operator)
    .EfCoreSave(//Save to database operator)
```

In this example, the input stream is « inStream » and two operators are successively applied : « Where » and then « EfCoreSave ». The output stream is kept in a variable called « outStream ». Also, a unique name has to be given to each operator. So, a way to implement that in our language must be found.

In our language, there are four important aspects to keep in mind :

- this is a stream base language
- operators must be implemented
- operators must have unique name
- operators must also be able to perform some sub operations. A typical scenario is when an operator must be able to perform equality. These sub operations are called « expressions ».

2.1.2.1 COBOL Inspiration

COBOL [11] stands for Common Business Oriented Language, it is a very old language that first appeared in 1959. It was designed for business, finance and administrative systems and it was a high-level language that relied on an english-base syntax. The interesting aspect for us is that it is a Domain Specific Language designed to be used by business people, meaning they probably don't know much about programming. As this approach was really similar to ours, it will be our starting point and some inspiration will be taken from COBOL to achieve our own goal.

How does it work? COBOL is not a block-structured language, instead it uses divisions, sections, paragraphs and statements. It is also self documenting with an english-like syntax. Here is how a « hello, world » would look like in COBOL :

```
IDENTIFICATION DIVISION.  
PROGRAM-ID HELLO.  
PROCEDURE DIVISION.  
    DISPLAY "Hello, world".  
END PROGRAM HELLO.
```

First, there is an IDENTIFICATION division whose goal is to give parameters like an Id, author or date. The only important one is the PROGRAM-ID as this ID is used when called from another program. Then, there is a PROCEDURE division which is a procedure as you could guess by name.

Secondly, there are also other divisions that can be used like ENVIRONMENT division used to connect the program to the outside world, the DATA division used to define data and their format but we are not going to get into too many details since it is out of topic. Instead, a typical syntax for a COBOL algorithm will be reviewed.

The following example is a COBOL algorithm to compute a gross pay :

```
COMPUTE-GROSS-PAY.  
    IF HOURS-WORKED > 40 THEN  
        MULTIPLY PAY-RATE BY 1.5 GIVING OVERTIME-RATE  
        MOVE 40 TO REGULAR-HOURS  
        SUBTRACT 40 FROM HOURS-WORKED GIVING OVERTIME-HOURS  
        MULTIPLY REGULAR-HOURS BY PAY-RATE GIVING REGULAR-PAY  
        MULTIPLY OVERTIME-HOURS BY OVERTIME-RATE  
        GIVING OVERTIME-PAY  
        ADD REGULAR-PAY TO OVERTIME-PAY GIVING GROSS-PAY  
    ELSE  
        MULTIPLY HOURS-WORKED BY PAY-RATE GIVING GROSS-PAY  
    END-IF  
    .
```

As we see this is very straight forward to understand thanks to the english-based syntax, especially for people working in HR. The next step is to take inspiration from that and translate it in our DSL.

COBOL-like DSL The use of divisions, sections, paragraphs and statements will be reproduced in our DSL. One division will represent one stream and inside this division, there will be one or more sections that each represent one operator. Then each operator will have statements for its expressions. The following example represent one stream in this language :

```
PROCESS outStream
  inStream
  OPERATION getFile
    READFILESYSTEM
    PATH ./myfile.csv
  OPERATION parseFile
    PARSECSV
    COLUMNS (
      col1[string],
      col2[date]
    )
  SEPARATION ,
  OPERATION removeBadString
    WHERE
      col1 IS NOT "Bad String"
END
```

The PROCESS keyword was used instead of STREAM because the goal is still to hide the complexity of the language to the end user and, in our opinion, understanding it as a sequence of processes is easier than understanding what a stream is.

With this syntax, a stream has all that it needs to be defined :

- a name for the variable that will contain the output stream (« outStream »)
- the name of the input stream (« inStream »)
- some operators

Operators also complete all their requirements :

- a unique name (« getFile », « parseFile » and « removeBadString »)
- the type of the operator (« READFILESYSTEM », « PARSECSV » and « WHERE »)
- all the expressions that a specific operator needs (« PATH », « COLUMNS », ...)

The very english-like syntax was also kept and it is quite simple to understand what the program is doing.

This is a first good shot but there are a few things that could be improved. The main one being that using divisions and sections adds some words to the language that could be avoided. The next attempt will try to overcome that.

2.1.2.2 SQL inspiration

Structured Query Language (SQL) is a language used to manage relational databases and perform operations on data. It is also a domain specific language allowing to perform data queries, data manipulation, data definition and data access control. The interesting part about this language is that it is very data oriented and this is definitely something that could be transposed in

our language.

SQL was furthermore designed for simplicity. Christopher J. Date, a specialist about relational database theory and thus a SQL specialist, said in 1974 : « The relational model is a particularly suitable structure for the truly casual user (i.e., a non-technical person who merely wishes to interrogate the database, for example a housewife who wants to make enquiries about this week's best buys at the supermarket). In the not too distant future the majority of computer users will probably be at this level. »

The goal of SQL being used by an housewife might not be reached. However, SQL was thought to be very accessible and this is something that can be translated in our language.

How does it work ? It is divided into several components : clauses, expressions, predicates, queries, statements. Not too much details will be given since it is assumed that most people know about SQL.

```
SELECT studentID, FullName
FROM student
WHERE FullName = 'Maximo';
```

This is a query, « WHERE » is a clause and « FullName = 'Maximo' » is a predicate

SQL like DSL In our language queries will be replaced by streams, clauses will be replaced by operators and predicate will be replaced by expressions. The example from previous section would be as follow :

```
SELECT
    col1, col2
FROM
    inStream
READFILESYSTEM
    PATH ./myfile.csv
PARSECSV
    COLUMNS (
        col1[string]
        col2[date]
    SEPARATION ,
WHERE
    col1 != "BadString"
```

It is shorter and easier to understand than the previous try. However, there still are some concerns about the syntax. Starting with the fact there is no unique name for the operators. This can be easily overcome by adding an AS « unique name operator » expression at the end of each operator. Then, the order of the operation seems a bit odd because it starts with the SELECT operator whereas in reality it will be the last one to be applied. Also, it would make more sense to start with FROM which just specifies the input (NB : FROM is not an operator). After making these changes, the previous example becomes :

```

FROM
    inStream
READFILESYSTEM
    PATH ./myfile.csv
    AS getFile
PARSECSV
    COLUMNS (
        col1[string]
        col2[date]
    )
    SEPARATION ,
    AS parseCSV
WHERE
    col1 != "BadString"
    AS removeBadString
SELECT
    col1, col2
    AS fileStream

```

This is more complete and solves most of the problems. There is one last thing that is missing, there is no name for the output stream. The first option that comes to mind is to add an AS expression after the FROM as it was done for the operators. This is a fair solution but there is a better one : the last AS statement could be taken as the name of the stream. In the example, it would be « fileStream ». This isn't problematic since FROM isn't an operator, it just gives the input so it doesn't require a unique name.

This is the Backus normal form (BNF) that will be used for our DSL. The next steps are now to implement the DSL and to decide which operators will be created. The choice of the operators is important because it will define what the language is capable of. Those topics will be discussed in the following sections.

2.2 DSL Implementation

In this section, the implementation of the language will be explained as well as the choice of the tools that were used. The goal is to take the source code from the language that was created previously, from now it will be called EtlScript, and translate it into ETL.NET which is a .Net library. This process is called a source-to-source compiler or transpiler.

2.2.1 Structure of the transpiler

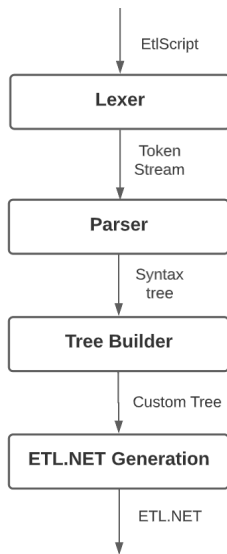


FIGURE 2.1 – transpiler’s structure

There are multiple steps in the transpiler. The input is some EtlScript code.

- The first step is to split the input into a token stream, it is performed by the lexer.
- As a second step, the parser will transform this stream into a data structure which is called a parse tree. It is easier to manipulate a data structure than a stream. The lexer and the parser are a set of rules, defined in a grammar, that the input code must follow. If not they will report errors.
- The third step is to build another tree that is more complete and more usable, we will call it the custom tree.
- The last step is to use this tree to generate some ETL.NET code. The general structure was given, the followings sections will go into much more details.

One of the first question that should be assessed before starting the implementation is the choice of the programming language. C# quickly seemed to be the best choice for some reasons. The first reason is that ETL.NET is a .Net library and is written in C#, so it would be more consistent to continue using C# in this environment. The second reason is that most FundProcess software is written in C#, since they will be using EtlScript and they might want to add some features in the future, it is easier for their developers to do it in C# since it is a language that they are used to. So for those reasons C# seemed to be the best choice.

2.2.1.1 Lexer

ANTLR was chosen to build the lexer but there were also other tools available to build lexical analysers from regular expressions (the most famous are lex and flex). Here are some of the points that were highlighted during the decision process :

- The transpiler is written in C#, so a lexical analyser that generates C# would be more consistent with the rest of the transpiler. ANTLR was originally designed to produce some java code but now, by tuning a bit the parameters, it can also generate C#. Lex and flex can’t generate C#, so in order to use one of them, a modified version like GPLEX/GPPG would have to be used.
- Lex and flex can only be used to perform lexing, so another tool for parsing is needed (usually Yacc or Bison). ANTLR looks like a good choice as it allows to perform both lexing and

parsing with the same tool, the 2 grammars can even be written in the same file.

- ANTLR is a top-down parser and it was designed for LL(1) grammar and our grammar has this property (see appendix A.1). Furthermore, the last version of ANTLR is more flexible, allowing language designers to focus more on writing accurate grammars. It implies that characteristics like LL(1) are less meaningful than before and they have little to no impact on users of the language.
- ANTLR has better error messages than the other alternatives. Since the grammar is LL(1), it is easier to show useful error messages as it can report which token was expected without ambiguities. It is helpful for the final user but it is also helpful for the developers since it is easier to understand and to debug.
- ANTLR has a higher complexity but the number of rules in the grammar remains quite small and EtlScript will mainly be used for short program that are less than 100 lines of codes. Thus, even if the complexity is a little bit higher, users won't even see the difference in a normal use case.

The goal of this first step is to transform EtlScript code into a stream of token. In order to achieve that, we have to define a set of rules in a grammar. Each rule will have the form of a regular expression, here are some examples :

```
FROM      : 'FROM' ;
NUMERAL   : [0-9] ;
NUMBER    : NUMERAL+ ;
```

It is in the grammar of the lexer that we define all the keywords of the language (FROM, AS, WHERE, int, string, ...), how we can name the variables, the comments, ...

As soon as the lexer will see a white symbol (it can be a space, a tabulation or an end of line), it will try to create a new token. It will check that this token is defined in the grammar. If not, an error will be thrown, e.g., a word starting with a number is not defined in the language. But if it is defined, it will create a token and give it a type. Here is an example (see appendix A.2 to have EtlScript grammar) :

```
FROM carStream WHERE carBrand != "audi" AS removeAudiCar
```

It will output the following stream of token (token name, token type) :

```
(FROM, FROM), (carStream, IDENTIFIER), (WHERE, WHERE),
(carBrand, IDENTIFIER), (!=, NOTEQUALS), ("audi", STRING_ID),
(AS, AS), (removeAudiCar, IDENTIFIER), (<EOF>, EOF)
```

2.2.1.2 Parser

The lexer can be seen as a dictionary that will check that all words are defined in the dictionary and the parser checks if those words can form a sentence. So, the grammar needs to be completed with all the rules to form sentences. ANTLR will also be used for this part for the reasons explained earlier.

The tokens that the lexer defines are the terminal nodes of the grammar whereas the rules that the parser defines are the non terminal nodes of the grammar. For example, the rule defining a stream is as follow :

```
stream : FROM input operation* output?;
```

We have a combination of terminal (FROM) and non terminal nodes (input, operation and output) that forms a regular expression. This regular expression defines one of the parsing rule. The role of the parser is to check that the source code can combine the tokens provided by the lexer to one of the parsing rule. If not, an error is thrown. If it can it will combine them in a structure called a parse tree. If we take the same example as before and our grammar (see appendix A.2), the parser will build the following parse tree :

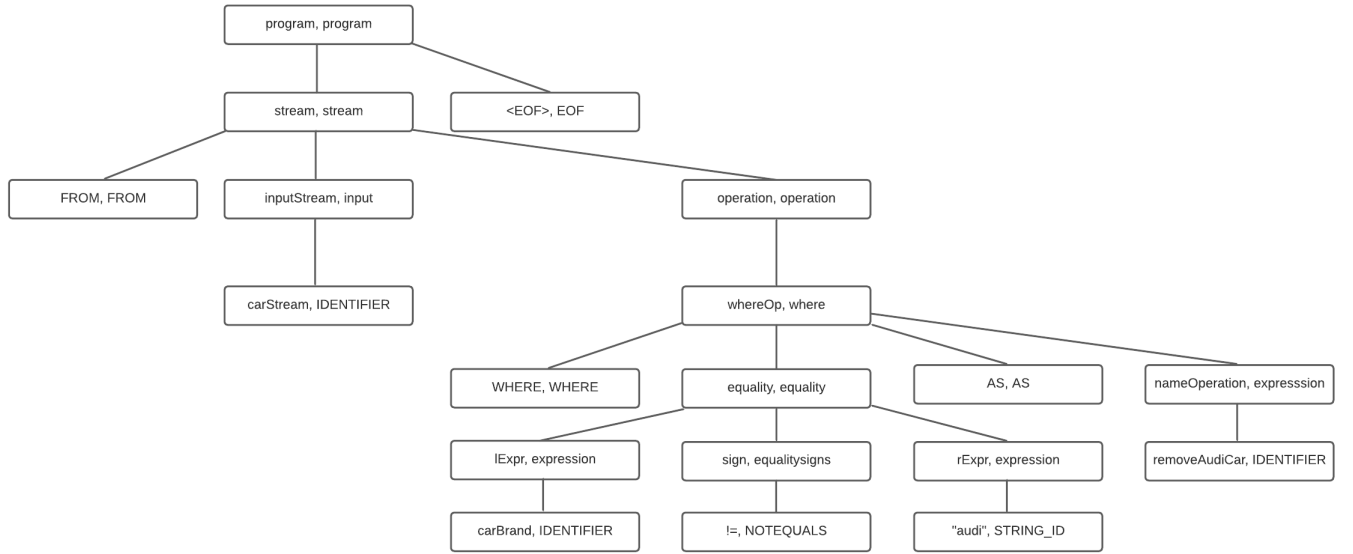


FIGURE 2.2 – parse tree

The leafs of the tree are the tokens given by the lexer and all the other inside nodes are non terminal symbols and are just a way of combining other nodes.

After this step, the grammar is now complete. As shown in appendix A.1, the grammar is LL(1) and one of the propriety of these type grammars is that they are not ambiguous. Thus, the grammar is unambiguous and has no conflicts.

2.2.1.3 Tree Builder

The initial input is now in the form of a structure which is more useful to us but it is still quite hard to manipulate. In this section, a custom data structure will be built. It will have the parse tree as basis but custom objects will be defined. These objects can be manipulated and they offer much more freedom and possibilities. This new data structure will be called the custom tree.

For this, a visitor will be created. A visitor is a way to add operations to a structure without having to change the structure itself. In more concrete terms, it means that a visitor class will be defined, this class has a *Visit()* method for each non terminal node defined in the grammar. Thus, we can go trough the parse tree and at each non terminal node, custom operations can be

performed without modifying the parse tree. One way to take advantage of the visitor is to return custom objects from the visit method. By doing this, the custom tree can be built using custom objects.

In order to define the customs objects, an interface called `IDslObject` will be created and all the other classes that represents the customs objects will inherit from this interface :

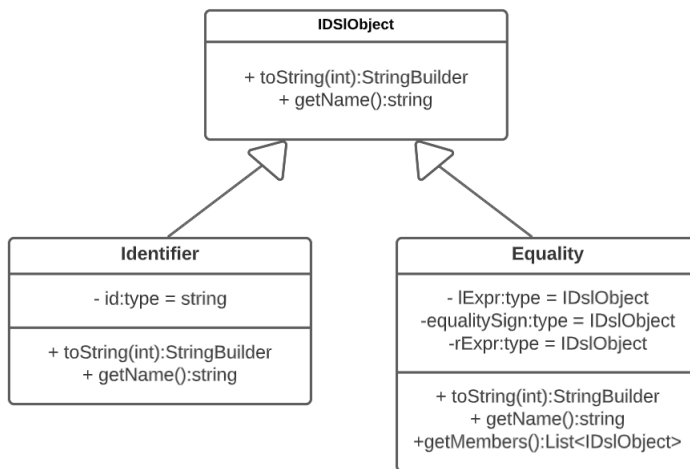


FIGURE 2.3 – class diagram

Two methods are defined in the interface and all the classes inherit from this interface. Here, we only represented two classes for the example.

Those classes will be used to build the custom tree we mentioned earlier. Each node will be represented by an object inheriting from `IDslObject`. By doing this, the newly created tree can be manipulated much more than the parse tree, e.g., we can add fields and methods to the classes.

Now that custom objects are defined, they can be returned from the different *Visit()* methods. Here is how the visitor class looks like :

```

class TreeBuilder
    IDslObject Visit(EqualityContext context)
        // Visiting the different parts of the equality
        return new Equality(//args)
  
```

All the *Visit()* methods will have the same pattern which starts by visiting the different parts of the context and then build an object and return it.

The full custom tree can be returned by calling the *Visit()* method on the root of the tree, here is the tree for the previous example :

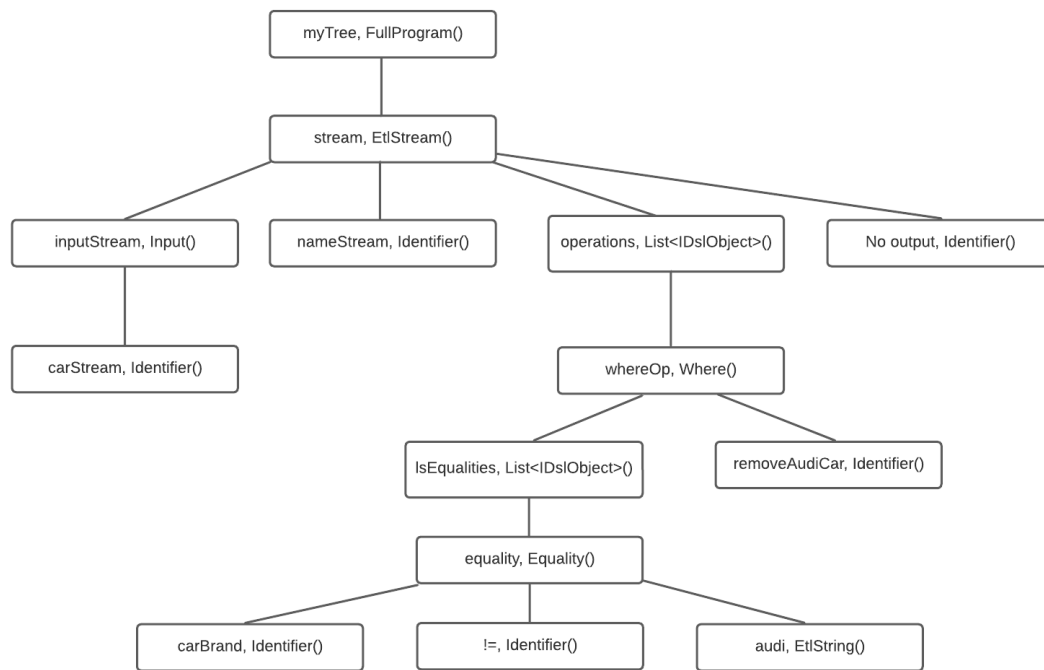


FIGURE 2.4 – Custom Tree

Figure 2.4 represents the custom tree. It contains less nodes than the parse tree and all nodes are custom objects.

2.2.1.4 ETL.NET Generation

This is the last step. The custom tree will be used to generate ETL.NET code. To achieve this, a method *toString(int)* was created in each class (as can be seen on Figure 2.3).

When calling the *toString(0)* method on the root node, it will recursively call the *toString(1)* method of its child and so on until they reach the leafs of the tree. Each node is responsible of generating its part of the ETL.NET code. The *int* argument of the *toString(int)* is just there to display some tabulations for better reading purposes. Here is the code that will be output for the previous example :

```

var removeAudiCar = carStream
    .Where("removeAudiCar", l => l.carBrand != "audi");

```

2.3 Possible operations with EtlScript

In the previous sections, some possible operations were presented along the way. It's time to present all the possible operations that the language can perform. This section is more a list of the possibilities the language offers than a documentation of the language. Each operation will only be briefly described because explaining everything in detail may be too long and off topic.

2.3.1 Stream

The language is stream based so the only way to use EtlScript is to define streams. A stream is defined like this :

```
FROM <source> List<operation> <destination>
```

FROM is the keyword to define a new stream. Then the source, a list of operations which are executed on the stream (the list can be null) and then an destination (can also be null) have to be specified.

2.3.2 Source

This section correspond to the load part of an ETL. We will define where data are coming from. It can be from an other stream but it can also be from an external source, e.g., a file or a database.

FLATFILE

```
FLATFILE
    NAME <file name>
    COLUMNS List<column>
    SEPARATION <separation sign>
    AS <operation name>
```

FLATFILE allows to access some text files, the most common being .txt and .csv files. The user has to specify the name of the columns after the COLUMNS keyword and by default, columns are assumed to be strings. If some columns have other types (e.g., number or date), it has to be specified.

TABLE

```
TABLE <table name> List<table operation> AS <operation name>
```

This is how the database can be accessed, if <table name> exists in the database, all its columns are accessible. A column can either be a property or a link to another table.

Then some operations can directly be performed on the data, <table operation> can be :

- WHERE <where operation> : filter used to extract records fulfilling a specified condition
- INCLUDE <table or property name> : adds other data columns than those in <table name>
- ORDER BY <property name> : sorts records

CONNECTOR

`CONNECTOR <connector name> AS <operation name>`

A connector is a way to access, e.g., a server or a mail box in FundProcess and it is defined by a name.

STREAM

`<stream name>`

The input of a stream can also be another stream. In this case, the name of the other stream just has to be given after the FROM keyword.

2.3.3 Operation

This section contains all the operations coming after the input. It refers to the Transform part of the ETL.

WHERE

`WHERE List<condition> AS <operation name>`

It is used to filter records fulfilling a specified condition between two elements. This condition can be : =, !=, >, >=, <, <=, CONTAINS, NOT CONTAINS.

DISTINCT

`DISTINCT List<name column> AS <operation name>`

It is used to make sure that every record has a different value for all the columns listed. It removes duplicates.

GROUPBY

`GROUPBY <name column> AS <operation name>`

It is used to group rows having the same values into summary rows.

SORT

`SORT List<name column> AS <operation name>`

It is used to sort the stream in a specific order, it can sort with multiple criteria.

LOOKUP

```
LOOKUP
    INTO <name stream>
    ON <equality>
    SELECT List<columns>
    AS <operation name>
```

LOOKUP is used to link two streams together. We are not going to go into too many details but it works nearly the same way as the SQL lookup. The pivots have to be specified after the ON keyword.

There is a second variant of LOOKUP which is LOOKUPDB. This one will link the current stream with a table from the database. The syntax is the same except it has one more option. A <cache operation> can be specified after the SELECT. This operation can be NOCACHE(<size>) or FULLCACHE. The reason is that by default lookup will keep the full data set in memory. But sometimes the data set is too big to be entirely stored in memory. So by activating this option, if the item it was looking for is not in the cache, then it can query the database again.

SELECT

```
SELECT
    OBJECT <object name>
    List<selected>
    AS <operation name>
```

It is used to select some data from the current stream. If an object name is specified, it can be used as a constructor for this object and newly created objects could be saved in the database in a future operation.

DELETE

```
DELETE <name Table> List<table operation> AS <operation name>
```

It is used to delete some data. <table operation> can be used to make certain conditions on which data needs to be deleted.

CORRELATE

```
CORRELATE
    WITH <stream name>
    SELECT List<columns>
    AS <operation name>
```

CORRELATE is used in the context of normalization of flat structure. Normalizing flat structure means breaking a table in several smaller table that reference each other in order to take less memory space.

SYNCHRONIZED

```
SYNCHRONIZED <stream name> AS <operation name>
```

ETLScript is a stream language and streams can be executed in parallel, thus SYNCHRONIZED is used when a stream needs to wait for another stream.

CROSS APPLY

CROSS APPLY <cross apply operation>

This operator is a way to execute multiple operations for each item of the original stream. Thus, for each original item, we can have multiple output items.

2.3.4 Destination

As for the source, the destination can have the form of a connector or a flatfile. The syntax is the same as for the source. However, the last one is a bit different.

TABLE

TABLE SEEKON List<column> AS <operation name>

This is used to store the content of the current stream in the database. Most of the time this operation is preceded by a SELECT operation where an object corresponding to a table in the database is created. Thus, this operation will look into the database and store the previously created objects.

The list of column is used when some items have to be inserted in the database with a pivot (or more) that is different from the primary key.

2.3.5 More operators ?

Now, it is clearer what EtlScript allows to do. Each operator previously presented uses one or more operators from ETL.NET and they are the ones that are the most commonly used. But as the focus was to try to simplify the language as much as possible and as time was limited to realize the project, there still remains a lot of ETL.NET operators that are not usable with EtlScript at the moment. Here is a list of the principal limitations of EtlScript as of today :

- The diversity of sources : Section 4 stated that one of the most important aspect of any ETL tool is the diversity of sources. In the context of this project, it was not possible to implement forty different sources due to the time factor, so we decided to only focus on three :
 1. FLATFILE allows to interact with files, more precisely with .txt and .csv files which are common to store data and easy to access.
 2. TABLE allows to interact with the database via Entity Framework (more in the next section).
 3. CONNECTOR allows to connect to multiple other sources by providing a json file with all the required information for the input setup. Of course it requires that connectors are created before using EtlScript and the connection is not directly done with EtlScript.

The first two sources give a way to access files and databases which already allows to build a lot of ETL systems because they represent two common sources for an ETL. The last one, CONNECTOR, is a simple way to add different sources that EtlScript can't handle by itself without having to implement them. In our opinion, starting with those three sources was the best choice since most ETLs can already be build from them.

- The number of operators : for the same reasons, all the operators present in ETL.NET couldn't be implemented. So, focus was on the most used ones, e.g., there are many ways to combine streams in ETL.NET (LeftJoin, Lookup, Union, UnionAll, Subtract, CorrelateToSingle, CorrelateToMany) and by reviewing some ETL.NET code, we noticed that Lookup

was the most used one by far and that `CorrelateToSingle` was mandatory in the context of normalization of flat structures which is a common way to use less memory space. So those two were implemented first. All the other operators used to combine streams work quite similarly, so by proving at least one works, the same recipe can be taken and applied to the others but time was spent on different features. This was an example for stream combinators but the same strategy was applied for many other different types of operators, each time by looking which one are the most used/important and by only implementing a few of them.

- The lack of flexibility in some operators : Even in the operators that `EtlScript` furnishes, sometimes some functionalities are missing. For the same reasons, the focus was on the most common behaviors rather than having all the possibilities implemented since day one. There is a trade-off between number/complexity of possible operations and simplicity of the language and as the goal is to design a language aimed to be used by people with little programming experience, simplicity was chosen over complexity.

2.4 Integration in FundProcess System

EtlScript is going to be used by FundProcess, so it will be integrated in FundProcess environment. This will also give access to a real database, this opportunity will be taken to interact with this database and to realise unit tests.

2.4.1 Different Modes

FundProcess uses three different modes and we will explain the particularities of each one of them.

File Retriever

This mode is designed to take some data from a database source, transform those data and load them as a file. So one of the challenges is to have an good system to extract data. The source to take items from the database in EtlScript is the TABLE source, it will be this one that will be used in this section.

```
string FileRetriever(string etlScript, DbContext dbContext);
```

This mode requires two inputs : a string and a DbContext. The string corresponds to the EtlScript code that is needed to translate into ETL.NET code. DbContext requires a little bit more explanation. It is a component of Entity Framework. Entity Framework is a .Net tool that does the mapping between the .Net objects and the tables in the database. It is useful because we don't need to directly interact with the database, tables and columns with SQL queries. It allows to work at a higher level of abstraction by interacting with .Net objects. DbContext can be seen as the data access of Entity Framework. So, information that we need can be retrieved from DbContext.

TABLE operator :

```
TABLE <table name> List<table operation> AS <operation name>
```

In the EtlScript code, the user has to specify the name of the table he wants to access. Then, dbContext will be used to check if that table exists. If yes, access to all its columns is given. A column can be a property of the table or it can be a link to another table. But if the user wants access to another item that is not in one of its columns, he can use INCLUDE which is one of the <table operation> and then he can specify the path to that item. The following example reproduces two tables in the database :

Table Book

string title

Author writer

Table Author

string publishingHouse

string email

The EtlScript code « FROM TABLE Book AS getBook » will create a stream with an access to all the properties of Book which means title and writer. Those can be used by the operators that will come next. As can be seen, the writer property is a link to another table but writer's properties (publishingHouse and email) are not accessible in the current stream and they can't be used by other operators. But this problem can be avoided with the INCLUDE keyword by specifying all the properties needed. The code « FROM TABLE Book INCLUDE writer.email AS getBook » will create a stream that has access to title, author but also to email.

This was a simple example to show how to give the stream access to a property of one of its row but this is not the only relationship that exists in a database. There is also some inheritance relationships, a property could be inherited from a super table or at the opposite, a super class may want a property from a sub table.

Table Book	Table Person	Table Author inherits Person
string title	string name	string publishingHouse
Person writer	int age	string email

An example of inheritance with the tables above is the code « FROM TABLE Book INCLUDE writer.email AS getBooks ». The type of the property writer is Person, so it will check if it has a property called email. It will see that is not the case. Then, it will search in its super table and its sub tables. In this case, it will find it in a sub table called Author. The thing to notice is that the syntax « INCLUDE writer.email » is the same as the one to access the property of a property even if the relation between the two tables is different. The syntax is also the same if we try to access a property of a super class.

It is much simpler for the user because he doesn't have to understand the different relationships existing in the database as the syntax is the same for all of them. The two examples that were presented were very simple and only had one level of depth but in a real use case it can quickly become complicated with longer path and different relationships along the way. But EtlScript solves that by hiding them to the user.

This is realised by an algorithm that takes the path the user gives as input and then checks all the different possible relations. If it doesn't find anything, it means that the input path was wrong. There can also be some ambiguities : a given path can end to more than one place in the database. A typical case is if two subclasses both define a property with the same name. In the previous example, it would be the case if another subclass of Person also defines a property called email. Thus, there is a priority list in the algorithm, it will first check if the input is a direct property of the class, then if it is a property of the super class and finally if it is a property of a subclass. In the latter case, if ambiguities still exist, it will just take the first one. It reduces the number of ambiguities but some are still possible. We made the choice to leave them and take the most logical one because removing all of them would induce that the user has to give more specific inputs and it will not hide the different types of relationship anymore. And in the few cases where the ambiguity is a problem, the user could still modify the ETL.NET code that is output. The syntax of the ETL.NET code would remain the same, it would just need to modify the name of some classes.

So a user might want to use the File Retriever mode when he needs to retrieve data from the database. The counter part is that this mode requires a DbContext as argument because without it, EtlScript has no idea of the database structure.

File Processor

This mode works in the opposite way as the previous one, it expects a file as source.

```
string FileProcessor(string etlScript, string pathToFile);
```

This mode also takes two inputs which are two strings. The first one corresponds to some EtlScript code and the second one corresponds to the path to the input file.

The source is a file so the FLATFILE source has to be used, but since the file was also given as input, many information the user won't have to write can be deduced from it, i.e., the name of the file, the name of the columns and the separation sign. So, another version of FLATFILE that doesn't require as much information was created. Here it is :

```
FLATFILE List<column type> AS <operation name>
```

The only information that can't be deduced is the type of the different columns. By default, it is assumed that they all are strings. If it is the case, the user doesn't need to specify anything. If not, he must specify all the types that are different from string.

In a case where a file has three columns (name, birth and age) that respectively have types : string, date and int. A valid EtlScript code is « FROM FLATFILE (,date,num) AS fileDefinition ».

Data Processor

This last mode is a mode that has a lot of freedom. Nothing is assumed about the source or the destination of the ETL. So, users have access to all sources and destinations of EtlScript.

```
string DataProcessor(string etlScript);
```

This mode only requires one string as input, it corresponds to some EtlScript code.

This mode has a lot of freedom but it doesn't have the advantage of the two others. It doesn't know the structure of the database, so the INCLUDE operator won't be able to deduce the relationships between tables. Thus, it will just assume everything coming after INCLUDE is a direct property of the table. Same thing if the input is FLATFILE, the user needs to specify more information than in the File Processor mode because he doesn't have access to the simplified version of FLATFILE.

So this mode has the advantage of being easier to use because it doesn't require anything else than some EtlScript code but it has the disadvantage of not being able to benefit from the most advanced functionalities of the other modes.

2.4.2 Unit Testing

Dependency injection with AutoFac

Before explaining dependency injection, dependency inversion will be defined. Dependency inversion is a programming principle for loosely coupling software modules. Traditionally, high-level modules depend on the implementation of low level modules. The idea behind this principle is to reverse that. Thus, high-level modules should not import anything from low-level modules, they both should depend on abstraction.

If an application follows this principle, it allows to disconnect pieces of the application from each other easily and allows to test part of our application independently.

Dependency inversion is a principle, dependency injection is how you make it work. With dependency injection, an object receives other objects that it depends on, so it doesn't have to construct them and it leads to loosely coupled program.

Autofac is a .Net library that helps to realise dependency injection. It builds containers. A container assembles components from different part of the application together.

FundProcess uses Autofac for their application. A container containing only the components strictly needed can be built. Those components are EtlScript implementation and FundProcess DbContext. So, a container with both of them is built using Autofac and our application can be tested inside this container.

XUnit

XUnit was used to realise unit testing. It is a free open source testing framework for .Net. It is widely used in the .Net community because it is better than the other .Net testing framework (NUnit and MSTest) on many points : isolation of tests, extensibility, assertion mechanism to only name a few.

All our tests have the same syntax :

```
Test()  
    var expected = // Expected ETL.NET Code  
  
    var input = // EtlScript Code  
    var actual = // Using one of the three modes on the input  
  
    Assert(expected,actual) // Verifying that output is right
```

In file retriever and file processor mode, a second input is needed, respectively a dbContext and a file. The same second input was used for all tests and they are inside the container built by Autofac.

The first tests performed are tests on the lexer and parser. Those are the same for the three modes. For the lexer, they verify that the EtlScript code is split into the right tokens. They also test that the lexer throws exceptions when the input EtlScript code is wrong such as characters that are not defined in the grammar. For the parser, they check that it throws exception when the input is wrong, e.g., when a keyword is misspelled or missing.

Then, every custom Object is tested separately. All the classes inheriting from IDslScript (see figure 2.3) must output some ETL.NET code and tests check that this output code is correct. Those tests have been divided in three categories, one for each part of the ETL (i.e., extract, transform, load).

Finally, the features of the three modes are tested separately. Tests check the INCLUDE operator returns the right path from the database in the file retriever mode. In the file processor mode, they check if the information we get from the input file is correct. Data Processor mode is not tested alone since it corresponds to the default behaviour, i.e., the one that was tested in the previous paragraph.

Object of the tests	number of tests
Lexer & Parser	8
Extract	10
Transform	18
Load	7
File Processor	3
File Retriever	13
Total	59

TABLE 2.1 – Tests

Table 2.1 summarises all the different tests and give the number of tests that were realised.

Chapter 3

Categorising features

3.1 ETL subsystems

The concept of an ETL seems quite simple to understand, you get the data out of its original source location (E), you do something to it (T), and then you load it (L) into its final destination. But this simple definition can lead to many interpretations and for a long time there was no real agreement on a common structure for an ETL solution. Then in 2004, Ralph Kimball and Joe Caserta wrote a book called « The Data Warehouse ETL Toolkit » that filled that gap. They listed 38 subsystems that needed to be implemented in an ETL. These subsystems provide a framework which helps to understand and categorise the implementation and management of an ETL solution. Since then, many organisations rely on those subsystems while building their own ETL. They were a few more editions of the book that restructured a bit the subsystems and condensed the list into 34 ETL subsystems. This chapter will refer to the third edition, edited in 2013, called « The Data Warehouse Toolkit » [1].

EtlScript was created to help people build ETLs, so we will verify which ETL subsystems can be built using EtlScript and which can't and why. The following tables will list the subsystems and give more explanation if the name is not relevant enough. Then, it will give a status of whether or not, it is possible to build this subsystem with EtlScript. Finally, the last columns will give a few information on how to build it, if it can. EtlScript produces ETL.NET code, thus some systems are not directly handled by EtlScript but they are by ETL.NET.

Before going any further, a topic needs to be clarified. Chapter 1 explained the concept of Business Intelligence (BI) and the role of ETLs in that matter. One of the role of ETLs is to process data and making it immediately available for managers to take business decisions. ETL.NET was created with a different mindset which was less BI oriented [9]. ETL.NET was thought to process data and store it in the data warehouse without having the intention of using it immediately. This is a pure software engineering approach of the technology and the distinction has to be made from the BI approach. This approach being different, it fits less the 34 subsystems considered in the traditional way of building ETLs described in « The Data Warehouse Toolkit ». This is especially true for the managing subsystems that will be described below.

3.1.1 Extraction subsystems

The first three subsystems are related to extracting, i.e., getting the raw data from its source systems and write it to the disk before any significant restructuring of the data takes place.

Subsystem	Status	Justification
1) Data Profiling : Gaining insight into the content and structure of the various data sources, e.g., statistics like number of columns or size of a table.	No	
2) Change data capture system : transfer only the relevant changes to the source data since the last update. If a data did not change, do not reload it.	No	
3) Extraction System : extracting data from a wide variety of source systems.	Yes	Sources can be files, databases and connectors.

TABLE 3.1 – Extracting

3.1.2 Cleaning and conforming subsystems

Subsystems four through eight describe systems whose role is to send source data through a series of processing steps in the ETL system to improve the quality of the data received from the source.

4) Data cleansing system : fixing or tidying up dirty data that comes into the ETL process.	Yes	Operators like WHERE, DISTINCT are there for this. They allow to perform actions such as removing irrelevant or null data.
5) Error Event Handler : record every error event that occurs during en ETL load.	Yes	Handled by ETL.NET.
6) Audit Dimension Assembler : contains the metadata when a table row is created, e.g., load date.	No	
7) Deduplication System : when the same information is stored at different places but should be merged, e.g., a client is stored twice but with names written differently or misspelled (Jean-Pierre Dumond and DUMOND JP).	Partially	DISTINCT operator can partially solve the problem if 2 records have the same key but not if keys are different. It would require some machine learning algorithm and EtlScript doesn't provide that.
8) Data Conformer : the fact of conforming incoming data from an outside source system into the right format so that it fits the company system.	Yes	Most of the operators are there for this purpose, specially SELECT which can create an object from the company system. Then this object could easily be loaded into the system because its type would be known by the system.

TABLE 3.2 – Cleaning and conforming

3.1.3 Delivering subsystems

Those subsystems are in charge of physically structuring and loading the data into the presentation servers. Some subsystems have not been covered in this section (see appendix A.3 to understand the reasons). There is also more information about fact tables in the appendix.

9) Slowly Changing Dimension Processor : handling an attribute value that has changed from the value already stored in the data warehouse (most of the time it corresponds to an « update » feature)	Yes	The OUTPUT TABLE operator calls the <i>EfCoreSave()</i> operator from ETL.NET which is an upsert operator meaning that it will check if the primary key already exists in the database. If yes, it will update the row, if not, it will create a new row.
10) Surrogate Key Creation System : generation of a meaningless key, typically an integer, to serve as a primary key for a dimension row.	Yes	Handled by ETL.NET.
11) Hierarchy Manager : possibility to build and to manage multiple, simultaneous, embedded hierarchical structures.	Partially	Possibility to build normalized data structure with CORRELATE operator. But there are other types of structure that EtlScript can't handle such as snowflakes structures.
12) Special Dimension Builder : possibility to support an organisation's specific dimensional design characteristics.	Yes	The FLATFILE operator gives the possibility to specify what are the type of the columns. It can handle strings, dates and numbers.
14) Surrogate Key Pipeline : It is a way to make the link between the dimensional table and the fact table.	Yes	This can be realised with LOOKUP operator this by taking the primary key as pivot.
15) Multivalued Dimension Bridge Table Builder : bridge tables are needed to allow variable depth hierarchies and variable kinds of relationships.	Partially	EtlScript can't build those kind of tables but them can deal with it while extracting data thanks to the INCLUDE operator.
16) Late Arriving Data Handler : data doesn't always arrive at the same time. In some cases, the fact table could be filled before the dimension table. In this case, there should be some error handling mechanism like creating a dummy value for the missing records.	No	

TABLE 3.3 – Delivering subsystems part1

19) Aggregate Builder : an aggregate table is a table that summarizes information, e.g., a table that contains all the sales by product in a month.	Yes	It can be done with the GROUP BY operator.
20) OLAP Cube Builder : OLAP is a type of database that presents dimensional data in an intuitive way, enabling a range of analytics users to slice and dice data.	No	
21) Data Propagation Manager : it is used to get data out of the data warehouse and send it to other environments for special purposes, e.g., submit data to the government organizations for reimbursement purposes.	Partially	EtlScript doesn't have a special tool for this purpose but it is possible to extract data and put it in the right format.

TABLE 3.4 – Delivering subsystems part2

3.1.4 Managing subsystems

The goal of those subsystems is to provide timely, consistent, and reliable data to empower the business. Criteria are : reliability (ETL must consistently run), availability (warehouse should be up and available) and manageability (constantly grows and changes along the business).

22) Job Scheduler : launching jobs on a schedule. The scheduler needs to be aware of and control the relationships and dependencies between ETL jobs, e.g., recognise when a file is ready to be processed.	No	
23) Backup System : a solution recommended by Ralph Kimball is to back up in three places : immediately after extracting, after transforming, after final preparation of the BI-accessible data sets.	No	
24) Recovery and Restart System : restart a job when it fails somewhere during the process.	Partially	If an EtlScript process fails and must restart, it will restart from the beginning but if some records were already loaded in the database they will not be loaded a second time.

TABLE 3.5 – Managing subsystems part1

25) Version Control System : capability to archive and recover all the logic and metadata of the ETL.	No	
26) Version Migration System : capability to migrate from one environment to the next, e.g., from development environment to test environment.	No	
27) Workflow Monitor : it provides a dashboard and reporting system taking many aspects of the ETL system into consideration, e.g., how many rows have been processed ? How fast ?	No	
28) Sorting System : Some operations requires data to be sorted, thus ETL must be able to sort by rows.	Yes	In the TABLE operator, there is the ORDER BY keyword that allows to sort data when they are extracted from the database. There is also the SORT operator that can be used to sort a stream on a specific criteria.
29) Lineage and Dependency Analyzer : Lineage is working backwards from the data to show where it is originated and what operations were performed on it. Dependency is starting from the data, show the following transformations and what their impact will be.	No	
30) Problem Escalation System : In case something goes wrong, you need to be notified as soon as possible.	Yes	EtlScript handles exception handling and error reporting
31) Parallelizing/Pipelining System : In order to obtain good performances, an ETL system must run tasks in parallel.	Yes	ETL.NET process streams in parallel, so does EtlScript. It also provides the SYNCHRONIZED operator in case a task needs to be finished before starting a new one.
32) Security System : An ETL has access to the database of an organisation, thus it can lead to some serious security breach.	No	EtlScript can access databases via Entity Framework. It relies on the security system that entity framework provides but it doesn't provide its own security system.

TABLE 3.6 – Managing subsystems part2

33) Compliance Manager : a compliance system is the interaction of several of the subsystem already described such as : lineage and dependency analysis (29), version control (25), backup and restore (23-24), security (32).	No	
34) Metadata Repository Manager : an ETL is responsible for the use and creation of most of the metadata involved in the data warehouse.	No	

TABLE 3.7 – Managing subsystems part3

3.1.5 Summary

Subsystems	Yes	Partially	No
Extraction	1	0	2
Cleaning and conforming	3	1	1
Delivering	5	3	2
Managing	3	1	9
Total	12	5	14

TABLE 3.8 – Subsystems summary

Table 3.8 shows that the biggest gap to fill is in the managing subsystems where 9 out of 13 subsystems are not possible to build with EtlScript. Managing subsystems provides functionalities such as dashboards, backup systems or changing from one version to another. As explained at the beginning of this chapter, ETL.NET approach is a bit different than the traditional approach and thus this is not really a surprise. However, ETL.NET provides a tracker functionality that displays diagrams of the current process. The tracker describes the execution plan and it gives a better overview of how the nodes representing the operators interact with each other. Some of the subsystems could be build from that.

The goal of this project was to create an easy language to produce some ETL.NET. The project is more about building ETLs than managing them which represent some very different tasks. The nature of managing subsystems is very different from the other types of subsystems (i.e., extracting, cleaning and conforming) and need a different approach. For this reason and the fact that ETL.NET is different from the traditional approach on that aspect, this work decided to leave them aside in the first place. However, it might be interesting to consider them in a future work that would see ETLs from a different angle. An approach where the focus would be on topics like version management or dashboard creation.

By analysing a bit more the missing subsystems, we identify that lots of them are related to metadata (1,2,6,25,27,29,33,34). As said, this project was about the creation of ETLs and not really

about data itself. So it might also represent some future work that will focus more about data, metadata and databases and which would deal with higher volumes of data and the challenges coming with it.

Chapter 4

Macro

4.1 Creating Macro With Data imported with the DSL

This section is not really related to what was done previously but we thought it would be interesting to show an example of how data can be used once they are inside the database of FundProcess. An ETL was used to insert records and then the macro designer will be used to display some computation details.

The term « Private Equity » refers to a financial investment in non-listed shares, meaning that they come from private companies that are not publicly listed on the stock market. Since mid-2020, the private equity investment fund sector in Luxembourg has been growing exponentially and particularly because of the Brexit.

A limited partnership is made between the general partner (GP) who runs the business and limited partners (LP) who invests money but don't run the business. This structure is often used in the private equity sector.

Once some profits have been made, it must be split between the partners and it is done according to the distribution waterfall method. The capital gained by the fund is allocated between LP and GP. It can be pictured as a set of buckets. When one bucket is full, the capital flows into the next bucket. The first are usually entirely for the LPs and the further we go, the most advantageous they are for the GP. It is usually done in four steps :

- **1) Return of Capital** : Return to the LP the amount he has invested
- **2) Preferred Returned** : Return to the LP until a specific internal rate of return (IRR) is reached.
- **3) Catchup** : Return to the GP (usually it is 20%)
- **4) Carried Interest** : Shared between LP and GP (most of the time 80-20)

FundProcess macro builder was used to build a macro that computes the four steps and we display the result :

STEP 1: Return of capital invested	
Capital invested	10 000 000,00
STEP 2: Preferred Return	
number of year	5,00
coefficient	1,47
Preferred return to LPs	14 693 280,77
Capital remaining after step 2	15 306 719,23
STEP 3: Catch up	
new base to compute catch up	18 366 600,96
Catch up amount for GP	3 673 320,19
Capital remaining after step 3	11 633 399,04
STEP 4: Profits Split	
Split GP	2 326 679,81
Split LP	9 306 719,23
Grand Total proceeds	
Total GP	6 000 000,00
Total LP	24 000 000,00

Chapter 5

Conclusion

This thesis described the creation of a Domain Specific Language (DSL) which allows to build ETLs in a simple way. The language relies on the ETL.NET library. It allows to extract and load data from files, databases and connectors. It can perform some actions like filtering, sorting or grouping thanks to some operators.

Three modes are implemented and they have their own characteristics :

- File Retriever : a better way to retrieve data from the database
- File Processor : a better better to retrieve data from files
- Data Processor : provides more freedom about input and output sources.

Then, the DSL was confronted to Ralph Kimball and Joe Caserta book called « The Data Warehouse Toolkit ». The book provides subsystems that need to be implemented in an ETL and these subsystems provide a framework that helped building the language. Each subsystem was taken individually to check if it was possible or not to build it using the DSL.

The DSL is designed to be used by casual users having little to no programming experience. Our approach to create a simple syntax reduces the need to understand some of the complex aspects of ETL.NET. By doing so, the learning curve is reduced making the ETL technology more attractive to a larger public.

Companies with small databases or students could be realistic users, e.g., a company which needs to generate a text report to monitor the sales of the week, but information needed is spread across multiple tables in the database. They could use EtlScript to solve this problem in a few minutes.

Our language is also useful for IT staff. Indeed, they can use it to build ETLs in a few minutes leading to more efficient work. But more importantly, companies will also be less dependent on them to build ETLs since other workers will be able to build them too. This will allow them to save time and focus on other features.

The implementation of the language does not pretend to be perfect and some areas of improvement were identified. The DSL allows to realize lots of basic ETL functions but it could be completed by adding new sources and new operators to be able to realise more operations and sometimes even more complex ones. However, this language was designed to be simple to use and there is a trade-off between number/complexity of possible operations and keeping the simplicity

of the language. So, if new functionalities were to be implemented, they should be designed keeping simplicity in mind so they remain simple to understand and simple to use.

As discussed in Section 3.1.5, an ETL should provide some managing functionalities and the DSL is very limited on that aspect. Some functionalities could be added such as dashboards with information about the ETL or the possibility to change from a version to another or to have backup systems. This broader view of the ETL brings many new challenges and it could be a very interesting topic.

Most of the other widespread ETL tools use a graphical interface with a drag and drop system. In section 1.4.5, we explained that ETL.NET approach was different but it could be a great asset to have both possibilities : using it as .Net library or using it through a graphical interface. It could be looking like Scratch (see Appendix A.4) which is a high-level block-based visual programming language.

As a more personal note, this work was a great opportunity for me and brought me some valuable experience. I had to learn many new technologies and new concepts that I will use in the future.

Appendix

A.1 LL(1) Grammar

A predictive parser is a top-down parser that doesn't require backtracking. The next input symbol and the current non terminal symbol perfectly determine the rule to apply. An LL(1) is a grammar for which predictive parsing is possible. LL(1) stands for :

- **L** : left-to-right scan of the tokens
- **L** : leftmost derivation
- **(1)** : one token of look ahead

The question that we may ask is "is our grammar LL(1) ?". A first intuition is that it seems quite logical looking at the syntax of the grammar, there are lots of keywords that remove ambiguities. A stream starts with "FROM" keyword then we know we expect an input. There are different inputs but they all start with a different keyword, if the next token after "FROM" is "FLATFILE", we know we have to apply the flatfile production rule with no ambiguities.

A more formal way to prove that our grammar is LL(1) is to build the predictive parsing table (thanks to *first()* and *follow()*). If one of the cell of the parsing table contains 2 production rules, we can say that the grammar is not LL(1). Since our grammar is quite big, it would be very time consuming to build the parsing table by hand. Fortunately, ANTLR provides a way to check that the grammar is LL(1). We just have to add 2 lines of code after creating the parser :

```
dslParser.AddErrorListener(new DiagnosticErrorListener());  
dslParser.Interpreter.PredictionMode = Antlr4.Runtime.Atn.PredictionMode.LL;
```

One of the property of LL(1) grammars is that they are not ambiguous. Since our grammar is LL(1), it is unambiguous and has no conflicts.

A.2 ANTLR Grammar

```
grammar EtlScript;  
@parser::header {#pragma warning disable 3021}  
@lexer::header {#pragma warning disable 3021}  
/*  
 * Parser Rules  
 */  
program      : stream+ | EOF;  
stream       : FROM nameInputStream=input operation* saveOperation=output?;
```

```

// Extract
input      : inStream=inputstream;
inputstream : connector
            | table
            | flatfile
            | expression
            | expressionOp
            | file
            ;
connector   : CONNECTOR connectorCode=expression AS nameOperation=expression;
table       : TABLE tableName=dbOperation;
flatfile    : regularflatfile
            | file
            ;
regularflatfile: FLATFILE
                (NAME filename = expression)?
                COLUMNS (typedexpr',')*typedexpr
                (SEPARATION sign=SEPARATIONSIGN)?
                AS nameOperation=expression;
file        : FLATFILE
            (LEFTPAR (typeornull',')*typeornull RIGTHPAR)?
            AS nameOperation=expression;

// Load
output      : OUTPUT (outStream=outputstream)?;
outputstream: connector
            | outobject
            | flatfile
            ;
outobject    : TABLE SEEKON (expression',')*expression AS nameOperation=expression;

// Transform
operation    : flatfile
            | where
            | distinct
            | groupby
            | sort
            | lookupOb
            | lookupDb
            | select
            | selectDb
            | deleteDb
            | correlate
            | etlSynchronized
            | crossapply
            ;
where        : WHERE (equality AND)*equality (AS nameOperation=expression)?;
distinct     : DISTINCT (expression',')*expression AS nameOperation=expression;

```



```

groupby      : GROUPBY key=expression AS nameOperation=expression;
sort         : SORT (expression',')*expression AS nameOperation=expression;
lookup       : INTO rTable=expression
              ON eq=assignment
              SELECT (selected',')*selected
              cacheOp=cache?
              AS nameOperation=expression;
lookupOb     : LOOKUP lookupOp=lookup;
lookupDb     : LOOKUPDB lookupOp=lookup;
select       : SELECT (OBJECT nameObject=expression)? (assignment',')*assignment AS nameOp
correlate    : CORRELATE WITH rTable=expression
              SELECT (selected',')*selected
              AS nameOperation=expression;
etlSynchronized: SYNCHRONIZED streamSync=expression AS nameOperation=expression;
crossapply   : CROSSAPPLY op=caoperation;
caoperation  : flatfile
              | selectDb
              | deleteDb
              | lookupOb
              | lookupDb
              | expressionOp
              ;
selectDb     : SELECTDB dbOp=dbOperation;
deleteDb     : DELETE dbOp=dbOperation;
dbOperation  : nameObject=expression
              inputTableOp*
              AS nameOperation=expression;
inputTableOp: where
              | include
              | orderby
              ;
include      : INCLUDE (expression',')*expression;
orderby      : ORDERBY (expression',')*expression;

// other
selected     : assignment;
assignment   : idl=IDENTIFIERS EQUALS idr=expression;
equality     : lExpr=expression sign=equalitysigns rExpr=expression;
expression   : IDENTIFIERS          #idExpr
              | STRING_ID           #stringExpr
              | NUMBER              #numberExpr
              | NULL                 #nullExpr
              | NOT NULL expr=expression #notNullExpr
              ;
expressionOp : expr=expression AS nameOperation=expression;
typedexpr    : expr=expression typ=type?;
equalitysigns: EQUALS
              | NOTEQUALS

```

```

        | GREATER
        | GREATEROREQUAL
        | SMALLER
        | SMALLEROREQUAL
        | CONTAINS
        | NOTCONTAINS
    ;
typeornull : type | ;
type       : realType=(
    BOOL
    | BYTE
    | CHAR
    | INT
    | DOUBLE
    | DATE
    | STRING)
    format=FORMAT?
    ;
cache      : NOCACHE (LEFTHPAR val=NUMBER RIGHTPAR)?
    | FULLCACHE
    ;

```

```

/*
 * Lexer Rules
 */

```

```

/*
 * Keywords
 */
FROM      : 'FROM' ;
TO        : 'TO' ;
WHERE     : 'WHERE' ;
NOT       : 'NOT' ;
NULL      : 'NULL' ;
ORDERBY   : 'ORDERBY' ;
GROUPBY   : 'GROUPBY' ;
SORT      : 'SORT' ;
IS        : 'IS' ;
IN        : 'IN' ;
WITH      : 'WITH' ;
DISTINCT  : 'DISTINCT' ;
LOOKUP    : 'LOOKUP' ;
LOOKUPDB  : 'LOOKUPDB' ;
INTO      : 'INTO' ;
NOCACHE   : 'NOCACHE' ;
FULLCACHE : 'FULLCACHE' ;
CROSSAPPLY : 'CROSS APPLY' ;
CORRELATE : 'CORRELATE' ;
FLATFILE  : 'FLATFILE' ;

```

```

NAME      : 'NAME' ;
PATH      : 'PATH' ;
COLUMNS  : 'COLUMNS' ;
SYNCHRONIZED: 'SYNCHRONIZED' ;
SEPARATION : 'SEPARATION' ;
TABLE     : 'TABLE' ;
INCLUDE   : 'INCLUDE' ;
CONNECTOR : 'CONNECTOR' ;
OBJECT    : 'OBJECT' ;
NEW       : 'NEW' ;
AS        : 'AS' ;
AND       : 'AND' ;
EQUALS    : '=' ;
NOTEQUALS : '!=' ;
GREATER   : '>' ;
GREATEROREQUAL : '>=' ;
SMALLER   : '<' ;
SMALLEROREQUAL : '<=' ;
DB        : 'DB' ;
ON        : 'ON' ;
CONTAINS  : 'CONTAINS' ;
NOTCONTAINS: 'NOT CONTAINS' ;
SELECT    : 'SELECT' ;
SELECTDB  : 'SELECTDB' ;
SAVE      : 'SAVE' ;
DELETE    : 'DELETE' ;
SEEKON    : 'SEEKON' ;
LEFTPAR   : '(' ;
RIGHTPAR  : ')' ;
OUTPUT    : 'OUTPUT' ;
ARROW     : '->' ;

```

```

BOOL      : 'boolean' ;
BYTE      : 'byte' ;
CHAR      : 'char' ;
INT       : 'num' ;
DOUBLE    : 'double' ;
DATE      : 'date' ;
STRING    : 'string' ;

```

```
/*
```

```
 * Identifiers
```

```
*/
```

```
STRING_ID : '"".*?""';
```

```
IDENTIFIERS : (LOWERCASELETTER|UPPERCASELETTER)(LOWERCASELETTER|UPPERCASELETTER|NUMBER|
```

```
SEPARATIONSIGN: ',|'|':'|';';
```

```
FORMAT : LEFTPAR'""(IDENTIFIERS|SEPARATIONSIGN)""RIGHTPAR;
```

```

WHITE_SYMBOL : [ \t\r\n] -> skip;
COMMENT      : '--' .*? ('\n'|EOF) -> skip;

LOWERCASELETTER : [a-z];
UPPERCASELETTER : [A-Z];
NUMERAL        : [0-9];
NUMBER         : NUMERAL+;
OTHERCHARACTER : '._'|'-'|'_'|LEFTPAR|RIGHTPAR ;

```

A.3 Subsystems not considered

- **13) Fact Table Loader** : this is not a process per se, it just explains the three different kind of fact tables which shows 3 different possibilities for an organization to keep records in the data warehouse. Fact tables hold the measurements of an organization. To give an example, if a sale is made, we will save information such as the content of the sale and the buyer in the “normal” table (called dimensional table) but information such as the hour of the sale, the expected delivery date and the actual delivery date will be saved in the fact table. We consider it is off topic in this context as we believe it is more a way to structure the data warehouse than a role of the ETL.
- **17) Dimension Manager System** : it is described as “the centralised authority who prepares and publishes conformed dimensions to the data warehouse community”. It consists of operations like implementing descriptive labels or adding new rows if there are some changes. This is more a way of organising things. So we have also considered it is off topic in this context.
- **18) Fact Provider System** : it is responsible for receiving conformed dimensions from the dimension managers and handling activities such as creating, managing and using fact tables. It works in parallel with system 17 and can also be considered as an organizational approach. Thus, we will not consider it.

A.4 Scratch

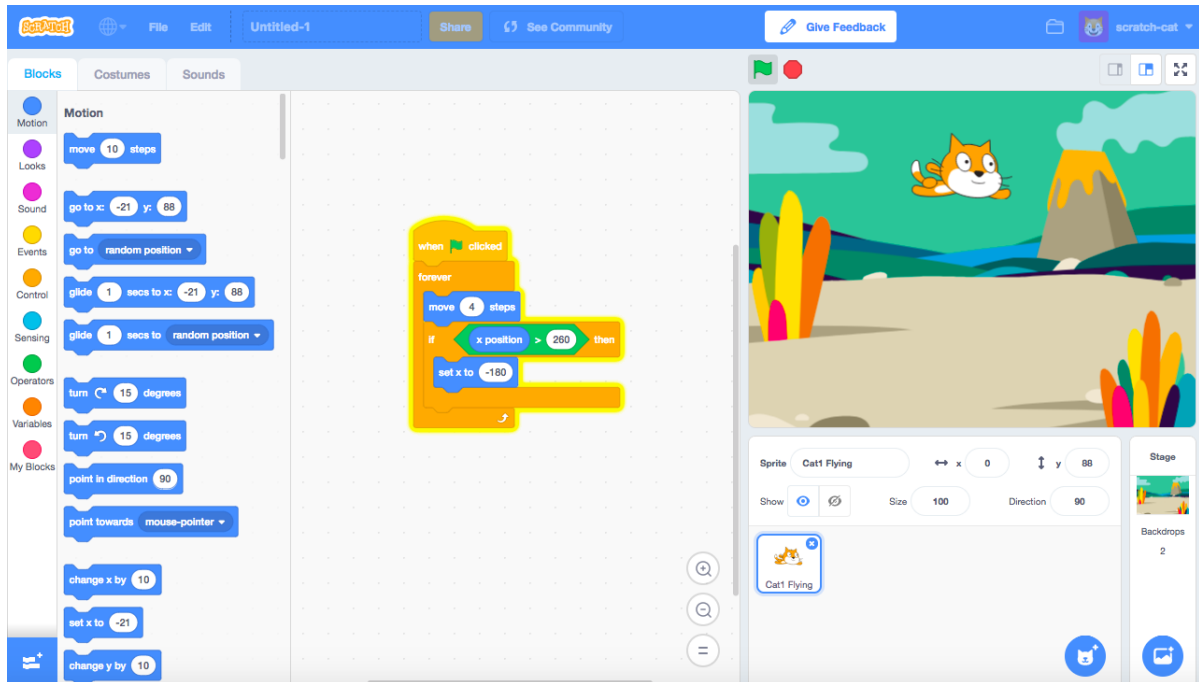


FIGURE 1 – Scratch

Scratch is an high-level block-based visual programming language. The concept of this language is to manipulate blocks instead of writing code. In our case blocks would represent operators and they could be added one after the other. We could also combine it with some managing functionalities, for example by giving some information like the number of rows and see what impact would an operator have.

A.5 EtlScript Example

A.5.1 Data Processor

```
FROM FLATFILE
  NAME file.txt
  COLUMNS FundCode, Date date, Isin, Weight boolean
  AS benchPositionFileStream

FROM benchPositionFileStream
DISTINCT FundCode AS distinctMinBenchmarkComposStream

FROM distinctMinBenchmarkComposStream
DELETE BenchmarkComposition
  WHERE FundCode = Portfolio.InternalCode AND Date>=Date
AS deleteNewerExistingBenchMarkCompoStream

FROM benchPositionFileStream
SYNCHRONIZED deleteNewerExistingBenchMarkCompoStream AS waitCompositionDeletion
```

```

DISTINCT FundCode, Date AS distinctComposition
LOOKUPDB
    INTO SubFund
    ON FundCode = InternalCode
    SELECT BenchMark, SubFund FULLCACHE
AS getRelatedSubFund
SELECT OBJECT BenchmarkComposition
    PortfolioId = SubFund.Id,
    Date = BenchMark.Date
AS CreateBenchCompo
OUTPUT TABLE SEEKON PortfolioId, Date AS benchCompositionStream

FROM TABLE SecurityInstrument AS securityStream

FROM benchPositionFileStream
LOOKUP
    INTO securityStream
    ON Isin = InternalCode
    SELECT BenchMark, Security
AS getRelatedSecurity2
CORRELATE WITH benchCompositionStream
    SELECT BenchMark, Security, Composition
AS getRelatedBenchComposition
SELECT OBJECT BenchmarkSecurityPosition
    BenchmarkCompositionId = Composition.Id,
    SecurityId = NOT NULL Security.Id,
    Weight = NOT NULL BenchMark.Weight.Value
AS CreatePosition
OUTPUT TABLE SEEKON SecurityId,BenchmarkCompositionId
AS benchPositionsStream

```

Once gone through the transpiler, it gives the following ETL.NET code :

```

var benchPositionFileStream = FileStream
    .Where("benchPositionFileStream file name pattern", fs=> fs.Name == "file.txt")
    .CrossApplyTextFile("benchPositionFileStream",
        FlatFileDefinition.Create(i => new {
            FundCode = i.ToColumn("FundCode"),
            Date = i.ToDateColumn("Date", "yyyy-MM-dd"),
            Isin = i.ToColumn("Isin"),
            Weight = i.ToBooleanColumn("Weight", "true", "false")
        }).IsColumnSeparated(',', ''))
    .SetForCorrelation("benchPositionFileStream: Set correlation key")
;
var distinctMinBenchmarkComposStream = benchPositionFileStream
    .Distinct("distinctMinBenchmarkComposStream", l => l.FundCode)
;
var deleteNewerExistingBenchMarkCompoStream = distinctMinBenchmarkComposStream

```

```

.EfCoreDelete("deleteNewerExistingBenchMarkCompoStream", o => o
    .Set<FundProcess.Pms.DataAccess.Schemas.Benchmarking.BenchmarkComposition>()
    .Where((l, r) => l.FundCode == r.Portfolio.InternalCode && l.Date >= r.Date))
;
var benchCompositionStream = benchPositionFileStream
    .WaitWhenDone("waitCompositionDeletion", deleteNewerExistingBenchMarkCompoStream)
    .Distinct("distinctComposition", l => new {l.FundCode , l.Date})
    .EfCoreLookup("getRelatedSubFund", o=>o
        .Set<SubFund>()
        .On(l => l.FundCode, l => l.InternalCode)
        .Select((l,r) => new {
            BenchMark = l,
            SubFund = r
        })
        .CacheFullDataset())
    .Select("CreateBenchCompo", l=> new BenchmarkComposition {
        PortfolioId = l.SubFund.Id,
        Date = l.BenchMark.Date
    })
    .EfCoreSave("benchCompositionStream", o=> o
        .SeekOn(l => new {l.PortfolioId , l.Date})))
;
var securityStream = ProcessContextStream
    .EfCoreSelect("securityStream", (o, r) => o
        .Set<FundProcess.Pms.DataAccess.Schemas.Pms.SecurityInstrument>()
    );
var benchPositionsStream = benchPositionFileStream
    .Lookup("getRelatedSecurity2", securityStream,
        l => l.Isin, l => l.InternalCode, (l,r) => new {
            BenchMark = l,
            Security = r
        })
    .CorrelateToSingle("getRelatedBenchComposition", benchCompositionStream,
        (l,r) => new {
            BenchMark = l.BenchMark,
            Security = l.Security,
            Composition = r
        })
    .Select("CreatePosition", l=> new BenchmarkSecurityPosition {
        BenchmarkCompositionId = l.Composition.Id,
        SecurityId = (l.Security != null) ? l.Security.Id
            : throw new Exception("l.Security value not found"),
        Weight = (l.BenchMark.Weight != null) ? l.BenchMark.Weight.Value
            : throw new Exception("l.BenchMark.Weight value not found")
    })
    .EfCoreSave("benchPositionsStream", o=> o
        .SeekOn(l => new {l.SecurityId , l.BenchmarkCompositionId})))
;

```

```

return FileStream.WaitWhenDone("wait till everything is saved",
    benchPositionFileStream, distinctMinBenchmarkComposStream,
    deleteNewerExistingBenchMarkCompoStream, benchCompositionStream,
    securityStream, benchPositionsStream);

```

A.5.2 File Retriever

Example of EtlScript code in File Retriever mode :

```

FROM TABLE SecurityInstrument
    INCLUDE Country, DataProviderCodes, Currency,
    Classifications.Classification, Classifications.ClassificationType
AS dbStream

FROM dbStream
WHERE Currency.Code = "EUR" AS selectEuros
SORT Name AS sortByName
SELECT
    SecurityCode = InternalCode,
    SecurityName = Name,
    CcyIso = Currency.IsoCode,
    ISIN = Isin,
    CountryIso2 = Country.IsoCode2,
    EodCode = DataProviderCodes.First().Code,
    BbgTicker = DataProviderCodes.First().Code,
    GicsCode = Classifications.First().Classification.Code
AS CreateExtractItem
OUTPUT FLATFILE
    NAME SecurityUniverse.txt
    COLUMNS SecurityCode, SecurityName, CcyIso, ISIN, CountryIso2,
        EodCode, BbgTicker, GicsCode
AS ExportToCsv

```

Once gone through the transpiler, it gives the following ETL.NET code :

```

var dbStream = ProcessContextStream
    .EfCoreSelect("dbStream", (o, r) => o
        .Set<FundProcess.Pms.DataAccess.Schemas.Pms.SecurityInstrument>()
            .Include(l => (l as RegularSecurity).Country)
            .Include(l => l.DataProviderCodes)
            .Include(l => l.Currency)
            .Include(l => l.Classifications).ThenInclude(l => l.Classification)
            .Include(l => l.Classifications).ThenInclude(l => l.ClassificationType)
    );
var ExportToCsv = dbStream
    .Where("selectEuros", l => l.Currency.Code == "EUR")
    .Sort("sortByName", l => l.Name)
    .Select("CreateExtractItem", l=> new {

```



```

        SecurityCode = l.InternalCode,
        SecurityName = l.Name,
        CcyIso = l.Currency.IsoCode,
        ISIN = l.Isin,
        CountryIso2 = (l as RegularSecurity).Country.IsoCode2,
        EodCode = l.DataProviderCodes.First().Code,
        BbgTicker = l.DataProviderCodes.First().Code,
        GicsCode = l.Classifications.First().Classification.Code
    })
    .ToFileValue("ExportToCsv", "SecurityUniverse.txt",
        FlatFileDefinition.Create(i => new {
            SecurityCode = i.ToColumn("SecurityCode"),
            SecurityName = i.ToColumn("SecurityName"),
            CcyIso = i.ToColumn("CcyIso"),
            ISIN = i.ToColumn("ISIN"),
            CountryIso2 = i.ToColumn("CountryIso2"),
            EodCode = i.ToColumn("EodCode"),
            BbgTicker = i.ToColumn("BbgTicker"),
            GicsCode = i.ToColumn("GicsCode")
        }).IsColumnSeparated(',','))
;
return ExportToCsv;

```

A.5.3 File Processor

Example of EtlScript code in File Processor mode :

```

FROM FLATFILE (num,num,num,date,num,boolean,num,num ) AS fileDefinition
DISTINCT PortfolioId AS takePortfolioOnce
SELECT
    BoolValue = Catchup,
    DateTimeValue = InitialDate,
    DoubleValue = PreferredReturn,
    Field = SplitLP,
    IntValue = CapitalInvested,
    PortfolioId = PortfolioId,
    StringValue = GPCatchup,
    Type = "JWA-Test",
    Value = Proceeds
AS CreatePortfolioExtensionFields
OUTPUT

```

Once gone through the transpiler, if the correct input file is given, the following ETL.NET code is created :

```

var CreatePortfolioExtensionFields = FileStream
    .CrossApplyTextFile("fileDefinition", FlatFileDefinition.Create(i => new {
        PortfolioId = i.ToNumberColumn<int?>("PortfolioId","."),

```

```

        CapitalInvested = i.ToNumberColumn<int?>(" CapitalInvested","."),
        Proceeds = i.ToNumberColumn<int?>(" Proceeds","."),
        InitialDate = i.ToDateColumn(" InitialDate","yyyy-MM-dd"),
        PreferredReturn = i.ToNumberColumn<int?>(" PreferredReturn","."),
        Catchup = i.ToBooleanColumn(" Catchup","true","false"),
        GPCatchup = i.ToNumberColumn<int?>(" GPCatchup","."),
        SplitLP = i.ToNumberColumn<int?>(" SplitLP",".")
    }).IsColumnSeparated(',')
    .SetForCorrelation("fileDefinition: Set correlation key")
    .Distinct("takePortfolioOnce", l => l.PortfolioId)
    .Select("CreatePortfolioExtensionFields", l=> new {
        BoolValue = l.Catchup,
        DateTimeValue = l.InitialDate,
        DoubleValue = l.PreferredReturn,
        Field = l.SplitLP,
        IntValue = l.CapitalInvested,
        PortfolioId = l.PortfolioId,
        StringValue = l.GPCatchup,
        Type = "JWA-Test",
        Value = l.Proceeds
    })
    .EfCoreSave("no output name")
;
return FileStream.WaitWhenDone("wait till everything is saved",
    CreatePortfolioExtensionFields);

```

Bibliography

- [1] Kimball, R., Ross, M. (2013). The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, Third Edition. Indianapolis, IN: John Wiley & Sons, Inc.
- [2] Casters, M., Bouman, R. Van Dongen, J. (2010). Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration. Indianapolis, IN: Wiley Publishing, Inc.
- [3] Keboola (2022). A guide to ETL vs ELT data pipelines.
- [4] Bill Inmon (2008). A brief history of ETL. [online] www.techtarget.com. Available at: <https://www.techtarget.com/searchdatamanagement/news/2240033946/A-brief-history-of-ETL>
- [5] Diouf, P. S., Boly, A., Ndiaye, S. (2017). Performance of the ETL processes in terms of volume and velocity in the cloud: state of the art. Cheikh Anta Diop University.
- [6] Oshi Varma (2020). 9 Best Big Data ETL Tools. [online] www.hevodata.com. Available at: <https://hevodata.com/learn/best-big-data-etl-tools/> [Accessed 30 May 2022]
- [7] FundProcess (2022) [online] Available at: <https://www.fundprocess.lu/> [Accessed 5 june 2022]
- [8] Paillave (2022). ETL.NET. [online] paillave.github.io. Available at: <https://paillave.github.io/Etl.Net/> [Accessed 23 May 2022]
- [9] Paillave (2021). Why ETL.NET ? [online] paillave.github.io. Available at: <https://paillave.github.io/Etl.Net/blog/2021/08/01/WhyETLNET> [Accessed 6 June 2022]
- [10] Matillion (2022). WHAT IS ETL? The Ultimate Guide. [online] www.matillion.com. Available at: <https://www.matillion.com/what-is-etl-the-ultimate-guide/> [Accessed 23 May 2022]
- [11] Martin, C.R. (2020). Brush up your COBOL: Why is a 60 year old language suddenly in demand? [online] stackoverflow.blog. Available at: <https://stackoverflow.blog/2020/04/20/brush-up-your-cobol-why-is-a-60-year-old-language-suddenly-in-demand/> [Accessed 23 May 2022]