LIÈGE université
Library

https://lib.uliege.be

matheO

https://matheo.uliege.be

Master thesis : On the Design and Implementation of an ETL Configuration DSL for Non-programming Experts

Auteur : Duchateau, Jakub
Promoteur(s) : Debruyne, Christophe; 17378
Faculté : Faculté des Sciences appliquées
Diplôme : Master en sciences informatiques, à finalité spécialisée en "computer systems security"
Année académique : 2022-2023
URI/URL : http://hdl.handle.net/2268.2/17649

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

School of Engineering and Computer Science

On the Design and Implementation of an ETL Configuration DSL for Non-programming Experts

Master's thesis completed in order to obtain the degree of
Master of Science in Computer Science by

Jakub Duchateau

Supervisor **Christophe Debruyne**
Montefiore Institute
University of Liège

Industrial Supervisor **Frédéric Duquenne**
FundProcess
CoFunder and CEO

Academic year 2022-2023
9 June 2023

Abstract

Extract-transform-load (ETL) tools are used in the business to ingest external data into their operational database. The design of an ETL process can be achieved through either code and libraries or a graphical tool featuring a graph of operations. However, the former is only accessible to programmers, while the latter lacks extensibility. This thesis proposes an approach to ETL configuration, based on a projectional domain-specific language (DSL), implemented with JetBrains MPS, and targeting ETL.NET. The ETL process is designed using a language that represents data as tables manipulated through sequences of operations. A prototype implementation is evaluated with a small user study, which shows that the DSL is accessible to non-programmers, and programmers prefer it over a tool based on graphs.

Résumé

Les outils Extract-Transform-Load (ETL) sont utilisés dans les entreprises pour importer des données externes dans leur base de données opérationnelle. La conception de processus ETL, se fait principalement avec du code textuel et des bibliothèques logicielles ou avec des interfaces graphiques présentant un graphe d'opérations. La première convient bien aux programmeurs tandis que la seconde est plus difficile à étendre en fonctionnalité. Nous proposons ici une méthode intermédiaire basée sur un langage de programmation spécifique (DSL) projectionnel, implémenté avec JetBrains MPS, et intégré à ETL.NET. Le processus ETL est exprimé dans un langage qui modélise les données sous forme de tableaux qui sont transformés avec des séquences d'opérations. Le prototype est évalué avec une petite étude utilisateur, qui montre que le DSL est accessible aux non-programmeurs, et que les programmeurs le préfèrent à une interface basée sur des graphes.

ii

Acknowledgements

I would like to thank my supervisor, Christophe Debruyne, for his guidance, support, and meticulous proofreading throughout this thesis. I would also like to extend my heartfelt thanks to the team at FundProcess, namely Frédéric Duquenne, Stéphane Royer, and Moyra Bonjean, who not only facilitated the selection of this thesis subject but also dedicated their time to testing various stages of the prototype and providing valuable feedback. I extend my sincerest gratitude to all the testers of the prototype for generously dedicating their time and providing valuable feedback without expecting anything in return. Lastly, I would like to extend my heartfelt thanks to my family and friends for their support and encouragement throughout this journey.

Contents

1 Introduction **1**
1.1 Business Problem . . . 1
1.2 Research Question and Objectives . . . 2

2 Related works **4**

3 Language design **7**
3.1 From Context to Guidelines . . . 7
3.2 Methodology . . . 8
3.2.1 Our Design Activities . . . 9
3.2.2 Case study . . . 9
3.3 ETLang Type System . . . 11
3.3.1 *Table* Type . . . 11
3.3.2 Types Descriptions . . . 12
3.4 ETLang Concepts . . . 13
3.4.1 Macro . . . 13
3.4.2 Operator . . . 14
3.4.3 Transform Block . . . 15
3.4.4 Statements . . . 15
3.4.5 Expressions . . . 16
3.4.6 Structure definitions . . . 16
3.5 Operators to Build ETLs . . . 16
3.6 Expressions to Build ETLs . . . 19

4 ETLang Implementation **22**
4.1 Choosing an Implementation Framework . . . 22
4.1.1 Utility Tools . . . 22
4.1.2 Language Workbenches . . . 23
4.1.3 Metalanguages in .NET Ecosystem . . . 24
4.1.4 Choosen Implementation Method . . . 25

iv

4.2 Language Design Adaptations . . . 25
4.2.1 Transform Blocks and Operators . . . 26
4.3 Implementing ETLang . . . 26
4.3.1 Concepts . . . 27
4.3.2 Editors . . . 30
4.3.3 Type System . . . 31
4.3.4 Scope System . . . 32
4.3.5 Migration . . . 32
4.3.6 Building the IDE . . . 33
4.3.7 Code generation . . . 33
4.4 Web Version of the Editor . . . 34

5 Evaluation **35**
5.1 Evaluation Quality Model . . . 35
5.2 User Study of Ease of Use . . . 37
5.2.1 Evaluation Methods . . . 37
5.2.2 User Evaluation Findings . . . 39
5.3 Evaluation of Functional Adequacy . . . 43
5.4 Discussion . . . 45

6 Conclusion and Future Works **47**
6.1 Conclusion . . . 47
6.2 Future Works . . . 49
6.2.1 Improving the Prototype . . . 49
6.2.2 Other Research Directions . . . 51

Glossary and Acronyms **53**

Bibliography **55**

A Questionnaire **58**

B Getting started **59**
B.1 Getting started tutorial . . . 59
B.1.1 Step 1: Create a new solution . . . 59
B.1.2 Step 2: Declare your file structure . . . 60
B.1.3 Step 3: Read the file . . . 61
B.1.4 Step 4: Modify the table . . . 62

B.1.5 Step 5: Save the table 63
B.1.6 Bonus Step 6: Defining symbols 65

vi

1 Introduction

The ability to empower individuals without programming skills to perform programming tasks is a significant challenge in the field of human-computer interaction. This master's thesis aims to address the question of how non-programmers can be enabled to undertake the ETL (Extract Transform Load) design task effectively. In order to explore potential solutions to this challenge, it is essential to establish the context in which this question arises. Subsequently, the objectives and research questions of this thesis will be presented.

1.1 Business Problem

FundProcess is a company that develops software for the financial industry. They developed an integrated approach for the Asset Management Industry, and within their toolkit, they have an ETL tool.

ETL, an umbrella term encompassing a variety of tools, is instrumental in the movement and transformation of data. These tools differ in multiple aspects, such as data loading capabilities, performance, configuration options (code-based or not), extensibility, range of operations, supported data formats, and modularity of the ETL architecture.

The need for an ETL tool is to be able to import data from external partners into their database in a standardized format. Financial institutions provide data in diverse formats, ranging from simple CSV files to complex PDF files.

To accommodate the diverse range of data sources, FundProcess has developed ETL.NET, a powerful tool that allows developers to define their import processes using .NET. The library, ETL.NET, was created to be able to handle the variety of data sources and to be able to write the import process in a programming language that integrates into the usual developer workflow, instead of visual approaches taken by other ETL tools. Its other main feature is its modular architecture, which allows one to choose only what is needed and to easily add new modules and ETL operators, enabling large extensibility.

However, due to its developer-centric nature, ETL.NET requires programming skills and a solid understanding of computer science concepts. Consequently, non-programming experts face challenges when attempting to configure and execute ETL processes using ETL.NET, even though they possess valuable domain knowledge about the data being processed. Currently, the responsibility of designing ETL processes lies with programmers, who are expected to possess both programming expertise and domain knowledge. Although complex ETL processes may necessitate specialized skills, FundProcess desires to delegate the configuration of simpler ETL processes to non-programming experts. This delegation would empower them to configure and execute ETL processes without the need for long programming training to acquire the knowledge.

To enable non-programming experts to configure and run ETL processes without programming skills, a new DSL (Domain Specific Language) can be created. Other viable approaches include using visual ETL tools that provide drag-and-drop or forms-based interfaces for configuring ETL processes. However, these approaches may not be as flexible or customizable as a DSL, and may not be suitable for more complex ETL processes. This thesis aims to create such a DSL to facilitate the ETL process configuration for this new user group.

Next, we will outline the objectives and research questions that the thesis will seek to answer.

1.2 Research Question and Objectives

The primary objective of this master's thesis is to address the following research question:

What DSL (Domain Specific Language) would enable non-programming experts to design and execute ETL (Extract Transform Load) processes efficiently and effectively?

Therefore, our overarching objective of this master's thesis is to design and implement a DSL tailored to the needs of non-programming experts to design ETL processes in an approachable way. By doing so, we aim to reduce the time users spend importing data into their database. To achieve this primary objective, we propose to decompose it into these sub-objectives:

1. Facilitate the learning curve of programming an ETL process.
2. Develop a system that is accessible to a wider range of users with limited programming expertise.

3. Ultimately, reduce the time spent by users on importing data into their database.
4. Propose a solution integrable with the existing ETL.NET library.

To reach these objectives, the thesis will be structured as follows:

The next chapter explore existing interfaces and proposed solutions for configuring ETL processes with a focus on proposition targeting non-programmers. This chapter will provide an overview of *related work* and highlight approaches addressing similar problems.

Then, Chapter 3 will outline the methodology employed to create our DSL, inspired by *human-centered design*, as an attempt to reach our second objective of accessibility. Additionally, the chapter will present the proposed DSL modelisation. And already from the design phase, we will try to reach our fourth objective of integration with the existing ETL.NET library, with an analysis of a sample an ETL scripts written with it.

Chapter 4 will present the *implementation* of the DSL and the IDE (Integrated Development Environment) to support it. This chapter emphasizes the evaluation of the language and its implementation as distinct entities, to enable a distinct analysis of the language's design and its implementation.

Subsequently, Chapter 5, will introduce the quality model adopted for evaluation purposes. The chapter will present the evaluation methodology employed to assess the DSL's design and prototype. Findings from the evaluation will be discussed in relation to each objective, including the learning curve (third objective) and efficiency (time reduction, fourth objective).

Finally, the Chapter 6 will conclude the thesis with a reflection on the whole process and, more personal notes and discuss future work, concerning this thesis and other potential research directions.

Related works

2

Previous attempts have been made to propose simple interfaces for business users to design ETL processes. For example, FundProcess developed ETL.NET because they found other tools to be insufficiently developer-oriented. In this chapter, we will examine other interfaces that have been proposed for non-developers to design and model ETL processes. Additionally, we will explore data preparation tools that offer interesting interfaces to facilitate parts of the ETL process, such as the extraction or transformation steps.

EtlScript

Trying to solve the same problem, Julien Wauthoz presents EtlScript [Wau22], an external DSL enabling to design of ETL processes. It is a textual language with a syntax inspired by COBOL and SQL. With this language, users manipulate streams of data and can apply operations to them. The set of operations in EtlScript is determined by the parser, which parses each operation differently based on a combination of keywords.

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

Listing 2.1: Example of EtlScript

However, EtlScript has two shortcomings: it is hard to extend because it is built with ANTLR, and it is not possible to add new operations without modifying the parser, because of the keyword-based syntax. It has no IDE; the user has to modify the raw text file.

We could imagine a language with a similar syntax, but with a more modular design, allowing to add new operations without modifying the parser, and with an IDE to ease the edition of the script. We will evaluate this idea in Chapter 4.

BPMN4ETL

El Akkaoui presents a conceptual and graphical DSL to model ETL processes based on BPMN (Business process model and notation) [El 14; El +12]. As we explicitly are excluding purely graphical languages, the interesting part was not the prototype itself but the way it was evaluated and the proposed quality framework. We will reuse part of the quality model while building our own in Chapter 5.

SSIS

SSIS (SQL Server Integration Services) has an graphical intrface with SSIS Designer [@Gu+23]. It also supports programming the process with `C#` via a specific API. The graphical editor has the data flow component, which features an ad-hoc graphical language with a drag-and-drop interface to configure the ETL process. SSIS was previously evaluated by FundProcess and they rejected it because it was difficult to adapt to their use case, due to the complex extension system, and it was not well integrable into their development workflow.

Arktos

Even if it is not the most recent work, it is still interesting to mention it because it features up to 3 interchangeable interfaces: a graphical language, a declarative textual language, and XML-based descriptive language. They provide two textual languages because they aime the XML-based language for reading as it is more verbose and the SQL-like for writing as it is more compact. We have taken inspiration from this idea of an alternative language for reading and another for writing by providing alternative syntax for some constructs of the language.

Enso

ETL.NET take advantage of being a `C#` library, to have a strong integration with the `C#` general purpose language. This allows users to switch between standard `C#` data structures and ETL.NET streams as needed within their ETL processes. Additionally, ETL.NET provides a visualization of the ETL data flow.

Enso is a dual representation general-purpose programming language. One representation is a functional-style textual language, the other a visual language. It may be interesting for future work to explore the possibility of a dual representation for ETL.NET or instead of creating a new language specialized language, provide specialized editor support for ETL.NET within the .NET ecosystem.

Wrangler

Within an ETL, there are the transformation and loading parts where you combine multiple data source, but one part of the ETL process is to extract the data from messy sources. Data preparation is then used to clean the data and make it usable for the ETL process.

During our research, we discovered "Wrangler, an interactive tool for data cleaning and transformation"(adapted from their website). It was proposed by Kandel et al. [Kan+11] and was considered the state-of-the-art tool in data cleaning. Now it is integrated into the commercial tool Trifacta and is still considered one of the most advanced tools in 2020 [HN20].

At its core proposition, the tool provides a large tabular pre-visualization of a sample of the data, which can be used to propose a modification of the data. From the manually edited data, the tool proposes guesses of how to formalize the action, and the user continues to edit the data until the tool finds the intended action. The sequence of action script of a sequence of transform operations. The interaction model is menu-driven and is operated in a Programming-by-Demonstration or Programming-by-Example style, since before selecting the guessed operation to apply, a demonstration of the operation is previewed. And to generate the operations, the user makes example modifications to the data.

It could be interesting to see in future work to propose to the user a sample preview of the data and see if programming by examples would also be doable in the ETL.NET context.

Closing word of Related works With this small overview of what is done in terms of human-computer interfaces in the field of ETL, we will be able to move on to the design of our proposed interface for ETL.NET. We shall not forget to avoid the problems of architectural extensibility of the previous proposals and although a graphical interface is potentially a long-term goal of FundProcess, we will focus on a text-like interface as planned.

Language design

3

A language is produced in a context; therefore, it is important to understand the context in which it will evolve. In this chapter, we will discuss the context of the language, and the methodology used to design it. After that, we pursue the general ideas that have guided the conception of the language. And finally, we will present the language design, with its type system, concepts, and aimed user experience.

3.1 From Context to Guidelines

The language is designed to be used by people with no prior programming experience. As observed by Green [Gre89], the alignment of the language with the domain greatly influences the ability to effectively express facts in that domain. In the case of ETL.NET, the language concepts and semantics are mapped to C# and suffer from the duality of its syntax, which introduces a gap between the conceptual view of the ETL process and its encoding in C#. According to Green, this duality entails an arduous cognitive dimension [Gre89] for the ETL designer.

We can take advantage of our users having an extensive experience with spreadsheets, we aim to make the language as familiar as possible in terms of naming conventions. By relying on standard language, we can facilitate the learning process and reduce the learning curve for users [El 14; Gre89].

Also, we will try not to introduce new concepts but rather reuse existing ones. And our ETL will work with CSV as a source of data and a relational database as the destination. A common property of both is that they can be modeled as tables. We will see whether we can exploit this representation in our language.

On a technical side, the guides were the possibility to extend the language with new operators and to be implemented with ETL.NET. Extensibility is already one of the main features of ETL.NET, and we want to keep it. We have a sample of ETL processes written with ETL.NET, and frequently these processes use custom operators specific to FundProcess. Also, users

do not define these custom operators, it is the developer of the language that will need to register.

It should be a DSL, generating `C#` with ETL.NET, for users who have significant experience with spreadsheets, and having an extensible operator set.

3.2 Methodology

Initially, established methodologies are examined, followed by an exploration of the specific process that led to the creation of our language.

Designing a DSL begins by developing a solid comprehension of the underlying domain concepts. The next step is to identify the concepts that are relevant to the domain and to define the language concepts that will be used to express them. Two approaches compiled from Barišić [Bar17] and Kleppe [Kle08] :

top-down approach with an extensive evaluation of domain, "discarding any existing implementation and focusing only on the complete description and categorization of the class of problems from which its users will use our new DSL" [Bar17, p. 14]. The top-down approach generally leads to *horizontal DSL*, and also called *business oriented DSL*.

bottom-up approach starts with the analysis of the existing API, namely ETL.NET, and identifies frequent patterns to infer a language design. Also called *technical oriented DSL*, it generally leads to *vertical DSL*.

In practice, Barišić [Bar17] says that DSLs are usually built with a mix of both approaches.

An iterative approach, influenced by the field of Human-Centered Design, has been proposed by Borum et al. [BNS21]. This approach proves particularly relevant in our study, considering the limited availability of domain experts and users, which poses challenges for conducting a large-scale user study. In our specific context, the objective is to extend the use of the ETL activity to a novel user group, namely non-programmers, who are not currently utilizing the ETL.NET's `C#` library.

The proposed approaches split the design process into two phases. First, an *exploration* phase, where the designer explores low-validity prototypes, does interviews with domain experts. The second phase is *design validation* phase. It is a phase where the designer will use a more mature prototype to validate the design with more users.

3.2.1 Our Design Activities

Our design activities involved several key steps:

1 Domain Analysis An analysis of the domain was conducted to gain a comprehensive understanding of the relevant concepts, requirements, and how the ETL.NET library is used.

2 Language Exploration Low-fidelity prototypes were developed to explore different individual design ideas and validate their relevance alone.

3 Low-fidelity Design Validation The complete low-fidelity design was validated to assess the coherence and orthogonality of its ideas, as well as their alignment with the overall goals of the project.

4 Prototype Validation The prototype was further evaluated in the context of its intended use, involving different groups of users. More details about the evaluation can be found in Chapter 5.

To mitigate potential risks and ensure the success of our design activities, we drew upon previous work and experiences documented in the literature. By adopting an iterative approach and combining top-down and bottom-up perspectives, we aimed to create a well-informed and user-centered DSL that effectively addressed the needs of non-programming experts in the ETL domain.

3.2.2 Case study

In this part, we describe a case study on our experience using a mix of top-down, bottom-up, and two phases approaches to design our language. It would be misleading to say that we followed a strict methodology from the start. Instead, we found that given what we were doing, we identified some risks, and we searched previous work to mitigate them.

1 Domain analysis We started by analyzing the domain, with our theoretical and practical knowledge of ETLs. And to get acquainted with ETL.NET, we started to implement some ETL processes with it. We also did some interviews with Stéphane Royer, the creator and maintainer of ETL.NET, with FundProcess, on how they use the tool, and what they would like to see improved. We also analyze a sample of ETL processes written with ETL.NET to identify common patterns, part of the result will be exposed

in Table 3.2. Also, this was not a purely linear process, we get back to this activity when we needed some more information.

2 Languages exploration We started to explore the language by creating some very low-fidelity prototypes, analogous to paper prototypes. Initially, we started with the syntax proposed with EtlScript by Wauthoz [Wau22], and with Moyra Bonjean tested variations keeping the one that was the most readable and appreciable.

For example, we experimented with different naming for operations and different representations.

One risk was to have a language that was too similar to EtlScript on the abstract concepts. To mitigate this risk, we have done a bottom-up analysis to try to identify frequently used patterns in ETL.NET processes, more on that while presenting the design and the set of operators Section 3.5. Additionally, we used also a top-down approach starting from the opportunities of the domain and input from the domain expert. For example, instead of just naming operations following spreadsheet naming, we tried tables as well, it was one of the demands, *what if we could just add remove columns, bring information from another table and just add these columns on the side of the table*. To be able to just add or remove columns, we need a strong data model than streams of any type, the table idea comes from this need.

3 Low-fidelity design validation After gathering a series of low-fidelity for different parts of the language, we conducted a design validation process to assess their effectiveness put together. This involved utilizing the paper prototypes to perform various tasks, both individually and with the participation of non-programmers. In these collaborative sessions, we simulated the editing of code using the envisioned syntax, guided by the users' instructions. The objective was to evaluate the expressiveness, consistency, and orthogonality of the core concepts embedded within the prototype. Based on the insights gained from this paper validation activity, we refined and further explored the design.

4 Prototype validation As the iterative design process progressed and the low-fidelity prototypes stabilized, while also considering the reduction in time, we proceeded to develop a minimum viable prototype. This allowed us to comprehensively evaluate the aspects that were not fully explored in the low-fidelity prototypes, including the actual editing experience and code generation. You will read more about the making of this prototype in Chapter 4. Then with this minimal working implementation, we get as many

participants as feasible to comment on it.[1] The evaluation of these user comments will be discussed in detail in Chapter 5.

This section presented the case study of our design process toward an easier-to-use language for ETL processes. The next sections present the results of this process: the language design. Then the following chapters present the implementation of the prototype and its evaluation.

3.3 ETLang Type System

In the context of ETLs, it is essential to define the data type that will be manipulated, maybe even before defining how it will be manipulated. This section presents the type of system that dictates what the data looks like. In the following section, we will present the concepts of the AST (Abstract Syntax Tree).

Our ETL system concentrates on a subset of ETL.NET capabilities. Namely, we focus on extracting data from CSV files, which ETL.NET support as `TextFile` and on the loading side the destination is a relational database.

Given that ETL.NET operates on sequences of `C#` entities and our objective is to generate `C#` content, we will employ types that possess similar characteristics or can be converted into `C#` types in order to ensure compatibility with their functionality.

3.3.1 Table Type

During our research, we found that users prefer to think of tables as a whole, and not as a stream of rows. Table 3.1 presents a representation of our *table* type.

Table 3.1: Table type representation

Table	<table name>		
	<column name 1>	...	<column name N>
	<value type 1>	...	<value type N>

[1] As note Borum et al. "it is unlikely that someone ends up with the possibility of performing too many usability tests when creating ADSLs"[BNS21].

Tables represent well, the supported `TextFile` from ETL.NET. It models the ordered columns with headers we frequently find in CSV files. Tables also imply ordered lines. And each table has a name. This is not the case for CSV tables, but it is the case for the destination of the ETL process, a relational database. Also, names are mandatory but note enforced unique, as in the language we have other means of uniquely identifying a table other than by its name, for example, the operator that produced it. The name of the table is good for communication with the user via messages or when loading the data in a database, to identify the table or entity it refers to. Column names are also mandatory, and their name is enforced uniquely within a table, as their name is used as a way to reference it from a table since we do not want to have to deal with ambiguities where we would need to specify the index in addition.

Column type can be of any simpler type, not a nested table. We considered the case where we could have a table as a column type, this would have a clean visualization but neither the source nor the destination data model supports it. It may be useful during the transformation phase of the ETL process when merging tables side by side or with data that can naturally be grouped together, such as address fields. Grouping data could be supported as a second-class citizen with tuples.

Next, are presented the types part of the core language, and potential extensions.

3.3.2 Types Descriptions

Primitive types

Boolean maps to `C#` bool.

Integer maps to `C#` integers. Implicitly convert to Number.

Number represents decimal numbers, and maps to `C#` double.

String is another first-class citizen type. We do not have specific charters except as strings of length 1.

Datetime are represented as `C# DateTime` type. And are placed in the core as dates and times are frequently used in the financial world.

Optional is a type that can wrap any other primitive or composite type, adding to the wrapped type an explicit empty value. It then can be used in the same way as the wrapped type but with the possibility of being empty.

Composite types

List is a list of values of the same type. They can contain any of the primitive types. They should behave like C# `List<T>` where T is the type of element in the list. They are not implemented in the core language and need an extension.

Abstract types

Table A table has a (non-unique) name and a predefined list of columns. Columns consist of a name and a primitive or composite type.

Potential extensions Implementations could imagine extending the language with more types. *Record* or *Tuple*, to store a composite type of different values in a column, could be an interesting extension to Composite types.

3.4 ETLang Concepts

In this section, we present the concepts that should be present in the language and that will be used to define the AST. We will not consider technical design constraints in this section, as the implementation framework influences the design of the DSL. However, we will discuss constraints on how to generate the C# code with ETL.NET, as they are part of the initial constraints.

Previously, we define the types that will be used in the language. With these types, we can now present the concepts of the language.

The list of concepts is as follows:

3.4.1 Macro

A Macro is a concept representing the processing done to a NAVPack to get it imported into the database. Given the NAVPAck, it will define how to interpret it, transform it and store it in the destination database. In other words, it is the whole ETL process.

Top symbol The Macro provide scope for global symbols, appearing as they are bound. A symbol is available in the scope after it is bound.

The term, Macro, comes from the naming of this concept by FundProcess, and also it is the name of small scripts in spreadsheets software programs. The Microsoft Office Excel documentation defines it as "A macro is an action or a set of actions that you can run as many times as you want." Microsoft [@Mic].

3.4.2 Operator

Operators are a concept that represents a family of operations on a table. For example, restructuring the table is a kind of operation parametrized by how the new structure is computed. They are equivalent to the ETL.NET operators and will transform the table parametrized by expressions.

To compose operators, one can simply list them sequentially. They will execute their effect in order, on the table provided by the previous operator if any. *An operator exposes its result as a table*, meaning the next operator arguments can access the table exposed by the previous operator.

In the following snippet if `Operator1` exposes the table `Table1` and `Operator2` exposes the table `Table2`, then `Operator2` arguments has in its scope `Table1` and `Operator3` arguments can access `Table2` but not `Table1`.

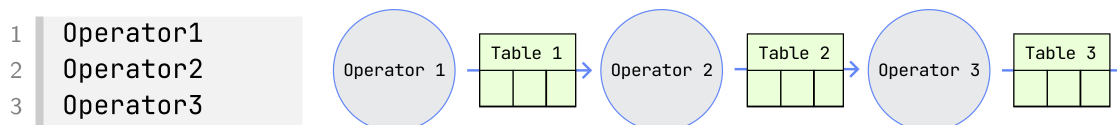


Figure 3.1: Operator sequence and table passing.

Based on this logic, we can identify two types of operators:

Source operators initialize the sequence and expose the first table. They have no previous operator to consume a table from.

Middle operators are all the operators after the source. They have access to the previous operator's exposed table.

We can also distinguish operators by the effect they have on the table:

Use operators uses the result built until now, as any other middle operator and performs action of the loading phase of the ETL process. An example would be saving the table to a file, or the database, or printing it to the standard output. We may expect them to be placed toward the end of the sequence. However, they may also appear as Source operator, if they initialize and uses the table directly. The purpose is to facilitate data path analysis.

To have a list of operators, see Section 3.5.

3.4.3 Transform Block

A transform block represents a consecutive list of operations. It must start with a source operator and it ends either with an explicit end of the transform block that may bind the output table to a top symbol.

This concept is inspired by the *stream* concept from EtlScript proposed by Wauthoz [Wau22]. It comes from the common pattern in ETL.NET:

```
1 var OutputStream = InputStream
2     .Operator1()
3     .Operator2()
4     .Operator3();
```

3.4.4 Statements

Is an abstract concept that regroups multiples of previously described concepts, by the fact they modify the global state of the Program, they have side effects.

Except for the already defined *Operator* and *Transform block* concepts, we will define the *Bind symbol* concept.

Bind symbol to an expression value is called in other languages variable declaration with initializer. The name of the symbol must be unique within a scope, but within nested scopes, a newly declared symbol can shadow the parent scope value.

3.4.5 Expressions

Expressions are concepts that resolve to value and a type. One of their uses is to parametrize operators. For example, the operator that restructures a table will take as a parameter an expression that will compute the new structure of the table. Expressions do not have side effects affecting the global state of the Pipeline in itself.

Composition of expression is done by imbricating them in a tree-like structure.

Notation is specific to each expression; they can be prefixed postfixed or infixed to ease readability and match usual notation.

To have a list of expressions, see section 3.6.

3.4.6 Structure definitions

To initialize a data source, basically a file, the compiler would benefit from knowing its structure, to be able to generate the C# code to read the file and to be able to check the rest of the user code and propose code completion and other analysis.

3.5 Operators to Build ETLs

In the previous section about operators, we described them as a concept without specifying the effect they have on the manipulated table. In this section, we will detail the effect of built-in operators. In addition, many more operators could be built and the design is open to new operators, mainly middle operators.

To select the operators to design as built-in, we will use a bottom-up approach. We analyze the usage of operators in a sample of ETL processes implemented in ETL.NET. From there, we will define the operators that will be implemented in the language.

Sample ETL.NET scripts Our sample is composed of 30 C# files, counting a total of more than 4163 lines of codes according to cloc. With 365 lines of comments, this low number of comments (8%) may be explained by the fact that each ETL.NET operator (represented by C# function) takes a description string as its first parameter, and many comments are commented-out code.

Table 3.2: Standard operators of ETL.NET with their usage in the example set.

ETL.NET operator	Description	Occurrence in CSV	PDF	Excel	Total
Select	Extended projection, modify the schema	84	36	17	137
Fix	Update conditionally attributes	1	1	1	3
Lookup	Left join	46	2	2	50
LeftJoin	Like a regular SQL left join	0	0	0	0
Union	Concatenate rows as they come	5	1	0	6
UnionAll	Concatenate all rows from one stream after the other	0	0	0	0
Where	Filter an array by a condition keeping only one succeeding at the condition	36	38	24	98
OfType	Filter rows based on a type	0	0	0	0
First	Get the first element of an array	7	22	14	43
Last	Get the last element of an array	3	16	7	26
Top	Take a number of elements from the start	0	4	0	4
Skip	Complement of top, pass a number of records	1	0	0	1
Distinct	Filter the sheet by keeping only distinct value	46	10	10	66
GroupBy	Group records on equal attributes into a list	0	2	0	2
Aggregate	Group records on equal attributes and do arbitrary reduce operations	7	0	1	8
Pivot	Aggregate values on equal attributes into a single record with aggregation operations	0	0	0	0
OrderBy	Sort the table by attributes	1	1	0	2
ToList	Convert the table to a list or records	0	24	11	35
EnsureSingle	Ensure the stream only contains one element	11	0	3	14
LookupCurrency	Specialized Lookup to provide currency from a text column	28	4	3	35

Continued on next page

Table 3.2: Standard operators of ETL.NET with their usage in the example set. (Continued)

EfCoreSave	Save the table to a database	78	14	11	103
EfCoreSelect	Read a table from a database	37	2	1	40
Ef-CoreLookup	Lookup a table from a database	3	3	2	8

From the previous table analyzing the operators in use in the sample, we propose the following list of operators to implement in the language, see Table 3.3.

Table 3.3: Operations available in core ETLang

ETLang operator	Description	Translated into ETL.NET
Restructure	Replace the table with an extended projection	Select
Add column	Add a column to the table, compute value from the previous line values	via Restructure
Update column	Update a column value, compute value from the previous line values	via Restructure or Update
Rename column	Rename a column of the table	via Restructure
Remove column	Remove a column from the table	via Restructure
Merge into	Merge two tables into a newly structured table by accessing records from both sides	Lookup or LeftJoin
Merge side by side	Merge two tables side by side by prefixing column names	Lookup or LeftJoin
Merge new column	Merge two tables by adding new columns to the table	Lookup or LeftJoin
Concatenate	Concatenate two tables by adding rows	Union or UnionAll
Filter lines	Filter the table rows by a condition	Where
First	Get the first row, produce a single-line table	First

Continued on next page

Table 3.3: Operations available in core ETLang (Continued)

Last	Get the last row, produce a single-line table	Last
Top	Get the first n rows	Top
Skip	Keep the lines after an arbitrary number of skipped lines	Skip
Remove duplicates	Remove rows with duplicate attributes, optionally merge partial rows	Distinct
Aggregate and reduce	Form groups of rows and perform reduce operations	Aggregate
Aggregate	Form groups of rows and perform aggregate operations on a list of values	Pivot or via Aggregate and reduce
Read files	Read a text file as a table at the provided path	CrossApply-TextFile
Read database	Read a database table with some constraints	EfCoreSelect
Merge with database	Merge a table with a database table	EfCoreLookup
Save to database	Save a table to a database table	EfCoreSave

3.6 Expressions to Build ETLs

The preliminary catalog of expressions was based on observation of the same C# sample used to analyze operations usage. Then we refined it with the help of our expert users, to see which kind of expressions they would use for each type of data.

The Table 3.4 presents the main expressions grouped by types.

Table 3.4: Expressions in ETLang

ETLang Expression		Signature	Description
Group	Expression		

Continued on next page

Table 3.4: Expressions in ETLang (Continued)

Value	Boolean Integer Number String Table	`*` → `Constant Expr`	Construct constant values
Table reference		→ `Table`	Reference a table
	Previous		
	Aggregate		
	Merge		
	Dot	`Table, ColumnDef` → `Expr`	Access a table column value
	Let in	`Expr, VariableDef` → `Expr`	Define a variable in an expression
Flow Control			
	Alternative	`Boolean, Expr, Expr` → `Expr`	
	When	`WhenBranch*` → `Expr`	
Unary	Not Floor Ceil Neg Abs	`Expr` → `Expr`	
Binary	and add sub div mul pow abs	`Expr, Expr` → `Expr`	
String Using			
	concat	`String*` → `String`	
	substr	`String, Integer, Integer` → `String`	Take the substring specified by the range of intergers
	template	`TemplateArg*` → `String`	
Regexp Using		`Regex, String` →	
	extract	→ `String`	
	match	→ `Boolean`	
	split	→ `one-line Table`	

Continued on next page

Table 3.4: Expressions in ETLang (Continued)

replace	$\rightarrow$ String

ETLang Implementation

4

With the design of the previous chapter, we will now implement a prototype of the language. We will see how the design can be implemented, and what are the challenges of implementing a DSL. Starting with the exploration of different implementation methods. A comparison of these methods already exists. Instead, we will discuss considered implementation methods and see how they could apply to our DSL, in Section 4.1. Since each tool will have an impact on the language design, in Section 4.2, we will see what has changed from the design to the materialized implementation. Afterward, we will get into the implementations comments of the DSL in Section 4.3.

4.1 Choosing an Implementation Framework

In this section, we evaluate different implementation methods with the needs of our DSL design. We have first considered approaches to build out the combination of utility tools, such as parser generator, AST transformations library, code generator, etc. Due to the need for editing support, we have also considered language workbenches that provide a complete environment for language implementation. And we also considered the possibility of implementing `ETLang` with a metalanguage of the .NET ecosystem.

4.1.1 Utility Tools

To construct a textual DSL with utility tools, we would need to combine at least a parser generator, an AST transformation library, a code generator, and a way to have editor support. From the textual representation we need to build an AST, and then either with model-to-model transformation or with a code generator, we can generate the code in the target language. Our preference goes to `C#` tools since it is the ecosystem in which ETL.NET gravitates.

22

Initially, we considered ANTLR, the major industry-grade parser generator. With its fourth version, ANLTR 4 stops trying to generate AST[1], instead, it generates a parse tree, from which we can continue processing with visitors or listeners.

We experimented with the C# ANTLR implementation as our goal is to generate C# code in the end and multiples libraries ease the C# code generation. Some of them are: T4 Templates, CodeGenCS a hybrid mix of code and text template, and Roselyn for C# analysis and compilation.

The missing part of transforming the parse tree into an AST, manipulating the AST, may be filled with StarLasu family of libraries. StarLasu is meant to be the link between the parse tree and structure to build and manipulate the AST. Its main implementation Kolasu, in Kotlin, is still in development but already provides usable structures to build an AST from the parse tree, manipulate an AST, automatically track node location, generate text from the AST, and EMF interoperability.

From the two quick prototypes we made, it seems clear that using a composite of utility tools will lead to a lot of boilerplate code, without needing that level of control, we would probably take advantage of a language workbench.

4.1.2 Language Workbenches

Many language workbenches exist, and we started with the state of the art of 2013 [Erd+13]. From 2013, many of these tools evolved, and some also disappeared. We already talked about Enso in Chapter 2, but we have also analysed more closely XText, textX, Rascal MPL, Spoofax, and MPS.

Based on their documentation, we estimated that MPS and Xtext were the most mature and complete tools. Xtext has the interesting feature of being able to generate a language server or web editor. MPS does not yet have an official web editor system, however, projects like LIonWeb seem promising. The advantage of MPS is elsewhere, and results from its *projectional* (or structural) editing system. Projectional editing means the user is editing the AST directly, instead of editing a text that will be parsed into an AST, they therefore avoid the parsing step. First, they can support syntax that cannot be easily parsed by text-based editors, such as tables or diagrams. Second, they can support language composition, typically

[1] "Because most ANTLR users don't build compilers, I decided to focus on the other applications for ANTLR v4: parsing and extracting information and then translations. For compilers, we need to convert everything into operations and operands – that means ASTs are easier." [@Par12]

language extensions, and mixing unrelated languages. These two features are very interesting for ETLang, as it would be a way to solve custom operator extensions, and to compose ETLang with other languages like regexp or even SQL. Projectional editing may come at a cost as described in an experiment [Ber+16]. Still, a big part of the language remains textual, expressions are an example; and the projectional editing may be less intuitive for the user, as they are not editing a text anymore. Fortunately, work has been done with Grammar Cells [Völ+16] in that direction, providing a way to mix in parsed text experience.

4.1.3 Metalanguages in .NET Ecosystem

Several metalanguages have been developed in the .NET ecosystem. They could allow us to add additional layers of abstraction to ETL.NET C# interface while retaining the flexibility of a general-purpose language and taking advantage of the .NET ecosystem. However, most of these metalanguages appear to be inactive, raising concerns about their suitability for new projects.

Among these metalanguages, Nemerle is a statically-typed, multi-paradigm language. Although it has been described as such, its development has significantly slowed down, with the last commit dating back to 2020. Nitra, a related project by JetBrains, was a Language Workbench released in 2014, but it appears to be abandoned as well.

Another metalanguage in the .NET ecosystem is Boo, which is also statically typed and features a syntax similar to Python. It provides capabilities for creating DSL, adding keywords, and generating code.

In contrast, F# stands out as another language within the .NET ecosystem. "F# allows you to write uncluttered, self-documenting code, where your focus remains on your problem domain, rather than the details of programming." [@Car+22]. While using F# for ETL processes in conjunction with ETL.NET may seem promising, it may not be a solution that will help non-programmers.

In summary, the .NET ecosystem offers various metalanguages, However, the lack of activity and questionable suitability for new projects raises doubts about their viability. On the other hand, F# demonstrates the potential for clean code development, we do not feel it will help substantially non-programmers by itself, but may be interesting to evaluate in conjunction with other aids.

4.1.4 Choosen Implementation Method

In selecting the implementation method for our project, several factors were taken into consideration. Ultimately, we opted to implement ETLang with JetBrains MPS for a variety of reasons.

First and foremost, MPS offers a projectional editor. This means that instead of relying on traditional text-based parsing, MPS allows for the direct manipulation and visualization of AST. This unique feature provides a flexible development experience, enabling us to define first the AST shape, thus the semantic, and only after, the syntax.

Additionally, MPS is known for its 'batteries included' approach. It provides a comprehensive set of tools and functionalities out of the box, such as code editors, generators, type systems, and scope management, which can significantly expedite the development process. This ready-to-use ecosystem minimizes the need for additional configuration or reliance on external libraries, streamlining the overall implementation effort.

Furthermore, one of the motivations behind choosing MPS is to leverage its projectional capabilities to create a more user-friendly and visually-oriented language for our project, while avoiding the downsides of a fully visual DSL. This decision aligns with our objective of developing a language that is accessible to non-programmers, as it allows for a more interactive and intuitive way of interacting with the language's constructs, and reduces syntactical errors.

In summary, the decision to employ MPS as our implementation method was driven by its projectional nature, which is justified by our non-programmers user group, as well as its comprehensive toolset and active ecosystem, which facilitates development.

4.2 Language Design Adaptations

Now that we have chosen MPS, and as for any implementation, we need to make choices where the design leaves open possibilities, and adaptations to the design to take into account the constraints and opportunities of the technology. In this section, we will discuss the choices and adaptations we made for this particular implementation with MPS.

4.2.1 Transform Blocks and Operators

With MPS, the user is modifying the AST directly. The designed AST concepts were not initially designed for this use way of editing, and we may need to relax some constraints.

Transform blocks and operators are two strongly linked concepts: a list of consecutive operators implies a transform block. The design also implies that it is the responsibility of the transform block to declare the symbol (or variable) name to which it will be bound. However, we may reconsider this design.

The input table is provided by the first operation (of type Source operator). To bind the output table to a symbol, we could let the transform block do it at its level, or add an operator that will bind the table of the preceding operator to a top symbol.

The second option allows the user to never have to manipulate transform blocks. A transform block in this design is implied by the Ⓘ IPiplineStart operator that marks the start of a transform block. The block ends with the last operation, which can be a Ⓒ ToSymbolPipe. Since Ⓒ ToSymbolPipe is an operation (Ⓐ BasePipelineStmt), by its semantics, it should provide a table as output. This has the implication, we can add directly after other operations that will consume the table. This is not possible with the first design, since the transform block is a concept that is not an operation and thus cannot be used as an operator.

The second design is more flexible. It, however, needs a bit more work on the generator part since we will need to reintroduce the transform blocks in the AST.

4.3 Implementing ETLang

This section presents the implementation of ETLang using MPS. It is important to note that familiarity with MPS is an advantage for understanding this section. We will begin by presenting the general architecture of the prototype, followed by various aspects such as the editor system, type system, scope system, and migration. We will then discuss the process of building the IDE and code generation.

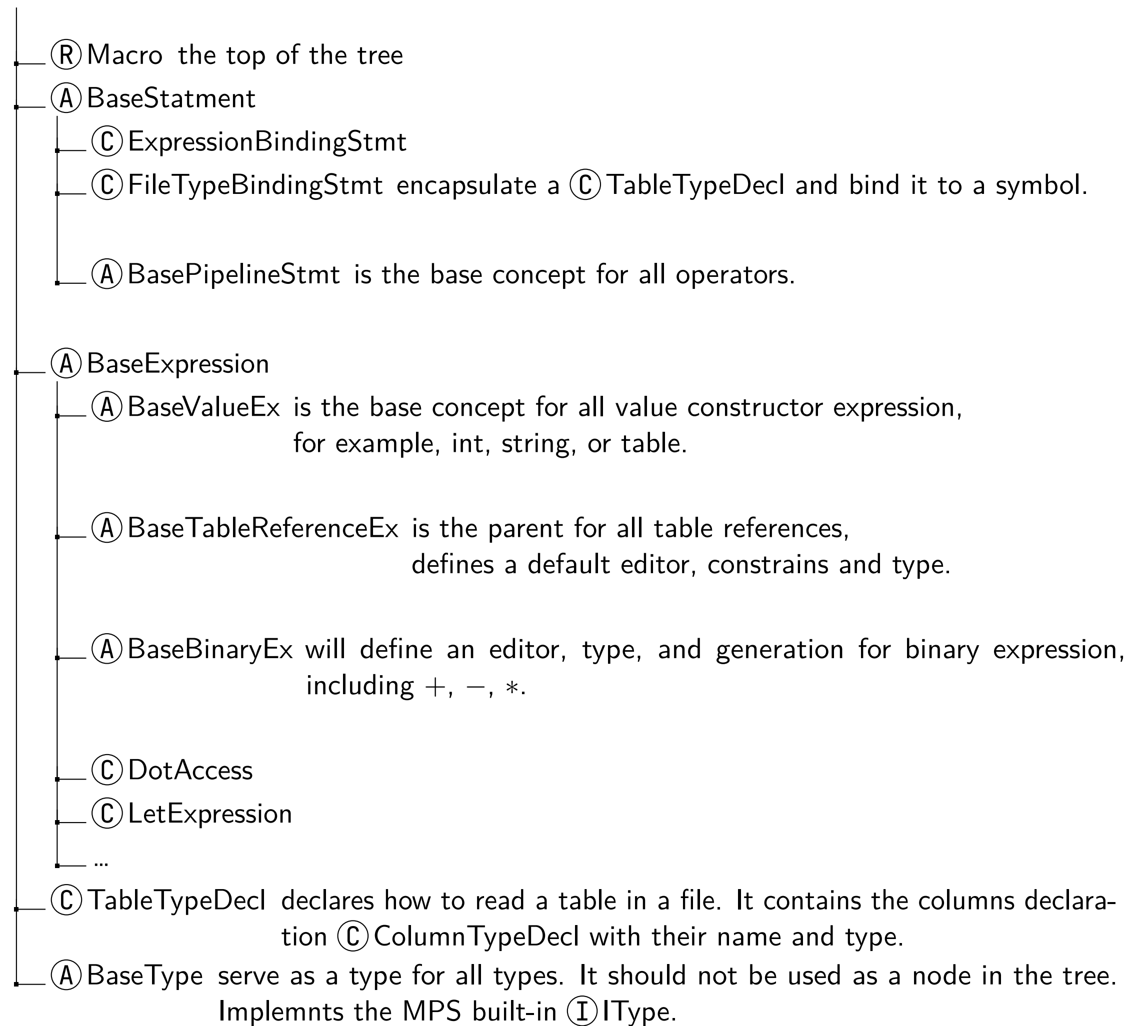
Note on the MPS conventions Implementing a DSL with MPS is done with special DSL mixed with the Java language. Therefore, many conventions are borrowed from Java. MPS has different kinds of concepts, and each concept can have multiple aspects. An aspect may be the structure of the concept in the AST, the editor used to display and manipulate it, the type it has, etc. Concepts may be Ⓡ Rootable, meaning they can be the root of a DSL tree, or Ⓐ Abstract, meaning they cannot be instantiated, similar to an abstract class in Java. Ⓘ Interfaces are also available and can be implemented by concepts. And Ⓒ Concepts may have properties, made of Java primitive types, children nodes, and relations, which are links to a node in the tree.

4.3.1 Concepts

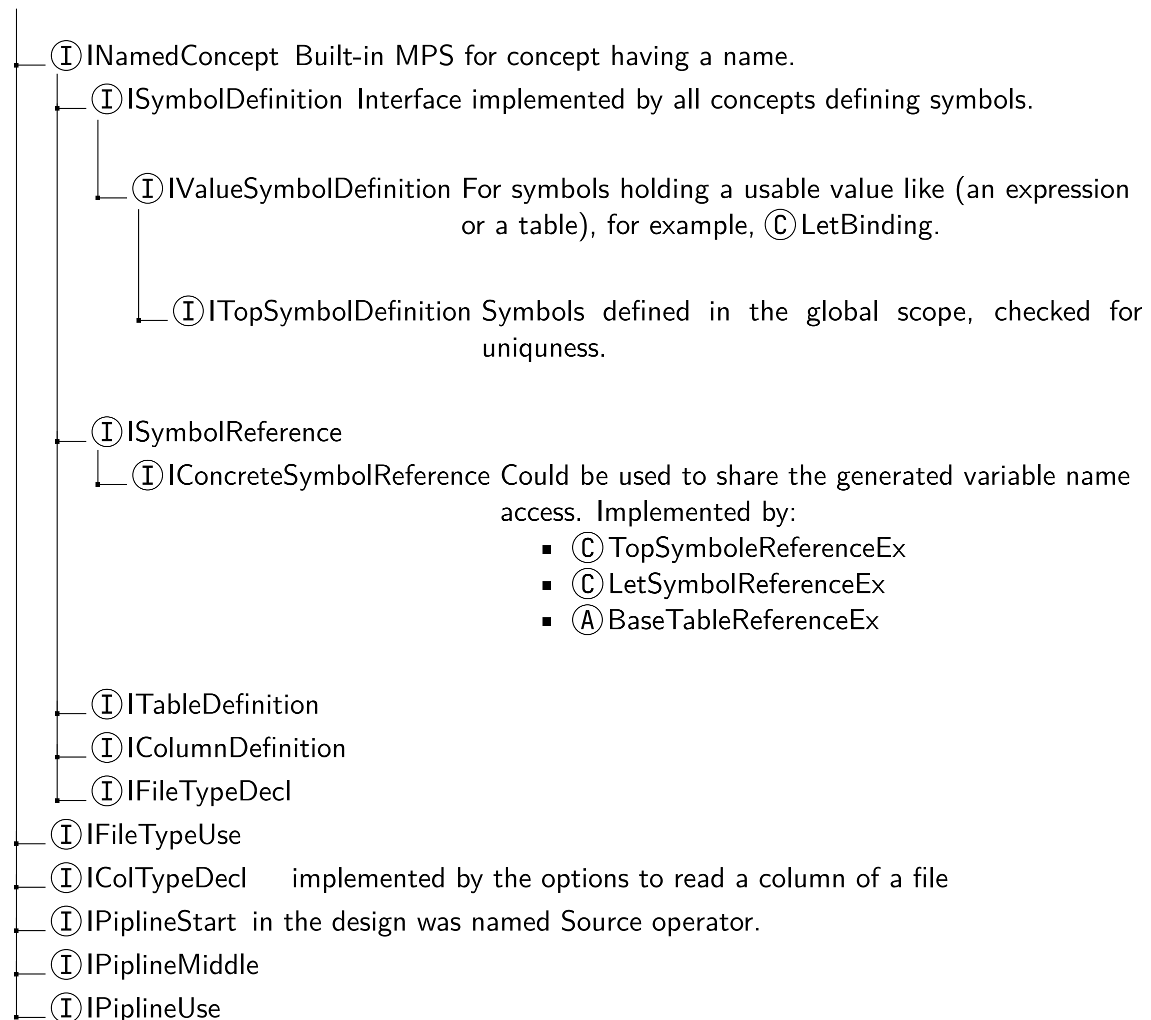
It is noteworthy that not all concepts will be implemented as part of this prototype. We have decided to implement a bit of every type of concept and aspect, starting with the most used ones. The objective is to explore the attainable outcomes rather than having a singular component that functions flawlessly while others remain nonexistent.

You will find a pruned version of the concepts tree in Tree 4.1, and a list of interfaces Tree 4.2.

Tree 4.1: Partial inheritance tree illustrating the concepts in ETLang, demonstrating the overall logical structure of the tree.



Tree 4.2: Complete tree diagram of the interfaces ETLang.



Patterns In the interface tree (Tree 4.2), several patterns are observed:

Define and Reference This is a straightforward pattern where a concept defines a value, and another concept references it for its usage. For instance, the concept Ⓒ LetBinding defines a value, which is referenced by Ⓒ LetSymbolReferenceEx. Incorporating interfaces for this behavior facilitates the sharing of editors or typing, as concepts sharing the interface can be used interchangeably.

Inline Definition This pattern allows users to define a structure in-place or use a pre-defined reference to the structure. It is commonly used in programming languages, where classes can be defined at the top level or as anonymous inner classes, and lambda expressions serve as anonymous inline functions. In the context of domain-specific languages, this pattern offers flexibility for language users to define certain elements inline, right where they are used, instead of separating them into distinct blocks [@Koš21]. It is used for example for the file definition as illustrated by Figure 4.1.

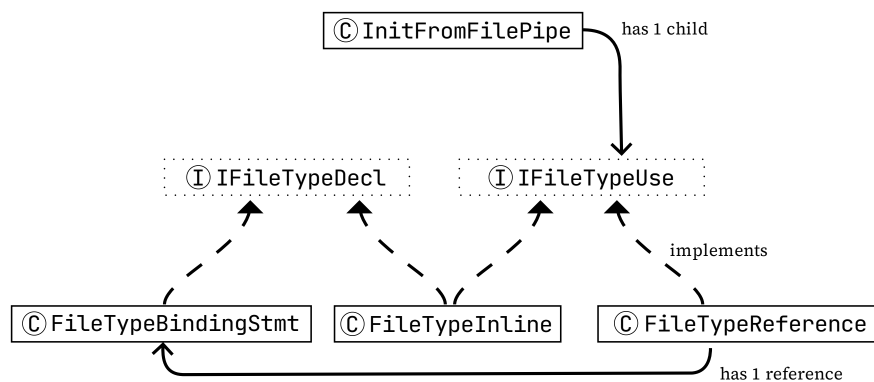


Figure 4.1: A file can be defined inline or as a reference to a named file definition beforehand. This adds flexibility for the user when using the file definition in a Ⓒ InitFromFilePipe that reads a text file with a structure.

As the language continues to expand, it will be necessary to refactor the interfaces to make better use of sharing behavior.

4.3.2 Editors

With the structure aspect, the editor is the second most important aspect of concepts. Editors define how the AST is presented to the user and how the user can interact with it.

Interestingly with projectional editing, we may define multiples editor for the same concept, then the user may choose between the default editor or one of the alternatives. For example, for Ⓒ TableValueEx that creates a table, we have a default editor that resembles structs in C for their compactness, but the user may choose to use a table. Both are fully usable in visualization and edition, Screenshot 4.1.

```
table SimplePerson {
  forename    ⟵ #Person.firstname
  surname     ⟵ #Person.lastname
  age         ⟵ currentYear - #Person.birthyear
  addr_street ⟵ #Person.street
  addr_num    ⟵ #Person.num
  addr_city   ⟵ #Person.city
}
```

a: Struct-like editor for a Ⓒ TableValueEx.

Table SimplePerson

forename	surname	age	addr_street	addr_num	addr_city
#Person.firstname	#Person.lastname	currentYear - #Person.birthyear	#Person.street	#Person.num	#Person.city

b: Table editor for a Ⓒ TableValueEx.

Screenshot 4.1: Alternative editors for `SimplePerson` a Ⓒ TableValueEx.

Usually with projectional editing, we need to construct the ast from root to leaves, which may be cumbersome to write expressions with infix visualization. For example, by default, to create an addition expression, the user has to create the addition by writing + and then the left and right operands. To allow fluent writing, we need to create the operands first and then when writing the addition charter, transform the tree and introduce the addition expression. This fluent writing behavior can be achieved with a *side transformation*, or by introducing an MPS Extensions, `grammarcells` will momentarily parse the user input with the available grammar, and construct the corresponding AST, and like any good parser, it handles priority and associativity.

4.3.3 Type System

With this prototype, the implementation of the type system started from the ground up because it is crucial to the language as it is based on inference and we do not ask the user to provide types.

Our type system is embarked with the primitive types: integer, boolean, and string as a starting point for this prototype. Then it has the table and column type. These are a bit

special because they track their origin. When we create a reference, we need a node to point, and when we do a reference to a column value, one needs to point to the column definition. When referencing the table exposed by the previous operator, the reference is computed and not stored in the reference node. Instead of trying to find in children of the previous sibling operation, it is easier to simply let the type track its origin. With the origin, we then can provide column completion when accessing a table.

When comparing table types, both the non-name attributes and the columns are compared. If the reference table lacks columns, the columns of the compared table are not verified. This functionality is useful, for instance, to simply validate whether a type is indeed a table or to validate only the table's attributes. Column comparison is based on their respective names and the type of content they contain.

In addition to assigning and verifying the types of AST nodes, the type system ensures compliance with other constraints of the AST that cannot be enforced solely through the AST structure. For example, it verifies that the transform block commences with an Ⓘ IPipelineStart, or that symbol names are unique.

4.3.4 Scope System

The scope system is rather simple. The top symbols are defined in the top-level scope provided by Ⓒ Macro and are available for use only after their definition, in following siblings and following siblings' children. Ⓒ LetBinding defines a scope, where the symbol defined by the Ⓒ LetBinding is available for use in the Ⓒ LetExpression and hides any existing symbol with the same name in the parent scope.

4.3.5 Migration

An unanticipated yet beneficial aspect of MPS is the capacity to migrate user code to a novel version of the language. Although not initially a primary concern, this functionality proved to be handy in deprecating a concept with ease and converting it to a more appropriate representation.

In contrast to general-purpose languages, where breaking changes are typically avoided, in DSLs, they are more commonly utilized [BS22]. By providing this feature, MPS significantly mitigates the expenses associated with breaking changes.

4.3.6 Building the IDE

To build the IDE, we needed to build a plugin with `ETLang` and ship our dependencies as well. MPS provides the building aspect to ease the generation of Apache Ant build files for our language and is capable of generating a plugin or a standalone IDE.

To be able to combine our dependencies, namely `grammarcells` and `plaintextgen`, we have used Gradle. Gradle with mps-gradle-plugin, have been used to manage the dependencies and coordinate the Ant tasks.

This process is then automated with GitLab CI, which builds the plugin and IDE and then publishes them to a package registry.

Users are then able to download and run the IDE on their machine, provided they have Java 11 installed, following the instructions available on `ETLang` install page.

4.3.7 Code generation

When we reached a minimal working state in terms of functionality, we stopped the development without having reached a complete implementation of the design. It was also desirable to prove the technical feasibility of code generation targeting ETL.NET.

MPS provides two-ways to generate code, via MMT (Model-to-Model Transformation) or directly TextGen. The concept of MMT is to have a model of the target language and to transform the actual high-level model into the target model, optionally using intermediate models. The TextGen aspect defines a model-to-text transformation. It works by appending lines of text to a buffer, with special support for indentation, and by calling the TextGen of the children nodes.

An additional way, contributed by MPS Extensions is to use *Plain Text Gen*, a model-to-model transformation, where the target model is a plain text matrix model. This approach enables the generation of text with a template-based approach, in contrast with TextGen.

The standard way in MPS is to use MMT. Officially supported languages are Java and Kotlin. `C#` is supported thanks to a school project in Charles University from 2019, with `mpscs`.

`C#` generation seems the most elegant approach, but some features like implicitly typed variables or anonymous types are not supported by `mpscs`, and these features are much used by ETL.NET code. Due to time constraints, we decided to use the Plain Text Gen approach,

which still brings use the benefits of the template-based approach. We were able to directly output text from our model without having to create an intermediate model for ETL.NET, because using a bottom-up approach during the design phase, the model is still fairly close to the target language. ETLang brings, however, some abstractions that will need an additional transformation to be outputted in ETL.NET.

In most cases, the concepts in the ETLang are converted directly into text, including the conversion of Ⓒ TableValueEx into C# anonymous types. As mentioned earlier, we need to explicitly introduce transform blocks in order to easily convert the AST into text. Because these transform blocks best represent the variable assignment of ETL.NET streams. This is made with a pre-processing script transforming the AST before the generation. A second transformation is needed to transform single-column table operations (e.g. Ⓒ AddColumnPipe) into Ⓒ RestructurePipe operation, that directly matches the `Select` operation in ETL.NET.

Furthermore, special consideration must be taken when referencing another variable. Since we are generating a text matrix, one cannot use the MPS reference mechanism. Instead, once a variable name has been generated, we store it in the source concept in a property prefixed with `gen_`. Then, when occurs a reference to the variable, we use the reference to the source concept again and use the previously generated variable name.

4.4 Web Version of the Editor

The zeitgeist has been to port applications to the Web. This is also true for development tools, and we have seen code editors being ported to the web, such as VSCode or JetBrains IDE. And the same goes for MPS multiple projects have attempted at bringing MPS to the web, with varying degrees of success, but non at this stage is considered the standard way to use MPS.

Different parties have started individually to deploy MPS on the Web [@Völ21]. Then they joined forces into Languages Interfaces on the Web organization, in short, LIonWeb.

While demonstrations showcasing MPS running on the Web exist, it is worth noting that this area remains under active development. Unfortunately, due to issues related to version compatibility and acquiring necessary dependencies, we were unable to present demonstrations of ETLang on the web. This remains an area of future exploration.

5 Evaluation

This chapter presents an evaluation of ETLang, focusing on the two main requested characteristics: ease of use and the reduced time it takes for users to become operational. While the notions can be subjective and open to various interpretations, we have developed a quality model that encompasses specific sub-characteristics related to these aspects. Additionally, we address the issue of extensibility, which was a particular concern given the limitations identified in previous research [Wau22]. Lastly, as one of the objectives was to integrate the ETL.NET ecosystem, we had to generate C# code using ETL.NET from ETLang. Therefore, we will also evaluate the generated code quality to ensure the functional appropriateness of our language.

In the following sections, we will first introduce our quality model, which outlines the sub-characteristics used to assess ETLang. Subsequently, we will elaborate on the methodology employed to evaluate our language, providing a clear understanding of the approach undertaken. Finally, we present our findings based on the evaluation and discuss these.

5.1 Evaluation Quality Model

While the characteristics we propose to evaluate ETLang are frequently cited in the literature, we have not found an established standard quality model for DSLs, and each article seems to pick its subset of quality characteristics. Indeed this is a field that is still in development [Pol+21], to build our usability framework we take inspiration from the one proposed by Zineb with BPMN4ETL [El 14] which targets business users and ETLs. But since BPMN4ETL is a graphical language we need to adapt to our projectional language. During the design phase we cited cognitive dimensions but using them now seems unpractical given the number of dimensions [Gre89; Bla00] and the limited time we have to conduct such an extensive experiment for this master's thesis. We may use some of them to justify our choices. We can also look at the internal properties of languages as proposed by Borning [@Bor02].

35

Tree 5.1: We propose to concentrate on the following characteristics, defined in conjunction with FundProcess and on the previously mentioned inspiration from the literature:

- Quality model
  - *Ease of Use* is the main charcteristic asked by FundProcess.
    - *Readability* needed when they show it to new users without fearing them with the `C#` syntax.
    - *Learnability* as they needed business users to be quickly operational, which is one of the pain points of the `C#` syntax.
    - *Expressiveness* comes with many studies, including [El 14; BNS21; Gre89; @Bor02; Pol+21]. Indeed we want to express our intentions with adequate language constructs, or "Expressiveness is defined as the ability of the language to describe all relevant aspects of the problem at hand" [El 14].
    - *Editing efficiency* can be evaluated by measuring the perceived efficiency of participants during a controlled experiment using projectional editing [Ber+16]. In our study, we will focus on the perceived efficiency of participants as well as the completion speed of the proposed task.
  - *Functionally adequate* seems the second requirement that needs at least a partial answer in a prototype. While we tried to improve usability, it evaluates whether we have succeeded to keep ETL.NET core functionality.
    - *Extensibility* via modularity ETL.NET enable extensions of every stadge of the ETL.
    - *Generated Code Quality* is not directly visible by end users. Nevertheless, users are indirectly affected by it, as insufficient generated code quality can undermine the intended abstractions, forcing users to confront their *conceptual model* with the underlying structures instead of relying on the convenience of higher-level abstractions.
      - *Correctness* of generated code, does the code compiles, and behave as intended.
      - *Readability* of the generated code.

5.2 User Study of Ease of Use

5.2.1 Evaluation Methods

In order to evaluate the subset of characteristics related to the 'ease of use' of our prototype, we primarily employed quantitative methods. These methods allowed us to gain insights and understanding of issues with a smaller group of testers. We complemented the quantitative approach with a qualitative method, namely the UMUX (User Metrics Usability Experience) [Fin10], which is a lightweight usability Likert scale test. However, due to time constraints, we acknowledge that the results obtained from this test may not be fully conclusive by themself taken alone.

There is a wide range of quantitative methods available for evaluating DSL. For a comprehensive list, we refer to Barišić's Ph.D. Thesis [Bar17] and the analysis of commonly used methods presented by Poltronieri et al. [Pol+21].

The utilization of triangulation in our evaluation process enhances the reliability and validity of our findings [Wil06]. By combining *qualitative and quantitative* methods, as well as considering the perspectives of different *user groups*, we can obtain a more comprehensive and robust understanding of the characteristics and usability of our prototype.

To conduct our evaluation, we employed the following methods:

Task solving Test participants were given two series of tasks to solve using our prototype. The first series is described and explained with a tutorial, the second without a tutorial. During the whole process, participants were encouraged to think aloud, providing insights into their cognitive processes and we observed their interactions with the system through screen sharing.

The tutorial goes through the basic concepts of defining a file structure, reading a file with the file structure, modifying the table and saving it to the database. These initial tasks have been chosen because they are the first steps that a user would take when designing a macro for ETL.NET. The tutorial consists of explicative text, video screen captures, and textual visualization of the code.[1] You can find the getting started tutorial online, in the documentation website of the prototype, or in Appendix B.

[1]The original code is structural, therefore the textual representation cannot be pasted into the editor, the tutorial contains an approximate representation of how the code looks like in the editor.

Then, as a second part, we asked participants to solve tasks without a tutorial, instead they may ask questions to the observer. The tasks involved reading a second file with a distinct file structure. Then they had to merge the two tables, based on a column, into a unified table that adhered to a specific structure, which was then to be saved to the database.

Semi-structured interviews At the end of the evaluation session, we conducted one-to-one semi-structured interviews with the participants. This allowed us to delve deeper into their experiences, gather feedback, and address any additional aspects that were not covered during the task-solving phase.

UMUX scale test We orally ran a modified UMUX scale test (see Appendix A) with the participants after the interviews. Along with the rating, we encouraged them to provide justifications for their responses, providing further qualitative insights.

Retrospective think-aloud exercise After one week without touching the prototype, testers were asked to re-explore their code and think aloud about what it does.

One-week-after refactoring After the retrospective think-aloud, we proposed a task involving modifying the code and adding operations to their ETL process.

Then, we used open coding to analyze the qualitative data obtained from the observations notes, the interviews and the justifications of the UMUX scale test.

Since the aim of ETLang is to facilitate ETL creation for business operators who are not necessarily programmers, we required a user group that could represent this target audience. Additionally, we wanted to ensure that the system remained accessible to programmers for simple ETL tasks that fall within the scope of working with tabular data. Consequently, we identified three user groups for our study:

1. *Business operators:* These users are non-programmers, although they may have had some exposure to SQL and basic programming concepts. Their familiarity lies primarily in working with spreadsheets. We included two users, one from FundProcess and a student with experience in Excel, in this group.

2. *Programmers:* These users possess general programming skills but are not specifically familiar with ETL.NET. In order to maintain a balanced representation, we conducted the study with two computer science master's students to counterbalance the business user group.

3. *ETL.NET experts:* These users are programmers who are actively using ETL.NET and possess expertise in ETLs. We had one user belonging to this category. This user group is relatively rare.

5.2.2 User Evaluation Findings

A total of five participants were interviewed for our research, all of whom were either University of Liège students or employees at FundProcess. To ensure anonymity, we assigned them pseudonyms throughout the study. Within the group, U1 and U3 identified themselves as business operators, while U4 and U5 were programmers. U2 possessed expertise in ETL.NET. We took the liberty of translating their quotations from French to English.

In this section, we present the findings of our user evaluation process regarding the ease of use. The objective was to assess whether our proposed solution achieved the goals outlined in our thesis and adhered to our quality model.

We will commence by discussing the positive aspects, followed by the issues encountered. One can note that some of these issues may be related to language design, while others may stem from implementation bugs or architectural choices.

Positive Aspects

Learning Our participants were commenting positively on the learning curve of ETLang, they found the tutorial and the ability to ask questions during the second phase of task solving a great help to learn the language. And after the session feel confident to use the language on their own. Non-programmers commented that they would need more resources in the form de video or manual with each operation, the form that these additional resources should take depends on the interviewee.

Autocompletion Was mostly considered helpful, only programmers explicitly commented that it was slowing them down sometimes when they have to use the completion due to the lack of allowing temporary erroneous code. A user commented (U2): "When you do not know what to do, you press Ctrl+Space and you are guided."

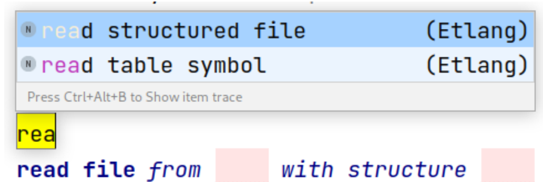
Template effect A non-programmer commented (U3): "it helps compared to C#, we have choices, it's not like starting from a blank page, it's logical to put things in the boxes". It probably comes from the absence of syntactical noise to write (separators are however visible in the editor) and it is not possible to have syntactical issues. A programmer commented, that it is nice to not have to care about the syntax, (U4) "like in other languages you have to end [statments] with a semicolumn, but it is cool to be able to forget it".

Multiple visualisations Being able to choose the visualization of table values (see Screenshot 4.1) seems to be appreciated, users commented that they would see themselves using the tabular visualization when re-reading their code.

Debugging types The ability to see the type of data in the editor was appreciated by the users, they commented that it was helpful to understand what was going on in the code, and they used it mainly to see the table structure as showcased in the tutorial.

Issues

We attempt to present the issues encountered during our evaluation process in an order that reflects their frequency, while also grouping them thematically.



Screenshot 5.1: You partially write `read structured file` and select from the completion, then the node is added to the AST and with the correction shown as on the second line

Closeness of naming and typing The initial testers found it disorienting, adding friction, that what they had written was too different from their projection in the editor. For example, they asked whether `read structured files` and `initialize from files` were the same things. Initially, you had to write the first, and the second would be displayed in the editor. This was corrected for subsequent testers and now displays as `read file from <file pattern> with structure <file structure>`, similar changes were made for other operators.

Optional parameters All participants asked about the meaning of `<no optionalFrom>` parameter in the file structure definition (for example Screenshot 5.2). Some participants simply asked the question and continued without dwelling on it, while others revisited the tutorial. Some participants posed the question again during the second phase. Although not an issue yet, this approach proved effective in highlighting features and allowing users to discover them. However, it is important to strike a balance and avoid excessive use of such techniques, as it may lead to verbose syntax and reduced readability, as mentioned by some users.

```
file structure type PersonFileStructure {
  // <no description>
  separator: ","
  table Person {
    firstname from <no optionalFrom> is string
    lastname from <no optionalFrom> is string
    birthyear from <no optionalFrom> is integer
    street from <no optionalFrom> is string
    num from <no optionalFrom> is integer
    city from <no optionalFrom> is string
    country from <no optionalFrom> is string
  }
}
```

Screenshot 5.2: File structure definition of a Person bound to the symbol PersonFileStructure, with 7 columns.

File structure definition Three users found it unclear why two different names were needed for the file structure definition and the table it defines.

Previous Table Access Except for our ETL.NET expert, all users had difficulty encoding access to the previous table. They tended to write `#Person` directly instead of `#table` to access the previous table, assuming the previous table was named ‘Person’.

Another issue arose when users created a reference to the previous table; to select a column, they had to add a dot and then select the column as explained in the tutorial. However, a bug caused the cursor (represented by ‘|’) to be placed at `#|Person` instead of at the end of the expression, making it disturbing for users.

Empty lines between operators Except for one tester, all participants instinctively inserted an empty line between their first two operators. Non-programmers paused and asked about the red underlining, while programmers initially disregarded it. However, when they encountered difficulties with accessing the previous table as described in the tutorial, they also paused and attempted to identify the error. Both user groups agreed that the error message was not clearly formulated, as it employed implementation-specific terminology, in this case, the message was ‘Error: Previous statement is not a pipeline producing a Table, try adding an initialize on the previous line.’

Error messages The error messages were not always clear, especially the ones encountered in the two previous issues. Users expressed that these messages contained unfamiliar terminology, consisting of implementation-specific concept names. Furthermore, the proposed solutions provided by the error messages were not helpful or relevant. The inclusion of quick fixes is a potential enhancement that could be considered for the language prototype, although it was not implemented in the early prototype stage.

Selection and Navigation Participants from various backgrounds reported being surprised by the selection and navigation in the editor. Further investigation would be needed over the long term, as even we, as developers of `ETLang`, were initially surprised but found it quite pleasant with practice. Participants also noted that they would appreciate the ability to copy and paste code from outside the editor, such as from the tutorial.

Merge The first users who tested the merge operation found it confusing and unclear. It has four fields to fill, and the field names alone were not sufficient to understand what to do. A second version with partially pre-filled fields greatly helped subsequent users. In any case, the concept of symbols has to be explained beforehand and said that they need to bind the table they want to merge with a symbol.

Symbols Programmers had no issues and found the term 'symbol' acceptable once they understood that it was equivalent to variables in other languages, specifically constants. *Non-programmers* had more difficulty understanding the concept of symbols. The term 'variable' was unfamiliar to them, 'constant' was clearer. They suggested the term "value container" instead. Binding expression to a symbol seems intuitive for our testers, however, the need to bind a table to a symbol was not as intuitive as some remarked that tables are already named.

Implicite Transform blocks As we have seen the transform blocks have been made implicit in the prototype, to have a more flexible and line-based edition. However, based on the comments about the empty lines between operators, or the table-to-symbol binding, it seems that making the transform blocks simple may not be the best idea. It moves the validity checking to the type system instead of being directly constrained by the AST. An A/B test could be conducted to determine which approach is the most intuitive.

Autocompletion and programmers Programmers found autocompletion useful but occasionally felt it slowed them down. Sometimes, they missed the option of being able to write incorrect code and then correct it, which is impossible in this prototype due to the chosen projectional technology. This should be further assessed over time with a smoother prototype to determine if it remains perceived as a problem.

In conclusion, it should be noted that users may be more inclined to comment on defects rather than positive aspects, due to a phenomenon known as negativity bias [@Lor16]. However, many of the issues mentioned are related to the prototype rather than the language itself. A detailed discussion of the ease of use will be presented in the Section 5.4.

5.3 Evaluation of Functional Adequacy

In this section, we will assess the functional adequacy of the system based on two key requirements: extensibility and code generation quality, as outlined in our quality model.

Extensibility

Extensibility of languages comes at a low cost with MPS. However, we must be careful not to break it, as with any other object-oriented programming language.

Adding a new operator that corresponds to an operation in ETL.NET is a straightforward process. To illustrate this, we will use the example of the 'Distinct' operator, which can be implemented in less than 5 minutes. The steps involved are as follows:

1. Create a new language that extends `ETLang`.
2. Define a concept for the new operator, such as 'DistinctPipe'.

   This concept should extend the appropriate concept from the base language and implement the necessary interfaces. Additionally, define any child nodes required, such as the key on which the distinct operation will be performed.

3. Define the editor, system, and generation aspects for the new operator. With these defined, the minimal implementation is complete.

Generating code out of the stream of operators is not currently possible with the prototype architecture. To enable this functionality, the language would need to define another concept that generates the out-of-the-flow code. This concept could then be inserted at the appropriate location in the AST using a pre-mapping script. In addition to operators, the prototype is open to extensions of new expressions and types, including both primitive types and compound types such as lists or tuples.

Code Generation

Among the three sub-characteristics associated with code generation, our evaluation will focus solely on correctness and readability. It is important to note that the primary objective of this prototype was to demonstrate the feasibility of generating code, rather than fully optimizing the output.

In terms of readability, our ETL.NET expert determined that the generated code was sufficient for understanding. The variables generated by the system maintain a resemblance to the original concept names, with changes in capitalization, the addition of underscores, or the use of prefixes and postfixes. Although the indentation may not adhere to strict `C#` standards, it is present in the generated code.

To ensure, the production of correct code, ETL.NET would need to enforce stricter AST. We have identified two notable issues that require attention:

- First, column names are not transformed and used as `C#` identifiers, the motivation for that choice is to be compatible with externally defined tables via anonymous types or classes. To solve this issue, we would need to check the column names to be valid identifiers.
- The second issue pertains to the handling of features that are currently not implemented in the code generation aspect. Specifically, when defining a key for a table in merge or aggregate operations, the AST allows for an arbitrary number of columns to be considered as keys. However, the code generation aspect currently considers only the first specified column. A potential solution would involve implementing code generation for each of these operators individually, although a more robust and generic approach is desired for future improvements. This would involve designing a concept capable of supporting multiple key columns and generating the corresponding code accordingly.

5.4 Discussion

In terms of *ease of use*, the testers seem to be overall satisfied, although some bugs and early-stage improvements have slightly tarnished their experience.

Readability was evaluated rather positively in the qualitative data. Due to the limited number of respondents in our questionnaire, we cannot provide a definitive conclusion concerning the qualitative data.

The *learning curve* was considered to be good, with the tutorial and the ability to ask questions, in the second phase of the task solving, being a great help to learn the language. Testers declared that they feel able to do it alone next time, with a manual for more complex operations such as aggregate or merge. We have verified with some of them the learnability during the one-week-after session, they seem and perceived that it was "easier" (U1) or "smoother" (U3).

The evaluation of *expressiveness* did not encompass complex or complete ETL macros. The prototype did not include all the necessary expressions to develop complex macros. However, the testers were able to easily express the simple macros they were tasked with. The abstraction layer and concepts proposed by `ETLang` were generally well-received. One aspect that testers appreciated was the ability to add a column to a table without the need to specify all the existing columns of the table, as in a `restructure` operation. When asked to keep only certain columns, some users instinctively wanted to delete the columns they did not need, while others preferred to select the columns to retain. Although these syntactic sugar operators were not implemented in the prototype, the language concept is open to their inclusion.

When it comes to the *editing experience*, the computer scientists were the most critical concerning projectional editing. They tried to write a full line of code without using the completion system, therefore without constructing a valid AST incrementally. During the interview, they expressed (U4) "I am used to being able to write code with errors and correct them afterward instead of having to rewrite everything."

Other testers initially struggled with navigation and deletion, but by the end of the session, they expressed appreciation for the ease of editing provided by the templated approach, where they only need to fill in the required information.

Even if users had the error once and then do not repeat it, empty lines between operators should probably be allowed to reduce the first friction with the language, and maybe redesign

how operators are linked together to make it more visible as discussed with the implicit transform blocks.

Looking back at the *functional characteristics* of our quality model, it is evident that the prototype is not yet suitable for production due to missing features. Nevertheless, we have demonstrated the feasibility and extensibility of the language by incorporating a distinct operator as an example. The generated code has been verified to be readable according to our expert user, and correct with samples of `ETLang` code. Although the generated code may not attain the level of efficiency achievable through manual coding, this is an anticipated outcome resulting from the inclusion of abstractions. We identified areas where optimization of code generation could be implemented.

Regarding the implementation of code generation, a template-based methodology was adopted for this thesis assessing the language feasibility and viability. However, for a production version, a more robust approach such as utilizing the MMT approach towards a `C#` AST would be recommended, even if more complex. This would allow for validation of the `C#` AST, and better support from MPS code generation aspect.

6 Conclusion and Future Works

As we approach the conclusion of this study, it is time to reflect on what has been accomplished, what has been learned throughout this work, and what could be done in the future to further develop the current prototype. Additionally, it is an opportunity to propose other original solutions that may be worth investigating.

6.1 Conclusion

Throughout this master's thesis, we have embarked on a design journey that allowed us to explore the creation of ETL user interfaces and address the needs of non-programming users. It has been an intriguing design problem, as it involved more than just proposing a potential solution, but rather engaging in the entire design process, having an important creative part.

Our work adhered to several principles of user-centered product design in the realm of human-computer interactions, which we discovered and studied at Aalto University. It was through the design aspects, rather than technical considerations, that we learned the most and experimented at various stages. This experience also provided an opportunity to delve deeper into the field of ETL, building upon our previous integrated project related to e-learning platform migration.[1] A common point of both projects was to empower users to have control over their data transformations, since full automation even if desirable was not possible.

While our previous project focused primarily on technical aspects, this master's thesis explored the area of design. It was approached from a perspective that values design culture, emphasizing the role of abduction or creative hypothesis, rather than the scientific culture of objectivity and neutrality. However, it is important to note that working on a design project individually does not naturally encourage documentation and the complementarity of ideas. As with any design process, restarting the project would yield different results.

[1] I have worked, with my group of Integrated Project, on a proof of concept transferring and converting Blackboard courses to Moodle.

47

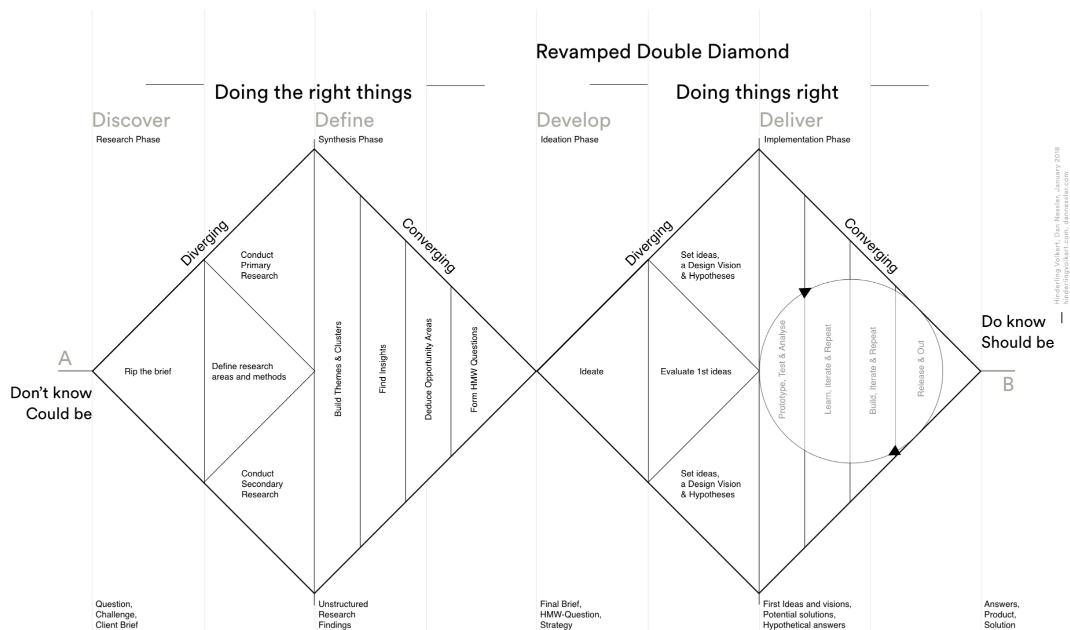


Figure 6.1: Revamped Double Diamond Framework [@Nes18]

Let us revisit our initial objectives and assess our progress at this point. Our goals were to facilitate the learning process of creating ETL processes and enable non-expert users to create ETLs while reducing the time required for their development, regardless of the users' programming expertise. We placed our focus on ETL processes targeting relational databases, with data sources primarily consisting of CSV files and other similar formats managed by ETL.NET as `TextFiles`.

Through retrospective analysis, we can examine the undertaken work using the *double diamond framework* (also on Figure 6.1), which has been adapted to accommodate a preferred solution in the form of a DSL:

1. The first stage involved investigating whether a DSL could serve as a solution to the problem at hand, answering the question 'are we doing the right thing ?'. During the divergent research phase, we engaged in discussions with domain experts and researched ETL processes, various types of programming interfaces, and related topics. Mind mapping was employed as a means to structure our ideas during the convergent synthesis phase. It would have been beneficial to document our findings more extensively for this thesis. At the conclusion of this phase, we were able to define the objectives of our project and identify the associated constraints with greater clarity. We discussed our

approach to the problem and potential solutions in the "Related and Future Works" section.

2. The second stage focused on 'doing the DSL right'. During the ideation phase, we created and evaluated several paper prototypes, following a similar approach employed by Borum et al. [BNS21]. This two-phase design process, akin to the diverging and converging stages of the diamond model, was discussed in detail in the design section (see chapter 3). The two phases of Borum et al. do not serve the same goals, the first is common to both methodologies, the second for Borum et al. is a final evaluation of the construction prototype, which we also conducted in the previous chapter. For the double diamond, the converging phase, or implementation phase, involves the actual implementation of a functional prototype for ETLang, which was indeed done before the final evaluation. The final evaluation phase allowed us to determine if the developed prototype adequately met the requirements of users and other stakeholders.

At the end of the process, we think a projectional dsl with spreadsheets naming and conventions, materialized by ETLang design and implementation, may be an answer to the objectives we initially set.

However, for this solution to be effectively utilized, further implementation effort and refinement with continuous user involvement would be necessary to make the language evolve with its users.

Furthermore, we would recommend considering exploring alternative solutions, either as complementary approaches or as potential alternatives to domain-specific languages.

Although further refinement and implementation efforts are necessary for the proposed DSL, we hope this design research provides a foundation for investigating alternative solutions and facilitating the ongoing evolution of the language in collaboration with its users.

6.2 Future Works

6.2.1 Improving the Prototype

To further enhance the prototype, several areas of focus can be identified:

Addressing Identified Issues

During the evaluation, we identified several issues. Addressing them would be a first step.

Expansion of `ETLang` Language Constructs

The prototype can be extended by introducing additional language constructs such as more operators, diverse types, and expressions. These extensions would provide users with a broader range of functionalities to express complex ETL macros. As we have seen in the extensibility evaluation.

Refactoring

The concept interfaces within the language can be reorganized to improve the sharing of behaviors and facilitate better node reference scoping. This restructuring would enhance the maintainability of the language.

Feasibility Study of MPS on the Web

Exploring the feasibility of adopting MPS on the web would open up new possibilities for accessibility and collaboration, allowing users to work with the language through a web-based interface.

Automatic File Structure Deduction

Investigating the feasibility of automatically deducing file structures from example files. Possibly implemented with MPS `PasteHandler` would streamline the process of defining file structures in the language, enhancing user productivity and reducing manual effort.

Standard Tables for Anonymous Types

Introducing standard tables, tables with predefined columns, validated by database-related operations, would enhance reliability and reduce errors.

Optimize AST for Code Generation

We have identified opportunities to optimize the AST for code generation. For example, we could merge multiple operations of adding columns and merge them into a single restructure operation, leading to fewer operators in the generated code.

Code Generation with MMT to `C#` and ETL.NET

Exploring code generation capabilities using MMT to generate stub `C#` code and ETL.NET code would provide users with more seamless integration with existing frameworks and tools.

Data Path Analysis

Performing data path analysis specifically focused on use operators, would enable users to gain insights into data flow and dependencies within their ETL programs. This analysis would enhance error messages and facilitate optimization.

Side Panel in MPS

Exploring the idea of adding a side panel within MPS that provides a preview of results of the selected steps of the ETL process would potentially enhance user productivity and facilitate real-time feedback during program development.

6.2.2 Other Research Directions

In addition to the presented enhancements to our existing prototype, our research has unveiled several other research directions that we think merit exploration. It is crucial to emphasize that these concepts have not undergone rigorous evaluation in terms of feasibility or their potential to enhance usability. Therefore, they are presented solely as potential ideas for future research.

Live-coding of ETL

Exploring methods to enhance interactivity and provide rapid feedback in the design of ETL processes represents an interesting research direction. ETLs systems, including ETLang, commonly involve a separation between the design and execution phases. Investigating approaches to integrate these phases and facilitate faster user feedback during the design phase would be a valuable endeavor.

One promising direction to explore is the concept of live coding for ETL, where users can observe the program results in real-time as they modify their program. This approach has been implemented in another context by Heyvaert et al. [Hey+16] and is also possible with Enso (see Chapter 2), where users can view the data output through interactive placed windows at choose stages of their programs. This live-coding capability would represent an initial step towards enhancing the user experience in ETL design.

Building upon this foundation, further possibilities arise, such as modifying program behavior by directly manipulating the displayed results, such as Programming-by-Example, which we will discuss next. Another approach would be to propose advanced

aids for comprehending and constructing ETL programs, drawing on the vision of *Learnable Programming* presented by Bret Victor [@Vic12].

However, several challenges must be addressed when implementing live coding for ETLs. These challenges include visualizing the complexity of ETL processes, ensuring good performance to provide real-time feedback, finding a balance between power and simplicity to improve usability, and guaranteeing the safety of operations as it should not have any side effects on data sources and destinations.

Programming-by-Example

Inspired by the success of the programming-by-example approach in data preparation tools such as Wrangler, investigating how it could be transposed to ETL design would be an interesting direction with an original user experience. This entails investigating how users can modify sample data and generate ETL programs based on these modifications, offering a unique and innovative user experience.

However, it is important to acknowledge the potential challenges associated with this approach. One concern pertains to the time, and frustration due to repetitive action, required to provide a sufficient number of examples to derive the desired action, particularly in cases involving conditions. In some instances, the time invested in providing an adequate number of examples may exceed the time required to describe the desired action using an alternative interface, language, or menu-based system.

Despite these potential challenges, exploring the feasibility and effectiveness of incorporating programming-by-example into ETL design presents an intriguing avenue for future research. It has the potential to offer users a more intuitive and efficient means of creating ETL programs by leveraging their existing knowledge of the data.

Glossary and Acronyms

Glossary

.NET .NET is a free, cross-platform, open-source developer platform for building many different types of applications. iv, 1, 6, 22, 24, 53

ETLang ETL Language, working around Tables. iv, v, 11–15, 18–22, 24–35, 38, 39, 42, 43, 45, 46, 49–51, 58, 59

C# C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. 5, 7, 8, 11–13, 16, 19, 22–24, 33–36, 44, 46, 50

F# F# is a functional programming language for .NET developed by Microsoft. 24

ETL.NET ETL library in .NET created by Stéphane Royer, sponsored and used by FundProcess. 1–18, 22, 24, 33–39, 43, 44, 48, 50

FundProcess FundProcess is a company that develops software for the financial industry, they provided the subject for the work presented in this thesis. 1, 2, 4–7, 9, 14, 36, 38, 39, 53

MPS JetBrains MPS, or Meta Programming System, is a language workbench that allows to create projectional DSL. 23, 25–29, 32–34, 43, 46, 50

MPS Extensions "The MPS Extensions aim to ease language development within MPS." They are maintained in open-source by itemis and JetBrains. 31, 33

Acronyms

AST Abstract Syntax Tree. 11, 13, 22, 23, 25–27, 30–32, 34, 40, 42, 44–46, 50

BPMN Business process model and notation. 5

DSL Domain Specific Language. 2–5, 8, 9, 13, 22, 24, 25, 27, 32, 35, 37, 48, 49, 53

ETL Extract Transform Load. iv, 1–9, 11–13, 15–21, 24, 35, 36, 38, 39, 45, 47, 48, 50–53

IDE Integrated Development Environment. 3, 4, 26, 33

MMT Model-to-Model Transformation. 33, 46, 50

SSIS SQL Server Integration Services. 5

UMUX User Metrics Usability Experience. 37, 38, 58

Bibliography

[Bar17] Ankica Barišić. "Usability Evaluation of Domain-Specific Languages". en. PhD thesis. Universidade NOVA de Lisboa, Dec. 2017 (cit. on pp. 8, 37).

[Ber+16] Thorsten Berger, Markus Völter, Hans Peter Jensen, Taweesap Dangprasert, and Janet Siegmund. "Efficiency of Projectional Editing: A Controlled Experiment". In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. Seattle, WA, USA: Association for Computing Machinery, 2016, pp. 763–774. DOI: 10.1145/2950290.2950315 (cit. on pp. 24, 36).

[Bla00] A. Blackwell. "Dealing with New Cognitive Dimensions". In: Workshop on Cognitive Dimensions. University of Hertfordshire, Dec. 8, 2000. URL: https://www.cl.cam.ac.uk/~afb21/publications/CDWorkshop.pdf (visited on May 19, 2023) (cit. on p. 35).

[BNS21] Holger Stadel Borum, Henning Niss, and Peter Sestoft. "On Designing Applied DSLs for Non-Programming Experts in Evolving Domains". en. In: *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, Oct. 2021, pp. 227–238. DOI: 10.1109/MODELS50736.2021.00031 (cit. on pp. 8, 11, 36, 49).

[BS22] Holger Stadel Borum and Christoph Seidl. "Survey of established practices in the life cycle of domain-specific languages". en. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*. Montreal Quebec Canada: ACM, Oct. 2022, pp. 266–277. DOI: 10.1145/3550355.3552413 (cit. on p. 32).

[El 14] Zineb El Akkaoui. "A BPMN-based conceptual language for designing ETL processes". en. Publisher: Université libre de Bruxelles. PhD thesis. Université libre de Bruxelles, June 2014. URL: http://hdl.handle.net/2013/ (visited on Apr. 4, 2023) (cit. on pp. 5, 7, 35, 36).

[El +12] Zineb El Akkaoui, José-Norberto Mazón, Alejandro Vaisman, and Esteban Zimányi. "BPMN-Based Conceptual Modeling of ETL Processes". en. In: *Data Warehousing and Knowledge Discovery*. Ed. by Alfredo Cuzzocrea and Umeshwar Dayal. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 1–14. DOI: 10.1007/978-3-642-32584-7_1 (cit. on p. 5).

[Erd+13] Sebastian Erdweg, Tijs van der Storm, Markus Völter, et al. "The State of the Art in Language Workbenches". In: *Software Language Engineering*. Ed. by Martin Erwig, Richard F. Paige, and Eric Van Wyk. Cham: Springer International Publishing, 2013, pp. 197–217 (cit. on p. 23).

55

[Fin10] Kraig Finstad. "The Usability Metric for User Experience". In: *Interacting with Computers* 22.5 (May 2010), pp. 323–327. DOI: 10.1016/j.intcom.2010.04.004. eprint: https://academic.oup.com/iwc/article-pdf/22/5/323/1992916/iwc22-0323.pdf (cit. on p. 37).

[Gre89] Thomas R G Green. "Cognitive Dimensions of Notations". In: *Proceedings of the Fifth Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*. UK: Cambridge University Press, 1989, pp. 443–460. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.128.270 (cit. on pp. 7, 35, 36).

[HN20] Mazhar Hameed and Felix Naumann. "Data Preparation: A Survey of Commercial Tools". In: *SIGMOD Rec.* 49.3 (Dec. 2020), pp. 18–29. DOI: 10.1145/3444831.3444835 (cit. on p. 6).

[Hey+16] Pieter Heyvaert, Anastasia Dimou, Aron-Levi Herregodts, et al. "RMLEditor: A Graph-Based Mapping Editor for Linked Data Mappings". In: *The Semantic Web. Latest Advances and New Domains*. Ed. by Harald Sack, Eva Blomqvist, Mathieu d'Aquin, et al. Cham: Springer International Publishing, 2016, pp. 709–723. DOI: 10.1007/978-3-319-34129-3_43 (cit. on p. 51).

[Kan+11] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. "Wrangler: interactive visual specification of data transformation scripts". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2011) (cit. on p. 6).

[Kle08] Anneke Kleppe. In: *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. 1st ed. Addison-Wesley Professional, 2008. Chap. 6.3 How to Create an Abstract Syntax Model (cit. on p. 8).

[Pol+21] Ildevana Poltronieri, Allan Christopher Pedroso, Avelino Francisco Zorzo, Maicon Bernardino, and Marcia de Borba Campos. "Is Usability Evaluation of DSL Still a Trending Topic?" In: *Human-Computer Interaction. Theory, Methods and Tools*. Ed. by Masaaki Kurosu. Vol. 12762. Cham: Springer International Publishing, 2021, pp. 299–317. DOI: 10.1007/978-3-030-78462-1_23 (cit. on pp. 35–37).

[Völ+16] Markus Völter, Tamás Szabó, Sascha Lisson, et al. "Efficient Development of Consistent Projectional Editors Using Grammar Cells". In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*. SLE 2016. Amsterdam, Netherlands: Association for Computing Machinery, 2016, pp. 28–40. DOI: 10.1145/2997364.2997365 (cit. on p. 24).

[Wau22] Julien Wauthoz. "Creation of a domain specific language for an Extract-Transform-Load system". MSc thesis. Université de Liège, June 26, 2022. URL: https://matheo.uliege.be/handle/2268.2/14583 (visited on Oct. 23, 2022) (cit. on pp. 4, 10, 15, 35).

[Wil06] Chauncey E. Wilson. "Triangulation: The Explicit Use of Multiple Methods, Measures, and Approaches for Determining Core Issues in Product Development". In: *Interactions* 13.6 (Nov. 2006), pp. 46–63. DOI: 10.1145/1167948.1167980 (cit. on p. 37).

Webpages

[@Bor02] Alan Borning. *CSE 341 – Evaluating Programming Languages*. 2002. URL: https://courses.cs.washington.edu/courses/cse341/02sp/concepts/evaluating-languages.html (visited on Dec. 7, 2022) (cit. on pp. 35, 36).

[@Car+22] Phillip Carter, Michaël Hompus, OlegAlexander, et al. *What is F#*. Oct. 13, 2022. URL: https://learn.microsoft.com/en-gb/dotnet/fsharp/what-is-fsharp (visited on May 16, 2023) (cit. on p. 24).

[@Gu+23] Grace Gu, Randolph West, Jason Roth, et al. *SSIS Designer*. Mar. 3, 2023. URL: https://learn.microsoft.com/en-us/sql/integration-services/ssis-designer (visited on May 16, 2023) (cit. on p. 5).

[@Koš21] Sergej Koščejev. *Language design patterns: Inline definitions*. June 7, 2021. URL: https://specificlanguages.com/articles/patterns/inline-definitions/ (cit. on p. 30).

[@Lor16] Hoa Loranger. *The Negativity Bias in User Experience*. Oct. 23, 2016. URL: https://www.nngroup.com/articles/negativity-bias-ux/ (visited on June 5, 2023) (cit. on p. 43).

[@Mic] Microsoft. *Automate tasks with the Macro Recorder*. URL: https://support.microsoft.com/en-us/office/automate-tasks-with-the-macro-recorder-974ef220-f716-4e01-b015-3ea70e64937b (visited on Apr. 28, 2023) (cit. on p. 14).

[@Nes18] Dan Nessler. *How to apply a design thinking, HCD, UX or any creative process from scratch — Revised & New Version*. Feb. 6, 2018. URL: https://uxdesign.cc/how-to-solve-problems-applying-a-uxdesign-designthinking-hcd-or-any-design-process-from-scratch-v2-aa16e2dd550b (visited on May 22, 2023) (cit. on p. 48).

[@Par12] Terence Parr. *Tree rewriting in ANTLR v4*. Dec. 9, 2012. URL: https://theantlrguy.atlassian.net/wiki/spaces/~admin/blog/2012/12/08/524353/Tree+rewriting+in+ANTLR+v4 (cit. on p. 23).

[@Vic12] Bret Victor. *Learnable Programming. Designing a programming system for understanding programs*. Sept. 2012. URL: http://worrydream.com/LearnableProgramming/ (visited on June 7, 2023) (cit. on p. 52).

[@Völ21] Markus Völter. *Deployment options for MPS*. May 26, 2021. URL: https://www.itemis.com/en/it-services/methods-and-tools/dsls-mps-deployment-options (visited on May 12, 2023) (cit. on p. 34).

Questionnaire

The used questionnaire was written in French, here is a translation of the questions.

The 3 first questions were rated on a scale of 1 to 7 stars, 1 being the worst and 7 being the best.

1. Could you evaluate the *readability* of `ETLang`'s language?
2. Could you evaluate the *ease of learning* of `ETLang`?
3. Could you evaluate the *ease of editing* of `ETLang`?

The second standard part was the UMUX, with scales of 1 to 7 going from *strongly disagree* to *strongly agree*:

1. The capabilities of this prototype meet my requirements.
2. Using this prototype is a frustrating experience.
3. This prototype is easy to use.
4. I have to spend too much time correcting things with this prototype.

58

B Getting started

The tutorial was converted for conservation from the original documentation website of `ETLang`. It is available at `https://etlang.org/docs/getting-started/`. Styling has been adapted to fit LaTeX typesetting. Videos have been replaced by reconstructed screenshots. The styling of keyboard shortcuts and menu items has been adapted as well as the styling of admonitions.

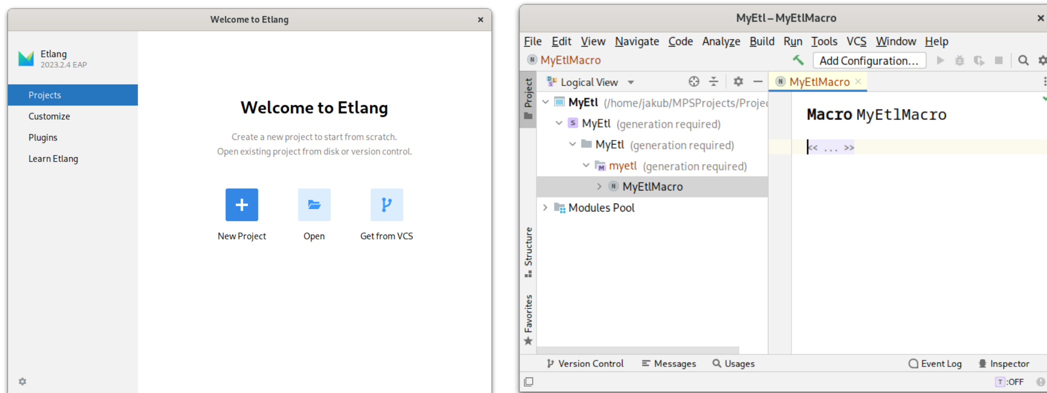
B.1 Getting started tutorial

In this tutorial, will see to create your first ETL process with Etlang, and we will go through some essential language constructs. We assume you have a working IDE, you can get one working with the installation page.

B.1.1 Step 1: Create a new solution

First, we need to create a *new project*. Follow the same steps as in the video, or the textual description.

> **Warning**
>
> Videos were done with a previous version of the language. **Mutustru** has been fully rebranded to **Etlang**.



Video B.1: Create a new project, first and last windows of the original video

Use the New project wizard, give it a name, and make sure to keep Solution project selected.

Once your new solution is created, you can create a model. Look at the Logical View (left panel), with the right-click menu select New > Model. A window opens, and in the Used languages tab, add `Etlang`.

Info

Models declare which languages they use and hold a logical set of ETL processes also called Macros.

From there, you can create a new Macro, which defines your ETL process. Right-click on your model, and select New > Macro. In the main windows, you can give your Macro a name, for example, MyEtlMacro.

B.1.2 Step 2: Declare your file structure

Etlang is designed to handle tabular data, like CSV files, with a fixed number of columns.

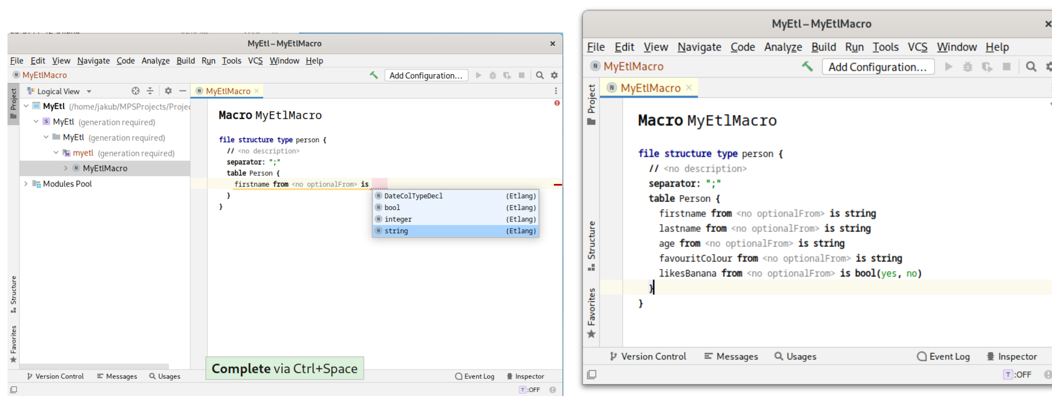
A table for Etlang has a name, and a set of columns. Each column of a table has a unique name and contains a predefined type of value, for example, number, text, date, etc.

Here is our example table, with 5 columns:

Table B.1: My hello.csv table

firstname	lastname	age	favouritColour	likesBanana
John	Doe	42	blue	true
Jane	Doe	42	red	false
Alice	Dreamer	36	green	true

Let's define it in Etlang, so that it knows how to read the CSV file. In your Macro, in the central view, trigger the completion menu with Ctrl + Space then start to write `file structure` and select it among the completions.



Video B.2: Declare a file structure. We see how the completion works, and the final result at this stage.

A file structure is named and defines a table. Give a name for both, `person` for example. Then in the table section, you will need to provide the columns of your table, as well as their type.

The `optionalFrom` is used if you want to use a different name than the header in the CSV file. In that case, use the name you want, and in `optionalFrom` put the exact header used in the CSV file.

B.1.3 Step 3: Read the file

Now that we have declared our table, we can read the CSV file with the table structure defined above.

To read the file, we need to add a new instruction. After the file structure definition, go to a *new line*, trigger the completion with Ctrl+Space, write and select `read structured file`.

This operator needs a file path and a table structure. The file path should be a string thus type " followed by the filename, for example, `"people.csv"`. For the *table structure*, your previously defined `person` should be proposed, by the completion.

```
1 read file from "hello.csv" with structure person
```

B.1.4 Step 4: Modify the table

Now that we have read the file, we can modify the table, for example, we can add a new column, with the birth year of the person.

It should look like this:

```
1 add column birthyear ← 2023 - #Person.age
```

To write it, go to the next line after the `read from file` operation, and trigger the completion with Ctrl+Space. Select `add column` from the completion menu, then write the name of the new column, and press Enter.

You have something like this:

```
1 add column birthyear ←
```

The only thing left is to give a value to the column. Write `2023 -`, then you need to reference the value of the previous table read from the file.

The previous table is given by **`#table`** (it will be displayed as `#Person`). **Access a column** of the table, use a dot. In our case we want the age, thus write `#Person.age`.

Analyse table columns

After some operators, you may lose track of which column is in the table.
To see the columns of a table, use the action **Show type** on an operator: Place your cursor on the operation name, Ctrl + ⇧ + p.
You will see the column `birthyear` has been added.

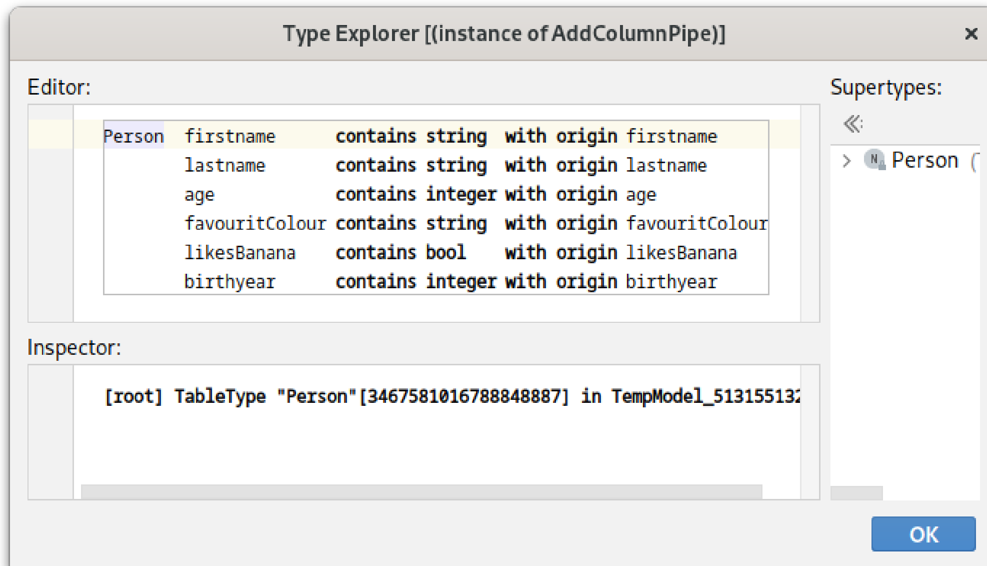


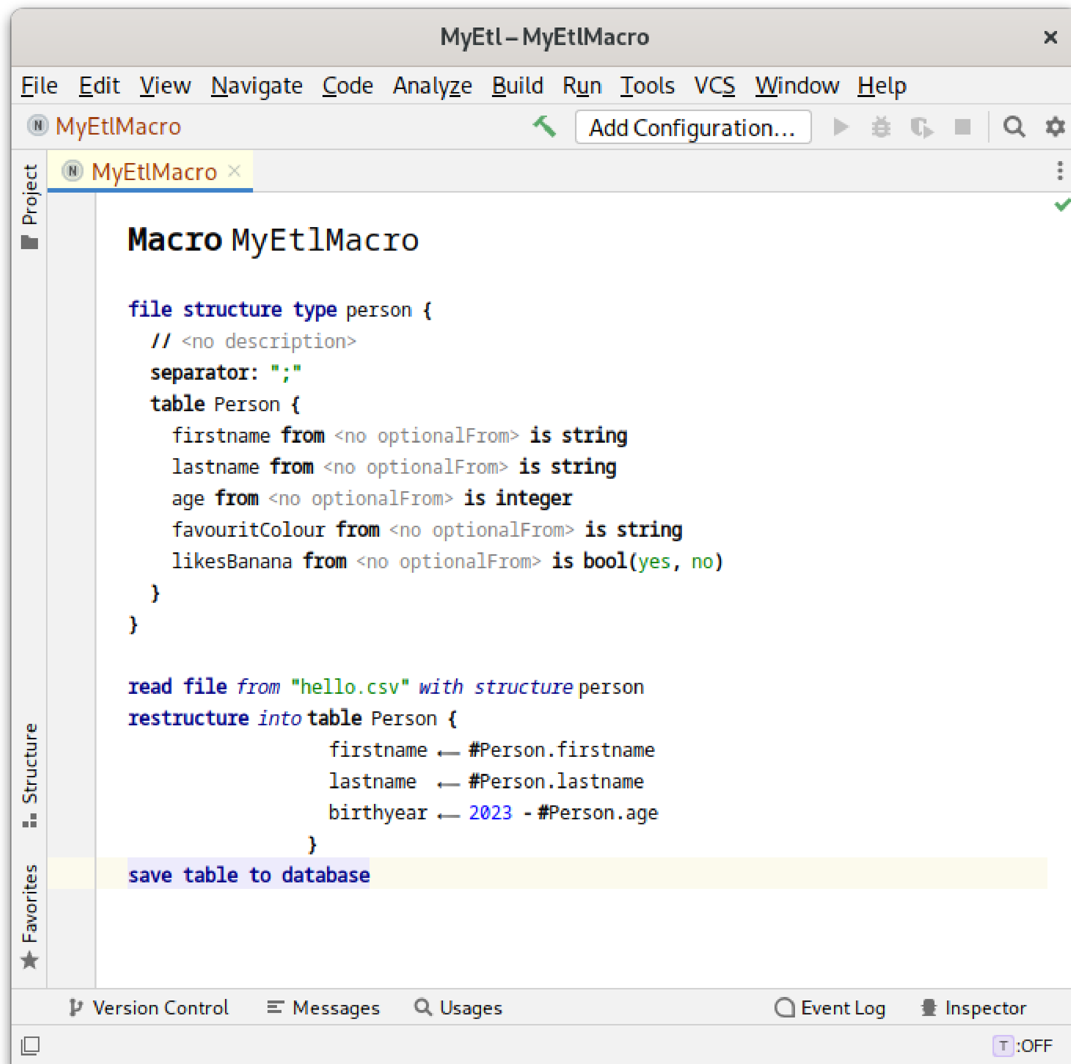
Figure B.1: Table type inspector

B.1.5 Step 5: Save the table

To save the table in our database, we need to have only columns: first name, last name, and birth year.

We can add one more operator to keep only these 3 columns, by *restructuring* the table, with the `restructure` operator. Restructuring a table is like replacing the table with a new one, you will therefore need to provide a new table with the `table` expression (not #table, this is to access the previous table).

Hereafter is a complete recap of what we have done with a video:



Video B.3: The video shows the whole editing of the file from the file structure definition, instead we propose the final result.

B.1.6 Bonus Step 6: Defining symbols

The current year is a bit hardcoded into the expression, we could extract the value and *define an expression* for later reuse.

To do that, go on a blank line before the `initialize from file` operator (to avoid cutting the sequence of operation) and write a `define expression` statement. For example define `2023`, with the name `currentYear`. Then in place of the `2023` while adding a column, use the name of your expression, `currentYear`.

Why not cut the sequence of operation ?

You should keep the sequence of operations together without blank lines, or other things than operators. *Otherwise the reference to the previous table does not work as there is no table produced on the line just above.*

Success

We have written our first ETL process, and we have seen some essential language constructs. You can go deeper by reading the language reference.