Everton Nascimento

Student Nº 43593 (MIEI)

A Model-Driven Approach to the Generation of Front-Ends

Dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Advisers:	Jácome Cunha, Assistant Professor, Universidade Do Minho
	Vasco Amaral, Assistant Professor, Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa

Examination Committee

Chairperson:	Name of the male committee chairperson
Raporteurs:	Name of a female raporteur
	Name of another (male) raporteur
Members:	Another member of the committee
	Yet another member of the committee

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2018

A Model-Driven Approach to the Generation of Front-Ends

Copyright © Everton Nascimento, Faculty of Sciences and Technology, NOVA University of Lisbon.
The Faculty of Sciences and Technology and the NOVA University of Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

This document was created using the (pdf)LaTeX processor, based in the "unlthesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Lorem ipsum.

Abstract

Today most companies have many ways to connect and gather data from their clients. Some companies provide spreadsheets to be filled and submitted, others have websites with forms to collect information or even an Android or iOS app, etc. Multiple types of front-ends are now the norm in the modern world. This brings advantages of connectivity, interactivity, and the data gathering possibilities are great.

But in a fast-evolving world, time is important. Therefore, ensuring a faster and more efficient way to generate, update and guarantee the coherence between front-ends would be very relevant. In this work, we will try to tackle the problem of generating in a coherent manners multiple types of front-ends from the common component which is the database schema. Our aim is to create a solution that automatically generates front-ends from a database schema definition in SQL code. Our solution is grounded in foundations and tooling found in model-driven development and DSLs.

We want to automatically generate from a database schema specification in SQL: spreadsheets and web pages (and in the future possibly any desired template) already containing the restrictions and verifications needed. By doing this, we can decrease the likelihood of errors and guarantee the efficiency and correctness of all elements of the problem.

As a case study, we have the goal of solving a specific real-world case of the Portuguese General Inspection of Finance (IGF). In IGF, we have identified the need for synchronization and consistency between database, spreadsheets, and web pages. By addressing this need we are trying to create a robust solution that solves various problems without the need to learn new techniques or technology with a significant learning curve.

Our solution was also tested with multiple schemas and we are able to generate web pages from the schema definitions of multiple SQL dialects.

Keywords: Spreadsheets, Web Pages, Front-Ends, SQL,Model-Driven Development, Domain Specific Languages

Resumo

Hoje em dia, a grande maioria das empresas possui várias maneiras de conectar e colectar dados dos seus clientes. Algumas empresas fornecem folhas de cálculo para serem preenchidas e enviadas, outras têm sites com formulários para receber informação ou até mesmo aplicações para Android ou iOS, etc. Vários tipos de front-ends são agora a norma. Isso traz vantagens de conectividade, interatividade e as possibilidades de extracção de dados são ótimas.

Mas em um mundo de rápida evolução, o tempo é importante. Portanto, garantir uma maneira mais rápida e eficiente de gerar, atualizar e garantir a coerência entre front-ends seria muito relevante.

Neste trabalho, tentaremos abordar o problema de gerar de maneira coerente vários tipos de front-ends a partir do componente comum, que é o schema da base de dados. Nosso objetivo é criar uma solução que gera automaticamente front-ends a partir de uma definição de um schema de base de dados em código SQL. Nossa solução é feita com fundamentos e ferramentas de desenvolvimento orientado a modelos (MDD) e DSLs.

O nosso objetivo é gerar automaticamente a partir de schema de base de dados escrito em SQL: folhas de cálculo e páginas Web (e no futuro, possivelmente qualquer tipo de front-end desejado) contendo as restrições e verificações necessárias. Ao fazer isso, podemos diminuir a probabilidade de erros e garantir a eficiência e correção de todos os elementos do problema.

Como caso de estudo, temos como objetivo abordar um caso real e específico da Inspecção Geral de Finanças (IGF). Nas finanças, identificamos a necessidade de sincronização e consistência entre base de dados, folhas de cálculo e páginas web. Ao abordar essa necessidade, tentamos criar uma solução robusta que resolve vários problemas sem a necessidade de aprender novas técnicas ou tecnologias que possuem um tempo de aprendizagem significativo. Nossa solução também foi testada com vários schemas e podemos gerar páginas Web a partir das definições de schema de vários dialetos da linguagem SQL.

Palavras-chave: Folhas de Cálculo, Páginas Web, Front-Ends, SQL, Desenvolvimento Orientado a Modelos, Linguagens de Domínio Específico

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 The Rapid Growth of The Number of Front-Ends

Most companies or enterprises in the modern world have numerous ways of connecting with their intended users. Some companies use websites and applications, others have Android and iOS apps coupled with a website (in Figure 1.1 we can see how the number of websites grows with the years). It does not matter what type of business or goal the company has, having a strategy centered in at least one type of front-end is helpful for the success of a business [Gol+03]. The reason is that not only provides means of communication with the intended target audience but also consists of multiple ways to gather user data and other types of important information.

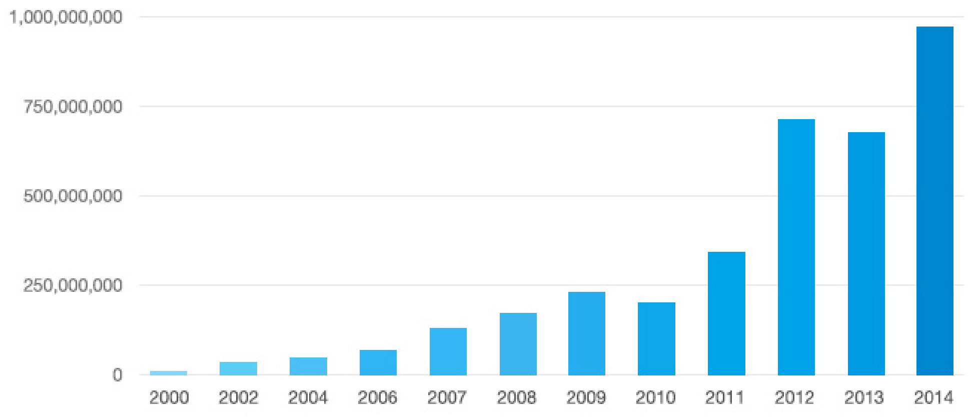


Figure 1.1: The number of websites in the internet along the year has increased immensely. [Num]

1.2 Problem Statement

When companies embrace the decision of maintaining the offer of several front-ends, it comes with a price. The multiple types of front-ends must be updated and coherent with each other. It is easy to find places or situations where having these technologies not synchronized would be troublesome.

For instance, the case of a company that provides simultaneously a spreadsheet, an Android app, an iOS app, Windows app, and web page. Many of these technologies also come in different programming languages, written by different teams at different times. Having different technologies that were worked on by different teams at different times can cause problems. If the need for changes arises, it is easy to guess that most likely all of these pieces have to be looked at individually by someone or some group. Assuming that the need to update this set of tools appears, the initial steps that would need to be taken to make sure everything is up-to-date and synchronized would be:

- First, the domain experts would have to analyze the system and see what needs to be updated, removed or improved.
- Second, after each development team analyses their technology, it would be necessary for them to meet and agree on the changes that need to be adressed.
- Third, it would be needed to ensure that the communication between teams is constant to avoid missing functionalities and big differences in layout.

In the case that, for instance, we have a company that uses as front-ends a spreadsheet, a web page (to receive information) and an Android app to collect information from its users. The fictional company in our example has at least four elements that have to be in synchronicity: spreadsheet, a web page, an Android application and the common link between them which is the database.

Traditionally, all of them would have to be updated manually. In many cases by different teams. This makes coherence between the elements specifically difficult and error prone. Every time that, for example, a table in the database is updated or altered, the app and website will need to be updated manually. Having to update and maintain the elements manually is less than ideal. It is known for some time that in the development of software, the higher costs go to the maintenance phase. In the mythical man month book by Fred Brooks, it is stated that 90% of the costs arise in the maintenance phase and the software subsequently will have to be maintained [Bro95].

The notion of difficulty for software maintenance is nothing new. In 1974, Lehman and Belady started to formulate the laws of software evolution. In 1980, professor Lehman defined an E-Program as: "An E-program is written to perform some real-world activity; how it should behave is strongly linked to the environment in which it runs, and such a program needs to adapt to varying requirements and circumstances in that environment" [Leh80].

For this type of systems, 8 software evolution laws were defined. Some of these laws are relevant to our problem, such as:

- Law I - **Continuing Change** - "E-type systems must be continually adapted else they become progressively less satisfactory" [Leh+97]. All of software eventually will have to be adapted, so having a efficient way to perform the adaptions will always be of value.

- Law VI - **Continuing Growth** - "The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime" [Leh+97]. The growth of software is also to be expected. Some parts have to be updated and its functionalities increased. Other parts will have to be added from scratch depending on what is needed.

- Law VII - **Declining Quality** - "The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes". [Leh+97]. Ease of maintenance is also a main focus of our work.

In this perspective, it is interesting to note that new ways of developing apps are also being developed and utilized. Developing apps manually is something that takes some time and low-code platforms and solutions that involve less hand-made code are gaining a lot of interest. The market for these tools is projected to grow significantly [RR16]. The possibility of developing something without having to code everything by hand is something that is very interesting and useful.

We defend in this thesis that the utilization of tools that use model-driven development and DSLs can make the process of maintenance of complex systems more efficient and less complicated. By creating a tool that utilizes model-driven development paradigms coupled with DSLs we can improve development time, development effort and make the process of actualization and verification easier. This thesis is limited to the creation of spreadsheets and web pages but in the future it could be extended to support more transformations and a bigger number of front-ends.

1.3 Proposed Solution

We wish to provide a solution that automatically generates front-ends without having a significant learning curve. We will use model-driven development coupled with the development of DSLs to achieve the goal of generating models that represent the elements of the database. With these models, we will generate two types of front-ends: a web page and spreadsheet.

We wish to create a tool that in the future could be extended to express more front-ends and include more types of databases. We created a DSL using Xtext. This DSL acts as a copy of SQL (and many of its most used dialects), the user only needs to copy the SQL code and a model that contains the information of the schema will be created by a generator written in the Xtend language. We then transform this model utilizing ATL (a DSL that was created for model transformation) rules. The model that contains the information of the schema is transformed to an intermediate model of the GENSS project [Tei16] written in the SSDSL language. We then utilize this intermediate model coupled with the EGL language (a project contained in the Eclipse IDE) to generate a spreadsheet and a web page respectively. There are tools that generate websites, some that produce web apps and etc. But most of these tools require the user to study and learn new techniques or technology.

To test and validate our solution, we will utilize a repository called OLTPBenchmark [Dif+13]. This repository has many SQL files of different types of dialects. We will also apply this to a case study from the Portuguese General Inspection of the Finance Department (IGF) [Igf].

In IGF, if there is a need to update the database, it would be necessary in this case to hire someone to re-do the website manually.

Our proposal addresses this by automatically generating spreadsheets and web pages from the database specifications. The database was chosen to be the principal element in our solution because it is the central point of almost all systems that have multiple front-ends. The information always has to come back to a common database. As many front-ends have multiple users that are not technology proficient our solution does not require a software expert to use it.

1.4 Contributions

This thesis will bring as contribution a framework that can receive a database schema and automatically generate the desired template.

The main contributions are:

- The design and implementation of a tool that can be used in a number of real-world use cases;
- A DSL representation of SQL in Xtext;
- ATL transformations of our DSL to the GenSS DSL;
- Creation of a complete solution that can generate a website from a database schema which can be improved and extended in the future with multiple types of transformations between the models, the generation of templates, the definition of the schema, etc.

1.5 Structure of the Document

This thesis consists of 4 more chapters. In chapter 2 we have a look at the background knowledge necessary to understand the work of this thesis and talk about projects and solutions that are relevant and can be related to this work. We briefly analyze solutions that utilize MDD, projects for spreadsheets, we talk about bi-directional transformations, generation of web pages utilizing content management systems and low-code development platforms.

Chapter 3, is where we explain how our solution was developed. We briefly explain each technology used in the project and what it was used for. After we explain how each piece of the solution is connected to each other, what each part generates and how the models are transformed and used to generate the front-ends.

Chapter 4 is dedicated to the validation section of the thesis. We show how the solution was tested and what was added and removed from the solution as the project evolved.

The final chapter 5 is used to show our conclusions and what could be done to improve this work in the future.

Chapter 2

Related Work

This chapter presents the similar and related approaches to the problem (or parts of) discussed in this thesis.

2.1 Model-Driven Development

Since the start of computer science, programmers and engineers have always tried to create ways to do more with the machine without having to write more code. Raising the level of abstraction to deal with complexity and increase productivity.

After the multiple types of developments and enhancements that occurred since the days of the first FORTRAN compiler, we have gotten to a point where model-driven development [Mel+03; Sel03] is a logical continuation of this trend.

In this work, a model is a simplified representation of something. In our case, the models will be representations of artifacts of the systems and solutions we want to model.

A model always conforms to a meta-model. The meta-model is the model of a model. The abstraction of our abstraction that is used to highlight the properties of the model itself (Figure 2.1, shows how the meta-meta-model, meta-model and model are related). In simpler terms, a meta-model decides what can and what cannot be represented in a valid model of a certain modeling language [Sei03]. Models and meta-models make up the basics of *model-driven development* (MDD).

Designing software is becoming more complex. There are many requirements that need to be satisfied and different bodies of knowledge we have to apply to the formulation of a solution. The main goal to be accomplished is to design something that not only works but works well.

Model-Driven development is a effective approach to deal with complex systems development. Model-to-code generation technologies years ago were expected to generate

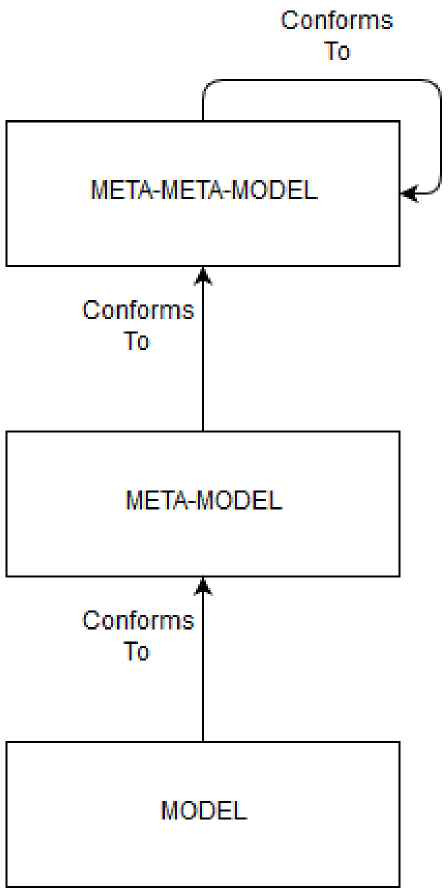


Figure 2.1: Relationships between model, metamodel and metametamodel.

code with memory and performance efficiency that could be on average between 5% to 15% better than hand-written code [Sel03]. In these days it would not be surprising if these number have grown.

MDD is model-driven because the models are the main drivers of the software development. These models (target or source, sometimes both) will expand, evolve and grow with the process. The implementation process needs to be derived from the starting models. It is useful because model-driven development automates the transformation of models from one form to another. One of the main advantages of using this type development (and models for software development) is the possibility of enhanced productivity [Bak+05].

MDD is applied to the problem of designing models at a high level of abstraction. Models guide the process of creating software, shifting the focus to specific parts of the problem and presenting a better way to manage the complexity of the problem.

A good example of an application of MDD is when a business needs software to be crafted which will make process within the company more efficient. Here, the workers of a company know how to perform an action that could be improved, or how to solve a particular problem. We can call these people the domain experts, by being experts they know all the details and the better way to solve the problem. The problem here is, the domain experts are not software experts (most of the time). They do not understand how to develop a software solution. The software engineer knows how to develop but does not have the knowledge of the expert.

Model-Driven development bridges this gap between domain expertise and software development knowledge by creating models that represent the systems we want. This helps us design solutions for the problems that arrive during the development process.

It is also interesting to note that MDD has an extensive list of benefits claimed by many authors, such as:

- Improvement of code quality [Bor04; MF05; SS09].
- Improvement of code consistency [Bor04; Mid].
- Elimination of repetitive code [SS09].
- Improvement in reuse, development of new versions and maintainability [HT06].
- It is a technology easier to understand [MF05; Sel03].
- Reduces developer effort [Bor04; HT06; Sel03; SK03; SS09; Mid].
- Improves productivity [Bra+12; MF05; Sel03].

MDD has also been used in Industrial settings. Motorola has a 15 year report on their application of MDD and its implications in effort, quality and productivity [Bak+05]. This report states that their results in code and automatic test generation have decreased defects by 1.2 and 4 times, productivity is between 2 and 8 times greater and effort is 2.3 times less.

2.2 Model Transformation

In this thesis, we use model transformations. A transformation can be defined as the automatic generation of a target model from a source model with a specific intent [Syr11]. This transformation is made by a set of transformation rules that are used to describe how the source model in a source language can be used to generate a transformed model in the target language. These transformation rules are the description of how the constructs in the source language and model will be transformed to target language and model. Using these basic definitions we can define model transformation as the manipulation of input models to produce output models.

A model transformation can have multiple source models and multiple target models. An example of a type of model transformation that uses multiple models is model merging where multiple source models are combined to give us a resulting target model. For a transformation example of the latter, we could take the one source model and transform it in multiple models (Figure 2.2 shows the transformation from a platform independent model (PIM) to multiple platform-specific models (PSM). The transformations can also be unidirectional (from source to target) or bidirectional (from target to source or vice versa).

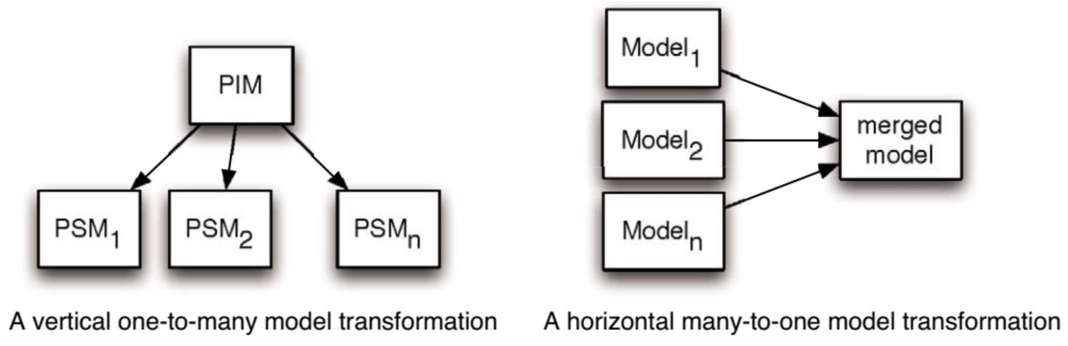

PIM
PSM1
PSM2
PSMn
A vertical one-to-many model transformation
Model1
Model2
Modeln
merged model
A horizontal many-to-one model transformation

Figure 2.2: Examples of model transformations [MVG06].

There are also other two types of model transformations categories: model-to-model and model-to-text (sometimes called model-to-code transformation). The first category consists of rule-based transformations between models. A model-to-model transformation as the name suggests transforms a model into another model. These transformations are made by rules that describe how the constructs of one model are mapped to the constructs of another model. The second category includes an automatic generation of text from a model which can be executable text (spreadsheet code, HTML, Python, etc.) or any other type of textual artifact. We will use a DSL that was made in the GENSS project [Tei16] and other tools to generate the code necessary for the desired templates.

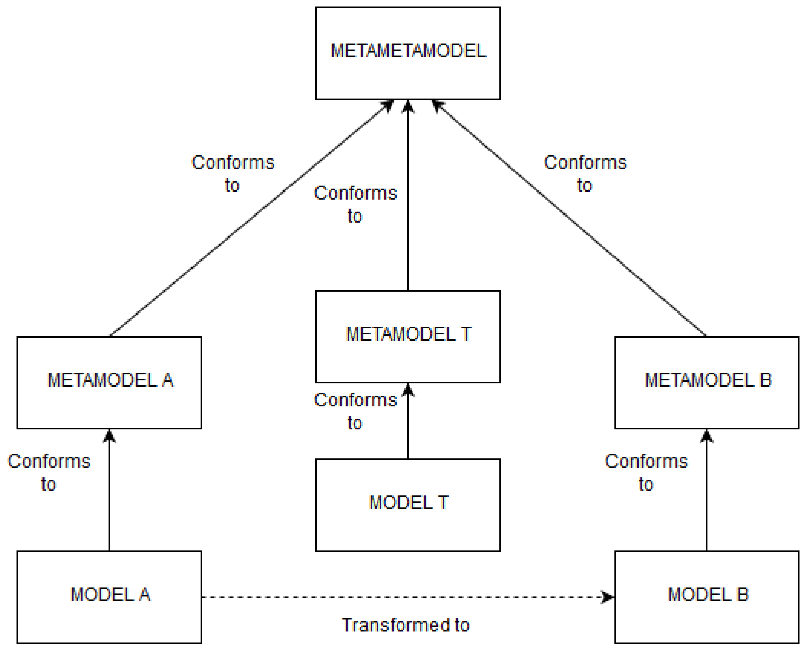


Figure 2.3: Summarizing the model transformation process.

In our project, we will use only one source model (The SQL code that will be represented in a model we generate) and will generate a model in the target language that was already developed in other project. This is a one-to-one transformation. We will also use both types of transformations. Model-to-model in the transformation of our source model (which is generated by our DSL) to the target model (represented by the SSDSL language created in the GENSS project). The model-to-text (or code) transformation, will happen when we use the transformed model (written in the SSDSL language) to generate the code for the desired type of spreadsheet and the code for the web page. These are the two types of targets we want to generate. Our transformations will also be unidirectional since we desire to go only from the database schema to generate our front-ends. Although, making the transformations bi-directional would be a very interesting thing to add to the project in the future.

2.3 Domain-Specific Languages

Programming languages can be classified in multiple ways: declarative vs imperative, high-level vs low-level, domain-specific languages (DSLs) vs general purpose languages (GPLs) and etc.

In our project, we will also use domain-specific languages. A domain-specific language is a programming language that is made with focus on a particular problem domain [VD+00]. We decided to create a DSL to help us generate the model that will represent the schema information that it is written in SQL. We've also already had a DSL created in another project [Tei16] that is used to help generate spreadsheet code and will be used to generate the web page.

The most used programming languages of today can be designated general purpose languages. As the name suggests this types of languages are employed in a wide variety of domains (E.g. Python, Java, C++ etc). These languages are used to provide tools to solve general problems and are not specifically made for a single scope.

The difference between domain-specific language and general-purpose languages is not always so clear. Some languages can be designed for a specific scope but used for in a bigger scope than planned. Perl was originally created to be a language used for text-processing but then it became a GPL.

Domain Specific Languages are languages that focus on a specific domain. There is a great number of programming languages that can be called domain specific. Some examples of well known DSLs are:

- **SQL** [Cha12] - A language that is used to manipulate databases. Its main purpose is to express how to insert, change or extract data from a relational database. It includes many types of dialects (MySQL, nuoDB, Oracle SQL, PostgreSQL, the Microsoft SQL Server, etc.).

- **The Extensible Markup Language** (XML) [Xml] - defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. XML-based formats have become the default for many office-productivity tools, including Microsoft Office (Office Open XML), OpenOffice.org, Libre Office (OpenDocument) etc.

- **LaTeX** [Lat] - this language was created to provide writers functions for defining the formatting and layout (headers, footnotes, references, bibliographies and etc) of their work. it is mainly used in academic circles for the publication of scientific documents and a wide variety of fields (physics, computer science, economics, mathematics and many more fields).

These are some the languages that are used and related to the work we developed in this thesis.

In the context of DSLs, they can also be divided into two main types:

- Internal: these are contained inside the host language (GPL) and reuse its syntax, type system and run-time systems (sometimes they are difficult to differentiate from an API) [Fow10].

- External: These have a parser, syntax and execution infrastructure (sometimes) independent of the host language [Fow10].

In MDD, DSLs can also be used to help in the generation of the desired models or artifacts.

The external are the ones in focus in this project, as they can be related to MDD because some of its complementary tools are also DSLs, such as OCL, QVT (domain-specific transformation language) and even ATL could be characterized as a DSL.

As we referred before, DSLs are utilized to focus on a particular aspect of a system/problem, they are used to encapsulate domain knowledge and facilitate the development of projects were domain experts and software engineers collaborate.

2.4 Model-Driven Development Projects

This section highlights some projects that use model-driven development concepts for their functionality and have models as a central part of its solutions.

2.4.1 INTEGRANOVA

The INTEGRANOVA M.E.S. (Model Execution System) is a business application that uses model-driven architecture to generate applications in multiple types of programming languages and environments. "The most advanced MDA (Model Driven Architecture) solution, capable of programming 100% of the final application's code, including the more complex business logic, instead of being limited (as other systems) to obtain basic functionality and/or frameworks of programs" [Int].

The goal is to generate a model-driven development framework that can also make it easier for domain experts and developers to communicate. Raising the level of abstraction in the development process is the way which they try to close this gap between domain expertise and software development.

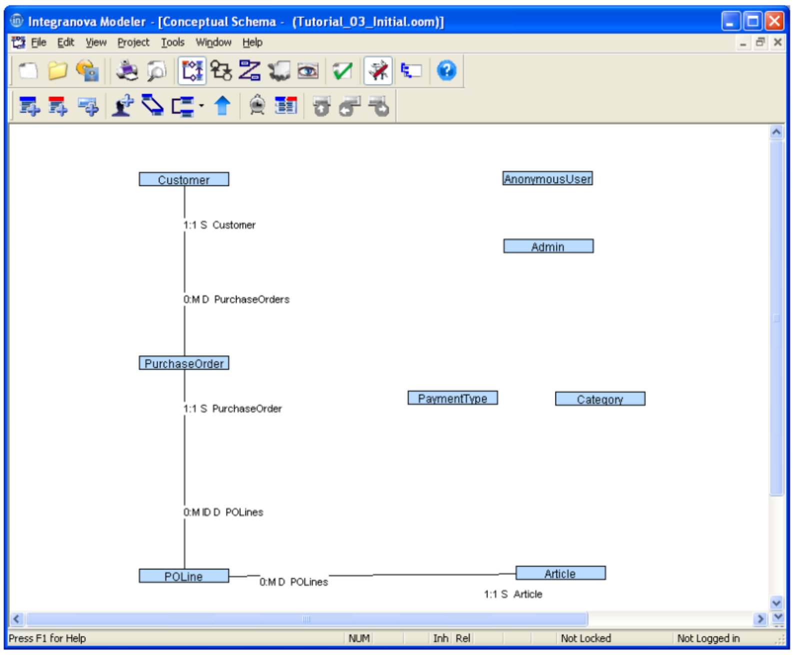


Figure 2.4: The class diagram of the Integranova software [Int].

The software has a simple structure. It has a class diagram, where the user can create the classes that he/she wants (see figure 2.4). In each class the user can define attributes and the actions that can happen to the various attributes.

After defining how the models interact, what actions and what permissions they have, it is possible to generate an application in multiple types of programming languages and environments (as we can see in figure 2.5).

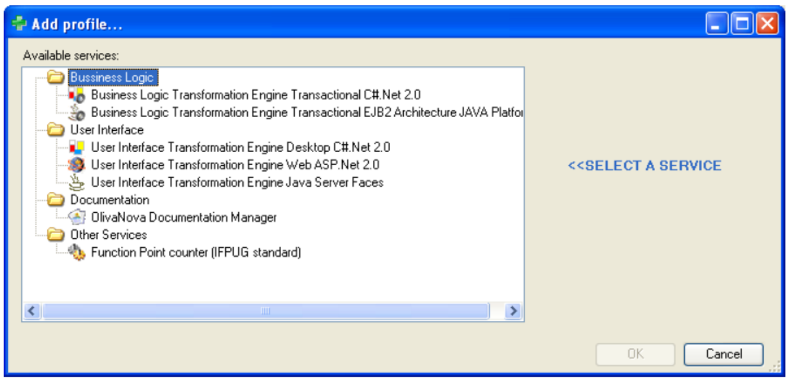

Add profile...
Available services:
Bussiness Logic
Business Logic Transformation Engine Transactional C#.Net 2.0
Business Logic Transformation Engine Transactional EJB2 Architecture JAVA Platfo
User Interface
User Interface Transformation Engine Desktop C#.Net 2.0
User Interface Transformation Engine Web ASP.Net 2.0
User Interface Transformation Engine Java Server Faces
Documentation
OlivaNova Documentation Manager
Other Services
Function Point counter (IFPUG standard)
<<SELECT A SERVICE
OK
Cancel

Figure 2.5: Generation of the final application in the Integranova environment [Int].

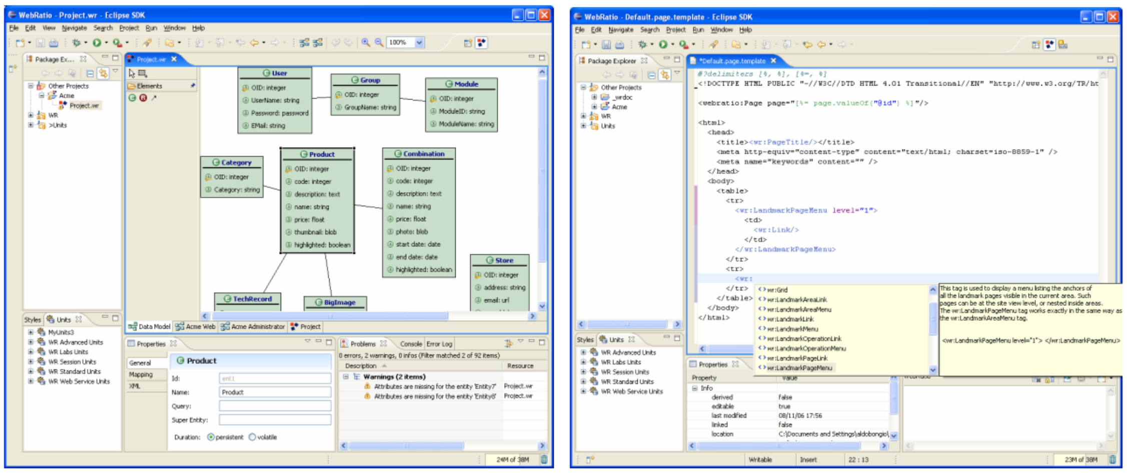
Although it is an easy software to master and a effective tool to generate code in multiple types of programming languages, the INTEGRANOVA software does not support the SQL language and the models have necessarily to be made by hand (at least in the first iteration).

2.4.2 WebRatio

The project WebRatio [Ace+04] is also a project the uses model-driven development. This project allows the user the capture business requirements in models to generate business applications.

WebRatio uses graphical languages to specify the generation of code for data-intensive Web applications, process management primitives, calls to Web services, and integration of heterogeneous data sources. These applications are made using the Entity-Relationship (ER) model for data requirements, and the WebML language for the functional requirements. It also has been created along Eclipse (see figure 2.6) as a set of plug-ins for the IDE [Ace+04; Ace+07; Bra+08].

The users of WebRatio can focus on the requirements of the app being developed and on high-level design because the documentation and code are automatically generated and verified by the tool.



```
#?delimiters [%, %], [%=, %]
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/ht

<webratio:Page page="[%= page.valueOf("@id") %]"/>

<html>
  <head>
    <title><wr:PageTitle/></title>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
    <meta name="keywords" content="" />
  </head>
  <body>
    <table>
      <tr>
        <wr:LandmarkPageMenu level="1">
          <td>
            <wr:Link/>
          </td>
        </wr:LandmarkPageMenu>
      </tr>
      <tr>
        <wr:
      </tr>
    </table>
  </body>
</html>
```

This tag is used to display a menu listing the anchors of all the landmark pages visible in the current area. Such pages can be at the site view level, or nested inside areas. The wr:LandmarkPageMenu tag works exactly in the same way as the wr:LandmarkAreaMenu tag.

<wr:LandmarkPageMenu level="1"> </wr:LandmarkPageMenu>

Figure 2.6: The WebRatio IDE [Ace+07].

WebRatio like most of the solutions only focus on a part of the problem we are trying to tackle. In this case it utilizes model-driven development to generate web and mobile applications and but it does not use any part of a common database between this two types of front-ends. The applications have to be created in a environment similar to the Eclipse IDE that has a signficant learning curve.

2.4.3 A Methodology For The Development Of Complex Domain Specific Languages

As a work that precedes the GENSS project (see section 2.5.3), we have a project worked on by Dr. Matteo Risoldi during his PhD which is a methodology for the development of complex domain specific languages [Ris10].

The goal of the thesis was to tackle the domain of interactive systems and apply a DSML-based workflow which moves from a system specification to the prototyping of a Graphical User Interface (GUI) [Ris10]. The domain chosen by the thesis was the control systems (CSs). A control system manages, commands, direct, or regulates the behavior of other devices or systems using control loops. It can range from a single home heating controller using a thermostat controlling a domestic boiler to a large industrial control systems that can be used to control machines in factories [DB98].

We can group objects with others to form more complex control systems. These more complex control systems will have a composite structure where each object can be coupled with others and the composite objects can also form larger objects. This forms a hierarchical tree in which the root represents the whole system and the leaves are its most elementary devices [Ris10].

This thesis delivers a definition of a method for DSML engineering along with a workflow that allows for the generation of interactive user interfaces, system simulation, validation, and verification. In this method, a language engineer makes the effort of creating the DSML for a specific domain and integrating it to the workflow, while specifying the system and prototyping the software goes to a system expert. The latter can use the DSML to interact with the workflow using concepts and metaphors near to his view on the system, without requiring (to a certain degree) a deep computing science knowledge.

2.5 Spreadsheet Solutions

For spreadsheet generation and verification there are some solutions that were created. In this section we outline the ones more related to our project.

2.5.1 ViTSL

To address some of the problems created by errors in spreadsheet conception, ViTSL (Visual Template Specification Language)[RA05] was devised. This is a language made to model templates and control the evolution of spreadsheets.

In ViTSL the conception of spreadsheets is split into two phases:

- We create a computational template and this defines headers, data cells, and computations. This part is made in the ViTSL extension.
- The spreadsheet is filled with info, and now we have the possibility to delete or insert data into the rows and columns. This part is performed with Gencel.

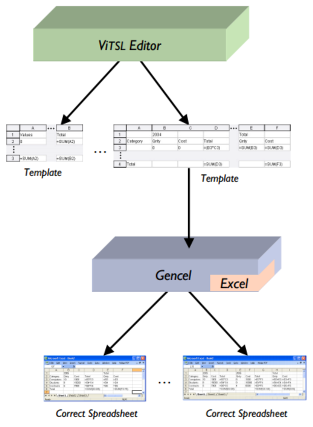
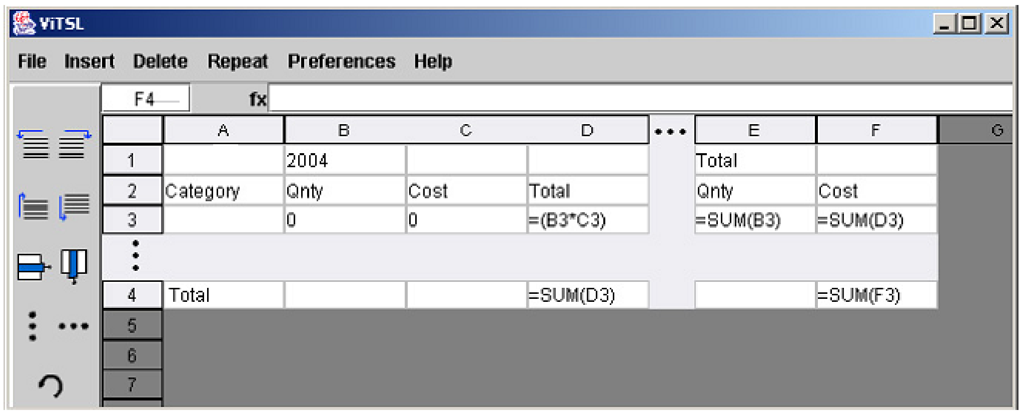


Figure 2.7: Representation of the ViTSL/Gencel architecture [RA05].



	A	B	C	D	•••	E	F	G
1		2004				Total		
2	Category	Qnty	Cost	Total		Qnty	Cost	
3		0	0	=(B3*C3)		=SUM(B3)	=SUM(D3)	
⋮								
4	Total			=SUM(D3)			=SUM(F3)	
5								
6								
7								

Figure 2.8: The ViTSL editor [RA05].

The two-level approach is useful because a user can update spreadsheets, but with limitations that make it impossible to introduce any type, omission and reference errors. As an example of the usefulness of this approach, for example, a company could construct with domain experts ViTSL specifications, and the employees need not know how they are implemented to work with the spreadsheets.

The second level of the approach would make Gencel act as a way to manage the quality of the data inserted, making sure users only insert data compliant with the specifications made by the domain experts.

Although ViTSL eliminates formula and reference errors from spreadsheets, it still offers a poor comprehension of the logic and structure of spreadsheets, another drawback is that after creating spreadsheets we cannot change its template.

2.5.2 ClassSheets and Embedded ClassSheets

After the implementation of ViTSL, the project ClassSheets[EE05] was developed in an attempt to further improve the solution and remove some mistakes that are still relevant to the problem.

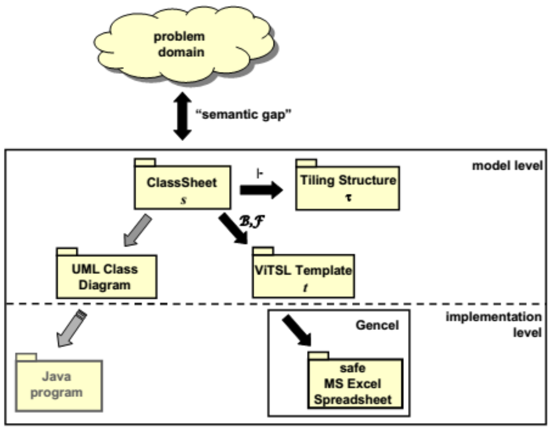


Figure 2.9: Graph of the ClassSheet solution [EE05].

A ClassSheet represents some characteristics of a spreadsheet, namely the details of the attributes, its references, and inheritance. For the classes and objects, it represents its relationships and structure. It also derives the grid-layout, the safety and correctness features of the ViTSL project, also offering the possibility to create a diagram as documentation for the classes of the ClassSheet.

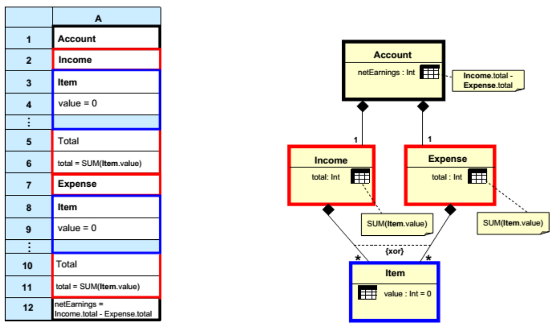


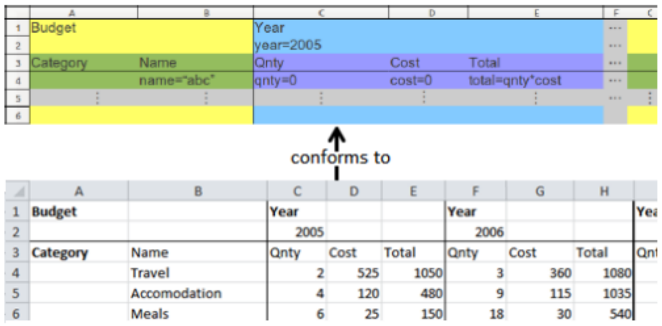
Figure 2.10: A 2D ClassSheet [EE05].

ClassSheets was created to offer an additional layer for the modeling of spreadsheets, therefore making it explicit for the developer (with indicated colored structures contained in the ClassSheet model) the different design possibilities and effects of decisions, and consequently reducing the gap between the requirements of the domain and its representation in the spreadsheet model.

The principal problems with this approach are that it still does not address the domain errors, making it impossible to express constraints related to the business logic, some popular spreadsheet structures are not supported, data insertion and model creation are in separated environments and when we modify a template a new spreadsheet is generated and all the data of the destroyed spreadsheet is lost.

To correct mistakes and improve the robustness of ClassSheets a related work emerged called Embedded ClassSheets [Cun+].

This new work makes available for end users to use a language similar to SQL, to help create queries in a familiar environment. Unlike ClassSheets, this Embedded ClassSheets allows the parallel evolution of a spreadsheet and template, meaning that the model evolve with its instance and vice versa.



	A	B	C	D	E	F	C
1	Budget		Year			...	
2			year=2005			...	
3	Category	Name	Qnty	Cost	Total	...	
4		name="abc"	qnty=0	cost=0	total=qnty*cost	...	
5	⋮	⋮	⋮	⋮	⋮	...	⋮
6							

	A	B	C	D	E	F	G	H	
1	Budget		Year			Year			Yea
2			2005			2006			
3	Category	Name	Qnty	Cost	Total	Qnty	Cost	Total	Qnt
4		Travel	2	525	1050	3	360	1080	
5		Accomodation	4	120	480	9	115	1035	
6		Meals	6	25	150	18	30	540	

Figure 2.11: Model and instance in conformity [Cun+].

But some of the former stated problems still stand, it still doesn't address the constraints and business logic errors, it does not scale well for larger spreadsheets and does not cover some of the more popular spreadsheet environments.

2.5.3 GenSS - A Tool to Generate Spreadsheets

To offer a solution for some of the problems that remain after the creation of the earlier referenced works, one project was developed called GenSS.

GenSS[Tei16] is a visual language which is based in UML. It provides the possibility of defining a spreadsheet via a graphical interface (see figure 2.12). Besides this, it also offers the possibility to use OCL invariants for the creation of constraints. OCL is used to make the model more robust thus preventing errors.

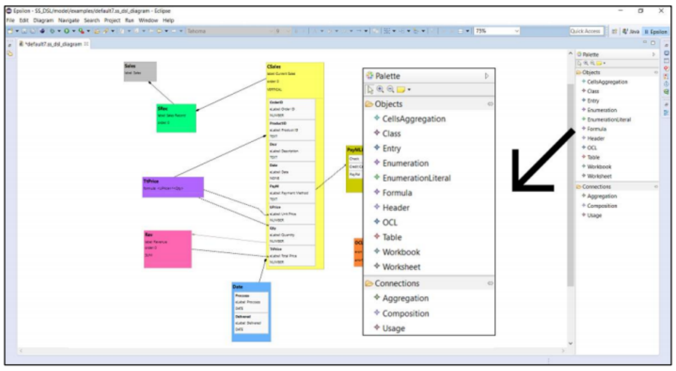


Figure 2.12: The GenSS user interface [Tei16]

The Advantage GenSS offers in comparison with the works we already explained, is the possibility to create domain constraints to prevent domain errors. The big disadvantage of the project is that one needs to be an expert and really know the technologies involved (Eclipse, EGL, etc) to be able to properly use the GenSS tool.

2.6 Bidirectional Transformations

Bidirectional transformations are a mechanism for maintaining the consistency of two (or more) related sources of information. Researchers from a wide range of areas are investigating the use of bidirectional transformations to solve a diverse set of problems.

These transformations apply to our case because they can compute and synchronize artifacts of software [AC08; Sch95; Ste07; YXM07], update tables and views [BS81; Boh+06; DB82; Kel86] and translate values between languages[Ben05; Ram03].

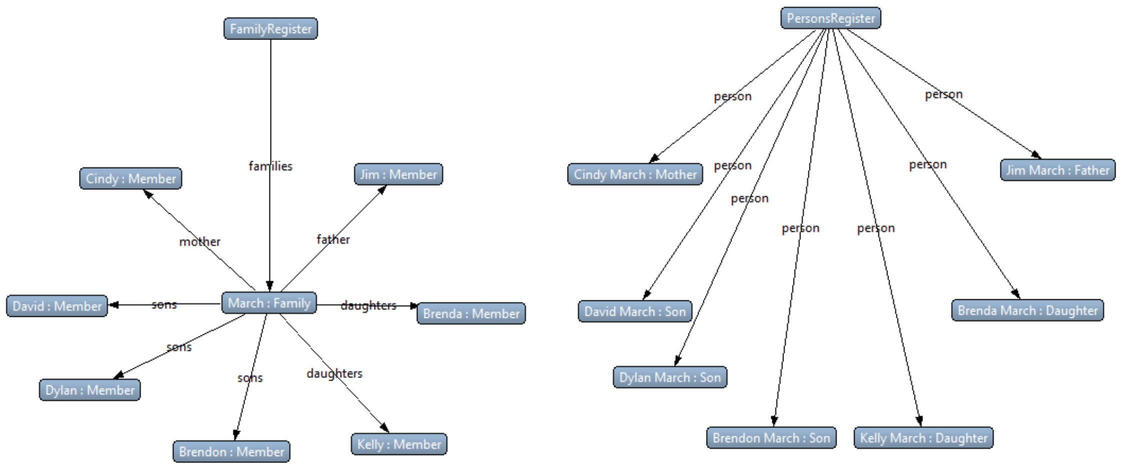


Figure 2.13: This is a well known example from the ATL use case library, the family to persons transformation [Web].

In figure 2.13, we have a source and target model. The source model is the FamilyRegister model. This model contains the number of families and its members. Each member belongs to its family through its role and this roles can be of a mother, father, son or daughter.

The target model has almost all the same information as the source model but it does not contain the concept of family. Every person belongs to PersonsRegister. The way to differentiate between people is via their last names, that we assume are unique for simplicity.

To relate this case to bidirectional transformations, for example, if we move a member in the source model to another family, it should cause the appropriate change in the target model and vice versa.

To implement solutions that use bidirectional transformations in different scopes, we have projects being developed such as:

- Boomerang is a bi-directional programming language used to create bidirectional transformations that work on ad-hoc, textual formats[Boh+08].
- Augeas, a configuration editing tool, that parses the configuration files we feed into a tree and make transformations from it[Lut08].

2.7 Website Generation

For website generation, there are not many projects/or tools being developed specifically for generation of a web page or web app in parallel with a spreadsheet template. Although, many languages have libraries that can help in generating code for web development.

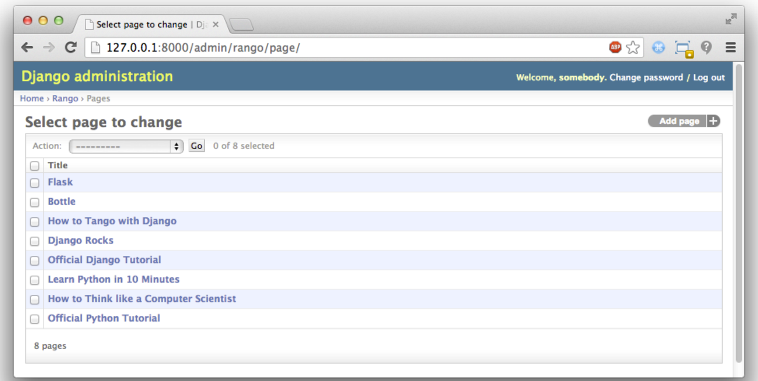
2.7.1 Ruby on Rails

Rails is a web application framework for the development of websites that acts on the server side. It provides everything that is needed do create a database, a web service and web pages. It facilitates the use of technologies such as JSON, HTML, CSS, JavaScript [Han]. In this application it is possible to define a data model, and after that it is easy to generate a web application.

2.7.2 Django

Like rails, Django is also a web application framework. It is based on the Python programming language and uses the model-view-template architectural pattern.

The goal of Django is to make it easier to create complex websites. It is focused on diminishing the number of lines of code that should be done, re-usability, loose coupling and quick development [Dja19a]. This framework also provides the create, read, update and delete (CRUD) interface for the admin models.


Select page to change | Dj
127.0.0.1:8000/admin/rango/page/
Django administration
Welcome, somebody. Change password / Log out
Home › Rango › Pages
Add page
Action:
Go
0 of 8 selected

Select page to change

Title
Flask
Bottle
How to Tango with Django
Django Rocks
Official Django Tutorial
Learn Python in 10 Minutes
How to Think like a Computer Scientist
Official Python Tutorial

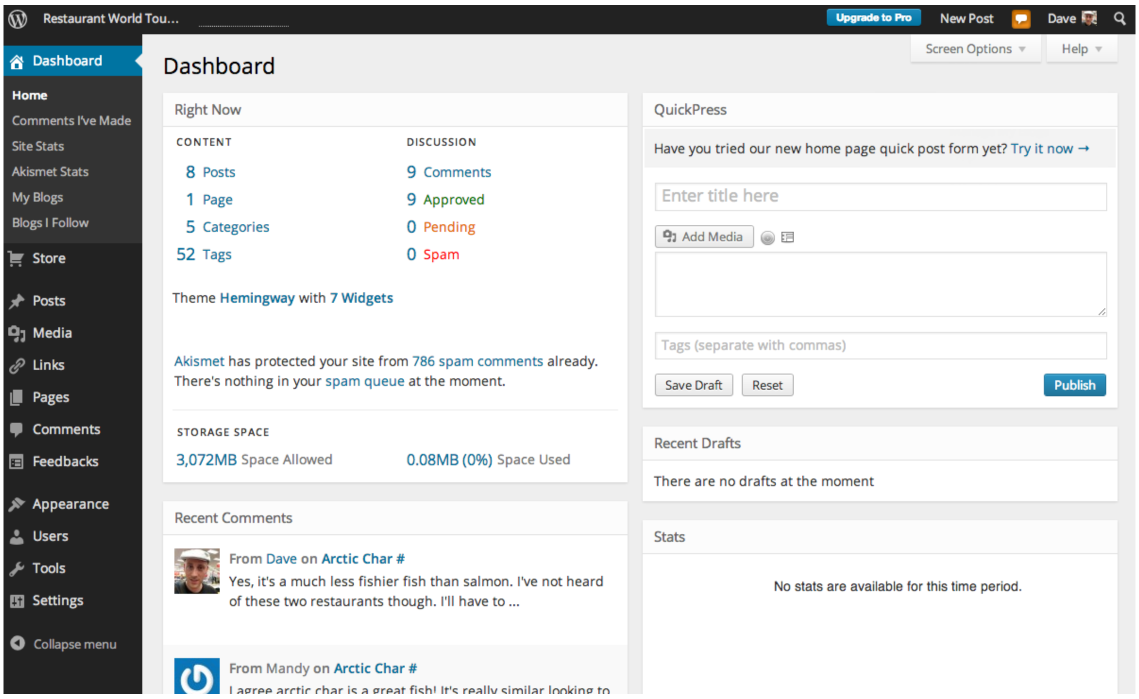
8 pages

Figure 2.14: The Django admin interface. [Dja19b]

The problem in relation to the problem we tackle on this thesis is that they are focused only on website generation. On ruby for example is not easy to use an existing database, due to the fact that it would have to be consistent with the rails conventions. Our objective is to generate more than one type of front-end.

2.7.3 Content Management System

A CMS (Content Management Systems) is a web application, that provides the possibility for different types of users with varying levels of knowledge create and manage data of a website project (website, blog, etc.). In the CMS domain, manage can mean many things such as create, publish, generate or archive data. These systems are perfect for users with little or no experience with any type of web development language (HTML, JavaScript, CSS). Tons of content management solutions are offered online. Some of the more popular ones are WordPress, Drupal, Joomla! and Blogger.



CONTENT
8 Posts
1 Page
5 Categories
52 Tags

DISCUSSION
9 Comments
9 Approved
0 Pending
0 Spam

Theme Hemingway with 7 Widgets

Akismet has protected your site from 786 spam comments already. There's nothing in your spam queue at the moment.

STORAGE SPACE
3,072MB Space Allowed
0.08MB (0%) Space Used

Have you tried our new home page quick post form yet? Try it now →

There are no drafts at the moment

No stats are available for this time period.

From Dave on Arctic Char #
Yes, it's a much less fishier fish than salmon. I've not heard of these two restaurants though. I'll have to ...

From Mandy on Arctic Char #
I agree arctic char is a great fish! It's really similar looking to

Figure 2.15: WordPress is one of the most popular CMS in the world [Cle].

They offer a comfortable way to create a website that requires minimal or no technical knowledge for the end users. It is almost an out of the box solution to generate websites.

This solution of course is always limited to the templates that the suppliers leave available to the users, they are not very customizable for specific business solutions.

2.8 Low-Code Development Platforms

The low-code development platforms [CR14] are types of software that provide for programmers and users without much programming practice, tools for fast development of applications through graphical interfaces instead of the traditional way of development based solely on learning to code.

These platforms are focused on the design and development of many types of applications: databases, business processes, web applications, etc. This software can help provide entirely operational applications.

The aim of these platforms is to decrease the amount of hand-coding done by the developers, thus enabling a sped up development and delivery process of applications. Another benefit is the possibility of inclusion of a wider-range of collaborators in the process, not only those with computer science and programming skills. They also diminish the costs of training, deployment.

The main utilities that the main low-code platforms provide are:

- Visual modeling of business logic and workflows, with the ability to extend with custom code.
- Visual definition of data models and integration components.
- Drag-and-drop implementation of modern user interfaces for multiple devices.
- Application change and life-cycle management.

2.8.1 OutSystems

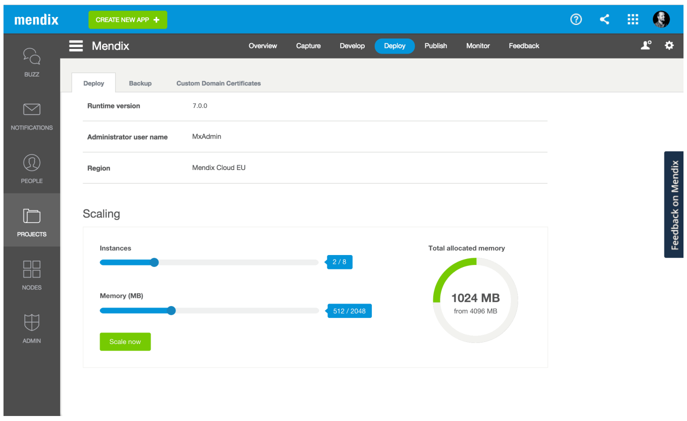
The OutSystems Platform [Out] is a platform as a service (PaaS) intended for developing and delivering enterprise web and mobile applications, which run in the cloud, on-premises or in hybrid environments.

It is a low-code platform that was created to let users visually develop entire applications, integrate this applications with existing systems and add hand-made code when necessary.

Figure 2.16: The OutSystems Development Platform [Rev16].

2.8.2 Mendix

Founded in 2005, Mendix like OutSystems provides tools and architecture to design, build, test, integrate, deploy, manage and optimize service-oriented business applications (SOBAs).



Runtime version	7.0.0
Administrator user name	MxAdmin
Region	Mendix Cloud EU

Instances 2 / 8

Memory (MB) 512 / 2048

Total allocated memory 1024 MB from 4096 MB

Figure 2.17: Mendix Low-Code Platform cloud [Men].

Mendix's technology enables companies to streamline and automate complex processes across people and systems. It also provides a run-time environment which means it does not provide code generation – the model is fully executable.

2.9 Summary

The aim of our work is to make a system that combines the solutions that are currently offered.

For the spreadsheet generation and verification, these are projects that provide a part of what we want to offer for our end users. They are limited in terms of accessibility for a user with little technical knowledge, they do not have flexibility in resolving domain problems related to business logic and also most of them only offer one type of spreadsheet environment template. These reasons make these projects one dimensional and not flexible in terms of scalability.

As for the bidirectional transformations, they are very related to what we want to perform. It would be very interesting to add to our project the possibility of bidirectional transformations of our models (database to front-end and vice versa). Even though, a solution that combines database integration and that generates multiple types of front-ends is not yet available.

The content management systems are able to offer solutions that are very accessible for any type of end user, facilitating immensely the job of creating and managing a website requiring no technical knowledge. But they are one dimensional, only provide website management and only for the templates that a given system is made to support. In this sense the user is limited to what type of template or website the system offers.

The low-code platforms are very interesting solutions and provide something very similar of what we want. We also want to minimize the amount of coding needed in the transformation from one model to another. These solutions are also mostly designed for the fast development of business oriented small apps.

We want to combine all of these solutions and offer a multidimensional project that can be user-friendly in the knowledge sense, offer many different types of templates generated from a database.

Chapter 3

A Model-Driven Approach To The Generation Of Front-Ends

In this Chapter, we describe our approach regarding the generation of front-ends using the schema from a database.

To resolve this multi-layered problem a multi-layered solution was devised. We have 4 main parts:

- The ESQL Language
- The ATL Transformations
- The Spreadsheet Generation
- The Web Page Generation

Firstly, we developed a language we called ESQL. the objective of this language is to emulate a simple version of SQL. In our case this DSL also supports some of the principal dialects of SQL such as: MySQL, nuoDB, Oracle SQL, PostgreSQL, and the Microsoft SQL Server.

Next, we developed in the ATL transformation language the transformations to go from the model generated in our ESQL language to the SSDSL language.

After having the ESQL model translated to the SSDSL language, we can use work that was already developed in the GENSS project [Tei16] to generate a spreadsheet. We also use the translated model along with model-to-text transformations (M2T) made with the EGL language to generate the web page files needed in our web site.

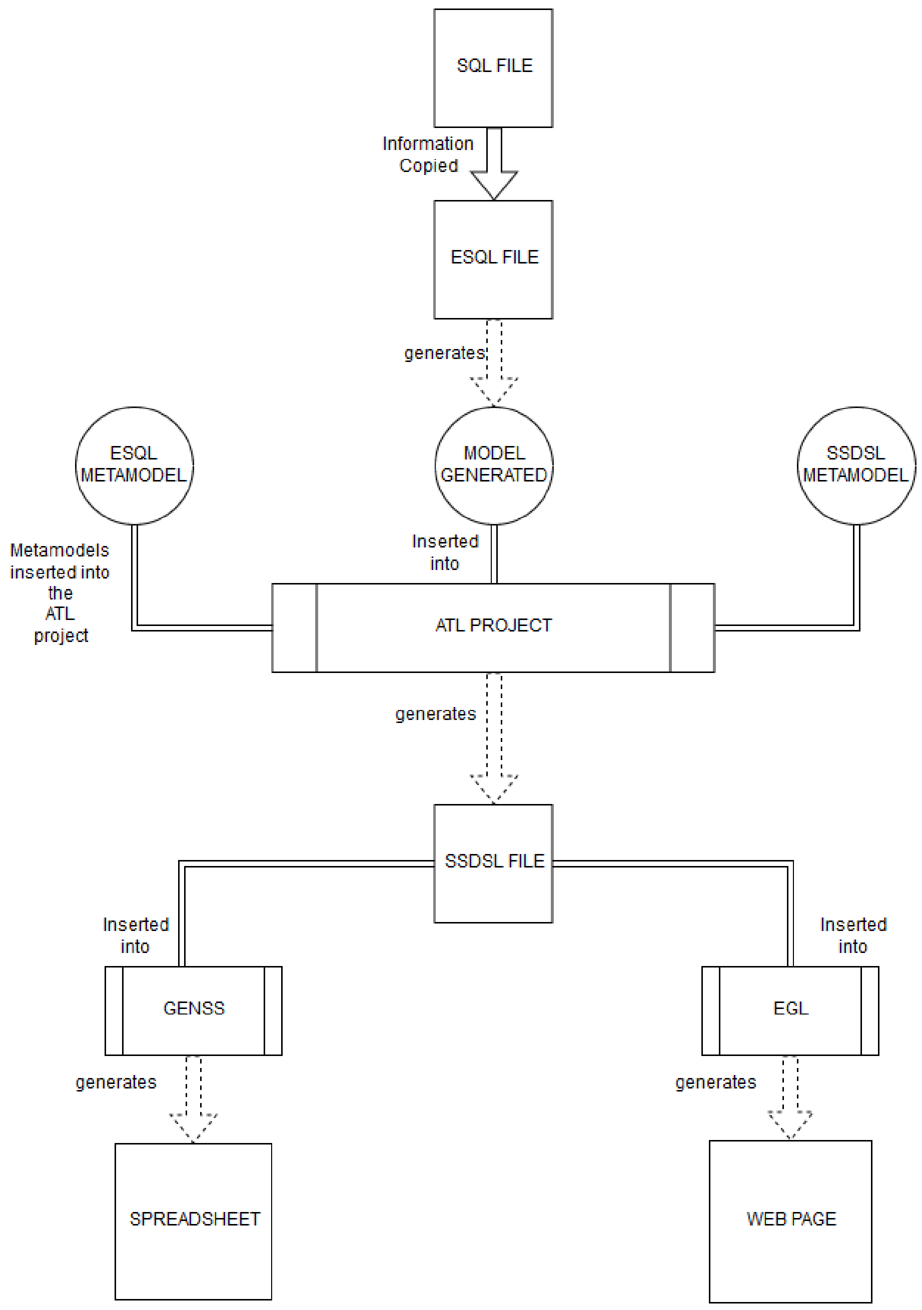


Figure 3.1: Diagram depicting how the parts of the solution connect.

3.1 Technologies Used

This section describes all the technologies that are related and are used in the development of our project.

3.1.1 Epsilon

For the development of the project we choose the Eclipse IDE as it is an familiar coding environment for us and because it works well with all of the technologies that we've used.

Along with eclipse, we will also use the Epsilon Platform (Extensible Platform of Integrated Languages for model management). This platform provides us with many tools that can be employed in model-driven development tasks like transformation, comparison, merging, validation of models and code generation [DK17].

Among a multitude of tools, Epsilon also gives us many languages that were created specifically for model-driven development. In our project, we will use:

- **EOL** (Epsilon Object Language) [Pai+09], that is a general purpose model oriented imperative language that gives us the capacity to create, query and modify EMF models (see section 3.1.2). This language is used in the project for spreadsheet generation [Tei16], to translate the Ecore models to GMF models.

- **EGL** (Epsilon Generation Language) [Ros+08], as the name suggests it is a model-to-text language used to generate code, documentation and other textual artifacts. This will be the language used to generate the templates of spreadsheets and other types of files we want.

Each one of these languages comes with development tools and an interpreter for its code.

3.1.2 Eclipse Modeling Framework

The EMF (Eclipse Modeling Framework) [Fou13] is going to be used to help us build models of the DSLs of our project.

EMF has a meta-modeling language called Ecore. In our case the Ecore files are used to define the meta-models of the languages.

This tool was also chosen because like the technology explained before, it comes with many available tools already implemented that can be used along with the Eclipse IDE.

In our work, we are concerned only with SQL expressions made to represent tables and columns in a schema.

The expressions for the schema, table and column are written in a specific way (see listings 3.1 and 3.2). Even though there is a standard definition for SQL alterations are still needed when switching from one dialect from another.

```
CREATE SCHEMA schema_name;
```

Listing 3.1: A schema declaration in SQL.

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype);
```

Listing 3.2: A table declaration in SQL.

In our case, it is not our objective to represent completely every detail of the SQL language, seeing that this would be a big task that would require more time and personnel than we can offer. In our language we represent the main commands for the creation of a database schema with multiple tables and tables with multiple columns.

3.2 The ESQL Language

We utilized the Xtext language in the Eclipse IDE to develop a DSL that we call ESQL. This language acts as a copy of the SQL language. We copy the information of a SQL schema to a file of the ESQL language. After this, our DSL is made to generate a model with the information contained in the original SQL schema.

In Xtext, we define rules for our grammar. These rules define how the syntax of our language is going to be made.

The first rule of our grammar describes where the parser starts and the type of root elements the DSL has.

```
Model:

  Schemas+=Schema*
  Tables += Table*
  Associations += Association* ;
```

Listing 3.3: The root element of the ESQL language.

In the ESQL language, a program is a collection of schemas, tables, columns and associations. All of this elements are stored in the model object.

The elements of the language are all stored in collections that are assigned with the "+=" sign (see listing 3.3). These data structures are similar to the ones implemented in other programming languages such as Java, Python and etc.

We've also created an rule called Elements (see listings 3.4) that makes a exclusive selection (denoted by the pipe "|" operator) between the tree main rules in our grammar: schema, table and column.

```
Elements:

Schema|Table|Column
;
```

Listing 3.4: The elements rule offer the possible choices for an element in the grammar.

3.2.1 Schema

A valid schema declaration statement starts with the `Create Schema` keywords (see listing 3.5), followed by a name which is of the "`Anything`" type. This type was created to be able to contain almost any type of combination for a name. Initially this was made taking in to account names that only contained the regular A-Z and 0-9 characters, but as the number of dialects expanded and the test cases got more complex, the need to create this data type arose, so now a name can contain the normal characters plus dots and special characters.

This ID is assigned to the feature called `name` of the parsed schema model element.

```
Schema:

("CREATE␣SCHEMA"  name = Anything  Tables += Table*)
source=[Elements]? Refnumber=INT?;
```

Listing 3.5: The create schema rule.

The schema rule also contains an optional reference to a source element. In our case this can be any type of element but generally a schema will not have a source element because it is the "root" element of the file. The rule also contains a `Refnumber` variable that is also optional and that functions as a reference number for the current schema and a `Tables` collection that contains all the tables that belong to the current schema.

3.2.2 Table

```
Table:

("CREATE␣TABLE"   name = QualifiedName )
("(" (Columns += TABLE_ELEMENTS)* ")"? ";"?
source =[Schema]? Refnumber=INT?);
```

Listing 3.6: The create table rule.

The table rule starts with the `Create Table` keywords (see listing 3.6). Afterwards it receives a name but of the `QualifiedName` type. This is a different type of name that of the schema name because some of the dialects tested allowed for names that were composed of two word linked by an ".", so we decided to create this rule to be able to contain two words of the type `Anything` connected by a dot.

After the name variable, the table rule contains the `Columns` collection which is composed of another rule called TABLE ELEMENTS (see listings 3.7).

```
TABLE_ELEMENTS:

  Column | KEY_TYPES | Check_Rule | KEY_RULE | UNIQUE_RULE
;
```

Listing 3.7: The elements that can belong in a table rule.

The table elements rule contains a column, the key types rule, the check rule, the key rule and the unique Rule. these are all of the different types of elements that can appear and be combined in any form in a table rule.

The table rule also has the source element, which in this case will also have the possibility of having all the three types of elements as a source and a INT for a reference number.

3.2.2.1 Key Types

To represent the types of keys that can appear in a table declaration, the key types choice was created. This rule contains the two types of keys of SQL: primary and foreign Keys.

In SQL, The primary key is comprised of a column or a set of columns that contains data that is utilized to uniquely identify each row in a given table. It could be thought of as like an ID for a person. A citizen card or ID is something that uniquely identifies a person, so a primary key can be compared to that but in relation to rows in a table.

The foreign key is also a column or set of columns that are a reference to the primary key of another table.

```
KEY_TYPES hidden(WS):

  Foreign_Key | Primary_key

;
Foreign_Key hidden(WS):
   {Foreign_Key} (("FOREIGN␣KEY" "("(Foreigns += Key)*")"
   (R = Rererence)) &(De ?= DELETE)?  &(A ?= ACTION)? ) ","?

;
Primary_key hidden(WS):

  {Primary_key} ("PRIMARY␣KEY" ("("(Primaries += Key)*")")? )","?

;
```

Listing 3.8: The types of keys a table can have.

The primary key rule contains an object definition of the primary key. After this, the key words "Primary Key" are called and the primaries collection is made. The `primaries` collection is a collection of variables of the type `key`. A key is just a name with a "," character. The coma is to give the possibility of having multiple names in a primary key definition.

The foreign key rule also contains the object instantiation of the rule. It has the "FOREIGN KEY" keywords and a collection of keys called `foreigns`.

Adding to this, the foreign key rule also contains the reference rule:

```
Rererence:
  {reference} "REFERENCES"
  "["?(Src_Table = [Elements|Anything])"]"? "(" (srcs += SRCs*) ")";
```

Listing 3.9: This rule creates the references of the key types.

The reference rule was created to complement the foreign key rule. A foreign key (most of the times) is coupled with a reference to a different table. This rule also contains the `src` table variable that can be an element or a string, and contains a `srcs` collection that contains the names of the sources for this foreign key.

The rule also contains the flags for delete and the types of actions that can be taken on the foreign keys.

```
DELETE:
  "ON␣DELETE␣CASCADE"  | "ON␣DELETE"
;

ACTION:

  "ACTION" | "NO␣ACTION"
;
```

Listing 3.10: The delete and action types of flags.

3.2.2.2 The Check Rule

In SQL, the check constraint is used to limit the range that a value that belongs to a column can have.

The rule is called by the check keyword and has left and right operators. These operators are all defined bellow in another rule.

```
Check_Rule:
  ("CHECK" "(" Left = [Column] Op = Operator Right = [Column] ")") ","?
  ;
//SQL Comparison Operators
Operator:
    '<>' | '==' | '>' | '<' | '<=' | '>=' | '='
  ;
```

Listing 3.11: The check rule.

3.2.2.3 The Unique Rule

Another constraint that we choose to represent in our DSL is the unique constraint. It ensures that all values in a column are different. A primary key automatically has a unique constraint.

The rule is called by the unique keyword and stores the keys in a collection called primaries.

```
UNIQUE_RULE:
  {UNIQUE_RULE}("UNIQUE" ("("(Primaries += Key)*")")? )","?;
```

Listing 3.12: The unique rule.

3.2.3 Column

The column rule starts with the optional keyword constraint, a variable name of the `QualifiedName` type, a type which is of the type `columntype` (this is an enumerate that contains all of the possible types for a column, in our case we included more than 20 types), a source element, reference number and a integer or two integers (one is the normal size of a column with just one number for its size and the other is a range type of size that accepts two numbers and this is the range of values that a column can have).

```
Column:

  "CONSTRAINT"? ((  name = QualifiedName  ("["?type = ColumnType "]"?)
  (ide = IDENTITY)? (size = Typeofsize)? )
    &('UNSIGNED')? &(De ?= DELETE)? &(A ?= ACTION)? &(N ?= NullnotNull)?
    &(D ?= DefaultTypes)? &(N2 ?= NullnotNull)?
    &(R ?= Rererence)? )('auto_increment')?
    (source = [Table]? Refnumber = INT?)(",")? ;


Typeofsize:

   NormalSize | RangeSize

;

NormalSize:
  ("(" S1 = INT ")");

RangeSize:
  ("(" S1 = INT ","  S2 = INT ")");

enum ColumnType:

   FOREIGNKEY | PRIMARYKEY | string | CHAR | clob | DATE
   | longvarchar | binary | ENUM |
   VARCHAR |varchar2 | number| varbinary | tinyint | INT
   | integer | SMALLINT| BIGINT |
   FLOAT | DOUBLE | BOOL | blob | tinyblob | mediumblob | serial
   | datetime | bit | money
   | smallmoney | TEXT | timestamp | numeric | mediumtext | decimal
   | Anything;
```

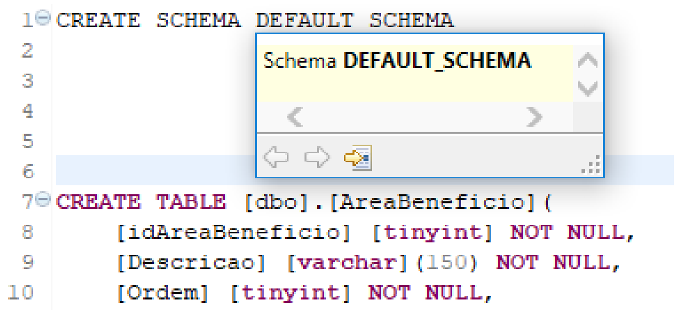
Listing 3.13: The column rule.

3.2.4 Association

The association rule is composed of an source element, a target element and a reference number of the association. This a rule made to help create associations when we create the model via Xtend.

```
Association:

  Source = [Elements] Target = [Elements] Refnumber=INT?
;
```

Listing 3.14: The association rule.



```
CREATE SCHEMA DEFAULT SCHEMA





CREATE TABLE [dbo].[AreaBeneficio](
    [idAreaBeneficio] [tinyint] NOT NULL,
    [Descricao] [varchar](150) NOT NULL,
    [Ordem] [tinyint] NOT NULL,
```

Figure 3.2: A example of schema, table and columns defined in our language.

3.2.5 Xtext Terminals

It is also important to note that many keywords and constructs of the dialects we choose to represent in our DSL, were represented as terminals in the Xtext language. This was made to facilitate the testing of the languages and also because the relevant information for the creation of the models for our project are the schema, table and column information.

```
terminal ML_COMMENT: '/*' -> '*/';
terminal SL_COMMENT: '//' !('\n'|'\r')* ('\r'? '\n')?;
terminal SL_COMMENT2    : '--'  !('\n'|'\r')* ('\r'? '\n')?;
//terminal UNIQUE    : 'UNIQUE'  !('\n'|'\r')* ('\r'? '\n')?;
terminal DROP      : 'DROP' -> ';';
terminal BEGIN     : 'BEGIN' -> 'END;;';
terminal ALTER     : 'ALTER' -> 'END;;';
terminal DECLARE   : 'DECLARE' -> 'END;;';
terminal TRIGGER   : 'CREATE OR REPLACE TRIGGER' -> 'END;;';
terminal CREATE_SEQUENCE      : 'CREATE SEQUENCE' -> ';';
terminal CREATE_SEQUENCE2      : 'create sequence' -> ';';
terminal CREATE_INDEX  : 'CREATE INDEX' -> ';';
terminal CREATE_INDEX2  : 'create index' -> ';';
terminal IF  : 'IF' -> ';';
terminal FUNCTION   : 'create or replace function' -> 'end;';
terminal ENGINE   : 'ENGINE' -> ';';
terminal VIEW   : 'CREATE VIEW' -> ';';
terminal SET   : 'SET' -> ';';
terminal ALTER_TABLE     : 'ALTER TABLE' -> ';';
```

Listing 3.15: The terminals of the ESQl language.

All of this represents the triggers, sequences, indexes and similar SQL commands. These were represented in terminals to be recognized as comments in our DSL. They are made to be comments in our language because at the moment they are not relevant to our project.

3.3 The ESQL Generator

After the DSL was created, we needed to specify how the model was going to be generated after an ESQL program is made. Xtext has a generator that is written in the Xtend language. This is where we created the methods for the generation of our ESQL model.

The ESQL generator class is a class that only contains the following methods:

- **DoGenerate**: This method (see listing 3.16) acts in a similar way as the main method in a Java class; it generates the XMI model file after running the called methods. It has a loop that runs through all of the files in the folder, and for each file runs the createxmi method. This makes the generation of multiple models from multiple files automatic.

```
  for (file : resource.contents){
              fsa.generateFile(file.eResource.URI.trimFileExtension +
              '.xmi',
        createxmi(file as Model))}
  }
```

Listing 3.16: The start for the model generator of the ESQl language.

- **createxmi**: This method (see listing 3.17) iterates through all of the schema objects stored in the model that is received. After that it checks to see if the schemas collection is empty, if true then it runs the generatedefaultSchemaCode which is a method that creates a default schema for the ensuring tables. This was needed because some of the files that where tested did not have a schema definition. If false, it runs just the generateSchemaCode.

```
def createxmi(Model m) {
  '''<?xml version="1.0" encoding="UTF-8"?>
  <myDsl:Model
      xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:myDsl="http://www.xtext.org/example/mydsl/MyDsl1"
    xsi:schemaLocation="http://www.xtext.org/example/mydsl/MyDsl1
    MyDsl1.ecore">

      IF m.schemas.isEmpty
        m.generateDefaultSchemaCode()

      ELSE
        FOR o : m.schemas
          o.generateSchemaCode()
        ENDFOR

      ENDIF


</myDsl:Model>
            '''
  }
```

Listing 3.17: The model (XMI) code that is generated.

- **GenerateSchemaCode** and **generateDefaultSchemaCode** (see listing 3.18): The first method receives a schema object and writes the information (name, reference number, source element) on the model. After that it iterates through all of its tables and calls the method generateTableCode for each table.

  The second method is very similar to the first one. The main difference is that instead of receiving a schema object, it just writes the information as a "Default-Schema". It also receives directly the model and its information and utilizes that to move on to its equivalent generateTablecode.

```
    def generateSchemaCode(Schema schema) '''
    <Schemas xsi:type="eSQL:Schema" name="«schema.name»
    "Refnumber="«schema.refnumber»
    "source="//@Schemas.«schema.source.refnumber»">
    «FOR t : schema.tables»
      «t.generateTableCode()»
      «ENDFOR»
      </Schemas>
    '''

   def generateDefaultSchemaCode(Model m) '''

    <Schemas name="Default_Schema"
             source="//@Schemas.0"
             Refnumber="0">

    «FOR element : m.tables»
      «element.generateTableCode2()»
    «ENDFOR»

    </Schemas>
     «Scounter++»
     «Tcounter = 0»
     '''
```

Listing 3.18: The two types of cycles for the schema code generation.

- **GenerateTableCode** and **GenerateTableCode2** (see listing 3.19): Both methods receive a table object that writes its information (name, reference number, source element), iterates through all of its columns and calls the method generateColumnCode for each column. For the table name in the model we removed all characters that are not a-z, A-z or 0-9, and removed blank spaces between names.

```
  def generateTableCode(Table table, Schema S) '''
  «table.refnumber = Tcounter»
  «table.source = S»
  <Tables name="«table.name.replaceAll("[^a-zA-Z0-9]", "")»"
        source="//@Schemas.«table.source.refnumber»"
        Refnumber="«table.refnumber»">

  «FOR c : table.columns»
           «if (c instanceof Column)
                 c.generateColumnCode(table)»
        «ENDFOR»
  </Tables>
  «Tcounter++»
    «Ccounter = 0»
  '''
def generateTableCode2(Table table) '''
   «table.refnumber = Tcounter»
  <Tables name="«table.name.replaceAll("[^a-zA-Z0-9]", "")»"
          source="//@Schemas.0"
        Refnumber="«table.refnumber»">

  «FOR c : table.columns»
           «if (c instanceof Column)c.generateColumnCode2(table)»
        «ENDFOR»
  </Tables>

  '''
```

Listing 3.19: The two types of cycles for the table code generation.

- **GenerateColumnCode** and **GenerateColumnCode2**: The last methods of the generator receive each column of the owner table, a table as the source element and with this writes its information (name, type, size, reference number and source element).

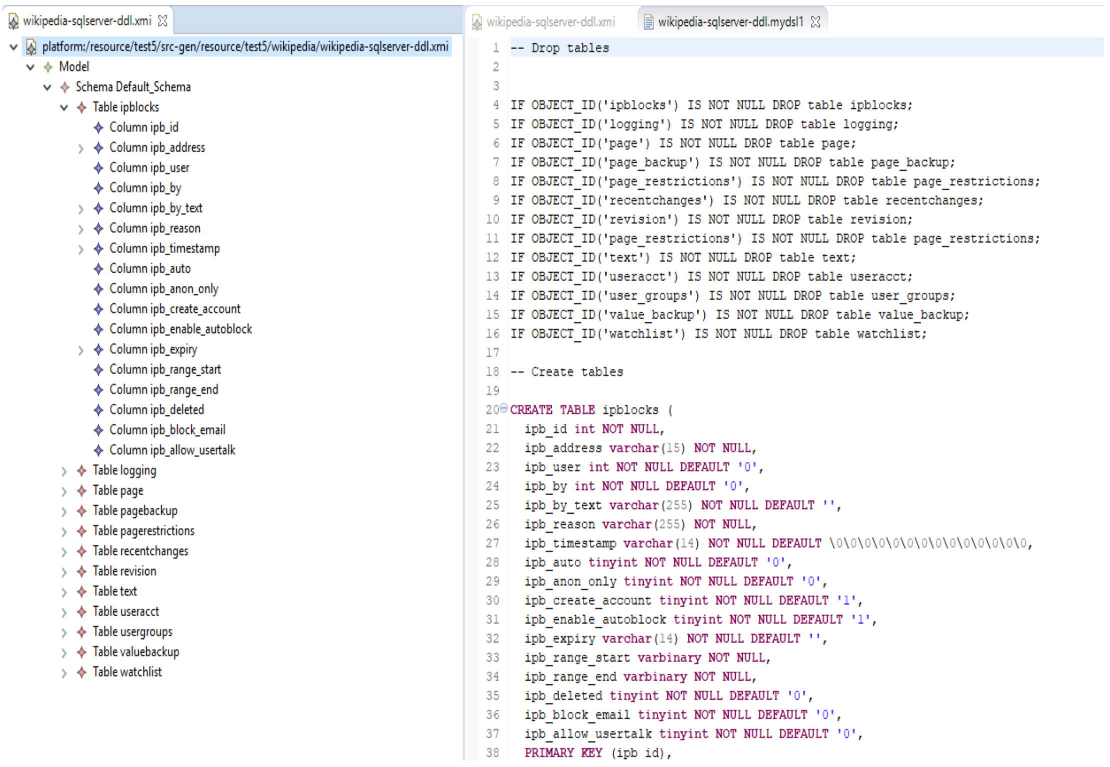
  We've also included the size or range method, to show how the two types of sizes are differentiated in the Xtend generator.

```
def generateColumnCode(Column column,Table source) '''
  <columns name="«column.name»" type="«column.type»"
  size="«column.size»"
  source=//@Tables."«column.source.refnumber»"/> '''

```

```

def generateColumnCode2(Column column, Table table) '''
«column.refnumber = Ccounter»
      «column.source = table»
<Columns
         xsi:type="myDsl:Column"
         name="«column.name»"
         source="//@Schemas.0/@Tables.«column.source.refnumber»"
         type="«column.type»">
         «column.size.size_or_range»
</Columns>
«Ccounter++»
'''
```

Listing 3.20: Like the examples before the column generation is also made with two types of cycles.

After the development of this two elements of our solution, we can represent SQL schema as a model in our project (see figure 3.3). By having this we can advance to the next part of our model-driven solution, which is to transform this model that contains the information of the schema to other model of the SSDSL language of the GENSS project [Tei16].


wikipedia-sqlserver-ddl.xmi
wikipedia-sqlserver-ddl.xmi
wikipedia-sqlserver-ddl.mydsl1

```
platform:/resource/test5/src-gen/resource/test5/wikipedia/wikipedia-sqlserver-ddl.xmi
  Model
    Schema Default_Schema
      Table ipblocks
        Column ipb_id
        Column ipb_address
        Column ipb_user
        Column ipb_by
        Column ipb_by_text
        Column ipb_reason
        Column ipb_timestamp
        Column ipb_auto
        Column ipb_anon_only
        Column ipb_create_account
        Column ipb_enable_autoblock
        Column ipb_expiry
        Column ipb_range_start
        Column ipb_range_end
        Column ipb_deleted
        Column ipb_block_email
        Column ipb_allow_usertalk
      Table logging
      Table page
      Table pagebackup
      Table pagerestrictions
      Table recentchanges
      Table revision
      Table text
      Table useracct
      Table usergroups
      Table valuebackup
      Table watchlist
```

```
-- Drop tables


IF OBJECT_ID('ipblocks') IS NOT NULL DROP table ipblocks;
IF OBJECT_ID('logging') IS NOT NULL DROP table logging;
IF OBJECT_ID('page') IS NOT NULL DROP table page;
IF OBJECT_ID('page_backup') IS NOT NULL DROP table page_backup;
IF OBJECT_ID('page_restrictions') IS NOT NULL DROP table page_restrictions;
IF OBJECT_ID('recentchanges') IS NOT NULL DROP table recentchanges;
IF OBJECT_ID('revision') IS NOT NULL DROP table revision;
IF OBJECT_ID('page_restrictions') IS NOT NULL DROP table page_restrictions;
IF OBJECT_ID('text') IS NOT NULL DROP table text;
IF OBJECT_ID('useracct') IS NOT NULL DROP table useracct;
IF OBJECT_ID('user_groups') IS NOT NULL DROP table user_groups;
IF OBJECT_ID('value_backup') IS NOT NULL DROP table value_backup;
IF OBJECT_ID('watchlist') IS NOT NULL DROP table watchlist;

-- Create tables

CREATE TABLE ipblocks (
  ipb_id int NOT NULL,
  ipb_address varchar(15) NOT NULL,
  ipb_user int NOT NULL DEFAULT '0',
  ipb_by int NOT NULL DEFAULT '0',
  ipb_by_text varchar(255) NOT NULL DEFAULT '',
  ipb_reason varchar(255) NOT NULL,
  ipb_timestamp varchar(14) NOT NULL DEFAULT \0\0\0\0\0\0\0\0\0\0\0\0\0\0,
  ipb_auto tinyint NOT NULL DEFAULT '0',
  ipb_anon_only tinyint NOT NULL DEFAULT '0',
  ipb_create_account tinyint NOT NULL DEFAULT '1',
  ipb_enable_autoblock tinyint NOT NULL DEFAULT '1',
  ipb_expiry varchar(14) NOT NULL DEFAULT '',
  ipb_range_start varbinary NOT NULL,
  ipb_range_end varbinary NOT NULL,
  ipb_deleted tinyint NOT NULL DEFAULT '0',
  ipb_block_email tinyint NOT NULL DEFAULT '0',
  ipb_allow_usertalk tinyint NOT NULL DEFAULT '0',
  PRIMARY KEY (ipb id),
```

Figure 3.3: After the code is run, a xmi file is created containing the information that was stored in the schema.

3.4 The ATL Model Transformations

The ATL language is the language we chose to help with the model-to-model transformation part of this thesis. We will utilize the model generated with our language (ESQL) and transform this to a model defined in the SSDSL language. The SSDSL language models are the models that we will use to generate spreadsheets and the files for the Django web page.

3.4.1 ATL Transformation Structure

The ATL model transformation language works by decomposing transformations into three main parts: a header, helpers and rules.

3.4.1.1 The Header Section

The header is used to declare general information such as the module name (this is the transformation name: it must match the file name), the source and target metamodels and imported libraries. The header section starts with the keyword **module** followed by the name of the module (in our example, in listing 3.21 is called defaultmodule). Then the keyword **create** indicates the target models. the target is the variable OUT and the metamodel is the outmodel. The keyword **from** and the variable (IN) are the source models and they conform to the metamodel called inmodel.

```
module defaultmodule;
create OUT : outmodel from IN : inmodel;
```

Listing 3.21: Example of a header section in the ATL transformation language.

The target model that is connected to the variable OUT is created from the source model IN.

3.4.1.2 Helpers

Helpers can be compared to Java methods but for the ATL project. These helpers are subroutines (based on OCL) [Jou+08] that are used to reduce code redundancy. To define a helper, we need:

- The **name** of the helper (in listing 3.22 the name is firstToLower()).
- The **context** of the helper. The context defines in which cases the helper is applied. The types of elements that the helper applies to. If the helper does not have a context defined, it is associated to the global context of the module.
- A **return** value. This is obligatory for every type of helper.
- A set of **parameters** (optional).

In ATL we can define two types of helpers: operation and attribute helpers.

- **Operation helpers**, have parameters and (sometimes) have recursion. They are used to define an operation to a model element or in general in the module.
- **Attribute helpers**, also have a name, context and a type but not input parameters. They are mostly used to associate read-only values to source model elements.

To show how an operation helper is made we can look at listing 3.22.

```
helper context String def: firstToLower() : String =
self.substring(1, 1).toLower() + self.substring(2, self.size());
```

Listing 3.22: Example of a helper section in the ATL transformation language [Jou+08].

The helper firstToLower is made for the type String and its values are strings. It has an OCL expression to calculate the values of the helper. These values are assigned after the '=' character. This helper changes the capital letter in the beginning of a string for a short case one.

3.4.1.3 Rules

Rules are the main logic behind the expression of the transformation made in an ATL program. These rules come in two types: matched rules and called rules.

Matched rules can also be called declarative rules. A matched rule has a name, source pattern and a target pattern. The source pattern is defined by using the keyword **from**. This allows the specification of a variable from the model element that has to match the type of the source elements.

The target pattern is declared by the keyword **to**. These pattern generates elements every time there is a match with the source pattern.

We can look at the listing 3.23 for an example of a declarative rule.

```
rule Member2Male {
from
    s : Families!Member (not s.isFemale())
to
    t : Persons!Male (
        fullName <- s.firstName + '␣' + s.familyName)
```

Listing 3.23: Example of a standard and lazy rule in ATL.

There are several kinds of matched rules differing in the way they are triggered.

- **Standard rules**, are applied once for every match that can be found in source models.
- **Lazy rules**, are triggered by other rules. They are applied on a single match as many times as it is referred to by other rules, every time producing a new set of target elements. Lazy rules are indicated by the keyword lazy;

- **Unique lazy rules**, are also triggered by other rules. They are applied only once for a given match. If a unique lazy rule is triggered later on the same match the already created target elements are used. Rules of this type are indicated by the unique lazy keywords.

A **called rule** is basically a procedure: it is invoked by name and may take arguments. They provide the imperative programming functionalites to the ATL transformations. These rules can be compared to helpers as they accept parameters and have to be explicitly called. But unlike helpers, they can actually create elements for the target model.

3.4.2 The Main Rule

In our model to text transformation, firstly we defined the module name (in our case is called translation). After this we defined the source and target metamodels and create the main rule. This rule acts as a starting point for the translation process and calls out the other rules for the transformation. In this main rule we created 4 variables: Wk (workbook), Ws(worksheet), Tables (set of tables) and Associations (set of compositions).

These elements correspond to the main elements of the SSDSL language that we use in this project. Each definition of these variables calls the corresponding translation rule and after that all of these elements are added in the main data structure and the translation is finished.

```
module Translation;
create OUT: ExcelLang from IN: ESQLLang;

rule main {
 from
  M: ESQLLang!Model
  using { Wk: ExcelLang!Workbook  = thisModule.WorkBook(M.Schemas.first());
        Ws: ExcelLang!Worksheet  = thisModule.Worksheet(M.Schemas.first());
      Tables:Set(ExcelLang!Table) = M.Schemas.first().Tables.asSet()
      -> collect(T|thisModule.table(T));
    Associations:Set(ExcelLang!Composition) =
    Tables -> collect(E|thisModule.associate(Ws,E));

    }
  to
    Elements: ExcelLang!SS_DSL (
      hasElements <- Set{Wk,Ws,Asc1,Tables,Associations}
    ),
    Asc1: ExcelLang!Composition (
      source <- Wk,
      target <- Ws
    )
    }
```

Listing 3.24: The Main rule for the beggining of the model to text transformation in ATL.

3.4.3 The Workbook Lazy Rule

The workbook lazy rule receives an element in our ESQL of the type schema and creates a workbook with that name.

```
lazy rule WorkBook{
  from
    S: ESQLLang!Schema (
      S.oclIsTypeOf(ESQLLang!Schema)
    )
  to
    Wk: ExcelLang!Workbook (
      name <- 'Workbook_' + S.name
    )}
```

Listing 3.25: The workbook lazy rule.

3.4.4 The Worksheet Lazy Rule

The worksheet lazy rule receives an element of the type schema and creates a worksheet element with the exact same name as the schema.

```
lazy rule Worksheet{
  from
    S: ESQLLang!Schema (
      S.oclIsTypeOf(ESQLLang!Schema)
    )
  to
    Ws: ExcelLang!Worksheet (
      name <- S.name
    )}
```

Listing 3.26: The worksheet lazy rule.

3.4.5 The Table Lazy Rule

The table lazy rule receives an element of the type table and creates a table element of the SSDSL language. This rule also calls the lazy rule column for the definition of its owned attributes as columns.

```
 lazy rule table {
  from
    T: ESQLLang!Table

    using{

    Columns:Set(ExcelLang!Columns) = T.Columns.asSet()->
    collect(c|thisModule.column2entry(c));
    }

  to
    Table: ExcelLang!Table (
      clientDependency <- Set{},
      name <- T.name,
      ownedAttribute <- Columns,
      type <- 'VERTICAL'
    )do {
    -- The corresponding name for the current ElementB is added in the map.
    -- This map will be used at the end of the transformation to create a
    link between ElementB and DefinitionB
    thisModule.TableToAssign(T.source.name, Table);
  }
}
```

Listing 3.27: The table lazy rule.

3.4.6 The Column Rule

This lazy rule translates column elements in our DSL to entry elements of the target DSL.

We also note that we have a helper rule to define the cType of the column. This rule has the three data types of the SSDSL language: text, number and none.

```
lazy rule column2entry {
  from
    C: ESQLLang!Column
  to
    Entry: ExcelLang!Entry (
      cType <- 'NONE',
      clientDependency <- Set{},
      name <- C.name,
      supplierDependency <- Set{}
    )
}

helper context mydsl!Column def: define_ct (column: mydsl!Column ):  String =

  if (column.type = #VARCHAR) or (column.type = #CHAR)
    then 'TEXT'
  else
  if (column.type = #SMALLINT) or (column.type = #INT)
  then 'NUMBER'
  else
       'NONE'
  endif
endif;
```

Listing 3.28: The column lazy rule and the define ct helper.

After these steps are executed, we currently have a path to generate from a SQL schema definition an equivalent program in the SSDSL language. This gives us a way to get from a SQL schema to a working spreadsheet made in the GENSS project.

The next step in the thesis is to create a CRUD web application to pair with our spreadsheet.

3.5 Generating Web Pages

One of the objectives of this work is to create a web page from the initial SQL schema definition. This web page is created to interact with the original database.

Currently, there are many types of solutions to create databases for example: CMS, Django, Ruby on rails and etc. But in these cases it is necessary to create most things from scratch and by hand.

In our project, we can already represent the SQL schema information in our generated models. Now we are going to use these models to generate CRUD web pages that are directly derived from a database schema.

3.5.1 The EGL Generator

EGL is a language that was made for model-to-text transformation (M2T) and thus it can be used to transform models into various types of textual artifacts. In our case it will be used to generate python code for the web site that we will create.

```
[%
var t = TemplateFactory.load("transformation.egl");
var workbook = Workbook.all[0];

t.populate("worksheets", workbook.targetAssociation.target);
t.process();
t.generate('models.py');
%]
```

Listing 3.29: The EGL code for the main method for the generation of the web site.

```
from django.db import models
from django.forms import ModelForm

[% for(ws in worksheets.sortBy(ws | ws.order)){ %]
  [% for(t in ws.targetAssociation.target.sortBy(t | t.order)){%]
  class [%=t.name%](models.Model):
    [%var attributes = t.ownedAttribute;
       for(a in attributes){%]
        [%=a.name%][% if (a.cType+"" == "NUMBER"){%]
        = models.IntegerField(blank=True, null=True)
        [%}else if (a.cType+"" == "TEXT"){%]
        = models.CharField(max_length=100, blank=True, null=True)
        [%}else if (a.cType == "NONE"){%]
    = models.CharField(max_length=100, blank=True, null=True)
    [%}else %] = models.CharField(max_length=100, blank=True, null=True)
   [% }%]
   def __str__(self):
      return self.name
  class Meta:
   managed = False
    db_table = '[%=t.name%]'


  [% }%]
[% }%]
```

Listing 3.30: The EGL code for the generation of the models for the website.

The EGL generator receives a model written in the SSDSL language and populates a variable called `workbook` in the main method. After the variable creation it iterates through its elements. Firstly, it finds the worksheets and iterates it one by one. For each worksheet, it finds its targets which are tables and for each table it writes its information in a python file that are for the models definition in a Django web page.

3.6 Summary

These transformations allow us to have a basic end-to-end transformation from a database schema to a spreadsheet and a CRUD web page developed in the Django environment.

This end-to-end transformation starts with a database schema written in SQL and that its code is copied to an ESQL file. The ESQL file is connected to an Xtend generator that automatically generates a model with the information contained in the file. This creates the first model needed to start our transformations.

After the generation of the model with the schema information, we transfer this model and the DSLs meta-models to the model transformation project made with the ATL language. This project contains all the rules for the model-to-model transformation that we need to make to obtain the transformation from our source model which is the ESQL model to our target model which is the SSDSL model.

After we run the ATL transformations and obtain the model translated to the SSDSL language, we then use the EGL generator. The EGL generator makes the model-to-text transformation of the information contained in the SSDSL model and generates the python file that contains the models in our website. We can also use the SSDSL model to generate a spreadsheet file by using the GENSS project.

Chapter 4

Validation

4.1 ESQL: Test Runs and Language Extension

As mentioned in the introduction of this thesis, the aim of this project is to create a tool that tries to makes it easier to synchronize and maintain back and front-end by making it easier to generate spreadsheets and web pages.

To summarize, the main objectives of this work are:

- To create a DSL in Xtext that represents the SQL language in our project.
- Create transformations using the ATL transformation language and the DSL created to generate models and translate this models to the GenSS DSL (a language created in the thesis that acts as a basis for this work).
- The creation of a complete and robust solution that generates a website and spreadsheet file from a database schema. This also should be able to be extended to allow further developments in the transformation of the models, the DSLs created and the generation of templates.

Initially, the DSL was created with the minimal abstractions of the SQL language because these abstractions are the ones needed for our project. It is best for a faster development to use the essentials of the language. The DSL contained the schema, table and column creation, and also the basic data types (`integer`, `float`, `double`, `char`, `varchar`). This is the information that is relevant in our project. As we validated our solution with more examples, the language and its constructs were extended to support not only more features of SQL but also to support multiple dialects of the SQL language.

To validate our solution, we used a repository from GitHub; the project is called OLTP-Bench [Dif+13] which is a multi-threaded load generator. The framework is designed to be able to produce variable rate, a variable mixture load against any JDBC-enabled relational database. This framework provides data collection features, e.g., per-transaction latency and throughput logs. This repository has a multitude of SQL files of multiple dialects and is a project that is utilized in the real world, so it provides valuable code.

We decided to make an iterative run through the repository files to validate and extend the ESQL language. During our first run, we tested the ESQL language for the Wikipedia SQL files of the benchmark suite. this folder similarly to most of the SQL folders of the repository contains the following dialects of SQL: MySQL, nuoDB, Oracle SQL, PostgreSQL, and the Microsoft SQL Server. At first, we only concerned ourselves with the running of the MySQL files as this is one of the most widely used dialects of SQL. The first run made us detect the index command but as the creation and/or utilization of indexes is not relevant for our work, we made the index command to be recognized as a type of comment written in our DSL.

This first run also extended the number of data types supported by our DSL. To the data types mentioned earlier we included the `varbinary`, `binary` and `tiny int`. Another feature added was the primary and foreign keys and also the unique predicate. In the second run, we used a more complex file of MySQL. This created the need to add the “default” predicate and default values for the columns and also the possibility to add references to tables.

The third test session was made with the nuoDB files, this dialect differs from MySQL and this caused more changes to our DSL. For these files, we added the `small int`, `longvarchar`, `datetime` and `big int` data types. In this session we also tested the Oracle SQL files, this is a dialect-heavy in triggers, indexes, sequences, and similar commands, so we decided to consider all the code for these features as comments as they are irrelevant for our project. The information we want to extract out of the files is the tables and columns information and this is going to be inserted into our models that will be translated. The Oracle files also made us add to the existing data types the `varchar2`, `number`, and `clob`. We also needed to add the support for a different type of size of the variables, a range type of size was added as an option for the size of the variable e.g. (column number(10,0)).

The repository also had other two dialects for most of the database files: PostgreSQL and Microsoft SQL Server. For the Wikipedia folder that was being tested these dialects didn’t require modifications to the language, the only addition was to consider the "IF" command as a comment and the text data type.

After this initial folder test, we changed folders and tested multiple folders at the same time as the runs were starting to produce few errors. When errors appeared, these were only minor and easily correctable. after running the project in this batch mode, we created a data type called `Anything` in our DSL. This data type comprises any possible combination of the String, ID, Int and the other types of characters supported, making it possible to have any type of name for the schemas, tables, and columns of our language. This is needed because many times names can have different types of characters and this makes it acceptable to have any type of character in a name.

The final batch of tests using the OLTPBench SQL files did not return any new errors thus making it unnecessary to introduce any new feature to our language. The only thing added was the possibility of having a primary key declared anywhere in the table (originally we only had the declaration made after all the columns were declared) and also the possibility to add multiple foreign key declarations in each table.

After our DSL was tested in all 84 files that contained SQL code in the repository and stopped returning any type of syntax error, we moved on to test the transformations of the models generated by our DSL as presented in the next section.

4.2 ATL: Model Transformation Tests

For the ATL transformations validation we used the ANSI SQL and MySQL files. These files contain two popular dialects of SQL.

An important point for the generation of the models is how scalable is our solution. It is important to know how the transformation will behave with a significant number of tables or a significant number of columns for each table. We tested for up to 500 tables which is database of significant size and the transformations behaved like we expected. For a bigger number of columns by table, we also tested with 5 tables with 100 columns each, also a decent sized database and the behaviour was also as expected and the time of execution remained less than 10 seconds.

There was no need to alter or add new transformations rules, and this was to be expected because the transformations only need the information supplied by the model generated in our DSL. In this case the changes added did not alter the information supplied to the transformation module.

As time is not altered by a big increase in tables, one of our focus of the tests was to see if the information from the tables contained in the schemas is correctly represented in the model generated by our ATL transformations.

To test this, we ran the transformations for the ANSI SQL and MySQL files of the Wikipedia, twitter and chbenchmark folders of the OLTPBenchmark repository. All of the models generated had all of the information we intended to extract from the SQL files and thus they were correct.

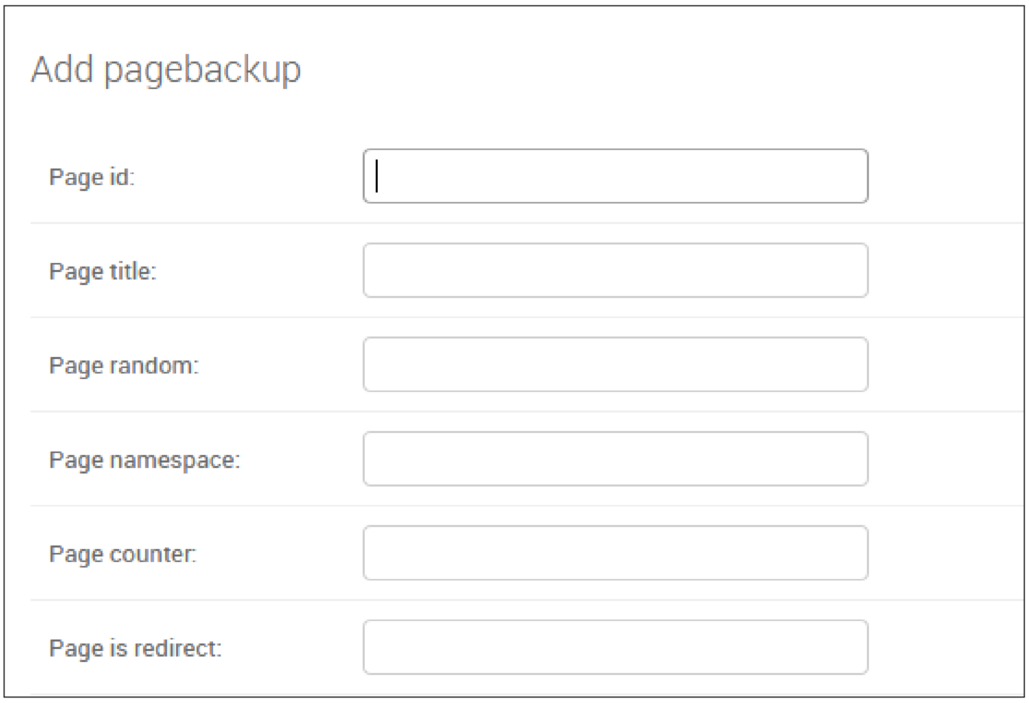
4.3 EGL: Web Page Generation Validation

For the web page validation part of the thesis we decided to create some web pages and verify if the fields of the tables are represented in the examples.

We've created web pages for the MySQL files of the Wikipedia (see figures 4.1 and 4.2), Twitter and chbenchmark folders. All of the web pages were verified by us and contained all the fields with correct information.

```
CREATE TABLE page_backup (
  page_id int NOT NULL,
  page_namespace int NOT NULL,
  page_title varchar(255) NOT NULL,
  page_restrictions varchar(1024) NOT NULL,
  page_counter bigint NOT NULL,
  page_is_redirect tinyint NOT NULL,
  page_is_new tinyint NOT NULL,
  page_random double NOT NULL,
  page_touched binary(14) NOT NULL,
  page_latest int NOT NULL,
  page_len int NOT NULL,
  PRIMARY KEY (page_id),
  UNIQUE (page_namespace,page_title)
);
```

Listing 4.1: The SQL code for the table pagebackup from the OLTPBenchmark Project.



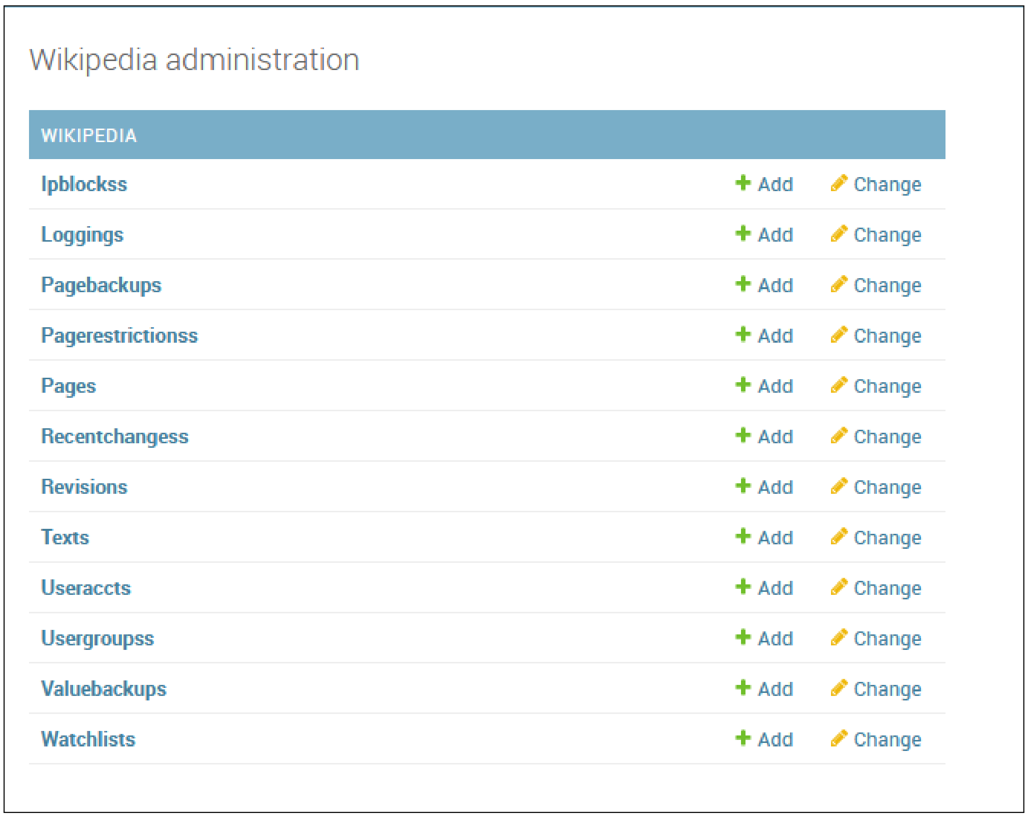
Add pagebackup

Page id:	
Page title:	
Page random:	
Page namespace:	
Page counter:	
Page is redirect:	

Figure 4.1: Web page form for one of the tables of the wikipedia SQL schema.

4.4 Case Study

As a case study, we have a real-world scenario that we also used for testing. This scenario is a real-world test utilizing the example database schemas of the Portuguese Finance General Inspection.



Wikipedia administration

WIKIPEDIA		
Ipblockss	+ Add	Change
Loggings	+ Add	Change
Pagebackups	+ Add	Change
Pagerestrictionss	+ Add	Change
Pages	+ Add	Change
Recentchangess	+ Add	Change
Revisions	+ Add	Change
Texts	+ Add	Change
Useraccts	+ Add	Change
Usergroupss	+ Add	Change
Valuebackups	+ Add	Change
Watchlists	+ Add	Change

Figure 4.2: Web page of the tables of the wikipedia SQL schema.

4.4.1 The IGF Case

The Finance General Inspection is a service of the integrated Ministry of Finance in the direct administration of the State. With administrative autonomy, it works directly dependent on the Minister of Finance, and its mission is the evaluation and strategic control of the financial administration of the State and specialized technical support to the Ministry of Finance. Its intervention covers all entities of the administrative and business public sector and the private and cooperative sectors.

IGF was kind enough to give us an example database schema that we utilized to run this test case scenario.

For this test case some modifications had to be made to our language as the schema that was given to us also contained some features that were not present in the SQL files that were tested in the repository. The main addition to our DSL was the addition of more types of SQL commands to be ignored (in the case of the IGF schema, we ignored the constrained commands utilized for the primary key, for what we need the information supplied by this command is not needed).

We composed the run of this project in the following steps:

1. The user copies the SQL schema code to a file of our DSL. After this a model (or models, depending on the number of files copied) containing the information of the file is generated.

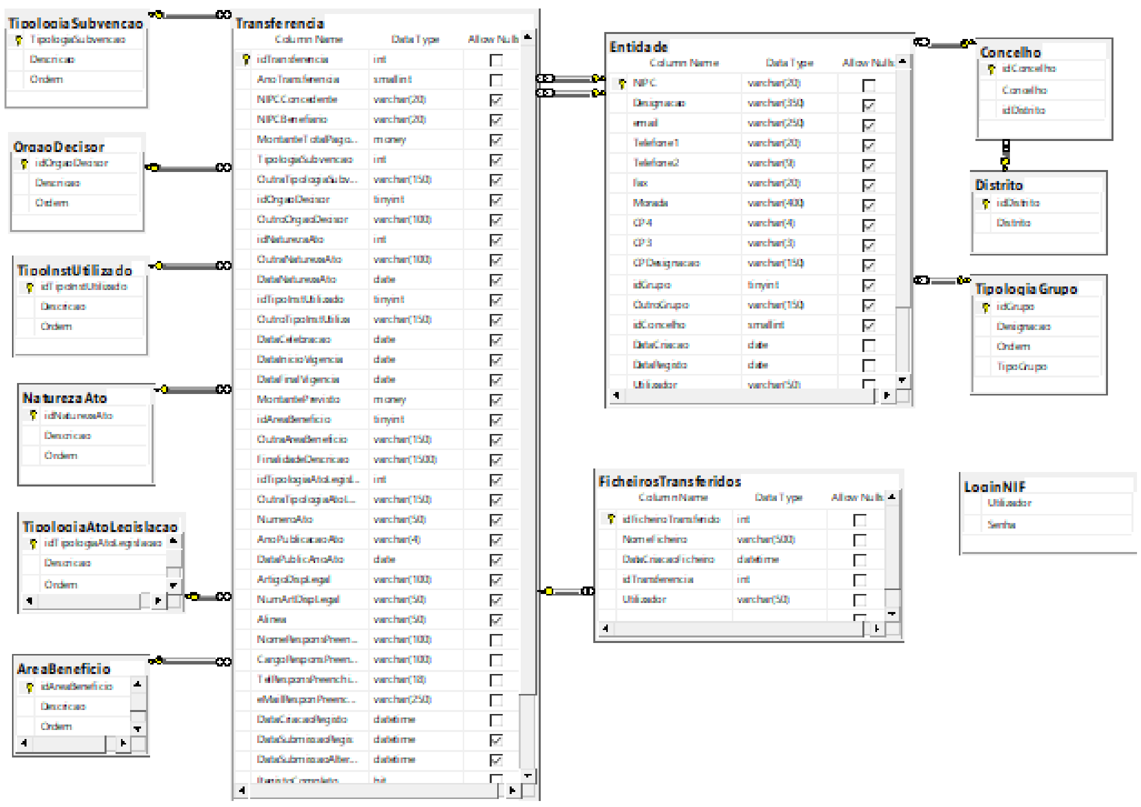


Figure 4.3: The example schema of IGF.

2. These generated models are copied by the user to the ATL transformations project.

3. The user now runs the ATL transformations on the models. This will create a SSDSL file which is the file that is used to generate the spreadsheet and the web page.

4. Generate the spreadsheet files with the GENSS project.

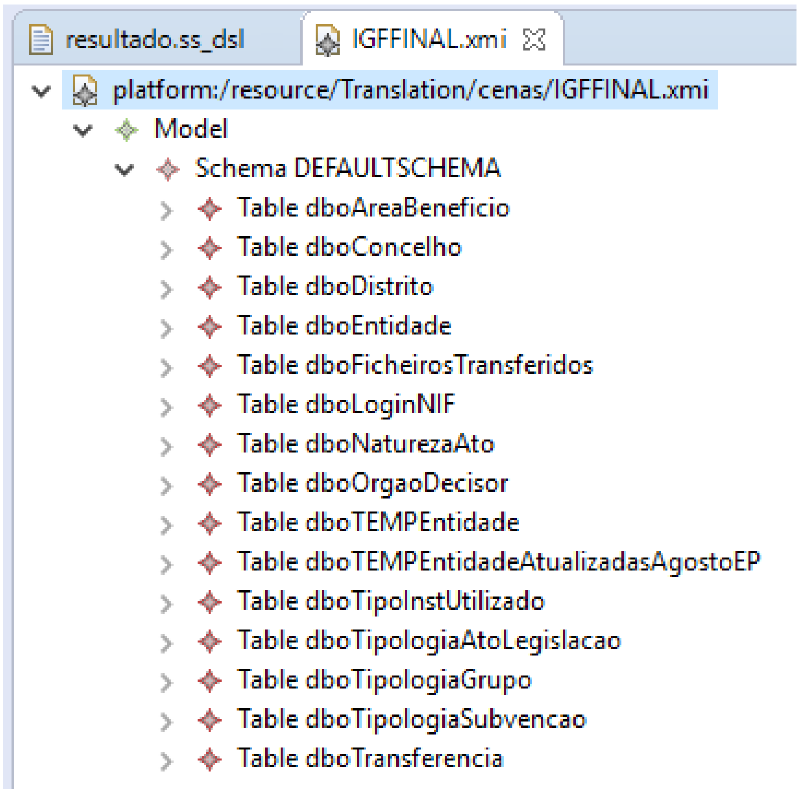
5. Finally, the user copies the SSDSL files to the EGL project and generate the Django files for the website.

To show how all of the elements of this work connect, we are going to utilize the schema provided by IGF (see figure 4.3) and show how the table `entidade` (see listing 4.2) is translated from SQL to our DSL. This generates a model that is translated to a SSDSL language model and after it can be used to generate a spreadsheet and a web page.

```
CREATE TABLE [dbo].[Entidade](
  [NIPC] [varchar](20) NOT NULL,
  [Designacao] [varchar](350) NULL,
  [email] [varchar](250) NULL,
  [Telefone1] [varchar](20) NULL,
  [Telefone2] [varchar](9) NULL,
  [Fax] [varchar](20) NULL,
  [Morada] [varchar](400) NULL,
  [CP4] [varchar](4) NULL,
  [CP3] [varchar](3) NULL,
  [CPDesignacao] [varchar](150) NULL,
  [idGrupo] [tinyint] NULL,
  [OutroGrupo] [varchar](150) NULL,
  [idConcelho] [smallint] NULL,
  [DataCriacao] [date] NOT NULL,
  [DataRegisto] [date] NOT NULL,
  [Utilizador] [varchar](50) NOT NULL,
  [EP] [bit] NOT NULL,
  [Estrangeiro] [bit] NOT NULL)
```

Listing 4.2: The ESQL code for the table Entidade.

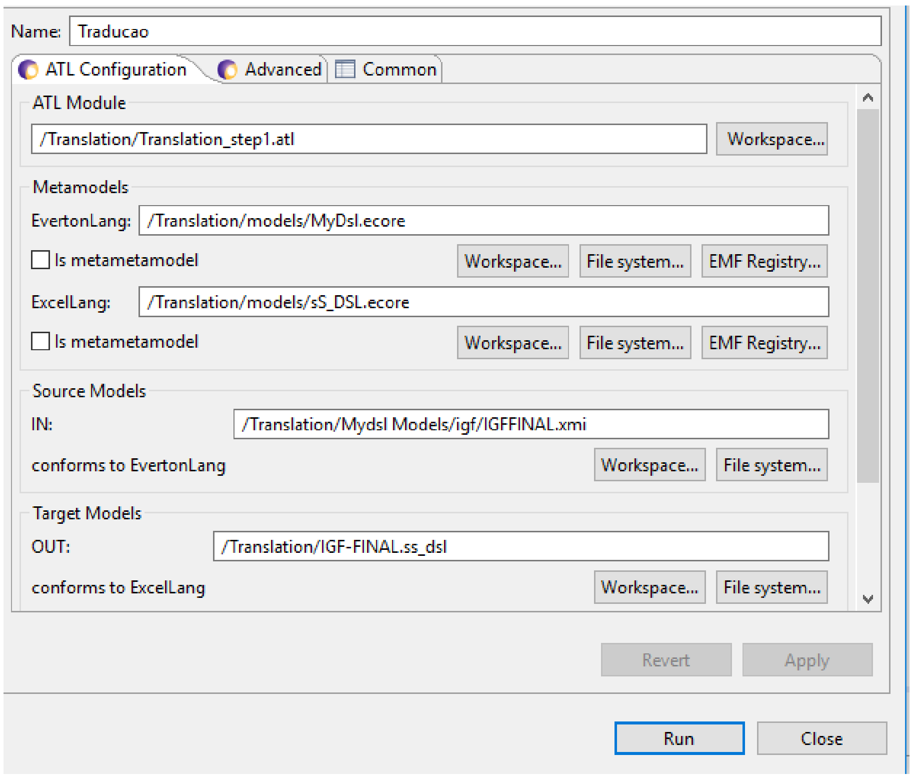
Firstly, we copied the SQL schema information to a file of our ESQL language. Our language then generates a model that contains the information of the IGF schema (see figure 4.4). Our generator was made to automatically generate XMI models as they are introduced in a project of the ESQL language. It also updates the models every time the ESQL file is changed and saved.



```
platform:/resource/Translation/cenas/IGFFINAL.xmi
  Model
    Schema DEFAULTSCHEMA
      Table dboAreaBeneficio
      Table dboConcelho
      Table dboDistrito
      Table dboEntidade
      Table dboFicheirosTransferidos
      Table dboLoginNIF
      Table dboNaturezaAto
      Table dboOrgaoDecisor
      Table dboTEMPEntidade
      Table dboTEMPEntidadeAtualizadasAgostoEP
      Table dboTipoInstUtilizado
      Table dboTipologiaAtoLegislacao
      Table dboTipologiaGrupo
      Table dboTipologiaSubvencao
      Table dboTransferencia
```

Figure 4.4: The model generated from the ESQL file.

After the generation of the model (or models) is finished, we transfer these models to the ATL translation project. In the ATL project, we manually run the transformations using the meta-models from ESQL, SSDSL and the model of the schema we want to translate (see figure 4.5).


Name: Traducao

ATL Module

/Translation/Translation_step1.atl

Metamodels

EvertonLang: /Translation/models/MyDsl.ecore

☐ Is metametamodel

ExcelLang: /Translation/models/sS_DSL.ecore

☐ Is metametamodel

Source Models

IN: /Translation/Mydsl Models/igf/IGFFINAL.xmi

conforms to EvertonLang

Target Models

OUT: /Translation/IGF-FINAL.ss_dsl

conforms to ExcelLang

Figure 4.5: The model-to-model transformation is made by the ATL project.

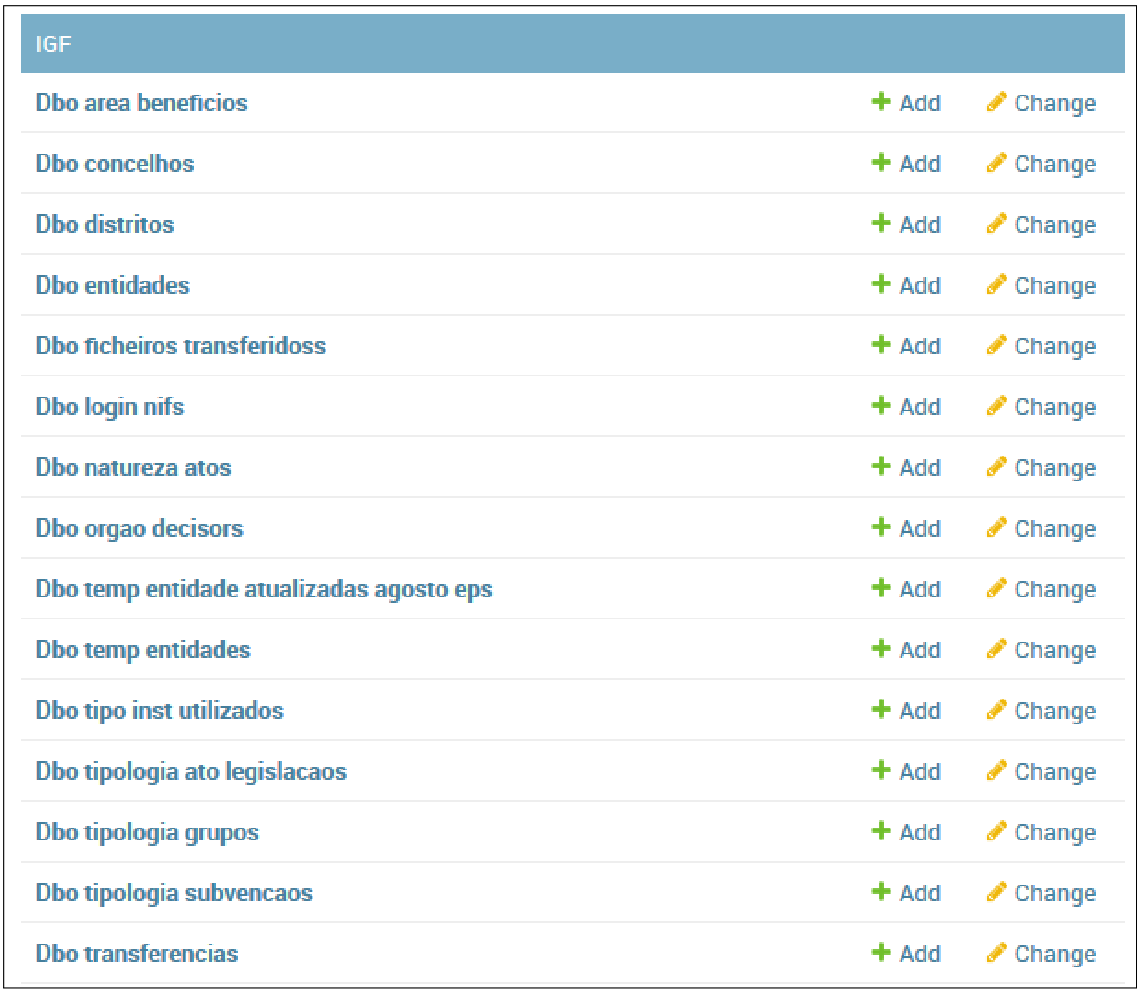
After we executed all the previous transformations, we can use the generated SSDSL files in two projects: the GENSS project and our EGL project.The GENSS project uses the SSDSL files to generate spreadsheet files written in XML. In a normal utilization of the GENSS project, those files would be generated by hand in an interface used in eclipse where the user can create a graph very similar to UML were the tables, columns and relationships between them are represented. In this case, we generate all the tables and relationships between them automatically with our transformations.

The EGL project generates files which represent the models that will be represented in the web page (see listing 4.3). Our Django project utilizes the models generated by them and creates the equivalent web pages (see figure 4.7 and 4.6).

```

class dboEntidade(models.Model):
        Fax = models.CharField(max_length=100, blank=True, null=True)
        EP = models.CharField(max_length=100, blank=True, null=True)
        Designacao = models.CharField(max_length=100, blank=True, null=True)
        Telefone2 = models.CharField(max_length=100, blank=True, null=True)
        CP4 = models.CharField(max_length=100, blank=True, null=True)
        email = models.CharField(max_length=100, blank=True, null=True)
        Morada = models.CharField(max_length=100, blank=True, null=True)
        idConcelho = models.CharField(max_length=100, blank=True, null=True)
        CPDesignacao = models.CharField(max_length=100, blank=True, null=True)
        Estrangeiro = models.CharField(max_length=100, blank=True, null=True)
        Telefone1 = models.CharField(max_length=100, blank=True, null=True)
```

```
    DataRegisto = models.CharField(max_length=100, blank=True, null=True)
    Utilizador = models.CharField(max_length=100, blank=True, null=True)
    CP3 = models.CharField(max_length=100, blank=True, null=True)
    DataCriacao = models.CharField(max_length=100, blank=True, null=True)
    OutroGrupo = models.CharField(max_length=100, blank=True, null=True)
    NIPC = models.CharField(max_length=100, blank=True, null=True)
    idGrupo = models.CharField(max_length=100, blank=True, null=True)
  def __str__(self):
        return self.name
class Meta:
  managed = False
    db_table = 'dboEntidade'
```

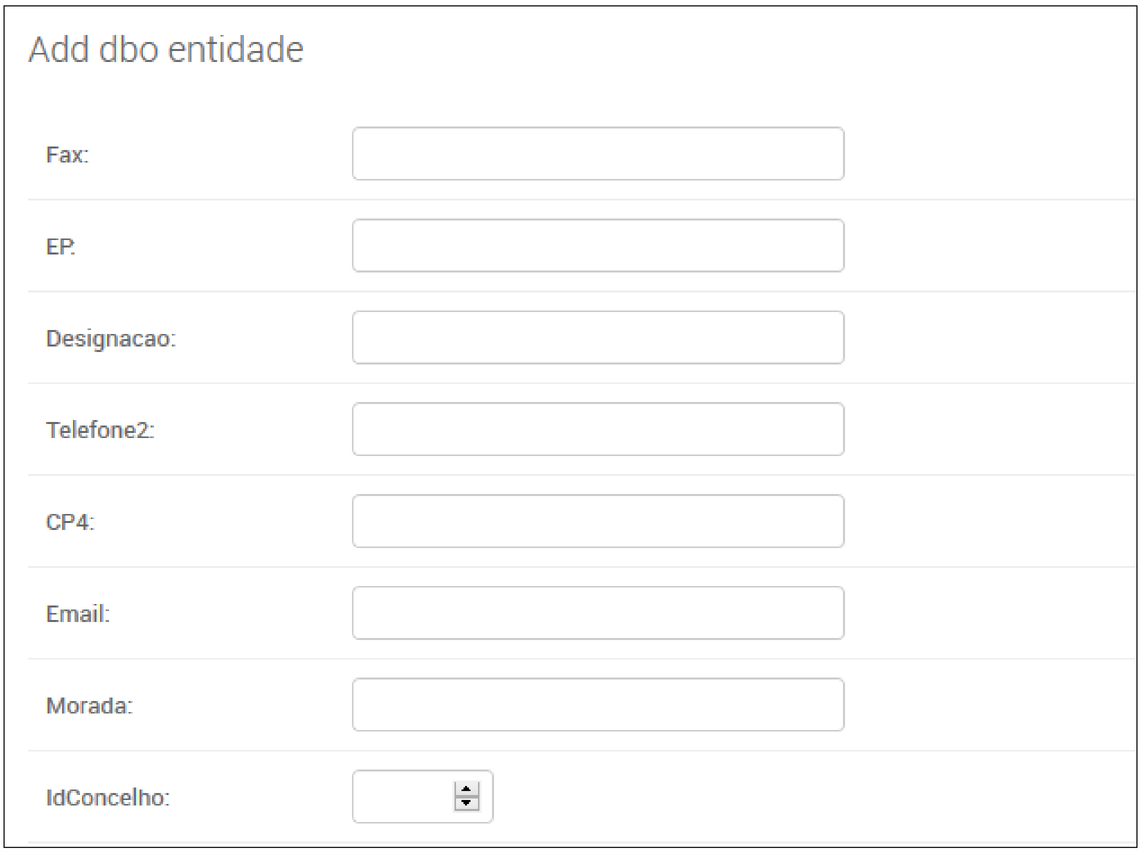
Listing 4.3: The Python code generated by the EGL project to represent the table Entidade in our web site.



IGF

Dbo area beneficios + Add ✎ Change
Dbo concelhos + Add ✎ Change
Dbo distritos + Add ✎ Change
Dbo entidades + Add ✎ Change
Dbo ficheiros transferidoss + Add ✎ Change
Dbo login nifs + Add ✎ Change
Dbo natureza atos + Add ✎ Change
Dbo orgao decisors + Add ✎ Change
Dbo temp entidade atualizadas agosto eps + Add ✎ Change
Dbo temp entidades + Add ✎ Change
Dbo tipo inst utilizados + Add ✎ Change
Dbo tipologia ato legislacaos + Add ✎ Change
Dbo tipologia grupos + Add ✎ Change
Dbo tipologia subvencaos + Add ✎ Change
Dbo transferencias + Add ✎ Change

Figure 4.6: The list of web pages and forms that were generated by the project from the schema from IGF.


Add dbo entidade

Fax:	
EP:	
Designacao:	
Telefone2:	
CP4:	
Email:	
Morada:	
IdConcelho:	

Figure 4.7: The generated web page form that represents one of the tables from the SQL schema supplied by IGF.

4.5 Summary

In this chapter we explained how we validated each part of the solution. For the ESQL language, we utilized a repository called OLTPBenchmark that contains a significant number of SQL files of different types of dialects. This repository made us extend our language to support more SQL concepts and different types of dialects.

The ATL transformation verification was made by running the transformation with multiple types of different models. Models with a large number of tables (up to 500) and models with tables with a large number of rows. Through all of the tests the information contained in the generated models was correct and the time of run for every test was practically the same.

For the EGL project, we generated multiple web pages and tested them in a Django project. Like in the ATL transformations, we verified if all of the fields were correctly represented and if the web pages were functional.

As a final test-case, we showed our transformation of the example SQL schema of the Portuguese Finance General Inspection to a web page that represents its tables. This is the full run of a real-world study case that utilizes our project, with this run we can show we accomplished the main goals of the project.

This project provides the possibility of utilization of multiple dialects of SQL (contains at least the most used ones such as MySQL, PostgreSQL, Microsoft SQL Server and more). This provides a lot of flexibility in the project's utilization. We also verified in our tests that the important information in the schemas provided is correctly translated to the generated intermediate models and to the front-ends in a quick and efficient manner.

In its current state, we can generate two types of front-ends: web pages and spreadsheets. But is important to note that this project can also be easily extended to contain more characteristics of the needed SQL dialects, more dialects and also to provide more complex and different front-ends.

Chapter 5

Conclusions

5.1 Achievements

In this thesis, we have presented our approach to the generation of front-ends by using model-driven development. To achieve our goals we created a DSL called ESQL and used another DSL called SSDSL which was made in a previous project.

We have also developed model-to-model transformations in a project called ATL. In this project, we defined the rules for our model-to-model translation of the model generated by our language to the target language. We've also utilized the EGL project to perform model-to-text transformations to generate the final front-end files for the spreadsheet and web page projects.

In summary, the main contributions of this work are:

- The creation of a DSL that mimics the SQL language in a model-driven environment. This language is coupled with a generator that creates models of the information contained in the DSL file.
- Model-to-Model transformations defined in the ATL language. We can transform models of our language to models of the SSDSL language.
- Model-to-Text transformations written in the EGL project. We can utilize the models that were translated to the SSDSL language and generate a CRUD web page of this information in a Django project.

More generally, using model-driven development and DSLs we've created a way to use a SQL schema definition to generate our desired types of front-ends.

We have also created a project that in the future can be extended to include more types of transformations (bi-directional), more dialects of SQL, more abstractions of SQL, different types of front-ends, etc.

5.2 Future Work

As is the case for many academic works, there are things which are not completely perfect and could be improved in this project in the future.

Some improvements and future work for this project could be:

- To extend the ESQL language to allow more types of dialects and to receive information from some of the commands of SQL. Some dialects have information in the commands that could be interesting to add (e.g the IGF case).
- Create a type of automatic verification for the model-to-model transformation in the ATL project. Currently, we verify the transformation by looking if all the information is there but this could be done in a more efficient manner.
- Create verifications for the models in Django project. The verification was made similarly to the ATL project where we looked to see if all the information is there.
- Create more polished and complex websites. We were not able to generate validations for foreign keys and other useful types of validations.
- Create a script/tool to allow the automatic execution of the complete process.

These are the main points for improvement of this work and the ones that could be used to enhance the project usability and effectiveness.

Bibliography

[Ace+04] R. Acerbis, A. Bongio, S. Butti, S. Ceri, F. Ciapessoni, C. Conserva, P. Fraternali, and G. T. Carughi. “Webratio, an innovative technology for web application development”. In: *International Conference on Web Engineering*. Springer. 2004, pp. 613–614.

[Ace+07] R. Acerbis, A. Bongio, M. Brambilla, and S. Butti. “Webratio 5: An eclipse-based case tool for engineering web applications”. In: *International Conference on Web Engineering*. Springer. 2007, pp. 501–505.

[AC08] M. Antkiewicz and K. Czarnecki. “Design Space of Heterogeneous Synchronization”. In: *In Generative and Transformational Techniques in Software Engineering II, International Summer School*, Volume 5235 (2008), pages 3–46.

[Bak+05] P. Baker, S. Loh, and F. Weil. “Model-Driven engineering in a large industrial context—motorola case study”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2005, pp. 476–491.

[BS81] F. Bancilhon and N. Spyratos. “Update Semantics of Relational Views”. In: *ACM Trans. Database Syst.* 6.4 (Dec. 1981), pp. 557–575. issn: 0362-5915. doi: 10.1145/319628.319634. url: http://doi.acm.org/10.1145/319628.319634.

[Ben05] N. Benton. “Embedded Interpreters”. In: *J. Funct. Program.* 15.4 (July 2005), pp. 503–542. issn: 0956-7968. doi: 10.1017/S0956796804005398. url: http://dx.doi.org/10.1017/S0956796804005398.

[Boh+06] A. Bohannon, B. C. Pierce, and J. A. Vaughan. “Relational Lenses: A Language for Updatable Views”. In: *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS ’06. Chicago, IL, USA: ACM, 2006, pp. 338–347. isbn: 1-59593-318-2. doi: 10.1145/1142351.1142399. url: http://doi.acm.org/10.1145/1142351.1142399.

[Boh+08] A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, and A. Schmitt. “Boomerang: resourceful lenses for string data”. In: *ACM SIGPLAN Notices*. Vol. 43. 1. ACM. 2008, pp. 407–419.

[Bor04] Borland. “Keeping your business relevant with model driven architecture (MODEL-DRIVEN ARCHITECTURE)”. In: (2004).

[Bra+08] M. Brambilla, S. Comai, P. Fraternali, and M. Matera. "Designing web applications with WebML and WebRatio". In: *Web Engineering: Modelling and Implementing Web Applications*. Springer, 2008, pp. 221–261.

[Bra+12] M. Brambilla, J. Cabot, and M. Wimmer. "Model-driven software engineering in practice". In: *Synthesis Lectures on Software Engineering* 1.1 (2012), pp. 1–182.

[Bro95] F. P. Brooks Jr. *The Mythical Man-month (Anniversary Ed.)* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-83595-9.

[Cha12] D. D. Chamberlin. "Early history of SQL". In: *IEEE Annals of the History of Computing* 34.4 (2012), pp. 78–82.

[CR14] C. M.A.C.D. W. Clay Richardson John R. Rymer. *New Development Platforms Emerge For Customer-Facing Applications*. Version 2016-11-18. Cambridge, Massachusetts: Forrester, June 9, 2014.

[Cle] D. Clements. *The New WordPress Dashboard Design (MP6) - Try It Now!* https://www.doitwithwp.com/the-new-wordpress-dashboard-design-mp6-try-it-now/. (Accessed on 02/19/2018).

[Cun+] J. Cunha, J. Mendes, J. Saraiva, and J. P. Fernandes. "Embedding and evolution of spreadsheet models in spreadsheet systems". In: *in VL/HCC'11. IEEE*, pp. 186–201.

[DB82] U. Dayal and P. A. Bernstein. "On the Correct Translation of Update Operations on Relational Views". In: *ACM Trans. Database Syst.* 7.3 (Sept. 1982), pp. 381–416. ISSN: 0362-5915. DOI: 10.1145/319732.319740. URL: http://doi.acm.org/10.1145/319732.319740.

[Dif+13] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux. "OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases". In: *Proc. VLDB Endow.* 7.4 (Dec. 2013), pp. 277–288. ISSN: 2150-8097. DOI: 10.14778/2732240.2732246. URL: http://dx.doi.org/10.14778/2732240.2732246.

[DK17] A. G.-D.R. P. Dimitris Kolovos Louis Rose. *The Epsilon Book*. 2017.

[Dja19a] Django. *Design philosophies | Django documentation | Django*. 2019. URL: https://docs.djangoproject.com/en/2.0/misc/design-philosophies/ (visited on 03/05/2019).

[Dja19b] T. W. Django. *6. Models and Databases — How to Tango with Django 1.7*. 2019. URL: https://www.tangowithdjango.com/book17/chapters/models.html (visited on 03/05/2019).

[DB98] R. C. Dorf and R. H. Bishop. "Modern control systems". In: (1998).

[EE05] G. Engels and M. Erwig. "ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications". In: *In 20th IEEE/ACM Int. Conf. on Automated Software Engineering*. 2005, pp. 124–133.

[Xml] *Extensible Markup Language (XML).* URL: https://www.w3.org/XML/ (visited on 02/05/2019).

[Fou13] E. Foundation. *an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model.* 2013. URL: https://www.eclipse.org/modeling/emf/.

[Fow10] M. Fowler. *Domain-specific languages.* Pearson Education, 2010.

[Gol+03] W. Golden, M. Hughes, and P. Gallagher. "On-line retailing: what drives success? Evidence from Ireland". In: *Journal of Organizational and End User Computing (JOEUC)* 15.3 (2003), pp. 32–44.

[HT06] B. Hailpern and P. Tarr. "Model-driven development: The good, the bad, and the ugly". In: *IBM systems journal* 45.3 (2006), pp. 451–461.

[Han] D. H. Hansson. *Ruby on Rails : A web-application framework that includes everything needed to create database-backed web applications according to the Model-View-Controller (MVC) pattern.* http://rubyonrails.org/. (Accessed on 02/19/2018).

[Igf] *Inspeção-Geral de Finanças.* http://www.igf.gov.pt/. (Accessed on 02/19/2018).

[Int] *Integranova Software Solutions | Integranova Software Solutions.* URL: http://www.integranova.com/ (visited on 02/11/2019).

[Jou+08] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. "ATL: A model transformation tool". In: *Science of computer programming* 72.1-2 (2008), pp. 31–39.

[Kel86] A. M. Keller. "Choosing a View Update Translator by Dialog at View Definition Time". In: *Proceedings of the 12th International Conference on Very Large Data Bases.* VLDB '86. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1986, pp. 467–474. ISBN: 0-934613-18-4. URL: http://dl.acm.org/citation.cfm?id=645913.671458.

[Lat] *LaTeX - A document preparation system.* URL: https://www.latex-project.org/ (visited on 02/05/2019).

[Leh80] M. M. Lehman. "Programs, life cycles, and laws of software evolution". In: *Proceedings of the IEEE* 68.9 (1980), pp. 1060–1076. ISSN: 0018-9219. DOI: 10.1109/PROC.1980.11805.

[Leh+97] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. M. Turski. *Metrics and Laws of Software Evolution – The Nineties View.* 1997.

[Lut08] D. Lutterkort. "Augeas–a configuration API". In: *Linux Symposium, Ottawa, ON.* 2008, pp. 47–56.

[Mel+03] S. Mellor, T. Clark, and T. Futagami. "Model-driven development - Guest editor's introduction". In: *Software, IEEE* 20 (Oct. 2003), pp. 14–18. DOI: 10.1109/MS.2003.1231145.

[MVG06] T. Mens and P. Van Gorp. “A taxonomy of model transformation”. In: *Electronic Notes in Theoretical Computer Science* 152 (2006), pp. 125–142.

[MF05] T. Meservy and K. Fenstermacher. “Transforming software development: An MDA road map”. In: *Computer* 38 (Oct. 2005), pp. 52 –58. DOI: 10.1109/MC.2005.316.

[Out] *OutSystems*. 2019. URL: https://www.outsystems.com (visited on 03/19/2019).

[Pai+09] R. F. Paige, D. S. Kolovos, L. M. Rose, N. Drivalos, and F. A. C. Polack. “The Design of a Conceptual Framework and Technical Infrastructure for Model Management Language Engineering”. In: *Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems*. ICECCS ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 162–171. ISBN: 978-0-7695-3702-3. DOI: 10.1109/ICECCS.2009.14. URL: http://dx.doi.org/10.1109/ICECCS.2009.14.

[Ram03] N. Ramsey. “Embedding an Interpreted Language Using Higher-order Functions and Types”. In: *Proceedings of the 2003 Workshop on Interpreters, Virtual Machines and Emulators*. IVME ’03. San Diego, California: ACM, 2003, pp. 6–14. ISBN: 1-58113-655-2. DOI: 10.1145/858570.858571. URL: http://doi.acm.org/10.1145/858570.858571.

[Rev16] M. Revel. *What Is Low-Code?* 2016. URL: https://www.outsystems.com/blog/what-is-low-code.html (visited on 02/11/2019).

[RR16] C. Richardson and J. R. Rymer. “Vendor Landscape: The Fracture, Fertile Terrain of Low-Code Application Platforms”. In: *FORRESTER, Janeiro* (2016).

[Ris10] M. Risoldi. “A methodology for the development of complex domain specific languages”. PhD thesis. Geneva U., 2010.

[RA05] S. K.-E. S. Robin Abraham Martin Erwig. *Visual Specifications of Correct Spreadsheets*. Tech. rep. Oregon State University, 2005. URL: http://web.engr.orst.edu/~erwig/Gencel/vitsl.pdf.

[Ros+08] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. C. Polack. “The Epsilon Generation Language”. In: *Model Driven Architecture – Foundations and Applications*. Ed. by I. Schieferdecker and A. Hartman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–16. ISBN: 978-3-540-69100-6.

[Sch95] A. Schurr. “Specification of Graph Translators with Triple Graph Grammars”. In: *International Workshop Graph-Theoretic Concepts in Computer Science*, (1995).

[Men] *Search Mendix*. 2016. URL: https://www.mendix.com/press/mendix-flexible-low-code-platform-cloud/ (visited on 02/11/2019).

[Sei03] E. Seidewitz. “What models mean”. In: *Software, IEEE* 20 (Oct. 2003), pp. 26 –32. DOI: 10.1109/MS.2003.1231147.

[Sel03] B. Selic. “The pragmatics of model-driven development”. In: *IEEE Software* 20.5 (2003), pp. 19–25. ISSN: 0740-7459. DOI: 10.1109/MS.2003.1231146.

[SK03] S. Sendall and W. Kozaczynski. “Model transformation: The heart and soul of model-driven software development”. In: *IEEE software* 20.5 (2003), pp. 42–45.

[SS09] Y. Singh and M. Sood. “Model Driven Architecture: A Perspective”. In: Mar. 2009. DOI: 10.1109/IADCC.2009.4809264.

[Ste07] P. Stevens. “Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions”. In: *International Conference on Model Driven Engineering Languages and Systems*, (2007), pages 1–15.

[Syr11] E. Syriani. “A multi-paradigm foundation for model transformation language engineering”. PhD thesis. McGill University Libraries, 2011.

[Tei16] R. Teixeira. “Controlled Specification and Generation of Spreadsheets”. MA thesis. Universidade Nova de Lisboa, 2016.

[Mid] *The Middleware Company: Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach, vol. 2004 (2003).* https://www.omg.org/mda/mda_files/MDA_Maintainability_Analysis_Report_040305.pdf.

[Num] *Total number of Websites - Internet Live Stats*. 2014. URL: http://www.internetlivestats.com/total-number-of-websites/ (visited on 02/01/2019).

[VD+00] A. Van Deursen, P. Klint, and J. Visser. “Domain-specific languages: An annotated bibliography”. In: *ACM Sigplan Notices* 35.6 (2000), pp. 26–36.

[Web] B. C. Website. *Bx Examples Repository: Family to Persons*. URL: http://bx-community.wikidot.com/examples:familytopersons.

[YXM07] Z. H.-H.Z.M. T. Y. Xiong D. Liu and H. Mei. “Towards automatic model synchronization from model transformations”. In: *In ASE ’07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering* (2007), pages 164–173.