# Utilizando o Framework Xtext na Implementação de uma Linguagem de Domínio Específico para Modelagem e Projeto de Bancos de Dados Relacionais

**Jonnathan Lopes**[1], **Maicon Bernardino**[1] **e Elder Rodrigues**[1]

[1]Laboratório de Estudos Empíricos em Engenharia de Software (LESSE)
Programa de Pós-Graduação em Engenharia de Software (PPGES)
Universidade Federal do Pampa (UNIPAMPA)

{jonnathan.riquelmo,eldermr}@gmail.com, bernardino@acm.org

***Abstract.*** *Developing a DSL is not a trivial task, as they are domain-specific programming languages that have their own grammar. Hence, it is necessary to use tools that support the definition of concepts inherent to a new language. This paper presents an experience report in DSL development using the Xtext framework. It aims to show an alternative for defining a textual notation for relational databases conceptual modeling.*

***Resumo.*** *O desenvolvimento de uma DSL não é uma tarefa trivial, pois são linguagens de programação de domínio específico que possuem uma gramática própria. Desta forma, faz-se necessário o uso de ferramentas que suportem a definição dos conceitos inerentes a uma nova linguagem. Este trabalho apresenta o relato de experiência no desenvolvimento de uma DSL utilizando o framework Xtext. Tem como objetivo apresentar uma alternativa para a definição de uma notação textual para modelagem conceitual de bancos de dados relacionais.*

## 1. Introdução

Conceitualmente, um modelo é uma representação, protótipo ou exemplo que se tem por objetivo reproduzir ou imitar de alguma forma. A elaboração de modelos são pontos centrais e importantes em diferentes áreas científicas. Na matemática, física e química, por exemplo, o emprego de modelos é tido como vital para a investigação teórica e prática em diferentes campos de estudo [Bailer-Jones 2009].

Em uma análise ampla, [Brambilla et al. 2017] discute que, considerando-se a premissa de que um observador e suas observações alteram a própria realidade, é possível se concluir que tudo na percepção de um indivíduo é um modelo, já que absolutamente nada pode ser processado pela mente humana sem ser modelado. Levando em consideração esse contexto, as Linguagens Específicas de Domínio (*Domain-Specific Languages*, DSLs) podem desempenhar um papel de destaque na Engenharia Orientada a Modelos (*Model-Driven Engineering*, MDE) [Schmidt 2006].

Para [Van Deursen et al. 2000] uma DSL é uma linguagem de programação ou linguagem de especificação executável que oferece, por meio de notações e abstrações apropriadas, poder expressivo focado e, geralmente, restrito a um domínio de problema específico. Assim como outras linguagens, as DSLs devem apresentar um conjunto de sentenças bem definidas por uma sintaxe e semântica própria.

Exposto isto, este estudo apresenta o relato de desenvolvimento de uma ferramenta que suporta uma DSL com abordagem textual, capaz de prover modelagem em nível conceitual de bancos de dados relacionais. Este desenvolvimento ocorre com o auxílio do *Language Workbench* (LW) Xtext, um *framework* para o desenvolvimento de linguagens de programação e modelagem. O resultado desse processo é uma solução chamada ERtext, gratuita e de código aberto.

O artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 resume os requisitos, gramática e arquitetura da solução construída. Uma visão geral da versão estável da ferramenta ERtext é fornecida na Seção 4 e, finalmente, a Seção 5 apresenta as conclusões.

## 2. Trabalhos Relacionados

Em geral, a abordagem textual é discutida há muitos anos em diversos estudos que investigam a modelagem de bancos de dados. Por exemplo, pode-se citar o trabalho de [Malhotra et al. 1989] que apresenta uma sintaxe para uma linguagem de programação ER integrada. Os problemas que surgem quando uma linguagem de consulta é incorporada em uma linguagem de programação de uso geral também são discutidos, apesar de não necessariamente ser apresentada a implementação da linguagem. Outro caso, mais recente, é o estudo de [Hossain et al. 2019] que apresenta uma linguagem natural controlada para especificar e verbalizar modelos de Entidade Relacionamento (ER). A ideia não é apenas resolver o problema de verbalização para esses modelos, mas também fornecer os benefícios de verificação e validação automáticas e *round-trip* semântico para tornar o processo de comunicação transparente entre os especialistas do domínio e engenheiros do conhecimento. Contudo, apesar da proposta apresentar a especificação em Prolog da gramática que define a linguagem natural controlada, maiores detalhes de implementações de uso são desconhecidas.

Desta forma, parte-se para uma comparação mais prática, analisando um conjunto de ferramentas com abordagens similares. Quando são considerados estudos e projetos citados em estudos secundários, pode-se mencionar o levantamento apresentado no trabalho de [Riquelmo et al. 2021]. O trabalho é uma revisão multivocal de literatura, ou seja, uma forma de mapeamento sistemático de literatura que inclui a literatura cinza, abrangendo um conjunto final de dez (10) estudos primários focados em DSLs, e que identifica 55 ferramentas já aplicadas na indústria e academia para modelagem ER em nível conceitual, lógico e físico. A Tabela 1 apresenta uma visão compacta das soluções mais significativas quando comparadas a proposta deste artigo. Além disso, é incluída também a *bigER*, um *plugin* lançado recentemente para VS Code e que utiliza o *framework* Xtext como base.

A ferramenta *Multi-paradigm System Modeling Tool* (MIST) foi desenvolvida na universidade de Novi Sad na Sérvia. A *dbdiagram.io*[1] é uma ferramenta web gratuita para o desenho de diagrama ER, desenvolvida por uma empresa de Singapura, com uma abordagem textual que implementa uma DSL própria. Esta DSL utiliza um modelo muito próximo do lógico. O diferencial da ferramenta é sua rápida curva de aprendizagem e, além disso, a apresentação de uma representação gráfica do que está sendo modelado. A apresentação dos elementos do diagrama pode ser organizada livremente pelo usuário

[1] https://dbdiagram.io/

**Tabela 1. Ferramentas de Modelagem Relacionadas.**

| | FERRAMENTAS DE MODELAGEM | | | | | |
|---|---|---|---|---|---|---|
| ***Características*** | **ERtext** | **MIST** | **dbdiagram.io** | **QuickDBD** | **RelaX** | **bigER** |
| Integração IDE | Eclipse | Eclipse | | | | VS Code |
| Web-based | | | ✓ | ✓ | ✓ | |
| Implementação de LSP | ✓ | ✓ | | | | ✓ |
| Open Source | ✓ | ✓ | | | ✓ | ✓ |
| Gratuito | ✓ | ✓ | ± | ± | ✓ | ✓ |
| Editor Textual | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Validação de Modelo | ✓ | | | | | ✓ |
| Visão de Diagrama | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Geração de Código | ✓ | | ✓ | ✓ | | ✓ |
| Avaliação Experimental | ✓ | ✓ | | | | |

*Legenda = IDE: Integrated Development Environment, LSP: Language Server Protocol, ±: Parcialmente.*

em tempo real. Entretanto é importante salientar que toda a modelagem de fato é realizada de modo textual. A ferramenta ainda oferece a geração automática de código SQL. Da mesma forma, a *QuickDBD*[2], desenvolvida por uma empresa na Irlanda, é uma ferramenta web com exatamente o mesmo modo operacional que a *dbdiagram.io*, também implementando uma DSL textual própria para modelagem de banco de dados. Contudo é uma ferramenta proprietária, ou seja, possui custo e tem o foco declaradamente na indústria. Ambas as ferramentas são muito similares também quanto a geração de representações gráficas da modelagem e apresentam diversos argumentos para sua adoção, como a rápida compreensão de suas DSLs, a perspectiva de realização de trabalhos fluídos, o acesso de qualquer plataforma e o compartilhamento dos modelos com outros usuários. A ferramenta web gratuita *RelaX (Relational Algebra Calculator)*[3] foi desenvolvida na universidade de Innsbruck, na Áustria, e é voltada ao ensino de álgebra relacional fazendo operações sobre bases de dados relacionais. Tem uma abordagem textual, utilizando uma DSL chamada RelAlg, e apresenta duas formas de operação: instruções de RelAlg e SQL. Finalmente, a *bigER* oferta recursos para especificar e visualizar modelos de dados conceituais de ER de maneira flexível, também com uma DSL própria. Esta solução ainda se destaca por incorporar o *Language Server Protocol (LSP)*.

## 3. Desenvolvimento

O desenvolvimento da proposta de ferramenta de modelagem é motivado pela continuação do trabalho de [Lopes 2019]. Este estudo definiu requisitos da linguagem, as decisões de projeto e a gramática inicial derivada. Neste ponto, o trabalho possuía a implementação de um protótipo de ferramenta com gerador para um modelo lógico em HTML.

Posteriormente, este gerador foi totalmente refatorado e evoluído, principalmente no que diz respeito a modularização de métodos e conformidade com o desenvolvimento dos outros geradores. Todos estes conceitos foram reutilizados para a implementação da nova infraestrutura do editor de suporte à DSL.

### 3.1. Requisitos da Linguagem

Esta seção lista os requisitos que foram definidos com base na literatura utilizada [Lopes et al. 2022], bem como no conhecimento prévio dos pesquisadores envolvidos na condução do estudo. Estes requisitos são relacionados diretamente com as decisões de

[2]https://quickdatabasediagrams.com/

[3]https://dbis-uibk.github.io/relax/

projeto. ***RQ1. A DSL precisa ser disponibilizada sob uma licença open source.*** Como o foco da proposta é no processo de ensino, entende-se ser fundamental que a linguagem seja de código aberto. A vantagem que este requisito proporciona é a posterior evolução e manutenção colaborativa com o envolvimento de outros desenvolvedores; ***RQ2. A DSL deve permitir representar textualmente modelos conceituais de banco de dados.*** Como é um objetivo que a solução seja outra opção em relação às abordagens gráficas, esse requisito se justifica. Isso permite o foco na compreensão do domínio e no desenvolvimento da DSL; ***RQ3. Os modelos conceituais devem dar suporte à definição de entidades, atributos, relações e cardinalidades.*** As ferramentas utilizadas para o desenvolvimento da linguagem precisam permitir que sejam implementados os conceitos de domínio que regem a estrutura de diagrama ER tradicional; ***RQ4. Os modelos conceituais devem dar suporte a definição de atributos identificadores, generalização/especialização, auto-relacionamentos e relacionamentos ternários.*** A linguagem deve permitir que conceitos mais sofisticados dos domínios sejam definidos; ***RQ5. A implementação da DSL deve realizar a transformação do modelo conceitual para o lógico.*** A solução deve realizar a transformação do conceitual para o lógico, exibindo o resultado gerado ao usuário; ***RQ6. A implementação da DSL deve gerar instruções SQL equivalentes, com base no modelo conceitual ou lógico.*** A solução precisa realizar a geração de instruções SQL para diferentes SGBDs; ***RQ7. A implementação da DSL deve gerar alguma forma de representação gráfica.*** A solução precisa realizar a geração de diagramas com base no modelo conceitual.

### 3.2. Decisões de Projeto

Nesta seção são descritas as decisões de projeto (DP) para criar a DSL textual que suporte todos os requisitos apresentados. Para cada decisão de projeto são indicados os seus requisitos associados. ***DP1. A solução deve adotar um language workbench (LW) open source no auxílio da implementação da DSL textual (RQ1, RQ2).*** Mediante a investigação conduzida no estudo previamente citado [Lopes 2019], foi selecionado o LW Xtext para o desenvolvimento da proposta por ser um *framework open source* focado no desenvolvimento de DSLs textuais, fornecendo toda a infraestrutura necessária. Além disto, o Xtext é uma ferramenta com alto nível de maturidade, documentação detalhada e uma comunidade extremamente ativa; ***DP2. A DSL deve fornecer uma representação textual que seja equivalente ao modelo ER gráfico usualmente utilizado (RQ3, RQ4).*** Para os requisitos cobertos por esta decisão de projeto foi adotada a estratégia de se realizar uma análise nas ferramentas averiguadas no mapeamento descrito em [Lopes 2019], bem como no livro referência de [Heuser 2009]; ***DP3. A solução deve realizar a transformação entre os modelos (RQ5).*** O Xtext usa modelos do EMF como a representação na memória de qualquer arquivo de texto analisado. Esse grafo de objetos na memória é chamado de árvore sintática abstrata (*Abstract Syntax Tree* - AST). Esses conceitos também são chamados de grafos de objeto de documento (*Document Object Graph* - DOM), modelo semântico ou simplesmente modelo. Desta forma, existe a representação do modelo da gramática na forma de um metamodelo central no núcleo do EMF, chamado de modelo Ecore. Possuindo o Ecore da DSL proposta como uma representação, é possível então aplicar regras de transformação, gerando assim outros modelos; ***DP4. A solução deve prover a integração entre a DSL e outras tecnologias (RQ6).*** A solução deve permitir a realização da exportação dos modelos construídos para um formato de instruções SQL, representando assim o modelo físico; ***DP5. A solução deve prover a geração e visualiza-***

***ção de diagramas dos modelos construídos com a DSL (RQ7).*** A solução deve permitir a visualização dos modelos construídos para um formato gráfico, representando assim o modelo conceitual. Inicialmente, essa integração foi realizada utilizando o PlantUML. O PlantUML é uma ferramenta de código aberto que permite a criação de diagramas a partir de uma linguagem de texto simples.

### 3.3. Gramática

Na fase atual da pesquisa, a linguagem no nível conceitual está atendendo aos requisitos de software da linguagem e decisões de projeto. Na Figura 1 está a gramática em Xtext, que por sua vez é inspirada na notação EBNF (*Extended Backus–Naur Form*).

```
grammar org.xtext.anonymous.anonymous.anonymous with org.eclipse.xtext.common.Terminals
generate anonymous "http://www.xtext.org/anonymous/anonymous/anonymous"

ERModel:
        ("Generate" targetGenerator= ("LogicalSchema" | "PostgreSQL" |
                                      "MySQL" | "Diagram" | "All") ";")?
        domain=Domain ";"
        ("Entities" "{") entities+=Entity+ ("}" ";")
        ("Relationships" "{") relations+=Relation* ("}" ";");

Domain:
        "Domain" name=ID;

Attribute:
        name=ID type=DataType (isKey?="isIdentifier")?;

Entity:
        name=ID ("is" generalization=("total/disjoint"    | "total/overlapped" |
                                      "partial/disjoint" | "partial/overlapped")
        is=[Entity])?
        ("{" attributes+=Attribute
        ("," attributes+=Attribute)* "}")?;

Relation:
        (name=ID) ("[" leftEnding=RelationSideLeft
        "relates"
        rightEnding=RelationSideRight "]")
        ("{" attributes+=Attribute
        ("," attributes+=Attribute)* "}")*;

RelationSideLeft:
        target=[Entity] | target=[Relation] cardinality=("(0:1)" | "(1:1)" | "(0:N)" | "(1:N)") ;

RelationSideRight:
        cardinality=("(0:1)" | "(1:1)" | "(0:N)" | "(1:N)") target=[Entity] | target=[Relation];

enum DataType:
        INT="int" | DOUBLE="double" | MONEY="money" | STRING="string" |
        BOOLEAN="boolean" | DATETIME="datetime" | BLOB="file";
```

**Figura 1. Definição da linguagem.**

### 3.4. Metamodelo

Qualquer DSL pode ser metamodelada, e a gramática concebida nesta pesquisa tem o metamodelo equivalente apresentado na Figura 2.

Os metamodelos no contexto de Arquitetura Orientada a Modelos (*Model-Driven Architecture - MDA*) são expressos usando padrões, *e.g.* MOF (*Meta-Object Facility*) - um padrão da OMG para MDE. Um metamodelo especifica a sintaxe abstrata de uma linguagem de modelagem e, como tal, é um tipo especial de modelo. Pode ser entendido como a especificação do conjunto de todos os modelos possíveis, expressos em uma dada linguagem de modelagem. Cada modelo deve estar em conformidade com seu(s) metamodelo(s), ou seja, deve seguir os padrões de sintaxe, semântica, regras e relações entre conceitos. O metamodelo da linguagem é importante na abordagem utilizada com o Xtext pois possibilita que a DSL e os geradores sejam construídos separadamente.

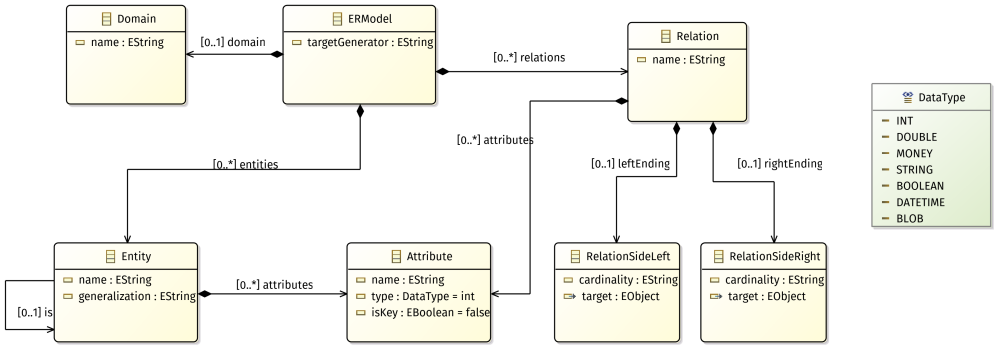


**Figura 2. O metamodelo da linguagem.**

## 3.5. Arquitetura

O *framework* Xtext gera grande parte da infraestrutura para linguagens fundamentalmente com base em gramáticas definidas, indo além dos geradores de *parser* tradicionais (ANTLR, PEG.js, JavaCC, etc). A principal diferença é que no Xtext a gramática também descreve como o *parser* deve instanciar um modelo a partir do texto. Enquanto um *parser* tradicional cria uma árvore de sintaxe genérica, um *parser* Xtext cria um modelo EMF estritamente tipado, incluindo referências cruzadas entre elementos. Dessa forma, é possível ver o Xtext analogamente a um *front-end* para instanciar modelos Ecore dentro do ecossistema Eclipse. A Figura 3 fornece uma visão geral, em um nível abstrato, da arquitetura final da ERtext, já com a adição de geradores que aplicam as transformação entre modelos.

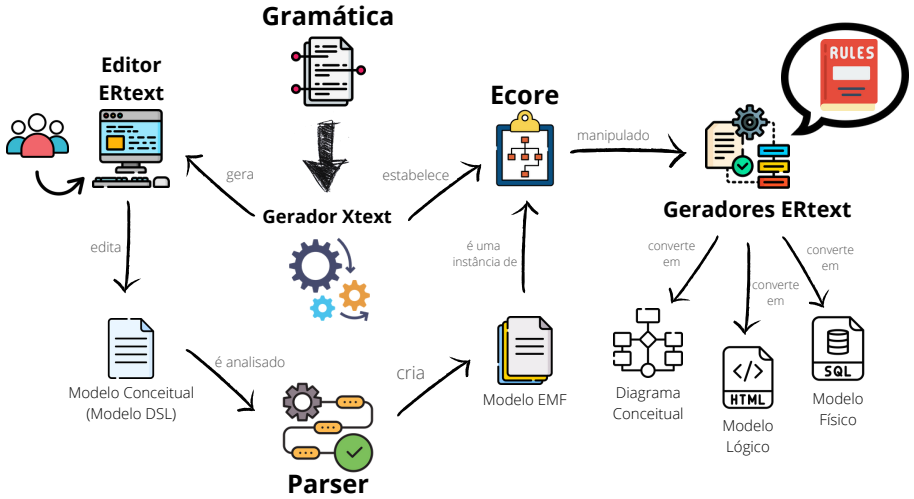


**Figura 3. Visão geral da arquitetura em alto nível.**

Quando uma linguagem própria é definida, a premissa é de que não haverá pessoas familiarizadas com ela. A possibilidade de ter um editor específico de linguagem (IDE) que orienta os usuários na criação de modelos válidos é o maior diferencial do uso deste *framework*. Com base na gramática, o Xtext ainda pode gerar a "inteligência específica" da linguagem para várias plataformas ou editores, *e.g.* Eclipse e IntelliJ IDEA. Isso inclui validação em tempo de execução para erros sintáticos e semânticos, sugestão de trechos

de código, coloração na sintaxe, identificação de referências cruzadas, rápida refatoração, entre outros benefícios.

Além da definição da linguagem e possibilidade de edição inteligente, o *framework* selecionado também viabiliza a criação de qualquer tipo de artefato a partir dos modelos criados. Para isso, o Xtext permite a construção e uso de geradores de código, que são executados automaticamente. Estes artefatos podem ser imagens, documentos, binários ou mesmo código em qualquer outra linguagem de destino, *e.g.* Java, HTML, SQL, etc. Se a linguagem alvo ainda necessitar ser compilada é possível automatizar o processo. Dessa forma, é possível ter o código escrito manualmente, e todos os artefatos gerados, no mesmo ambiente de trabalho. Os geradores da ERtext são escritos em Xtend, que é uma linguagem de programação de propósito geral com base Java e fortemente indicada quando o uso de *Template Expressions* é alto.

## 4. A Ferramenta

O código fonte da versão estável da ferramenta de modelagem produzida está disponível em um repositório público no GitHub[4]. A Figura 4 apresenta um fragmento visual da ferramenta em uso.

A legenda (1) na Figura 4 demonstra o editor com um modelo descrito na DSL implementada. A legenda (2) apresenta uma representação gráfica (diagrama) utilizando a integração com PlantUML. Para isto o modelo é transformado da DSL proposta para a notação necessária do PlantUML, gerando os diagramas utilizando bibliotecas do Graphviz. A legenda (3) mostra o modelo transformado para uma notação lógica, já resolvendo as relações, ou seja, as chaves primárias e estrangeiras. Finalmente, na legenda (4) é apresentado o código SQL gerado para PostgreSQL.

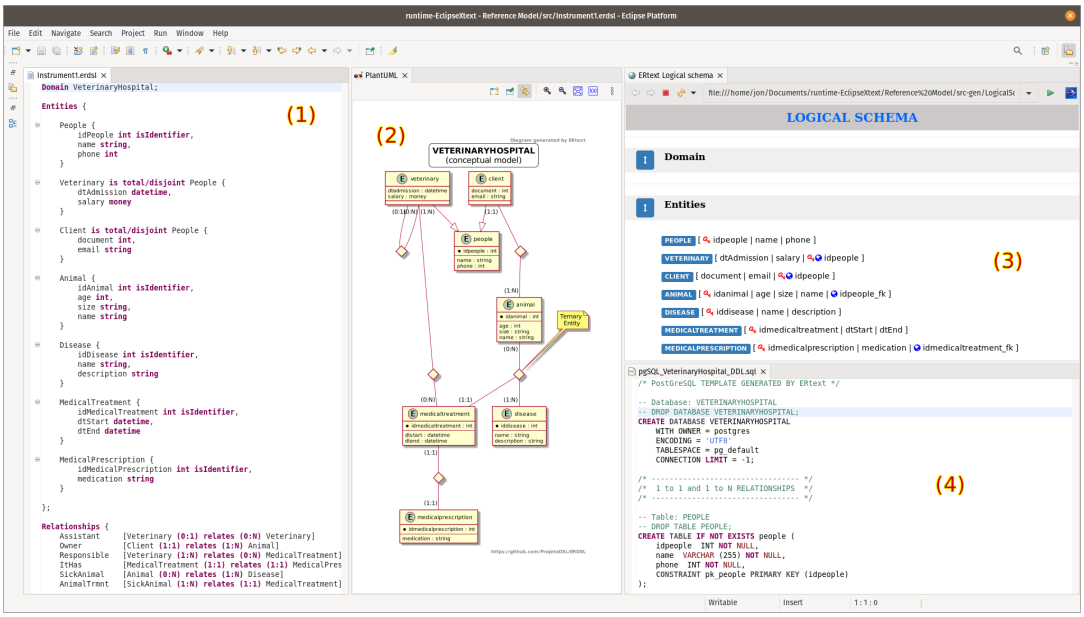


**Figura 4. Versão estável.**

[4]Repositório: `https://github.com/ProjetoDSL/ERDSL`

Para as transformações dos modelos foram avaliadas as opções de regras de transformações, descritas por [Heuser 2009]. Em relação aos relacionamentos do tipo 1:1, adotou-se a estratégia de adição de coluna para os três casos possíveis. Para os relacionamentos 1:N, seguiu-se a mesma premissa de transformação para os quatro (4) casos possíveis. Já para a transformação de relacionamentos N:N, optou-se pela única regra de transformação recomendada, ou seja, a regra de criação de tabelas próprias para os três (3) casos.

Basicamente, durante a atividade de modelagem, um arquivo de texto plano é criado com a extensão da linguagem (.erdsl) no sistema em que a ferramenta está operando. A cada vez que este arquivo é salvo, um modelo Ecore é gerado ou atualizado para então invocar os geradores que lidam com a criação dos artefatos. Isto acontece por meio da interface `IFileSystemAccess2`, que fornece os meios para percorrer os modelos criados e aplicar as regras de transformação. Esta interface oferece uma abstração para operações no sistema de arquivos, possibilitando o mapeamento de caminho lógico (artefato de saída). Esta interface compõe uma série de extensões para `IFileSystemAccess`, atuando juntamente com outras melhorias implementadas para necessidades específicas que podem ser encontradas para cada tipo de modelo de saída.

Para registro, osgeradores especializados utilizam o padrão de projeto *Template Method* na sua implementação. O *Template Method* é um padrão de projeto comportamental, que define o esqueleto de um algoritmo em uma superclasse, deixando as subclasses sobrescreverem etapas específicas do algoritmo sem modificar sua estrutura. Este padrão de projeto preconiza que o algoritmo precisa ser modularizado em uma série de etapas, transformando essas etapas em métodos, e colocando uma série de chamadas para esses métodos dentro de um único método padrão. Na ERtext as etapas dessa transformação são definidas em uma classe abstrata. Desta forma os geradores especializados implementam todas estas etapas abstratas, e eventualmente sobrescrevem algumas das opcionais quando necessário, mas nunca o próprio método padrão.

Esta versão estável ainda conta com a implementação de outros recursos de usabilidade. Entre os que pode-se destacar estão os *wizards* de criação de projetos e arquivos para a DSL, e as recomendações de *templates* de código durante o uso. Finalmente, mas não menos importante, essas funcionalidades não poderiam ter sido implementadas sem a ajuda da comunidade de desenvolvedores Xtext[5].

## 5. Conclusões e Trabalhos Futuros

Pdoe-se afirmar que o uso do *framework* Xtext fornece uma enorme vantagem inicial para criar uma IDE completa para a proposta de linguagem de modelagem conceitual para bancos de dados relacionais. Isso ficou ainda mais evidente a partir do momento em que compreendemos que todo o padrão genérico de infraestrutura provido conta com um baixíssimo nível de acoplamento, ou seja, trabalha com um alto grau de injeção de dependências. Isso permite a alteração de, virtualmente, qualquer configuração de seus componentes.

Desde a concepção da solução ERtext, realizou-se quatro (4) experimentos controlados que, quando combinados, somam cem (100) participações [Lopes et al. 2021a,

[5]Fórum da comunidade Xtext: `eclipse.org/forums/eclipse.modeling.tmf`

Lopes et al. 2021b]. As avaliações experimentais consideraram, de forma quantitativa e qualitativa, uma série suposições envolvendo a abordagem textual da ERtext contra uma abordagem gráfica. O objetivo principal era avaliar se a abordagem textual poderia ter um desempenho e aceitação de usuário tão efetiva, ou ao menos semelhante, à gráfica. Chegou-se a conclusão que ambas as abordagens são similares, sem diferenças estatísticas ou de qualidade de uso que se mostrem significativas dentre o conjunto de sujeitos em todos os experimentos.

A solução proposta utiliza uma perspectiva de *plugin* no momento. Isso se dá pela necessidade do desenvolvimento ágil de uma infraestrutura completa que contemplasse desde o *parser*, *linker*, *typechecker* a até mesmo o compilador completo da linguagem criada. Na atualidade não existem *frameworks* como o Xtext que disponibilizem isso diretamente, de forma especializada, em um contexto *web*. Contudo, apesar da limitação imposta por um *plugin* Eclipse, *i.e.* funciona apenas localmente, o Xtext fornece suporte completo a LSP, o que abre a possibilidade concreta da migração futura da nossa solução para uma plataforma *web*, capaz de ser executada em um navegador web.

Posto isso, existe agora a iniciativa dentro do projeto para a sua migração como um recurso que possa ser consumido via *web*, ou seja, transformar em Software como Serviço (*Software as a Service - SaaS*) [Turner et al. 2003]. Isso é plausível com o auxílio de um *framework* Sprotty de código aberto, uma opção viável para a geração de diagramas dinâmicos, uma vez que a versão atual funciona de forma *stand-alone* e gera diagramas estáticos. O lado do cliente será implementado em TypeScript, com imagens SVG para renderização e CSS para definir o estilo da apresentação. Por outro lado, o servidor remoto (*backend*) será mantido com a infraestrutura construída com o Xtext, já que o mesmo é compatível com o Sprotty, ou seja, será possível o reaproveitamento total do esforço de codificação realizado até o presente momento. Para a comunicação entre cliente e servidor, o Sprotty fornece ainda um protocolo JSON muito simples e extensível, graças ao compatibilidade com LSP. Esta migração se mostra possível quando observa-se a proposição da bigER, já citada nos trabalhos relacionados, que utiliza de uma lógica similar, porém dentro do ambiente VS Code feito em Electron, combinando tecnologias *web* como JavaScript e CSS para apresentação, e Xtext com Node.js em um servidor local.

Existem ainda outras melhorias que serão desenvolvidas. Entre os principais recursos, estão: **(i)** melhorar a validação com base em escopo, em especial para entidades ternárias (atualmente esta validação se dá de forma parcial); **(ii)** criar geradores de DDLs para procedimentos armazenados cobrindo operações CRUD; **(iii)** continuar a implementação do suporte de mineração de código[6], funcionalidade que atualmente está parcialmente implementada. **(iv)** criar mais geradores SQL, aumentando assim o conjunto de plataformas alvo. **(v)** implementar meios de personalização de coloração dos modelos conceituais (diagramas com a PlantUML), lógico (.html) e físico (SQL) gerados automaticamente. Atualmente a coloração é definida estaticamente nos geradores desenvolvidos. **(vi)** disponibilizar a geração automática de diagramas de ocorrência para relacionamentos indicados durante a modelagem. **(vii)** implementar técnicas de *Round-Trip Engineering* (RTE) [An et al. 2008], ou seja, métodos de engenharia reversa buscando maneiras de gerar os artefatos (modelos conceituais, lógicos e físicos) a partir de instruções SQL. **(viii)** investigar possíveis integrações com *frameworks Object Relational Mapper* (ORM) e lin-

[6]Code Mining: `https://www.eclipse.org/eclipse/news/4.8/platform_isv.php`

guagens de programação compatíveis.

## Referências


An, Y., Hu, X., and Song, I.-Y. (2008). Round-trip engineering for maintaining conceptual-relational mappings. In Bellahsène, Z. and Léonard, M., editors, *Advanced Information Systems Engineering*, pages 296–311, Berlin, DE. Springer.

Bailer-Jones, D. (2009). *Scientific Models in Philosophy of Science*. University of Pittsburgh Press.

Brambilla, M., Cabot, J., and Wimmer, M. (2017). *Model-Driven Software Engineering in Practice, 2nd Ed.* Synthesis Lectures on Software Engineering. Morgan & Claypool.

Heuser, C. A. (2009). *Projeto de Banco de Dados*. Bookman, Porto Alegre, Brasil.

Hossain, B., Rajan, G., and Schwitter, R. (2019). *CNLER: a controlled natural language for specifying and verbalising entity relationship models.*, page 126–135. Australasian Language Technology Association.

Lopes, J. (2019). Ertext: uma linguagem específica de domínio para a representação de modelos conceituais de bancos de dados relacionais.

Lopes, J., Bernardino, M., Basso, F., and Rodrigues, E. (2021a). Empirical evaluation of a textual approach to database design in a modeling tool. In *Proc. of the 23th International Conference on Enterprise Information Systems (ICEIS)*, page 8. SciTePress.

Lopes, J., Bernardino, M., Basso, F., and Rodrigues, E. (2021b). Textual approach for designing database conceptual models: A focus group. In *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development - MODELSWARD,*, pages 171–178. INSTICC, SciTePress.

Lopes, J., Bernardino, M., Basso, F., and Rodrigues, E. (2022). Entity-relationship modeling tools and dsls: is it still possible to advance the state of the art from observations in practice? In *Proc. of the 24th International Conference on Enterprise Information Systems (ICEIS)*, page 8. SciTePress.

Malhotra, A., Markowitz, H., Tsalalikhin, Y., Pazel, D., and Burns, L. (1989). An entity-relationship programming language. *IEEE Transactions on Software Engineering*, 15(9):1120–1130.

Riquelmo, J., Bernardino, M., Basso, F., and Rodrigues, E. (2021). Multivocal literature mapping on dsls and tools for entity-relationship modeling. In *Anais da V Escola Regional de Engenharia de Software*, pages 149–158, Porto Alegre, RS, Brasil. SBC.

Schmidt, D. C. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31.

Turner, M., Budgen, D., and Brereton, P. (2003). Turning software into a service. *Computer*, 36(10):38–44.

Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific Languages: An Annotated Bibliography. *SIGPLAN Not.*, 35(6):26–36.