Uma Linguagem Específica de Domínio para a Representação de Modelos Conceituais de Bancos de Dados Relacionais

Jonnathan Riquelmo[1], **Maicon Bernardino**[1],
Fábio Paulo Basso[1], **Elder Macedo Rodrigues**[1]

[1]Laboratory of Empirical Studies in Software Engineering (LESSE),
Engenharia de Software, Universidade Federal do Pampa (UNIPAMPA)
Av. Tiarajú, 810 - Bairro Ibirapuitã - Alegrete, RS

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

Abstract. *With the advancement of technology, databases have become vital elements in contemporary society. That said, the training in the area for professionals coming from academia must be constant. In order to contribute to a relevant open source alternative, this study proposes a Textual Specific Domain Language to support the teaching-learning process of conceptual database modeling. For this there was the selection of the framework Xtext to support the initial development. Requirements and design decisions were designed to then carry out the preliminary definition of a grammar. Implementing a working prototype and integrating DSL into an Eclipse Rich Client Platform (RCP) was done. Thus, there was the previous test where the modeling process with the new language gained native features like formatting, validation and textit syntax highlighting.*

Resumo. *Com o avanço da tecnologia os bancos de dados passaram a ser elementos vitais na sociedade contemporânea. Posto isto, a capacitação na área para profissionais oriundos da academia deve ser constante. Objetivando contribuir com uma alternativa open source relevante, este estudo propõe uma Linguagem de Domínio Específico textual para apoiar o processo de ensino-aprendizagem da modelagem conceitual de banco de dados. Para isto houve a seleção do framework Xtext para apoiar o desenvolvimento inicial. Requisitos e decisões de projeto foram concebidos para então ser realizada a definição preliminar de uma gramática. A implementação de um protótipo funcional e a integração da DSL em um RCP (Rich Client Platform) Eclipse foi feita. Dessa forma, houve o teste prévio onde o processo de modelagem com a nova linguagem ganhou recursos nativos como formatação, validação e syntax highlighting.*

1. Introdução

No processo de desenvolvimento de software, os modelos de dados podem sofrer diferentes transformações. A abordagem de modelagem de três níveis para projeto de banco de dados [ANSI 1975] sugere separar modelos de dados em diferentes graus de abstração, conhecidos como modelos conceituais, lógicos e físicos. A modelagem conceitual de sistemas de bancos de dados é a que requer o mais alto nível de abstração, sendo que a técnica mais difundida para sua elaboração é a proposta de modelagem conceitual de dados concebida por [Chen 1976] que, de tão bem aceita, passou a ser considerada uma referência definitiva para modelagem de banco de dados (BD).

A modelagem de BDs é uma parte importante do ciclo de vida de desenvolvimento de software, na qual geralmente é suportado por alguma Linguagem de Domínio Específico (*Domain-Specific Language* - DSL) [Voelter 2009], como a *Structured Query Language* (SQL). De acordo com [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. Como outras linguagens, uma DSL deve apresentar um conjunto de sentenças bem definidas por uma sintaxe e semântica em si. Para [Fowler 2010] uma DSL é definida como uma linguagem de programação de computadores com expressividade limitada e focada em um domínio particular.

Além das abordagens de *code-first*, a adoção de DSLs em processos de software promove um aumento significativo na produtividade, qualidade, facilidade de uso e flexibilidade [Mernik et al. 2005, Vara et al. 2014]. A maior vantagem no uso de DSLs é a abstração, na qual o conhecimento necessário para o desenvolvimento é levado a um nível menos complexo. Desta forma, especialistas em domínio podem entender, validar e modificar o código por meio de transformações e refinamentos de modelos [Van Deursen et al. 2000].

De acordo com [Mernik et al. 2005], as DSLs podem ser classificadas em três dimensões: origem (interna ou externa), aparência (textual, gráfica, tabular ou simbólica) e implementação (modo de execução). Em geral, a principal consideração para a criação de uma DSL deve ser sua origem, uma vez que cada abordagem tem vantagens e desvantagens específicas [Fowler 2010]. Uma DSL interna é projetada a partir das regras gramaticais de uma linguagem existente, que pode ser uma linguagem de programação de propósito geral (*General-Purpose Programming Language* - GPL) ou outra DSL. Uma DSL externa é uma linguagem com sintaxe distinta e depende de sua própria infraestrutura para análise léxica, sintática, semântica, interpretação, compilação, otimização e geração de código [Mernik et al. 2005]. Quanto a execução, conceitualmente, essa dimensão engloba desde a especificação de notações gráficas/textuais de linguagens até soluções implementadas de fato.

Desde os primórdios desenvolvedores utilizam texto para especificar produtos de software. As linguagens de programação aumentam o nível de abstração de maneira similar aos modelos, resultando assim em linguagens de modelagem textual. Essas linguagens geralmente são processadas por mecanismos que transformam as informações expressas em formato textual para modelos. Esses mecanismos baseiam sua execução na estrutura sintática de uma linguagem de modelagem textual, formalizada em uma gramática. Uma gramática define palavras-chave de uma linguagem, o aninhamento de seus elementos e também a notação de suas propriedades. Entre os potenciais ganhos em uma abordagem textual de modelagem pode-se citar a melhor transmissão de detalhes, maior coesão, edição rápida e possibilidade de uso de editores genéricos.

Posto isso, sabe-se que o ensino na área de BD é parte essencial da formação de profissionais de computação. Com a premissa de que existe uma crescente busca por instrumentos que apoiem o processo de ensino-aprendizagem na academia, o objetivo deste trabalho é propor a especificação e realizar a implementação de uma DSL com abordagem textual, que possua uma gramática de fácil uso e compreensão, para o projeto e modelagem conceitual de BDs relacionais.

Este artigo está organizado da seguinte maneira. A Seção 2 apresenta os trabalhos relacionados. A seguir, ocorre a demonstração da proposta na Seção 3. Finalmente, as considerações preliminares são discutidas na Seção 4, em que ainda é apontada a perspectiva para trabalhos futuros.

2. Trabalhos Relacionados

O estudo de [Dimitrieski et al. 2015], desenvolvido na universidade de Novi Sad na Sérvia, apresenta uma ferramenta chamada *System Modeling Tool* (MIST). Essa ferramenta utiliza uma DSL chamada EERDSL, uma linguagem com base no modelo aprimorado de entidade-relacionamento, do inglês *Enhanced Entity–Relationship* (EER). Apresenta uma abordagem de modelagem bidirecional (gráfica e textual) de modelagem de BD. O autor discute que tal decisão tem como motivo o entendimento de que a preferência sobre a abordagem de modelagem utilizada pode depender do domínio do problema, do conhecimento e das preferências pessoais de um projetista de BD. Descreve também uma experiência anterior, onde foi construída uma ferramenta para modelagem com uma abordagem baseada em formulários, e que a partir dos resultados obtidos nesta experiência foi concebida a ideia da MIST. O propósito da ferramenta é a aplicação tanto no mercado profissional quanto para o ensino de projeto e modelagem de BD no meio acadêmico. A MIST foi desenvolvida com o auxílio do Xtext para a notação da DSL textual, e inicialmente a Eugene e posteriormente a Sirius para a sua versão gráfica. Esta ferramenta ainda oferece suporte a geração de código SQL.

O *dbdiagram.io*[1] é uma ferramenta *Web* gratuita para o desenho de Diagrama Entidade-Relacionamento (DER), 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 em tempo real. Entretanto é importante se salientar que toda a modelagem de fato é feita 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 ferrramenta *Web* com exatamente o mesmo modo operacional que a *dbdiagram.io*, também implementando uma DSL textual própria para modelagem de BDs em nível lógico. Contudo é uma ferramenta proprietária e com o foco declaradamente na indústria.

Finalmente, pode-se citar a ferramenta *Web* gratuita *RelaX (Relational Algebra Calculator)*[3]. Esta ferramenta não foi encontrada no mapeamento, mas indicada por um pesquisador da área de DSLse BDs. Trata-se de uma ferramenta desenvolvida na universidade de Innsbruck, na Áustria, e voltada ao ensino de álgebra relacional fazendo operações sobre bases de dados relacionais. Tem uma abordagem textual, utilizando uma DSL chamada RelAlg, e apresentando inclusive duas perspectivas de operação: instruções de RelAlg e instruções em SQL. A RelaX utiliza uma abordagem de modelo já em nível físico para operações, como as DDLs de construção e DMLs para as consultas. Apesar de suas funcionalidades, a RelaX não se propõe a ser uma ferramenta de projeto e

[1] https://dbdiagram.io/

[2] https://quickdatabasediagrams.com/

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

modelagem de BD, mas de uso restrito ao ensino dentro da academia.

3. Proposta de DSL

Esta Seção apresenta a proposta central deste trabalho, sumarizando os requisitos da linguagem e as decisões de projeto concebidas. Após, é apresentada a implementação do protótipo seguido de uma demonstração de uso.

3.1. Requisitos da Linguagem e Decisões de Projeto

Esta seção lista os requisitos (RQs) que foram definidos com base na literatura utilizada neste trabalho, bem como no conhecimento prévio dos pesquisadores envolvidos na condução do estudo. Estes requisitos são relacionados diretamente com as decisões de projeto (DPs).

- ***RQ1. A DSL precisa ser disponibilizada sob uma licença open source.*** Como o foco da proposta é no processo de ensino é 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 BDs.*** Como é um objetivo que a solução seja outra opção em relação as 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 a 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 DER tradicional.
- ***RQ4. Os modelos conceituais devem dar suporte a definição de atributos identificadores, generalização/especialização, autorelacionamentos 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 modelos físicos equivalentes, com base no modelo conceitual ou lógico.*** A solução precisa realizar a geração de instruções SQL para diferentes SGBDs.

Para atender o propósito do trabalho foram tomadas algumas decisões de projeto para criar a DSL textual que suporte todos os requisitos levantados. Para cada decisão de projeto são indicados os seus requisitos associados.

- ***DP1. A solução deve adotar um LW open source no auxílio da implementação da DSL textual (RQ1, RQ2).*** Mediante a investigação conduzida durante este estudo 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 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 nos trabalhos relacionados, 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, do inglês *Abstract Syntax Tree* (AST). Esses conceitos também são chamados de gráficos de objeto de documento, do inglês *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*. Tendo 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. Essa integração será realizada para o SQL Server, o MySQL e o PostgreSQL.

3.2. Protótipo da DSL

Na fase atual a linguagem não se encontra totalmente finalizada. Existem tópicos relativos a validação de escopo, como no caso do tratamento de referências cruzadas indesejadas, e outras restrições inerentes a modelagem conceitual que devem ser analisadas e então implementadas. A Figura 1 exibe a definição da DSL criada.

O comando `grammar` especifica o nome da DSL, enquanto que a instrução `with` declara uma herança de outra linguagem. No caso da gramática proposta, será utilizado uma gramática padrão do Xtext, chamada `Terminals`, a qual fornece algumas regras predefinidas como, por exemplo, a regra `ID` para identificadores ou `INT` para inteiros. O comando `generate` é a instrução que produz a AST da linguagem.

A primeira regra, chamada de regra de entrada, define como é a estrutura geral da linguagem. Palavras e símbolos entre aspas duplas ou simples indicam as palavras reservadas. Por exemplo, o objeto `Entities` é obrigatoriamente precedido de "`Entities{`". Este objeto representa uma espécie de *container*, sendo isto indicado por meio do operador de atribuição +=. Ele é um objeto que pode conter outros objetos, no caso um ou mais `Entity` (entidades). É estabelecido que cada arquivo da DSL deve também ser composto de um `Domain` (domínio) e zero ou mais `Relation` (relações).

Para melhor entendimento, deve-se deixar claro que a multiplicidade é indicada por `*` (zero ou muitos), + (um ou muitos) ou `?` (zero ou um). Ao não se colocar nenhum desses operadores, implicitamente se espera então apenas uma ocorrência. Em relação as atribuições, quando apenas um = for especificado significa que o objeto da esquerda espera apenas um registro. Logo, para += espera-se então zero, uma ou mais ocorrências.

O objeto `Domain` é precedido de uma palavra reservada com o mesmo nome, seguido de um identificador. A entidade é definida pela palavra `Entity` e um nome identificador específico para este objeto. A definição de uma herança é opcional por meio da palavra reservada `isA`. Após a definição do nome, abre-se um corpo de chaves em

```
grammar org.xtext.unipampa.lesse.erdsl
with org.eclipse.xtext.common.Terminals

generate erdsl "http://www.xtext.org/unipampa/lesse/erdsl/"

ERModel:
        domain=Domain ";"
        ("Entities{") entities+=Entity+ ("};")
        ("Relationships{") relations+=Relation* ("};");

Domain:
        "Domain" name=ID;

Entity:
        name=ID ("isA" isA+=[Entity])*
        ("{" attributes+=Attribute ("," attributes+=Attribute)* "}")?;

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

Relation:
        (name=ID)?
        ("[" leftEnding=RelationSide
        "isRelatedWith"
        rightEnding=RelationSide "]")
        ("{" attributes+=Attribute
        ("," attributes+=Attribute)* "}")*;

RelationSide:
        ((minimalCardinality?="zero")?) maximumCardinality=CardinalityType
        target=[Entity] | target=[Relation];

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

enum CardinalityType:
        One="one" | Many="many";
```

Figura 1. Implementação do prótipo da DSL.

que são especificados os atributos da entidade. Uma entidade deve conter ao menos um atributo, mas ele não precisa ser identificador por conta da possível existência de entidades fracas.

As regras compostas só são realizadas devido a possibilidade de se agrupar expressões com o uso de parênteses, além da possibilidade de se utilizar outras regras por meio de referências cruzadas. Os colchetes entre a regra `Entity` servem para indicar que se almeja usar apenas o atributo `name` que identifica o objeto. Os atributos das entidades são definidos por um nome, herdando a regra `ID` de `Terminals`, e atributos `isIdentifier` opcionais para simbolizar chaves primárias.

A relação é definida, já dentro do corpo do bloco `Relationships`, com uma declaração opcional de sua identificação. Em seguida, são abertos colchetes e deve-se especificar dois elementos `RelationSide` como referência aos atributos `leftEnding` e `rightEnding`. Estes atributos representam os lados de uma relação. Estes objetos devem ser separados pela expressão `isRelatedWith`. Os lados da relação são definidos na regra *RelationSide*, composta de dois atributos. O atributo `minimalCardinality` é opcional, indicado pelo operador `?`, e aceita apenas a palavra reservada `zero`. O atributo `maximumCardinality` aceita um objeto `CardinalityType`.

Os tipos de atributo estão contidos em uma lista enumerada chamada `DataType`, na esquerda fica a representação no modelo *Ecore*. Na direita está a palavra reservada que é usada na linguagem. O símbolo condicional `|` significa o operador lógico *OR* (ou) e serve para separar cada definição `<chave> = <valor>` como uma opção dentro da lista. Das três cardinalidades possíveis, duas estão definidas em outra lista enumerada chamada `CardinalityType`.

Na Figura 2 mostra-se um exemplo de uso com um pequeno modelo que aborda uma universidade como domínio. Neste protótipo já se pode observar a modelagem de generalização/especialização, auto-relacionamento e relacionamento ternário.

```
Domain University;

Entities{
        Person{
                PersonID: int isIdentifier,
                Name: string
        }
        Teacher isA Person{
                Phone: int,
                Salary:money
        }
        Student isA Person{
                Course: string
        }
        OutsrcEmployee isA Person{
                OutsourcedEID: int isIdentifier,
                Company: string
        }
        Class{
                ClassID: int isIdentifier,
                Course: string,
                Semester: string
        }
        ClassRoom {
                ClassRoomID: int isIdentifier,
                Capacity: int
        }
};

Relationships{
        [many Student isRelatedWith many Class]
        TeacherClass [many Teacher isRelatedWith many Class]
        ClassSchedule [many TeacherClass isRelatedWith many ClassRoom]
                {ClassScheduleID: int, DayOfWeek: datetime, Discipline: string}
        Supervisor [one OutsrcEmployee isRelatedWith many OutsrcEmployee]
};
```

Figura 2. Exemplo de uso do protótipo da DSL no RCP Eclipse.

É importante destacar que neste exemplo são modeladas seis entidades e quatro relacionamentos. Contudo, no processo de transformação para o modelo lógico espera-se que seja gerado um esquema textual com mais três entidades, inferindo-se isso através de, por exemplo, relacionamentos *muitos para muitos*. Isso aconteceria, nesta amostra, no relacionamento sem identificação, atribuindo automaticamente a nova entidade o nome resultante da concatenação das duas entidades que se relacionam no conceitual. Também seriam criadas novas entidades a partir da derivação dos relacionamentos nomeados `TeacherClass` e `ClassShedule`, sendo que o último carateriza um relacionamento ternário. Por fim, o auto-relacionamento `Supervisor` de `OutsrcEmployee`

implicaria na adição de um novo atributo na entidade no novo modelo.

Com base nos modelos *Ecore* gerados em tempo real, como o gerado a partir do modelo descrito anteriormente, já está sendo implementada a transformação do modelo conceitual para o lógico. A implementação desta transformação está sendo realizada utilizando-se a Xtend, uma GPL baseada em Java.

4. Considerações Preliminares

Até o momento o Xtext mostrou-se um ambiente de desenvolvimento, conhecido como *Language Workbench*, capaz de suprir as necessidades iniciais do projeto, fornecendo suporte completo para a criação de gramáticas com notação *Backus-Naur Form* (BNF), uma meta-sintaxe amplamente usada para expressar gramáticas livres de contexto como nas estruturas de linguagens de programação no geral. Além de prover a validação da gramática criada, foi gerado um *plugin* tornando assim possível a realização do teste do protótipo em um RCP Eclipse.

Este estudo faz parte de um projeto de pesquisa realizado pelo *Laboratory of Empirical Studies in Software Engineering* (LESSE) da UNIPAMPA. O projeto tem como objetivo construir uma ferramenta que utilize esta DSL textual para modelagem conceitual de SGBDs relacionais, a fim de oferecer uma alternativa para o ensino de alunos de graduação em modelagem e projeto de BDs. Pretende-se que a proposta final não apenas realize modelagem, mas sim a transformação dos modelos conceituais gerados em *scripts* SQL para diferentes tecnologias SGBDs, *e.g.* PostgreSQL, MySQL e SQL Server.

Referências

ANSI, A. N. S. I. (1975). Interim Report: ANSI/X3/SPARC Study Group on DBMSs 75-02-08. *ACM SIG on Management of Data*.

Chen, P. P.-S. (1976). The Entity-relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36.

Dimitrieski, V., Čeliković, M., Aleksić, S., Ristić, S., Alargt, A., and Luković, I. (2015). Concepts and Evaluation of the Extended Entity-relationship Approach to Database Design in a Multi-paradigm Information System Modeling Tool. *Computer Languages, Systems & Structures*, 44(Part C):299–318.

Fowler, M. (2010). *Domain Specific Languages*. Addison-Wesley Professional.

Heuser, C. (2009). *Projeto de banco de dados : Volume 4 da Série Livros didáticos informática UFRGS*. Livros didáticos informática UFRGS. Bookman.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344.

Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages: An annotated bibliography. *ACM SIGPLAN Notices*, 35(6):26–36.

Vara, J., Bollati, V., Jiménez, A., and Marcos, E. (2014). Dealing with Traceability in the MDD of Model Transformations. *Transactions on Software Engineering*, 40(6):555–583.

Voelter, M. (2009). Best Practices for DSLs and Model-Driven Development. *Journal of Object Technology*, 8(6):79–102.