

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO AMAZONAS**  
**CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**  
**CAMPUS MANAUS CENTRO**

**Sérgio Roberto Pinto de Oliveira**

**CONCEPTER 3.0: MÓDULO GERADOR DE MODELOS RELACIONAIS**

MANAUS

2019

**Sérgio Roberto Pinto de Oliveira**

**CONCEPTER 3.0: MÓDULO GERADOR DE MODELOS RELACIONAIS**

Trabalho de Conclusão de Curso apresentado a banca examinadora do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Amazonas, como requisito para aprovação na disciplina TCC 2 – Desenvolvimento de Software.

Orientador: Prof. Msc. Marcelo Chamy Machado

MANAUS

2019

## RESUMO

O atual cenário da área de tecnologia da informação exige que profissionais estejam capacitados e familiarizados com os conceitos e ferramentas disponíveis no mercado. Especificamente, o profissional especialista em banco de dados, precisa usar ferramentas intuitivas que proporcionem uma melhor interação entre o trabalho que ele deseja realizar e a forma como este trabalho pode ser planejado. Um projeto de banco de dados é subdividido em algumas etapas, onde cada etapa tem como propósito criar uma visão relacionada a níveis de percepção. O software ConceptER, atualmente, trabalha na primeira etapa, que se trata do nível mais alto e abstrato, chamado de modelo conceitual. Porém esta limitação impede que profissionais, professores e alunos necessitem mudar de sistema para dar continuidade nos demais níveis de modelagem. Logo, este trabalho apresenta uma proposta de evolução desta ferramenta, através da concepção de uma nova interface que contemple tanto modelo conceitual, quanto o modelo lógico, além de integrar uma ferramenta com a funcionalidade de conversão entre estes modelos respectivamente.

**Palavras-Chave:** Banco de dados, Modelagem de Dados, Modelo Conceitual, Modelo Lógico, Conversão entre Modelos.

## **ABSTRACT**

The current scenario in the area of information technology requires that professionals are trained and familiar with the concepts and tools available in the market. Specifically, the database professional needs to use intuitive tools that provide a better interaction between the work he or she wants to accomplish and how this work can be planned. A database project is subdivided into a few steps, where each step is intended to create a vision related to levels of perception. ConceptER software currently works in the first stage, which is the highest and abstract level, called the conceptual model. However, this limitation prevents professionals, teachers and students from needing to change systems to continue the other levels of modeling. Therefore, this work presents a proposal of evolution of this tool, through the design of a new interface that includes both conceptual model and the logical model, besides integrating a tool with the conversion functionality between these models respectively.

**Keywords:** Database, Data Modeling, Conceptual Model, Logical Model, Conversion between Models.

## LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
AWT	<i>Abstract Windows Toolkit</i>
DDL	<i>Data Definition Language</i>
DER	Diagrama Entidade-Relacionamento
DML	<i>Data Manipulation Language</i>
EER	<i>Enhanced Entity–Relationship</i>
ER	<i>Entity–Relationship</i>
ERD	<i>Entity Relationship Diagram</i>
GUI	<i>Graphical User Interface</i>
IBM	<i>International Business Machines</i>
ID	<i>Identification Data</i>
IDEF1X	<i>Integration Definition for Information Modeling</i>
IE	<i>Information Engineering</i>
IFAM	Instituto Federal de Educação, Ciência e Tecnologia do Amazonas
JAXB	<i>Java Architecture for XML Binding</i>
JVM	<i>Java Virtual Machine</i>
N/A	<i>Not Applicable</i>
PNG	<i>Portable Network Graphics</i>
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	<i>Structured Query Language</i>
TADS	Tecnologia em Análise e Desenvolvimento de Sistemas
TCC	Trabalho de Conclusão de Curso
UC	<i>User Case</i>
UML	<i>Unified Modeling Language</i>
UTF-8	<i>8-bit Unicode Transformation Format</i>
XML	<i>Extensible Markup Language</i>

## LISTA DE ILUSTRAÇÕES

<b>Figura 1</b>	- Esquema gráfico de um modelo conceitual a partir de um minimundo .....	17
<b>Figura 2</b>	- Um diagrama de esquema ER para o banco de dados EMPRESA.....	17
<b>Figura 3</b>	- Notação do diagrama EER para representar subclasses e especializações.....	18
<b>Figura 4</b>	- Esquema gráfico de um modelo lógico .....	19
<b>Figura 5</b>	- Tabela .....	21
<b>Figura 6</b>	- Tabela Funcionários .....	22
<b>Figura 7</b>	- Visão geral do projeto lógico.....	24
<b>Figura 8</b>	- Transformação de entidade em tabela .....	25
<b>Figura 9</b>	- Opções para mapeamento de especialização ou generalização .....	25
<b>Figura 10</b>	- Correspondência entre os modelos ER e Relacional .....	27
<b>Figura 11</b>	- Visual Database Design.....	27
<b>Figura 12</b>	- ERD & Database Engineering.....	28
<b>Figura 13</b>	- ER (Entity-Relationship) diagrams.....	29
<b>Figura 14</b>	- Navicat Data Modeler 2.1 – Exemplo de Modelo Conceitual.....	30
<b>Figura 15</b>	- Tela principal da ferramenta ConceptER .....	31
<b>Figura 16</b>	- XML 1.0 – Exemplo de XML .....	34
<b>Figura 17</b>	- ConceptER 1.0 – Tela Principal com Diagrama.....	36
<b>Figura 18</b>	- Tela principal da ferramenta ConceptER 2.0 .....	37
<b>Figura 19</b>	- Tela principal do Módulo Gerador de Modelos Relacionais.....	38
<b>Figura 20</b>	- Visão geral do processo de conversão .....	39
<b>Figura 21</b>	- Visão geral do processo de conversão do modelo conceitual para o lógico.....	39
<b>Figura 22</b>	- Diagrama de Casos de Uso .....	42
<b>Figura 23</b>	- Diagrama Classes do Módulo Gerador de Modelos Relacionais .....	43
<b>Figura 24</b>	- Diagrama de Pacotes do Módulo Gerador de Modelos Relacionais .....	44
<b>Figura 25</b>	- Diagrama de Sequência: Criar Novo Modelo.....	45
<b>Figura 26</b>	- Submenu Novo .....	45
<b>Figura 27</b>	- JOptionPane – Descartar Alterações .....	46
<b>Figura 28</b>	- Ambiente de Edição com Novo Modelo .....	46
<b>Figura 29</b>	- Script actionPerformed para criar novo modelo.....	47
<b>Figura 30</b>	- Diagrama de Sequência: Criar Novo Modelo.....	47
<b>Figura 31</b>	- Submenu Abrir.....	48
<b>Figura 32</b>	- Script de desserialização do arquivo com modelo salvo .....	49
<b>Figura 33</b>	- Script de parseamento e decodificação dos objetos de criação do gráfico .....	49

<b>Figura 34</b> - Script de alimentação do HashMap da classe de propriedades de tabelas .....	49
<b>Figura 35</b> - Diagrama de Sequência: Converter XML Conceitual em XML Lógico .....	50
<b>Figura 36</b> - Diagrama de Sequência: Fluxo de Conversão Conceitual para Lógico .....	51
<b>Figura 37</b> - Script de desserialização do arquivo XML.....	51
<b>Figura 38</b> - Script de mapeamento de relacionamentos do modelo conceitual .....	52
<b>Figura 39</b> - Script de mapeamento de entidades dos relacionamentos mapeados.....	52
<b>Figura 40</b> - Script de mapeamento de atributos das entidades mapeadas.....	53
<b>Figura 41</b> - Script de mapeamento de atributos multivalorados.....	53
<b>Figura 42</b> - Script de criação de chave-primária genérica.....	54
<b>Figura 43</b> - Script de atribuição para um novo relacionamento lógico .....	54
<b>Figura 44</b> - Diagrama de Sequência: Editar Modelo .....	55
<b>Figura 45</b> - Nova Tabela Criada .....	55
<b>Figura 46</b> - Script de template predefinido para novas tabelas .....	56
<b>Figura 47</b> - Botão ExibirPropriedades .....	56
<b>Figura 48</b> - Script de ação para clique no botão ExibirPropriedades .....	56
<b>Figura 49</b> - Painel de Propriedades de Tabelas .....	57
<b>Figura 50</b> - Tabela Carregada .....	57
<b>Figura 51</b> - Tipos de Relacionamentos .....	58
<b>Figura 52</b> - Tabelas Relacionadas.....	58
<b>Figura 53</b> - Diagrama de Sequência: Salvar Modelo.....	59
<b>Figura 54</b> - Submenu Salvar .....	59
<b>Figura 55</b> - Script de captura de informações do gráfico .....	60
<b>Figura 56</b> - Codificação e serialização do gráfico e propriedades das tabelas .....	60
<b>Figura 57</b> - Interface principal do módulo gerador de modelos relacionais.....	61
<b>Figura 58</b> - Componente gráfico mxGraphOutline .....	62
<b>Figura 59</b> - Painel de propriedades ativo.....	62
<b>Figura 60</b> - Relacionamentos entre tabelas.....	63
<b>Figura 61</b> - Arquivo XML com modelo logico salvo.....	64
<b>Figura 62</b> - Arquivo XML com modelo conceitual salvo .....	65
<b>Figura 63</b> - Arquivo XML com modelo logico convertido .....	66
<b>Figura 64</b> - Fragmento do script incluído em uma nova classe mxMarkerRegistry .....	67

## LISTA DE QUADROS

<b>Quadro 1</b>	- Comparação entre os Trabalhos Relacionados .....	32
<b>Quadro 2</b>	- Requisitos Funcionais .....	40
<b>Quadro 3</b>	- Requisitos Não Funcionais .....	41
<b>Quadro 4</b>	- Cronograma de Atividades .....	69
<b>Quadro 5</b>	- Descrição do caso de uso para criar um novo modelo .....	76
<b>Quadro 6</b>	- Descrição do caso de uso para abrir um modelo salvo .....	76
<b>Quadro 7</b>	- Descrição do caso de uso para editar um modelo .....	77
<b>Quadro 8</b>	- Descrição do caso de uso para salvar um modelo .....	77



## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>10</b>
1.1	Contextualização	10
1.2	Problema	11
1.3	Justificativa	11
1.4	Objetivos	12
1.4.1	Objetivo Geral	12
1.4.2	Objetivos Específicos	12
1.5	Estrutura do Trabalho	13
1.6	Metodologia	13
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>15</b>
2.1	Modelagem de Dados	15
2.2	Modelo Conceitual	16
2.2.1	Modelo Entidade Relacionamento (ER)	17
2.2.2	Modelo Entidade Relacionamento Estendido (EER)	18
2.3	Modelo Lógico	18
2.4	Abordagem Relacional	19
2.4.1	Composição de um Banco de Dados Relacional	20
2.4.1.1	Tabelas	20
2.4.1.2	Chaves	21
2.4.1.3	Domínios	22
2.4.1.4	Valores NULL	23
2.4.1.5	Restrições de Integridade	23
2.5	Conversão do Modelos ER Conceitual para Modelo Relacional	24
2.5.1	Implementação de Entidades	24
2.5.2	Implementação de Relacionamentos	25
2.5.3	Implementação de Generalização e Especialização	26
<b>3</b>	<b>TRABALHOS RELACIONADOS</b>	<b>27</b>
3.1	MySQL Workbench	27
3.2	Visual Paradigm	28
3.3	Astah Professional	29
3.4	Navicat Data Modeler	29
3.5	ConceptER 2.0	30

3.6	Comparação entre os Trabalhos Relacionados.....	31
4	TECNOLOGIAS.....	33
4.1	Linguagem Java .....	33
4.2	Swing .....	33
4.3	JGraphX .....	33
4.4	XML .....	34
4.5	JAXB.....	34
5	IMPLEMENTAÇÃO .....	35
5.1	Funcionalidades .....	35
5.1.1	Modelagem Conceitual.....	35
5.1.2	Modelagem Lógica.....	37
5.1.3	Conversão entre modelos .....	38
5.2	Requisitos de Software .....	39
5.2.1	Requisitos Funcionais.....	40
5.2.2	Requisitos Não Funcionais .....	40
5.3	Casos de Uso.....	41
5.3.1	Diagrama de Caso de Uso .....	41
5.4	Diagrama de Classes.....	42
5.5	Diagrama de Pacotes .....	44
5.6	Diagramas de Sequência .....	44
5.6.1	Diagrama de Sequência Criar Novo Modelo.....	45
5.6.2	Diagrama de Sequência Abrir Modelo Salvo .....	47
5.6.3	Diagrama de Sequência Converter XML Conceitual em XML Lógico.....	50
5.6.4	Diagrama de Sequência Editar Modelo .....	54
5.6.5	Diagrama de Sequência Salvar Modelo .....	59
6	DETALHAMENTO DA FERRAMENTA .....	61
7	EXPERIMENTOS E RESULTADOS .....	67
8	CRONOGRAMA DE ATIVIDADES .....	69
9	CONSIDERAÇÕES FINAIS .....	70
9.1	Trabalhos Futuros .....	70
	REFERÊNCIAS BIBLIOGRÁFICAS.....	72
	APÊNDICE – DESCRIÇÃO DOS CASOS DE USO .....	76

# 1 INTRODUÇÃO

## 1.1 Contextualização

Atualmente as aplicações são frequentemente caracterizadas pela presença de um sistema de armazenamento de banco de dados, que fornece serviços complexos permitindo a recuperação e troca de informações. Os bancos de dados destas aplicações normalmente são concebidos através do resultado de uma modelagem, desde o mais alto nível (conceitual) até o baixo nível (físico), tendo como fonte e parâmetros o mapeamento das regras de negócios descritos sobre um mini-mundo<sup>1</sup>.

Entre as etapas de modelagem Conceitual e a Física, temos a Projeto Lógico (Modelagem Lógica) que segundo Elmasri e Navathe (2011), resulta em “um esquema do banco de dados no modelo de dados de implementação do SGBD”. Muitas questões de pesquisa estão relacionadas à tradução, compatibilidade e coerência entre os modelos de modelagem Conceitual e Lógica, porém produzem apenas embasamento teórico, deixando a desejar a sua aplicação prática em uma ferramenta que possibilite analisar a conversão de um modelo para o outro.

No ano de 2015, Allan Magnum Melo Mendonça, aluno do curso de Tecnologia em Análise e Desenvolvimento de Sistemas - TADS, do Instituto Federal de Educação, Ciência e Tecnologia do Amazonas – IFAM, desenvolveu em seu Trabalho de Conclusão de Curso uma ferramenta CASE, o ConceptER, que tem como propósito auxiliar na criação do Diagrama de Entidade Relacionamento (DER). Esta ferramenta abrange todos os componentes da Modelagem Conceitual e fornece a exportação do resultado da modelagem em formato XML, que segundo o autor visa o direcionamento para trabalhos futuros onde seja possível “gerar um modelo lógico de dados ou criar o banco de dados em si” (MENDONÇA, 2015).

No ano de 2017, Fernanda Soares Nascimento, também aluna do Curso de TADS do IFAM, realizou melhorias na ferramenta, como: *upgrade* de versão dos *frameworks* utilizados e do Java, portabilidade, remoção de *bugs*, melhorias visuais na interface e adição da funcionalidade de especialização contendo apenas uma única subclasse. Com essas melhorias, a ferramenta foi evoluída para a versão ConceptER 2.0.

---

<sup>1</sup> “Um banco de dados representa algum aspecto do mundo real, às vezes chamado de mini-mundo ou de universo de discurso” (ELMASRI E NAVATHE, 2011).

Em meio a estes fatos, torna-se necessário a criação de uma interface para modelagem no nível de Projeto Lógico que se integre à última versão do ConceptER e possibilite também a conversão de forma prática e intuitiva do modelo Conceitual para o Modelo Lógico utilizando o formato de arquivo XML, proposto em sua primeira versão, como fonte estruturada para fornecer os parâmetros de adequação desta transformação. Com intuito de satisfazer estas necessidades, será apresentado neste trabalho a proposta de solução com intuito de evoluir a ferramenta para a versão ConceptER 3.0.

## **1.2 Problema**

Diante da perceptível evolução das tecnologias de concepção e administração de banco de dados relacionais, ainda não existe um software capaz de oferecer recursos de modelagem de dados que integre de forma harmônica os modelos conceitual e lógico.

Várias ferramentas focam todos seus recursos na integração dos modelos lógico e físico, como: relacionamentos, tipificação, conversão lógica para física e reengenharia física para lógica. Porém o modelo conceitual é significativamente necessário, pois sua modelagem não está restrita a hardwares e softwares, onde se torna o modelo primordial do analista para entendimento do negócio, independente do ambiente de dados que possa compor o projeto.

Já outras ferramentas que englobam o modelo conceitual, pecam na integração deste modelo com os demais modelos, lógico e físico, além de graficamente propor elementos estruturais que não satisfazem as regras de notação do modelo entidade-relacionamento (ER) proposto por Peter Chen<sup>2</sup>.

## **1.3 Justificativa**

De acordo com Elmasri e Navathe (2011), a abstração de dados “se refere à supressão de detalhes da organização e armazenamento dos dados destacando recursos essenciais para um melhor conhecimento desses dados”.

A modelagem de dados, no que tange os sistemas de banco de dados, é o processo que visa disponibilizar um artefato para compreender níveis de abstração de dados. Compreende-se

---

<sup>2</sup> CHEN, P. The Entity-Relationship Model: Toward a Unified View of Data. EUA: ACM Transactions on DataBase Systems, v. 1, n. 1, pp. 9-36, 1976

neste aspecto que a modelagem de dados é essencial para descrever a estrutura de um banco de dados, podendo compreender seus relacionamento e restrições.

Para que uma estrutura de banco de dados seja compreendida por completo, é necessário que exista uma modelagem bem elaborada, do mais baixo ao mais alto nível de abstração, ou seja, desde o nível conceitual, atravessando o modelo lógico até o modelo físico.

Considerando estas desvantagens, é possível entender que os acadêmicos e profissionais das diversas áreas, que englobam a tecnologia de sistemas de banco de dados, não possuem uma ferramenta case capaz de satisfazer, através da modelagem de dados, uma visão completa e integrada das perspectivas fundamentais dos modelos conceitual e lógico.

## **1.4 Objetivos**

Nas subseções desta seção, são apresentados o objetivo geral e os objetivos específicos. O objetivo geral apresenta o sentido mais amplo sobre a ação definida para este trabalho. Os objetivos específicos estão divididos em quatro itens, que detalham as particularidades das ações desta temática, tendo como base o objetivo geral.

### **1.4.1 Objetivo Geral**

Criar uma ferramenta gráfica de modelagem de dados para evoluir o Software ConceptER através da integração de recursos entre os modelos conceitual e lógico.

### **1.4.2 Objetivos Específicos**

- Criar uma interface para construção de modelos lógicos;
- Definir estratégia de mapeamento de conversão entre os modelos conceitual e lógico;
- Integrar a interface de modelagem lógica à estrutura do ConceptER;
- Validar a ferramenta através da técnica de inspeção de software.

## 1.5 Estrutura do Trabalho

Posteriormente à introdução, este trabalho foi dividido em quatro partes principais da seguinte forma:

Primeiramente é apresentado a Fundamentação Teórica, que é o resultado da revisão dos materiais literários relevantes para concepção deste trabalho. Logo após, são citados os Trabalhos Relacionados, que contemplam as ferramentas disponíveis no mercado e pesquisas acadêmicas em que foram aplicados o benchmarking, comparando especificações que se assemelham ao objetivo geral do trabalho aqui proposto. Em seguida é apresentado a parte de Tecnologias, que cita e descreve as principais tecnologias que foram e serão utilizadas para a criação de artefatos e evolução da ferramenta ConceptER 3.0. E por fim a Proposta de Solução, que descreve a relevância deste trabalho para a comunidade acadêmica, bem como narra a solução através do detalhamento dos requisitos de software e diagramas UML.

## 1.6 Metodologia

Neste capítulo é apresentada a metodologia utilizada para a concepção deste trabalho. Inicialmente foi elaborado um *brainstorm* para identificar contextos de estudos relevantes ao projeto, no que abrange o presente objetivo geral. Os seguintes termos foram pontuados como significantes:

- a. Modelos de Dados: Conceitual e Lógico;
- b. Banco de Dados: Tipos de Dados e Interface SGBD;
- c. Linguagem SQL: DDL e DML;
- d. Java: JGraphx e JAXB;
- e. XML, e;
- f. ConceptER 2.0: Estrutura, Funções e Limitações.

Para fundamentação teórica foi realizada uma extensa pesquisa bibliográfica sobre os conceitos e técnicas de modelagem, bem como as regras de conversão entre os modelos conceitual e lógico. Foram identificadas cinco ferramentas como tecnologias correlacionadas para com objetivo de aplicar um *benchmarking* e obter informações relevantes para a definição de requisitos. Foram definidos requisitos funcionais e não funcionais a partir de uma análise das necessidades essenciais para garantir o comportamento em alto nível da ferramenta proposta

neste trabalho. Para a elaboração da estrutura do projeto de software, foi definido a utilização da UML (Linguagem de Modelagem Unificada). Na implementação, a ferramenta será desenvolvida usando as mesmas tecnologias de apoio da versão ConceptER 2.0, de forma que garanta a uma evolução compatível e adaptações futuras. A metodologia para planejamento e gestão do desenvolvimento escolhida foi a SCRUM, por se tratar de uma metodologia ágil eficaz para projetos de software, à qual seus papéis e ciclos foram adaptados para interação individual. A validação da ferramenta se dará através da inspeção de software utilizando um *checklist* predefinido, consistindo na verificação de artefatos em prol da detecção de falhas.

## 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados temas que convergem para delimitação do escopo deste trabalho. Diante das temáticas relacionadas a Banco de Dados, são tratados temas como: Modelagem e Modelos de Dados, Diagramas Entidade-Relacionamento, Regras de Mapeamento, Regras de Conversão entre os modelos conceitual e lógico, além da Conversão de dados relacionais para XML.

### 2.1 Modelagem de Dados

Modelagem de dados “é o estudo das informações existentes em um contexto sob observação para a construção de um modelo de representação e entendimento de tal contexto” (MACHADO, 2014).

A modelagem de dados representa graficamente as regras de negócio necessárias para explicar as características de comportamento de um software.

De acordo com Machado (2014), um Banco de Dados “representa algum aspecto do mundo real, o qual é chamado de minimundo; qualquer alteração efetuada no minimundo é automaticamente refletida no banco de dados”.

A observação do minimundo bem como sua breve, mas exata, descrição de como as coisas funcionam de fato, ajudam a formular a compreensão das regras de negócio que servirão como base para a fase de modelagem. Por isso, qualquer alteração no minimundo pode invalidar uma modelagem anteriormente realizada e até mesmo alteração física do banco de dados. Portanto faz-se necessário que seja bem definida pois normalmente não se espera mudanças drásticas em um projeto.

“Uma das principais características da abordagem de modelagem de banco de dados é que ela fornece níveis de abstração de dados que omitem do usuário final detalhes sobre o armazenamento dos dados” (MACHADO, 2014).

A hierarquia entre modelos de dados está dividida em três níveis, baseados na capacidade de abstração, são eles: conceitual, lógico e físico.

“O objetivo de um modelo de dados é ter certeza de que todos os objetos de dados existentes em determinado contexto e requeridos pela aplicação e pelo banco de dados estão completamente representados e com precisão” (MACHADO, 2014).



Normalmente o usuário é um ator representado no minimundo que fará o uso de alguma tecnologia que se comunicará com o banco de dados, e não há necessidade de ele conhecer aspectos técnicos sobre como os dados estão sendo armazenados e manipulados no servidor de banco de dados, a ele interessa somente as regras de negócio e a apresentação das informações.

Segundo Heuser (2008), “modelos de dados são usados para comunicação entre as pessoas da organização e até mesmo a comunicação entre programas”. Logo, nem sempre em um minimundo teremos um ator, pois em alguns casos o sistema de banco de dados poderá ter como finalidade apenas a troca de informações entre máquinas.

O modelo de dados tem como propósito fornecer diferentes tipos de visões para cada etapa da modelagem do banco de dados.

## **2.2 Modelo Conceitual**

O modelo conceitual “descreve a realidade do ambiente do problema, constituindo-se em uma visão global dos principais dados e seus relacionamentos (estruturas de informação), completamente independente dos aspectos de sua implementação tecnológica” (MACHADO, 2014).

Trata-se de uma descrição para o sistema em definição em coerência a razão de ser de sua existência, portanto deve englobar regras de ação e comportamento.

Apesar de sua macrodefinição, deve ter “a preocupação de captar e retratar toda a realidade de uma organização, processo de negócio, setor, repartição, departamento etc” (MACHADO, 2014).

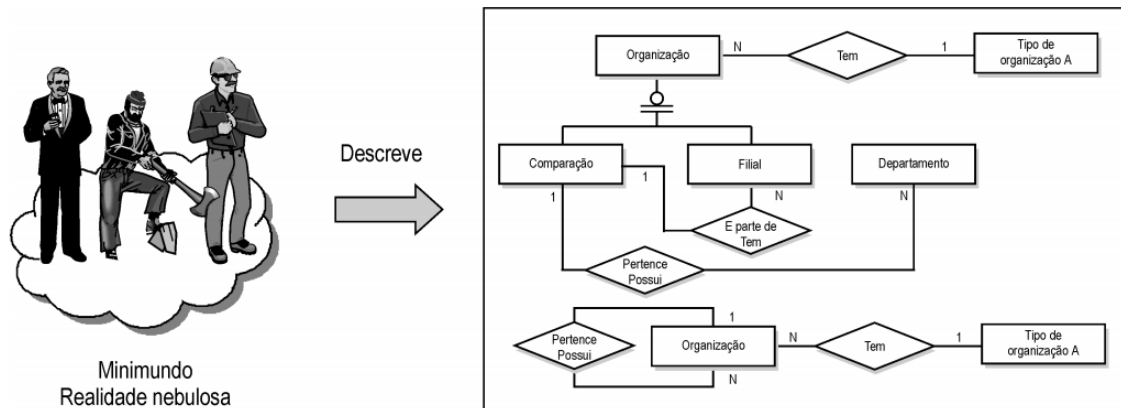
Este modelo é utilizado nas primeiras fases do projeto de interação, onde um conjunto de suposições abstraídas de um minimundo formarão as regras de negócio de um sistema.

Pode ser considerado como “o resultado de um modelo conceitual é um esquema gráfico que representa a realidade das informações existentes em determinado contexto de negócios, assim como as estruturas de dados em que estão organizadas essas informações” (MACHADO, 2014).

O esquema de um modelo conceitual ajuda a dar sentido aos processos de um minimundo, enriquecendo características de modelos complexos.

A seguir é apresentado, na figura 1, um esquema gráfico de um modelo conceitual baseado em um mini-mundo, sem a preocupação com formas de acesso e/ou estrutura de um SGBD (Sistema Gerenciador de Banco de Dados).

**Figura 1** – Esquema gráfico de um modelo conceitual a partir de um mini-mundo.



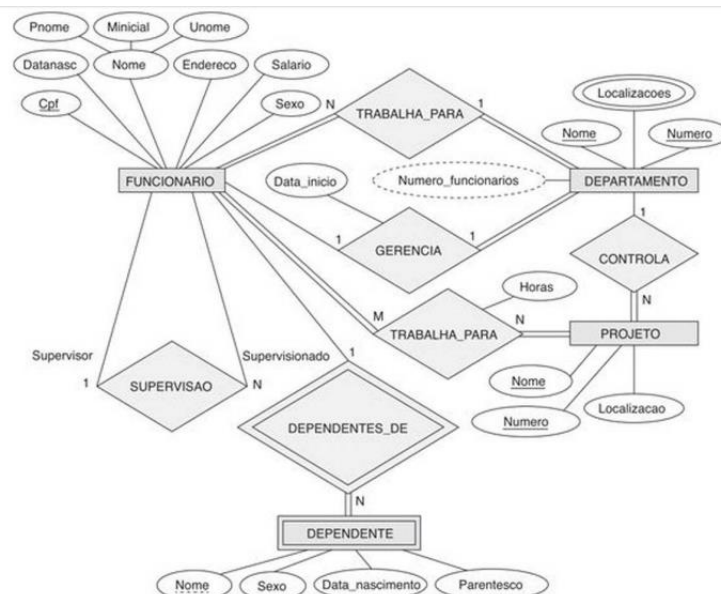
Fonte: Machado (2014).

### 2.2.1 Modelo Entidade Relacionamento (ER)

Um modelo Entidade-Relacionamento, “simplesmente declarado como modelo de ER, ele é conceitual e vê o mundo real como entidades e relacionamentos. Um componente básico do modelo é o diagrama de Entidade-Relacionamento, que visualmente representa objetos de dados” (MACHADO, 2014).

Representado na figura 2, este modelo descreverá os dados sob o domínio que engloba o minimundo em um nível conceitual. É utilizado como base de metadados estruturais para possível construção de um modelo lógico através das entidades de dados mestres.

**Figura 2** - Um diagrama de esquema ER para o banco de dados EMPRESA.



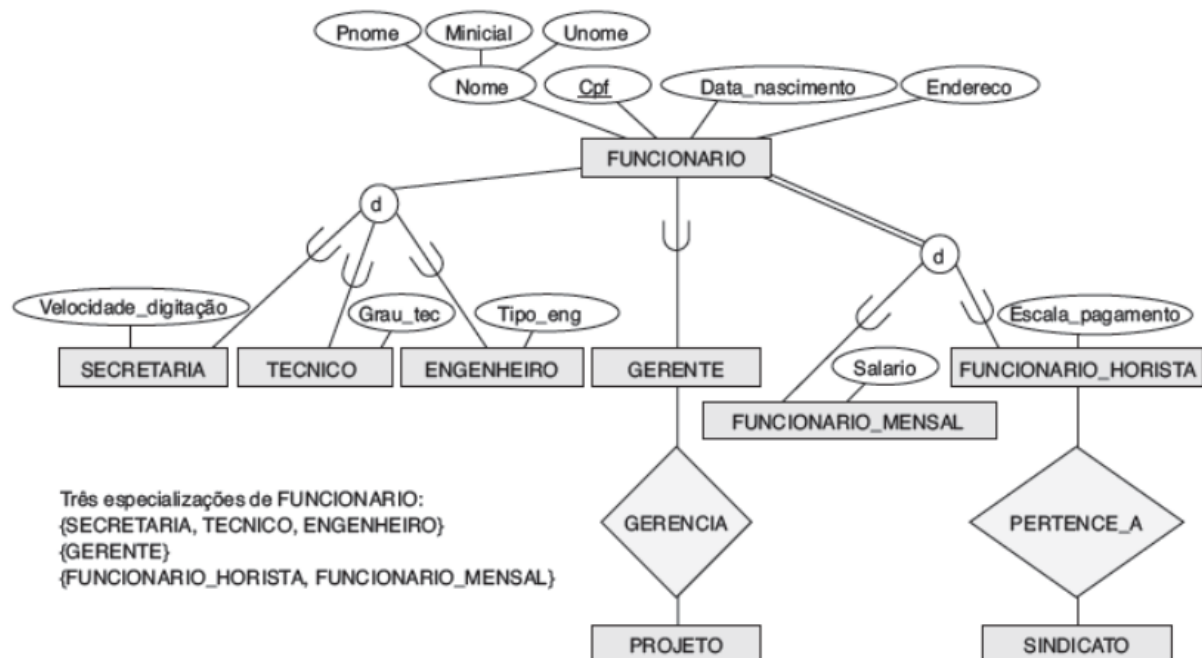
Fonte: Elmasri e Navathe (2011).

### 2.2.2 Modelo Entidade Relacionamento Estendido (EER)

“O modelo EER inclui todos os conceitos de modelagem do modelo ER (...). Além disso, inclui os conceitos de subclasse e superclasse e os conceitos relacionados de especialização e generalização” (ELMASRI e NAVATHE, 2011).

O modelo EER estende o modelo ER incorporando os conceitos semânticos a seguir: subclasses, superclasses, herança de atributos, especialização, generalização e categorias. A figura 3, ilustra a representação do diagrama EER para subclasses e especializações.

**Figura 3** - Notação do diagrama EER para representar subclasses e especializações.



Fonte: Elmasri e Navathe (2011).

### 2.3 Modelo Lógico

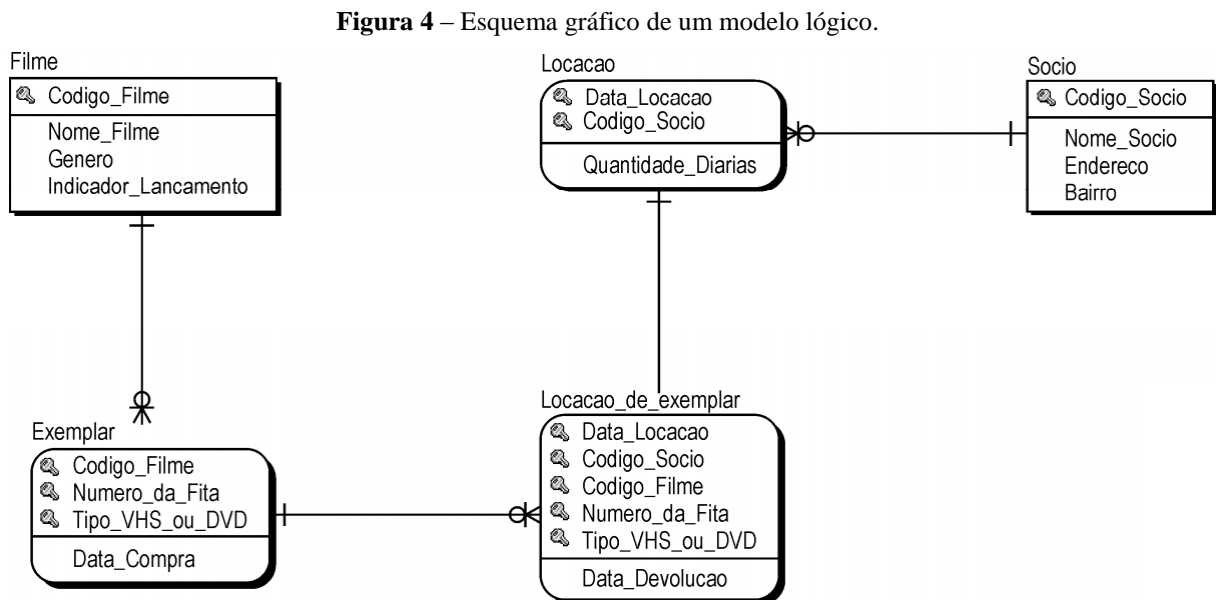
“Um modelo lógico é uma descrição de um banco de dados no nível de abstração visto pelo usuário do SGBD” (HEUSER, 2009).

As características de estrutura interna de um SGBD específico não são levadas em consideração. Normalmente deriva de um modelo conceitual existente no qual algumas regras para esta derivação são aplicadas.

“O modelo lógico descreve em formato as estruturas que estarão no banco de dados de acordo com as possibilidades permitidas pela sua abordagem, mas sem considerar, ainda, nenhuma característica específica de um SGDB” (MACHADO, 2014).

Esta possibilidade de não considerar características específicas de um SGDB, leva em consideração que alguns SGDBS possuem particularidades que diferem entre outros, logo evita um possível engessamento desta modelagem. Este modelo é responsável por definir as tabelas e seus relacionamentos sem incluir detalhes sobre seu armazenamento interno.

A figura 4 ilustra um esquema gráfico de um modelo lógico a partir de um mini-mundo baseado em uma locadora de filmes.



Fonte: Machado (2014).

## 2.4 Abordagem Relacional

A abordagem relacional trata sobre o modelo de dados relacional, que “foi introduzido inicialmente por Ted Codd, da IBM Research, em 1970, em um artigo clássico (Codd, 1970), que atraiu atenção imediata devido a sua simplicidade e base matemática” (ELMASRI E NAVATHE, 2011).

Esta base matemática é conceituada de relação matemática, que se constitui na teoria de conjuntos e logica de predicado de primeira ordem.

De acordo com Siebra (2010), o modelo relacional “possui a base mais formal entre os modelos de dados, entretanto é o mais simples e com estrutura de dados mais uniforme”.

Este modelo é agregado por uma coleção de tabelas que possuem nomes únicos que as identificam e diferenciam em relação às outras. Cada tabela possui uma ou mais colunas que são chamadas de campos de atributos que estão relacionados a um tipo de dado.

O modelo relacional apareceu devido às seguintes necessidades: aumentar a independência de dados nos sistemas gerenciadores de banco de dados; prover um conjunto de funções apoiadas em álgebra relacional para armazenamento e recuperação de dados; permitir processamento ad hoc. (TAKAI, ITALIANO & FERREIRA, 2005).

Logo, este modelo proporciona uma melhor flexibilidade para a solução de problemas na criação de banco de dados, na qual sua representatividade é adequada para um modelo implícito ao SGBD.

#### **2.4.1 Composição de um Banco de Dados Relacional**

“Um banco de dados relacional é composto de tabelas ou relações. A terminologia tabela é mais comum nos produtos comerciais e na prática. Já a terminologia relação foi utilizada na literatura original sobre a abordagem relacional” (HEUSER, 1998).

É adequado para implementar estruturas de dados relacionais organizadas e tenta prevenir através de regras de restrições de integridade que ocorra a incapacidade de representar ou até mesmo perder informações.

##### **2.4.1.1 Tabelas**

A Microsoft (2017), define tabelas como “objetos de banco de dados que contêm todos os dados em um banco de dados. Nas tabelas, os dados são organizados de maneira lógica em um formato de linha-e-coluna semelhante ao de uma planilha”.

Cada tabela possui um nome único e um conjunto de atributos identificados por nomes formando um domínio, no qual todos os valores de uma coluna possuem o mesmo tipo de dado.

“Na terminologia formal do modelo relacional, uma linha é chamada de tupla, um cabeçalho da coluna é chamado de atributo e a tabela é chamada de relação” (ELMASRI E NAVATHE, 2011)

Já a ordem das tuplas e também dos campos de atributos não possuem relevância, uma vez que estes registros podem ser identificados por meio da chave primária. Todo atributo possuirá um valor atômico, ou seja, um valor indivisível.

Uma linha é formada por campos que armazenam os valores dos atributos e formam um registro exclusivo, é academicamente conhecida como Tupla. Já uma coluna representa o conjunto de campos da tupla que possuem o mesmo nome de atributo.

Todo atributo em uma relação terá um nome único na relação e todas as tuplas devem ser únicas, mantendo uma estrutura heterogênea, formando um conjunto.

De acordo com as características apresentadas nesta seção, a figura 5 ilustra a composição de uma tabela genérica.

**Figura 5 - Tabela.**

CódigoEmp	Nome	CodigoDepto	CategFuncional
E5	Souza	Souza	C5
E3	Santos	Santos	C5
E2	Silva	Silva	C2
E1	Soares	Soares	-

Linha (Tupla) Coluna (Atributo) Nome do Campo (Nome do Atributo)  
Valor de Campo (Valor de Atributo)

Fonte: Adaptado de Helser (1998).

#### 2.4.1.2 Chaves

“O conceito básico para estabelecer relações entre linhas de tabelas de um banco de dados relacional é o da chave. [...] há ao menos três tipos de chaves a considerar: a chave primária, a chave alternativa, e a chave estrangeira” (HEUSER, 1998).

Chaves primárias servem para identificar registros exclusivos, ou seja, apresentar sem ambiguidade uma determinada tupla de uma relação.

De acordo com Machado (2014), chave estrangeira “é uma referência de um elemento de uma tabela a um elemento de outra tabela, uma relação entre as tabelas, uma ligação lógica entre elas”.

A chave estrangeira tem seus valores inseridos em uma chave primária de forma que crie uma restrição a uma tabela existente garantindo integridade nos dados referenciados entre uma relação de tabelas.

A figura 6 ilustra uma tabela de registro de funcionários, em que a chave primária é representada pela coluna identificada como “NumReg”, que armazena os números de registro de funcionários. Esta mesma tabela possui também duas colunas que representam chaves estrangeiras, “CdCargo” e “CdDepartamento” que armazena os códigos de cargo e códigos de departamento respectivamente, que por sua vez também são chaves primarias nas suas tabelas correspondentes.

**Figura 6** – Tabela Funcionário.

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/8/2003	M	C3	D5
104	Carlos Pereira	2/3/2004	M	C4	D6
134	Jose Alves	23/5/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
123	Pedro Sergio Doto	29/6/2003	M	Nulo	D3
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/2/2000	M	C2	D4

Fonte: Machado (2014).

#### 2.4.1.3 Domínios

“Quando uma tabela do banco de dados é definida, para cada coluna da tabela, deve ser especificado um conjunto de valores (...). Este conjunto de valores é chamado de domínio da coluna ou domínio do campo” (HEUSER, 1998).

O domínio consiste no tipo de dado que descreve os valores possíveis para a composição de uma coluna, ou seja, cada coluna de uma tabela possui um nome identificador que referencia um domínio da tabela, representando um conjunto de valores atômicos associados a um formato ou tipo de dado. Essa associação defere que tais propriedades não poderão sofrer decomposição em sua estrutura. A restrição de domínio tem como função determinar a regra de valores possíveis para cada conjunto.

#### 2.4.1.4 Valores NULL

“Um conceito importante é o dos valores NULL, que são usados para representar os valores de atributos que podem ser desconhecidos ou não se aplicam a uma tupla. Um valor especial, chamado NULL, é usado nesses casos” (ELMASRI E NAVATHE, 2011).

A existência de um campo vazio declara que ele não possui valor de seu domínio. Por regra, chaves primárias devem ser definidas como *not null*, para definir que a existência de dados seja obrigatória para coluna respectiva. Pois um identificador nulo, em uma chave primária, representa uma ocorrência não distinguível.

#### 2.4.1.5 Restrições de Integridade

Heuser (1998), afirma que “uma restrição de integridade é uma regra de consistência de dados que é garantida pelo próprio SGBD”.

Restrições de integridade são regras primordiais para especificação de um banco de dados relacional. Visão garantir a exata consistência dos dados em relação a realidade modelada a partir de um minimundo.

Algumas restrições de Integridade são classificadas por Heuser (1998), como:

- Integridade de domínio: Especifica que o valor de um campo deve obedecer a definição de valores admitidos para a coluna (o domínio da coluna);
- Integridade de vazio: Especifica se os campos de uma coluna podem ou não ser vazios (se a coluna é obrigatória ou opcional);
- Integridade de chave: Define que os valores da chave primária e alternativa devem ser únicos;
- Integridade referencial: Define que os valores dos campos que aparecem em uma chave estrangeira devem aparecer na chave primária da tabela referenciada.

Elmasri e Navathe (2011), afirmam que em “restrição de integridade de entidade nenhum valor pode ser de chave primaria pode ser NULL”.

Os valores de chaves-primárias nunca poderão ser nulos ou se repetir no seu respectivo campo da tabela. Este valor representará o identificador único para uma tupla de uma relação. Logo, em uma hipótese deste valor ser NULL em mais de um registro, não seria possível identificar a tupla correta e diferencia-la em uma determinada relação.

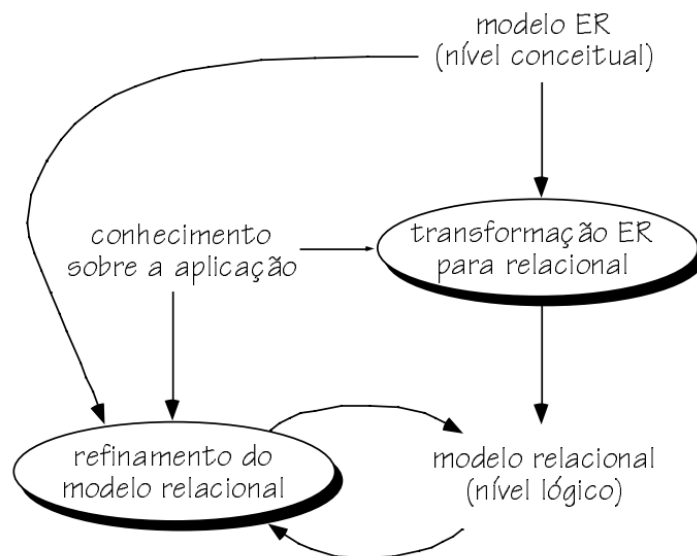


## 2.5 Conversão do Modelos ER Conceitual para Modelo Relacional

O modelo ER é baseado na modelagem de dados de maneira independente à um SGDB, onde é possível representar um modelo conceitual. Já o modelo relacional é direcionado à modelagem no nível de SGBD relacional, com a finalidade de representar um modelo lógico.

A figura 7 representa um esquema geral para conversão do modelo conceitual, intitulado como modelo ER, para o modelo lógico, intitulado como modelo relacional. Os procedimentos para esta conversão estão descritos nas subseções desta seção, e são eles: Implementação de Entidades; Implementação de Relacionamentos e; Implementação de Generalização e Especialização.

**Figura 7** – Visão geral do projeto lógico.

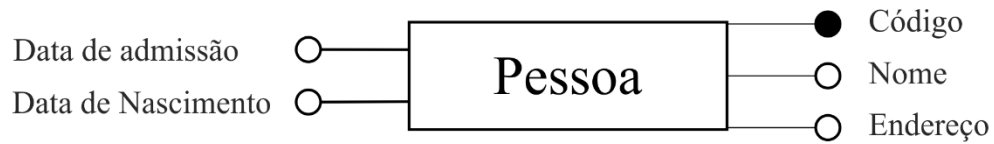


Fonte: Heuser (1998).

### 2.5.1 Implementação de Entidades

“Para implementação inicial de entidades, cada entidade é traduzida para uma tabela, cada atributo de entidade define uma coluna desta tabela e os atributos identificadores da entidade correspondem às colunas que compõem a chave primária da tabela” (HEUSER, 1998).

A figura 8 exemplifica a transformação de uma entidade em tabela, apresentando o Diagrama Entidade Relacionamento (DER) e o esquema relacional respectivo.

**Figura 8.** Transformação de entidade em tabela.

Esquema relacional Correspondente:

Pessoa (CódigoPess, Nome, Endereço, DadaNasc, DataAdm)

**Fonte:** Adaptado de Heuser (1998).

## 2.5.2 Implementação de Relacionamentos

“Um dos principais pontos a observar em um esquema relacional, ao contrário de um esquema ER, é que os tipos de relacionamento não são representados explicitamente” (ELMASRI E NAVATHE, 2011).

A regra específica que deve ser usada na tradução de um relacionamento é determinada pelas cardinalidades mínima e máxima das entidades envolvidas nos relacionamentos. A figura 9 apresenta opções relativas ao mapeamento de especialização e generalização.

**Figura 9.** Opções para mapeamento de especialização ou generalização.

MODELO ER	MODELO RELACIONAL
Tipo de entidade	Relação de entidade
Tipo de relacionamento 1:1 ou 1:N	Chave estrangeira ou relação de relacionamento
Tipo de relacionamento M:N	Relação de relacionamento e duas chaves estrangeiras
Tipo de relacionamento n-ário	Relação de relacionamento e n chaves estrangeiras
Atributo simples	Atributo
Atributo composto	Conjunto de atributos componetes simples
Atributo multivalorado	Relação e chave estrangeira
Conjunto de valores	Domínio
Atributo chave	Chave primária ou secundária

**Fonte:** Adaptado de Elsmari e Navathe (2011).

### 2.5.3 Implementação de Generalização e Especialização

“Para a implementação de generalização/especialização na abordagem relacional, há duas alternativas a considerar: uso de uma tabela para cada entidade e uso de uma única tabela para toda hierarquia de generalização/especialização” (HEUSER, 1998).

Neste trabalho será utilizada a alternativa que trata sobre o uso de uma tabela para cada entidade. Na figura 10 é possível observar a correspondência entre os modelos Entidade-Relacionamento e o modelo Relacional.

**Figura 10.** Correspondência entre os modelos ER e Relacional.

**(a) FUNCIONARIO**

Cpf	Pnome	Minicial	Unome	Data_nascimento	Endereço	Tipo emprego
<b>m</b>						
<b>SECRETARIA</b>			<b>TÉCNICO</b>		<b>ENGENHEIRO</b>	
Cpf Velocidade digitacao			Cpf Tnota		Cpf Tipo_eng	

**(b) CARRO**

Id_veiculo	Placa	Preco	Vefocidade_max	Numero_passageiros
<b>CAMINHAO</b>				
Id_veiculo	Placa	Preco	Numero_eixos	Capacidade_peso

**(c) FUNCIONARIO**

Cpf	Pnome	Minicial	Unome	Data_nascimento	Endereço	Tipo.emplo	Velocidade.digitacao	Graujec	Tipo_eng
<b>PECA</b>									
Peca_nr	Descricao	Tipojabr	Num.desenho	Datafabricacao	Numjote	Tipo_compr	Nomejornecedor	Preco	

**Fonte:** Elsmari e Navathe (2011).

A especialização e generalização, em geral, representam objetos do mundo real que tem atributos heterogêneos que possam ser categorizados e hierarquicamente representados com suas dependências entre entidades da mesma categoria.

### 3 TRABALHOS RELACIONADOS

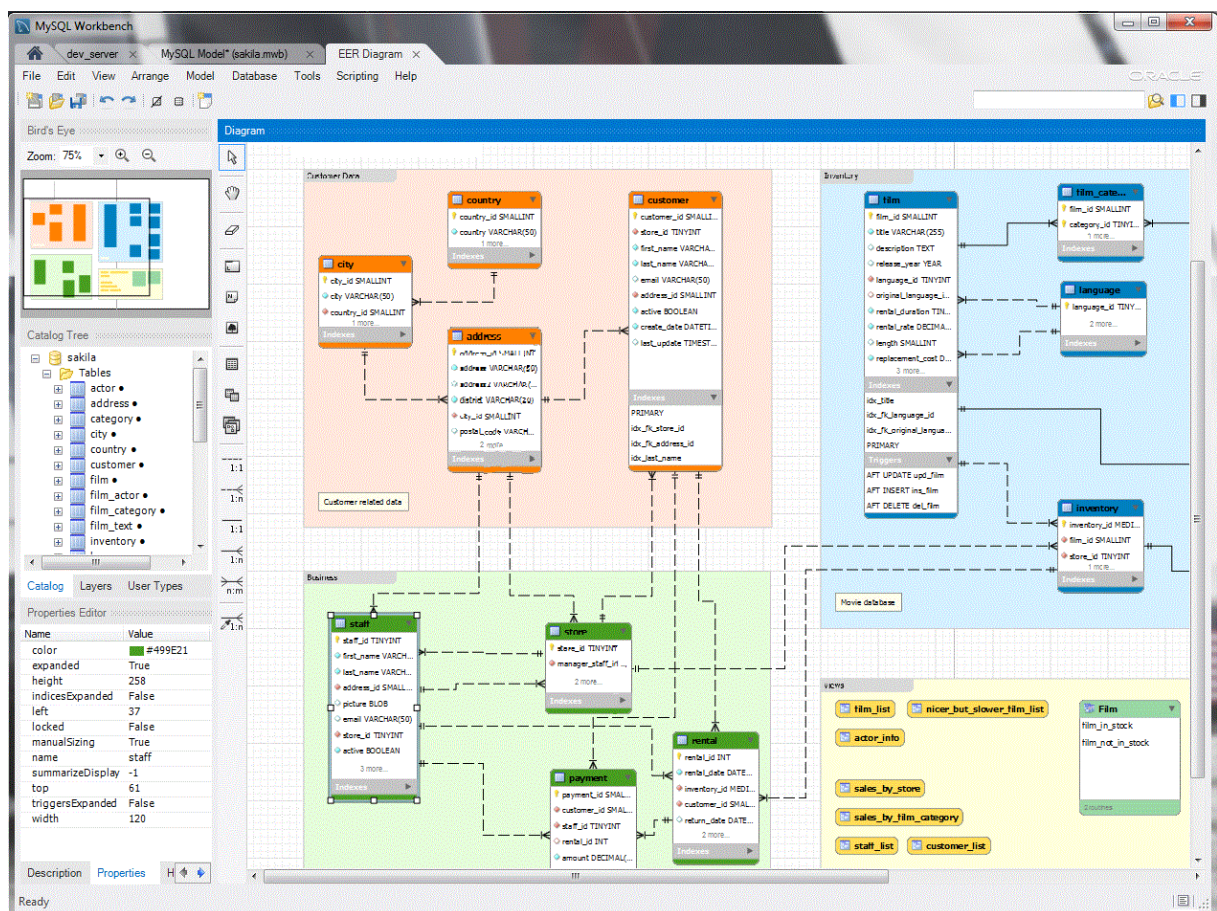
Neste capítulo serão apresentadas as ferramentas testadas, para avaliação das principais funcionalidades relacionadas com os objetivos deste projeto, assim como artigos nos quais os pesquisadores criaram propostas de conversões entre os modelos utilizados.

#### 3.1 MySQL Workbench

Atualmente o MySQL Workbench é uma ferramenta oficial da Empresa Oracle, sediada no Estados Unidos da América, que oferece um ambiente visual que permite trabalhar com servidores e bancos de dados MySQL. Suas principais funcionalidades são: modelagem de dados, desenvolvimento sql, administração de servidores e migração de dados.

A figura 11 apresenta a tela de edição de diagramas EER na ferramenta Visual Database Design.

Figura 11. Visual Database Design.



Fonte: Mysql (2018).

A Oracle atualmente oferece duas edições desta ferramenta, são elas: a *Community Edition* e a *Commercial Edition*, sendo uma gratuita e a outra paga, respectivamente.

Esta ferramenta não possui suporte e integração para importar e/ou desenvolver Modelos Conceituais.

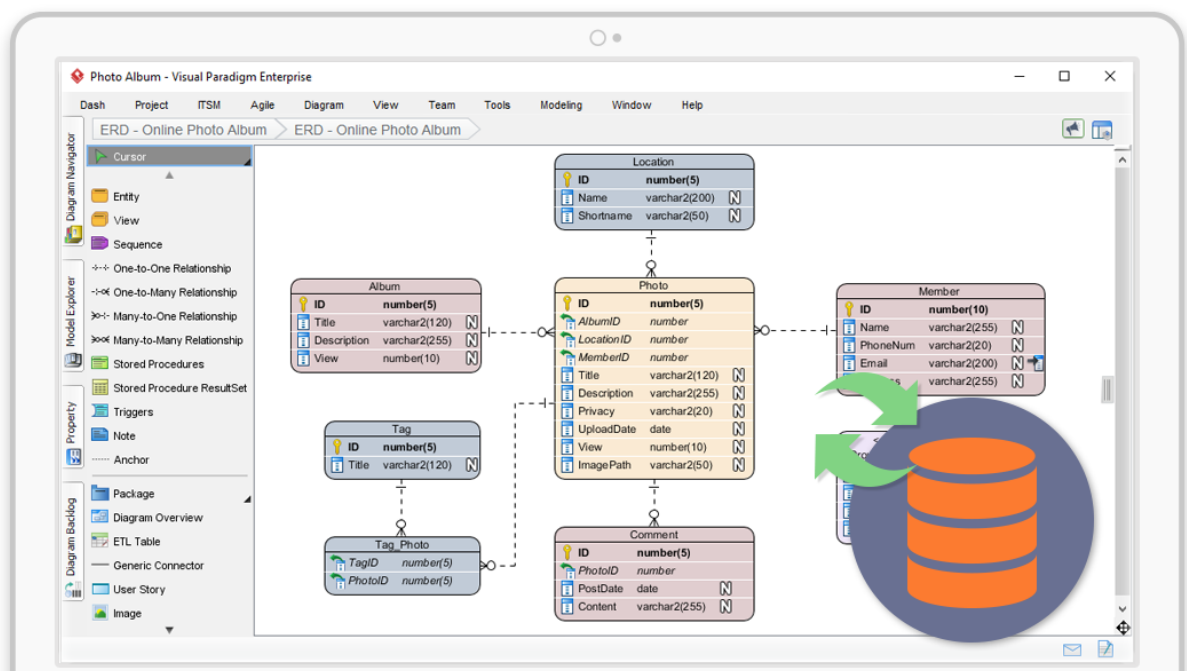
### 3.2 Visual Paradigm

O Visual Paradigm é uma ferramenta ofertada pela empresa Visual Paradigm International, sediada na China, que contempla diversos módulos de modelagem e gestão direcionado para o a análise e desenvolvimento de sistemas, projetado para suportar equipes de desenvolvimento ágil. Sua premissa é oferecer recursos em uma solução “one-stop-shop”, para: gerenciamento de projetos, arquitetura corporativa, desenvolvimento de software e colaboração em equipe.

Entre os módulos disponíveis, é destacado o módulo Database Design, que oferece o recurso para criação de Diagramas Entidade e Relacionamento, que tem como objetivo fornecer representação gráfica de um Banco de Dados Relacional.

A tela de apresentação de um modelo Relacional que pode ser utilizada no software Visual Paradigm, é apresentada a seguir na figura 12.

**Figura 12.** ERD & Database Engineering.



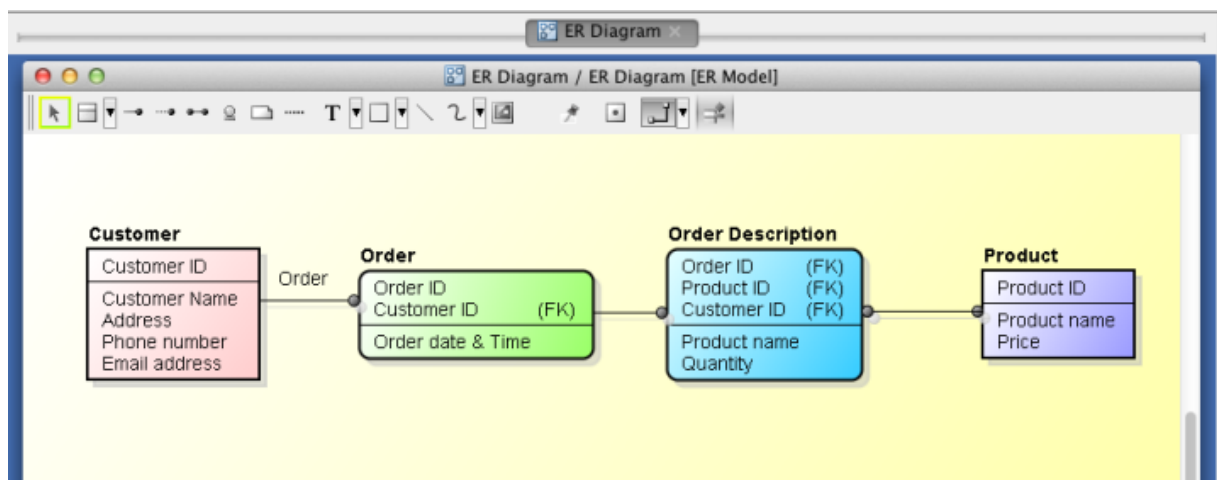
**Fonte:** Visual Paradigm (2018).

Em seu site, o Visual Paradigm destaca a possibilidade de criar modelagens de Banco de Dados como Modelo Conceitual, Modelo Lógico e Modelo Físico. Porém, na prática, o software oferece elementos visuais que se adequam aos três Modelos destacados anteriormente, mas não é possível realizar os relacionamentos do Modelo Conceitual de forma satisfatória, além de não possuir recursos de automação dos atributos para convertê-lo em Modelo Lógico.

### 3.3 Astah Professional

O Astah é uma ferramenta ofertada pela empresa Change Vision, sediada no Japão, que oferece recursos integrados para UML, Mind Mapping, Flowchart e ERD. A figura 13 refere-se interface para edição de Diagramas ER desta ferramenta.

**Figura 13.** ER (Entity-Relationship) diagrams.



Fonte: Astah Blog (2018).

O Astah Professional permite a criação de Diagramas Entidade Relacionamento em duas notações, IDEF1X ou IE. Permite a criação de Modelo Lógico e Modelo Físico, bem como a conversão e reversão entre esses dois modelos.

### 3.4 Navicat Data Modeler

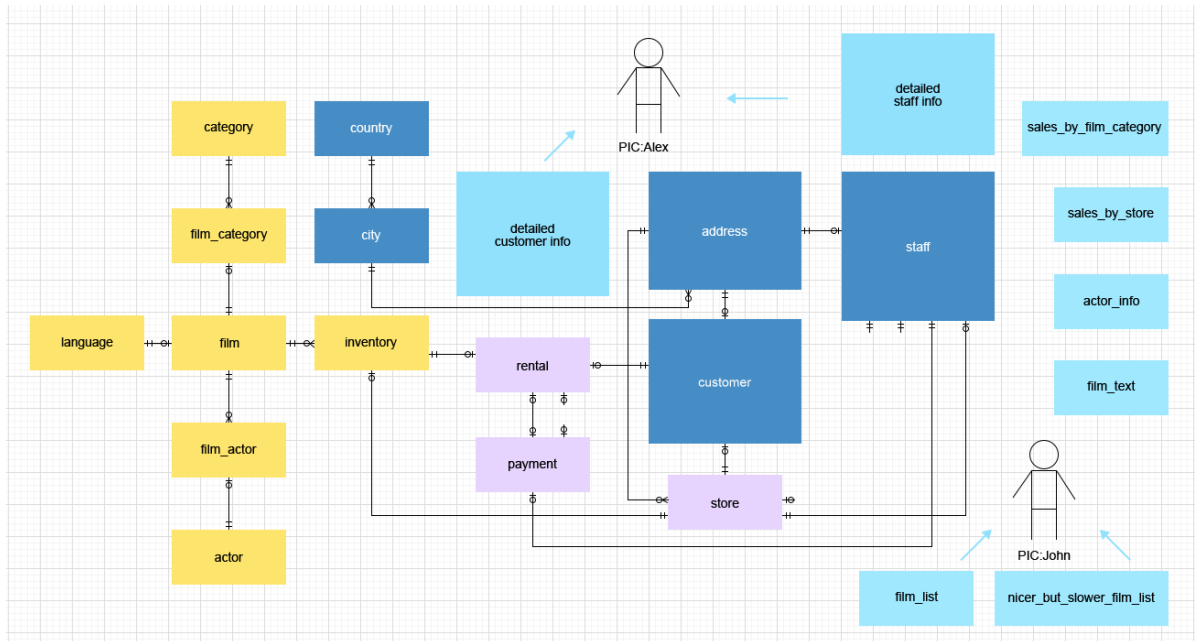
O Navicat Data Modeler é uma ferramenta ofertada pela Empresa PremiumSoft™ CyberTech Ltd, sediada na China, que permite a criação de Diagramas Entidade



Relacionamento. Possui recursos para criação de Modelo Conceitual, Modelo Lógico e Modelo Físico.

O Modelo Conceitual oferecido não contém a devida notação gráfica característica do modelo proposto, sua interface é apresentada na figura 14.

**Figura 14** - Navicat Data Modeler 2.1 – Exemplo de Modelo Conceitual



Fonte: Navicat (2018).

### 3.5 ConceptER 2.0

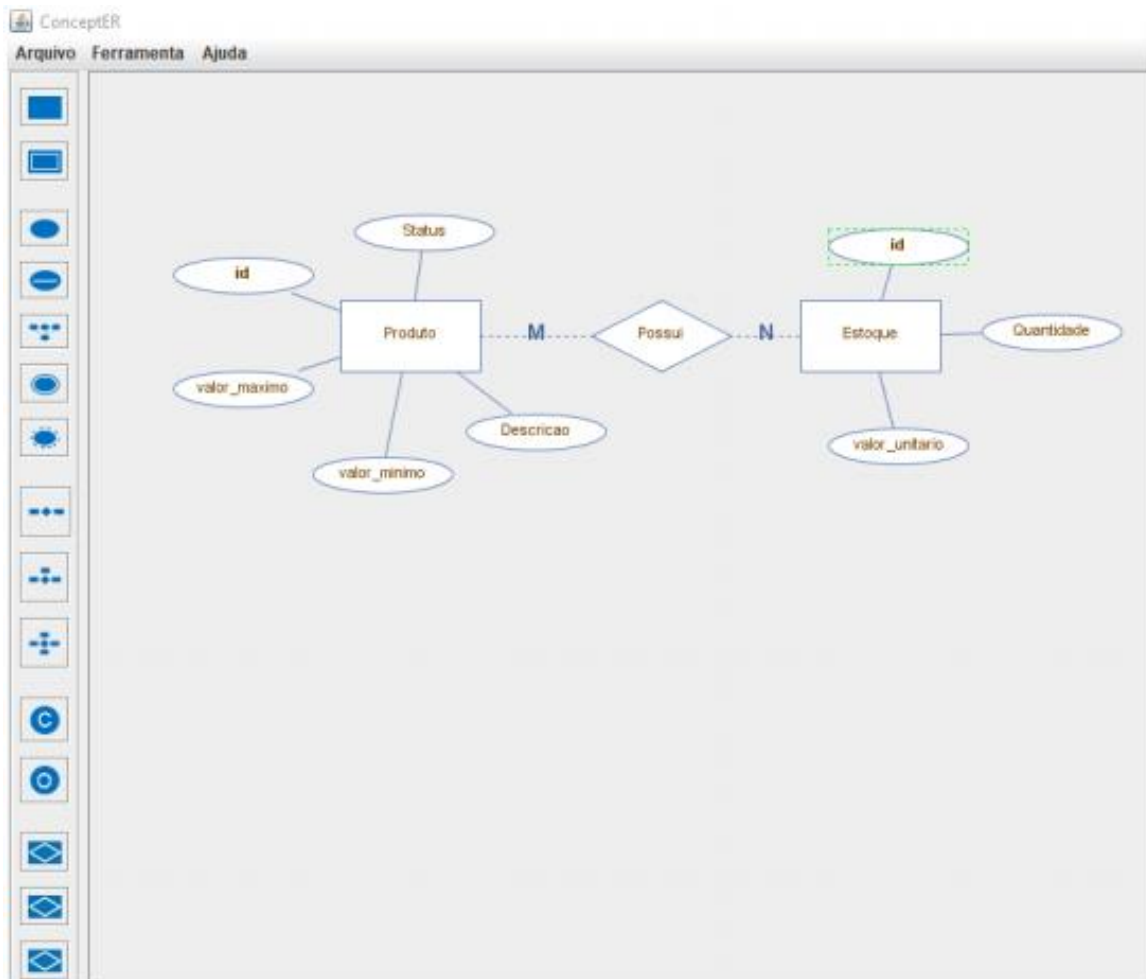
É uma ferramenta para criação de modelos conceituais, tendo como suporte componentes gráficos baseados no diagrama de entidade-relacionamento.

A principal função da ferramenta desenvolvida é tratar de forma teórica a modelagem conceitual, e sustentar o projetista nas melhores práticas implementadas no modelo. (MENDONÇA, 2015)

Atualmente o ConceptER está na versão 2.0, contemplando funcionalidades para modelagem de banco de dados em nível conceitual, através da criação de um Diagrama Entidade-Relacionamento. Ao realizar uma modelagem, usando o ConceptER 2.0, o usuário pode exportar o resultado do seu trabalho para o formato de imagem PNG e/ou salvar em um arquivo de extensão “.er0”. O arquivo salvo com a extensão “.er0” não possui criptografia e tem como base a formatação e notação de um arquivo XML 1.0 com encoding “UTF-8”, que em razão disto, pode ser aberto pela maioria dos softwares compatível com a extensão XML.

A tela principal do ConceptER é apresentada a seguir na figura 15, contemplando todos seus componentes disponíveis para modelagem de um diagrama Entidade-Relacionamento.

**Figura 15** -Tela principal da ferramenta ConceptER.



Fonte: Nascimento (2017).

### 3.6 Comparação entre os Trabalhos Relacionados

Esta seção tem como propósito apresentar a comparação entre os trabalhos relacionados, que está devidamente organizado no quadro 1. Os quadros são assinalados com “S” para sim, caso a ferramenta possua atributos relacionados aos itens encontrados na primeira coluna do quadro, e “N” para não, caso não possua. A primeira coluna apresenta itens que devem ser encontrados na versão 3.0 do ConceptER após a conclusão deste projeto que resultará na evolução da ferramenta.



**Quadro 1** – Comparação entre os Trabalhos Relacionados.

	<b>MYSQL WORKBENCH</b>	<b>VISUAL PARADIGM</b>	<b>ASTAH PROFESSIONAL</b>	<b>NAVICAT DATA MODELER</b>	<b>CONCEPTER 2.0</b>
<b>MODELO CONCEITUAL</b>	N	S	N	S	S
<b>MODELO LÓGICO</b>	S	S	S	S	N
<b>CONVERSÃO ENTRE MODELOS</b>	N	N	N	N	N
<b>WINDOWS</b>	S	S	S	S	S
<b>LINUX</b>	S	S	S	S	S
<b>MAC</b>	S	S	S	S	S
<b>OPEN SOURCE</b>	S	N	N	N	S
<b>SOFTWARE LIVRE</b>	N	N	N	N	S

Fonte: Elaboração do autor (2018).

## 4 TECNOLOGIAS

Neste capítulo serão apresentadas as tecnologias utilizadas para desenvolver a interface gráfica de modelagem lógica bem como a evolução da ferramenta ConceptER 3.0

### 4.1 Linguagem Java

Java é uma linguagem de programação orientada a objetos, proposta para ser independente de plataforma através do uso de máquina virtual, neste caso a Java Virtual Machine (JVM).

Este conceito de tecnologia permitirá que a ferramenta proposta nesse trabalho, possa ser executada em ambientes computacionais diferentes que suportem a JVM.

### 4.2 Swing

O Swing é uma API (*Application Programming Interface*) que oferece diversos componentes padrões de GUI (*Graphical User Interface*) que permitem uma combinação para construção da interface gráfica destinados ao Java.

O pacote AWT (Abstract Windows Toolkit) contém os primeiros componentes para interface gráfica destinados ao Java. Trata-se de uma biblioteca de baixo-nível que depende de código nativo da plataforma que traz alguns problemas de compatibilidade no quesito de apresentação da interface gráfica dependendo da plataforma utilizada.

O pacote Swing é o resultado da evolução do pacote AWT, possuindo a maioria dos mesmos componentes porem com acabamento melhorado, além de possuir diversos controles extras que resolvem o problema de compatibilidade de apresentação da interface gráfica em plataformas heterogêneas. Diferente do AWT, realiza a renderização dos elementos sem precisar delegar este processo para o sistema operacional.

### 4.3 JGraphX para Java Swing

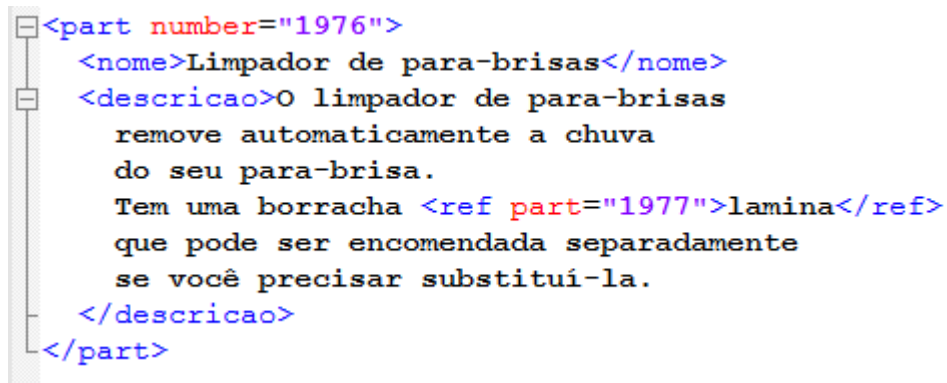
JGraphX é uma biblioteca de diagramas Java Swing que fornece funcionalidades para interação com gráficos matemáticos. Apesar do nome da biblioteca ser JGraphx, seus pacotes

são nomeados com mxGraph. Fornece funcionalidades necessárias para desenho, interação e associação de diagramas.

#### 4.4 XML

O *Extensible Markup Language* (XML) é um formato de arquivo computacional que permite estruturar informações baseadas em texto. Na prática, oferece diversos benefícios pelo detalhamento de sua marcação e também pela sua legibilidade devido ao fato de ser auto descritiva. Um exemplo de marcação nos padrões XML é demonstrado na figura 16.

**Figura 16** - XML 1.0 – Exemplo de XML.



```

<part number="1976">
  <nome>Limpador de para-brisas</nome>
  <descricao>O limpador de para-brisas
    remove automaticamente a chuva
    do seu para-brisa.
    Tem uma borracha <ref part="1977">lamina</ref>
    que pode ser encomendada separadamente
    se você precisar substituí-la.
  </descricao>
</part>
  
```

Fonte: Adaptado de W3C (2018).

#### 4.5 JAXB

O JAXB (*Java Architecture for XML Binding*) “fornece uma maneira rápida e conveniente de vincular esquemas XML e representações Java, facilitando para os desenvolvedores Java incorporar dados XML e funções de processamento em aplicativos Java” (ORACLE, 2018).

Uma implementação que utiliza o JAXB, contempla os seguintes componentes de arquitetura: compilador de esquemas, gerador de esquemas e; vínculo para estrutura de tempo de execução. O acesso, manipulação e validação do conteúdo XML se dá através das operações de *marshalling* (escrita) e *unmarshalling* (leitura).

## **5 IMPLEMENTAÇÃO**

Neste capítulo serão apresentados os tópicos definidos para a execução do projeto que fazem parte da estratégia utilizada no desenvolvimento do Módulo Gerador de Modelos Relacionais, bem como: as funcionalidades, requisitos funcionais e não funcionais, diagramas e descrições de casos de uso, diagrama de classes, diagrama de pacotes, diagramas de sequência e um breve detalhamento da ferramenta.

### **5.1 Funcionalidades**

Este trabalho tem como propósito evoluir a ferramenta ConceptER para a versão 3.0 através da integração de novas funcionalidades. Primeiramente foi criada uma interface que permite a criação de modelos de banco de dados relacionais em nível lógico, bem como suas funcionalidades que serão abordadas no decorrer deste capítulo. Logo em seguida, foi criado um módulo, integrado a interface de modelagem relacional, que permite a conversão do modelo conceitual originário do ConceptER 2.0 para o modelo lógico padrão do ConceptER 3.0.

#### **5.1.1 Modelagem Conceitual**

A interface que permite a criação gráfica do modelo conceitual através do diagrama de entidade-relacionamento teve sua base de componentes mantida. Este módulo da ferramenta pode ser apreciado nos trabalhos anteriores a este, os mesmos estão com seus trabalhos acadêmicos e repositórios de software estão inclusos nas Referências Bibliográficas no final deste trabalho.

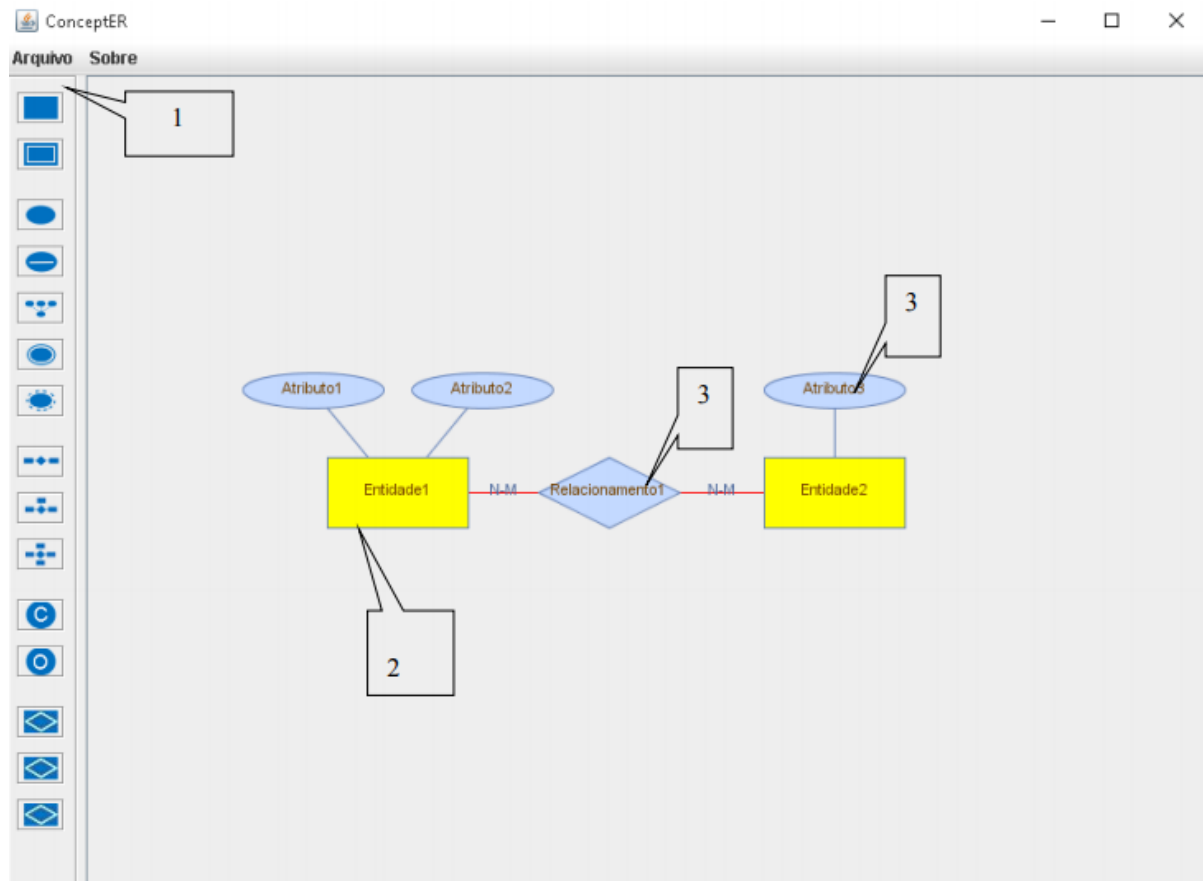
O primeiro trabalho, intitulado “FERRAMENTA PARA CRIAÇÃO DE MODELOS CONCEITUAIS DE BANCO DE DADOS”, elaborado pelo aluno Allan Magnum Melo de Mendonça, teve sua conclusão no ano de 2015 e deu origem a ferramenta ConceptER.

Inicialmente esta ferramenta teve como objetivo auxiliar na criação do Diagrama de Entidade-Relacionamento (DER), bem como validar o modelo gerado através da verificação de regras de validação.

A versão 7 do Java utilizada para implementação bem como a versão 2.5.0.1 da biblioteca de componentes gráficos do JGraph. A estrutura de armazenamento e recuperação dos modelos salvos compreende em um arquivo de metadados XML, contendo os dados das

entidades, atributos e relacionamentos. A figura 17 apresenta a tela principal da primeira versão do ConceptER.

**Figura 17** – ConceptER 1.0 – Tela Principal com Diagrama.



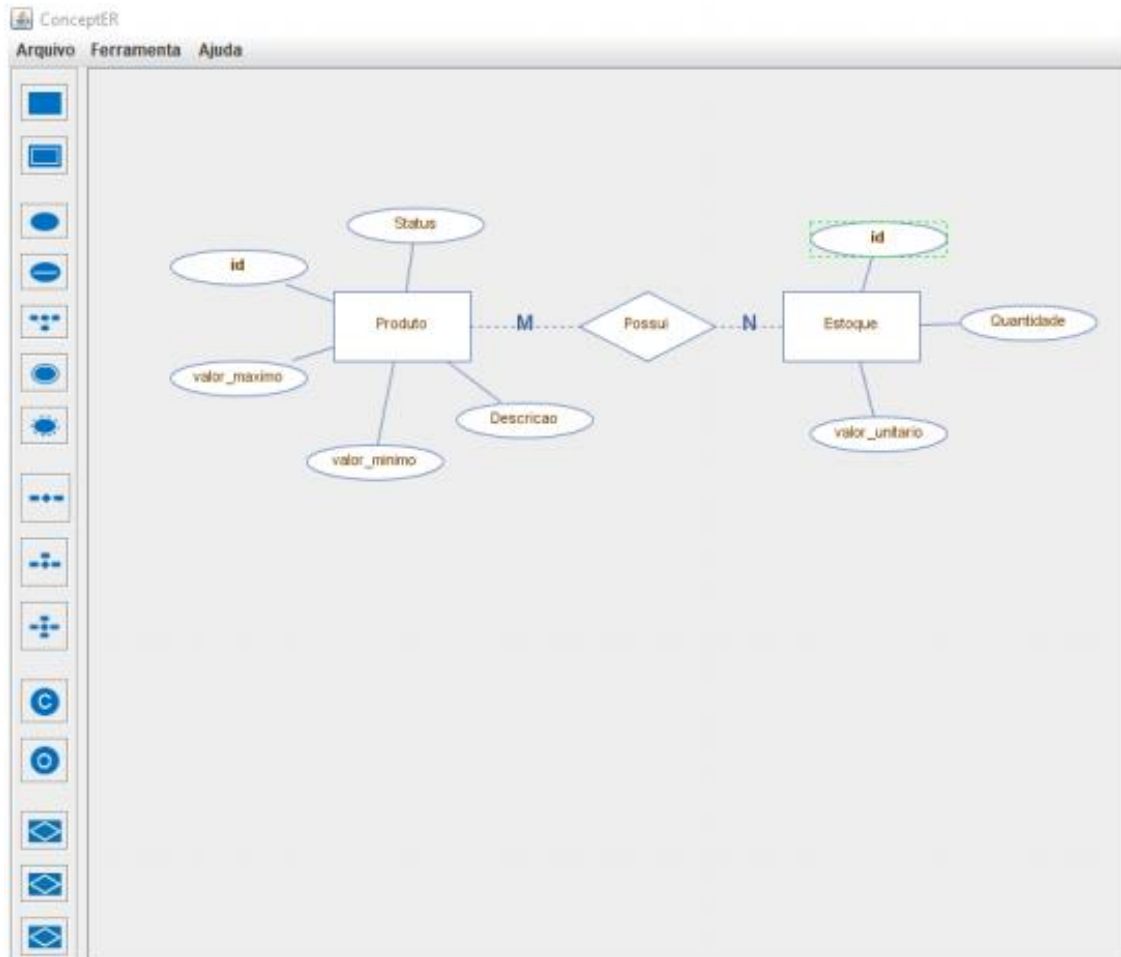
Fonte: Mendonça (2015).

O segundo trabalho, intitulado “CONCEPTER 2.0 - EVOLUÇÃO DA FERRAMENTA”, elaborado pela Aluna Fernanda Soares Nascimento, teve sua conclusão no ano de 2017 e deu continuidade ao trabalho da ferramenta.

Seu objetivo foi a evolução da ferramenta ConceptER, através de manutenções evolutivas e corretivas. Este trabalho realizou a correção de diversos bugs e aprimoramento de algumas funcionalidades. Entre estas realizações pode-se citar algumas, como: adição de um componente com funcionalidade de zoom, ação para permitir o salvamento ao sair da ferramenta, possibilidade de retomar a edição de um modelo criado anteriormente, correção na inserção do item de agregação, correção da representação de atributos chave, limitação de tela para criação de elementos de forma que todos estejam acessíveis, a versão do Java foi atualizada

para 8 e a versão do JGraph foi atualizada para 3.8. A figura 18 representa a tela principal da segunda versão do ConceptER.

**Figura 18** - Tela principal da ferramenta ConceptER 2.0.

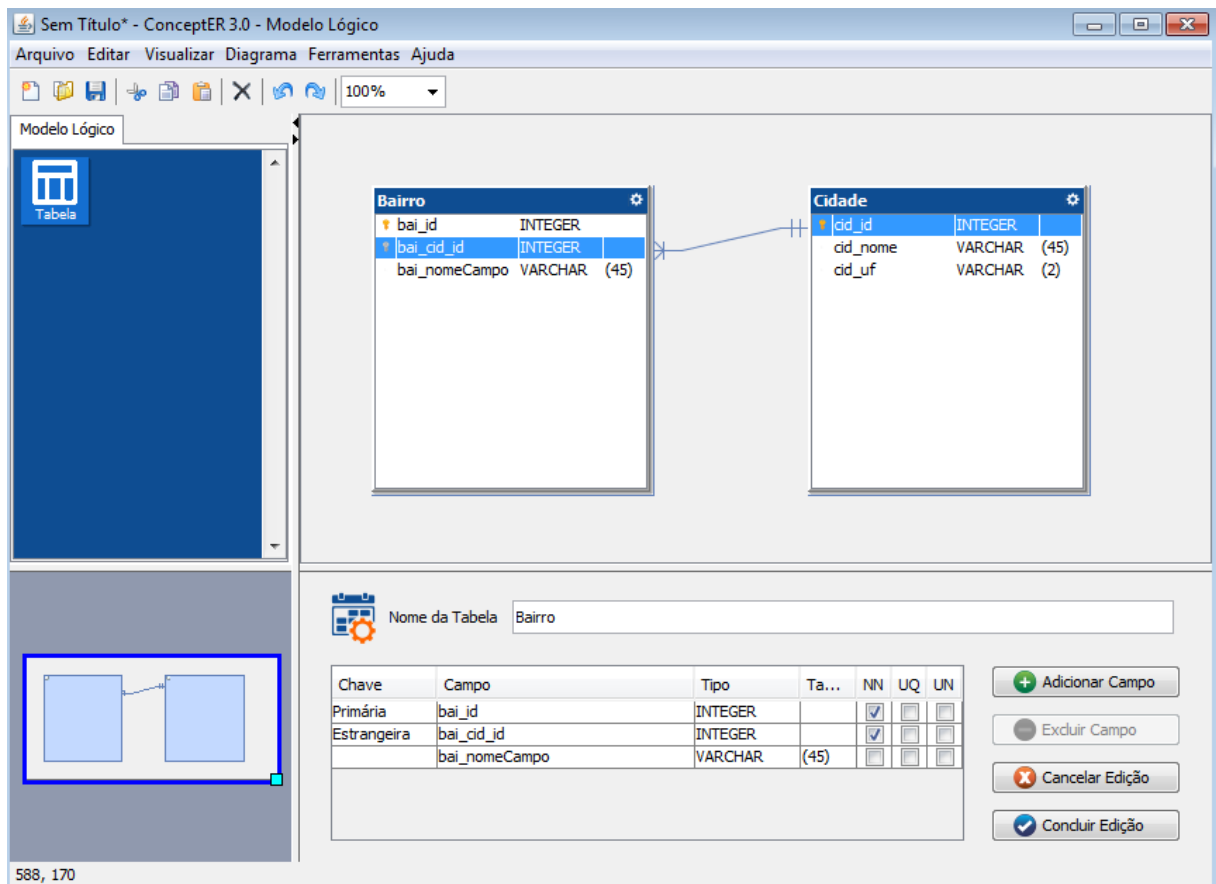


Fonte: Nascimento (2017).

### 5.1.2 Modelagem Lógica

O Módulo Gerador de Modelos Relacionais, integrado ao ConceptER, fornece uma nova interface com novos componentes que permite a elaboração de uma representação gráfica de um modelo relacional em nível lógico de banco de dados, como pode ser observado na figura 19. Esta figura também apresenta o conjunto de componentes que fazem parte da interface de edição, como: barra de menu, menu rápido, paleta de opções, paleta de *zoom*, componente gráfico de edição e painel de propriedades.

**Figura 19** -Tela principal do Módulo Gerador de Modelos Relacionais – ConceptER 3.0.

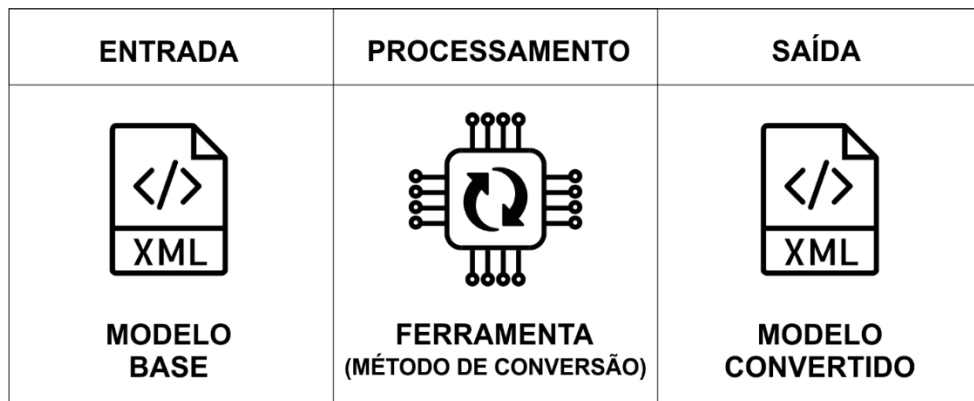


Fonte: Elaboração do autor (2019).

### 5.1.3 Conversão entre modelos

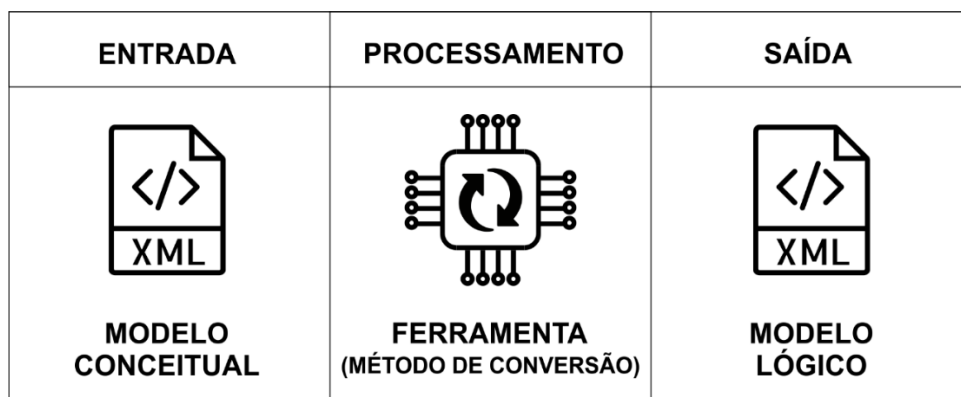
O processo de conversão entre modelos ocorrerá quando um usuário desejar transformar arquivo que contém um modelo conceitual em um arquivo contendo um modelo lógico. Para isso a entrada será sempre um arquivo com formatação XML, como modelo base, contendo as marcações predefinidas.

Após a escolha do arquivo XML de entrada, a ferramenta executará o método de conversão, que consistirá no mapeamento e transformação da estrutura do arquivo XML de entrada, de forma que este processo tenha como saída um novo arquivo XML, compatível com o modelo lógico como produto de saída deste processo, que deve possibilitar sua visualização e edição em qualquer editor XML, como é demonstrado na figura 20.

**Figura 20** – Visão geral do processo de conversão.

Fonte: Elaboração do autor (2018).

Para a conversão do modelo conceitual para lógico, o arquivo XML de entrada será baseado nas marcações realizadas para construção de um diagrama Entidade-Relacionamento no ConceptER 2.0. A figura 21 apresenta este processo.

**Figura 21** – Visão geral do processo de conversão do modelo conceitual para o lógico.

Fonte: Elaboração do autor (2018).

## 5.2 Requisitos do Software

Requisitos de software são descrições que direcionam como o sistema deve funcionar e o que fazer para atender a necessidade do cliente e/ou usuário, bem como suas restrições. De forma geral, os requisitos incluem as especificações e restrições de funcionamento. Nesta seção serão descritos os requisitos funcionais e não funcionais definidos para evolução do ConceptER.



### 5.2.1 Requisitos Funcionais

Um requisito funcional descreve uma interação entre o sistema e o seu ambiente (PFLEEGER, 2004), podendo descrever, ainda, como o sistema deve reagir a entradas específicas, como o sistema deve se comportar em situações específicas e o que o sistema não deve fazer (SOMMERVILLE, 2007). A seguir, no quadro 2, são apresentados os requisitos funcionais definidos para esta proposta.

**Quadro 2** – Requisitos Funcionais.

ID	Descrição
RF01	Criar novo modelo lógico a partir de um novo arquivo em branco.
RF02	Abrir modelo lógico a partir de um arquivo salvo.
RF03	Editar modelo lógico a partir de um novo arquivo em branco.
RF04	Editar modelo lógico a partir de um arquivo salvo.
RF05	Converter arquivo XML com modelo conceitual em arquivo XML com modelo logico
RF06	Salvar modelo lógico em arquivo para posterior edição.
RF07	Deve permitir desfazer e refazer ações.

Fonte: Elaboração do autor (2018).

### 5.2.2 Requisitos Não Funcionais

Requisitos não funcionais escrevem restrições sobre os serviços ou funções oferecidas pelo sistema (SOMMERVILLE, 2007), as quais limitam as opções para criar uma solução para o problema (PFLEEGER, 2004).

Um requisito não funcional de software é aquele que descreve não o que o sistema fará, mas como ele fará. Assim, por exemplo, têm-se requisitos de desempenho, requisitos da interface externa do sistema, restrições de projeto e atributos da qualidade. (DEV MEDIA, 2019).

A seguir, no quadro 3, são apresentados os requisitos não funcionais definidos para esta proposta:

**Quadro 3** – Requisitos Não Funcionais.

<b>ID</b>	<b>Descrição</b>
RNF01	Deve ser executado nas plataformas Windows, Linux e Mac com arquiteturas de processamento X86 ou X64 que possuam JVM (Java Virtual Machine) instalado.
RNF02	A documentação será fornecida na mesma pasta em que o software estiver disponível.
RNF03	O sistema será disponibilizado na língua portuguesa, padrão PT-BR.
RNF04	A ferramenta de conversão deve estar integrada a opção de conversão disponível no menu Ferramentas do ConceptER.
RNF05	O módulo gerador de modelos lógicos deve estar integrado no mesmo projeto do módulo gerador de modelos conceituais do ConceptER.

Fonte: Elaboração do autor (2018).

### 5.3 Casos de Uso

De acordo com Jacobson (1992), o caso de uso é “um documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo”.

Nas subseções desta seção, serão apresentados o diagrama geral de casos de uso, ilustrando as principais ações que serão implementadas na ferramenta. E logo a seguir, estas ações serão detalhadas nos quadros referentes às descrições de casos de uso.

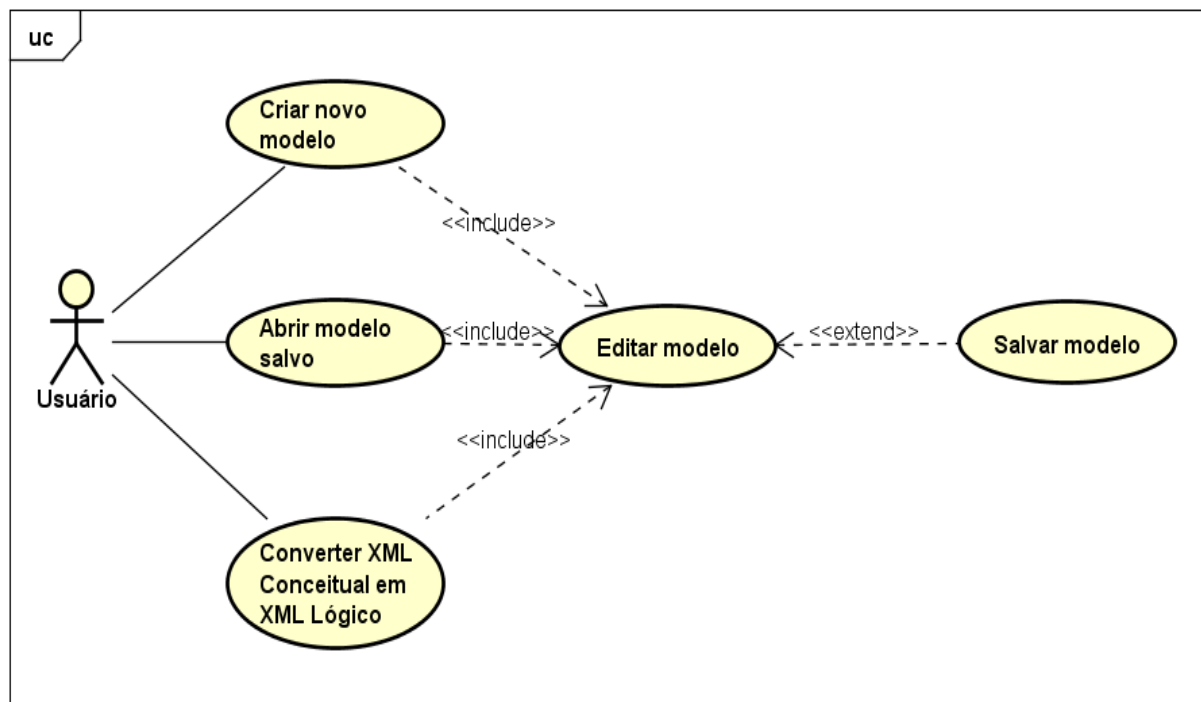
#### 5.3.1 Diagrama de Casos de Uso

Um diagrama de caso de uso documenta o que o sistema faz do ponto de vista do usuário. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. (DEV MEDIA, 2018)

O diagrama de casos de uso deste projeto é apresentado na figura 22, e apresenta a interação do usuário com 5 casos de uso, que são: Criar novo modelo, Abrir modelo salvo, Converter XML conceitual em XML lógico, Editar modelo e Salvar modelo.

A partir deste conceito, a figura 22 detalha as principais ações de rotina que serão implementadas na ferramenta proposta.

**Figura 22** – Diagrama de Casos de Uso.



powered by Astah

Fonte: Elaboração do autor (2018).

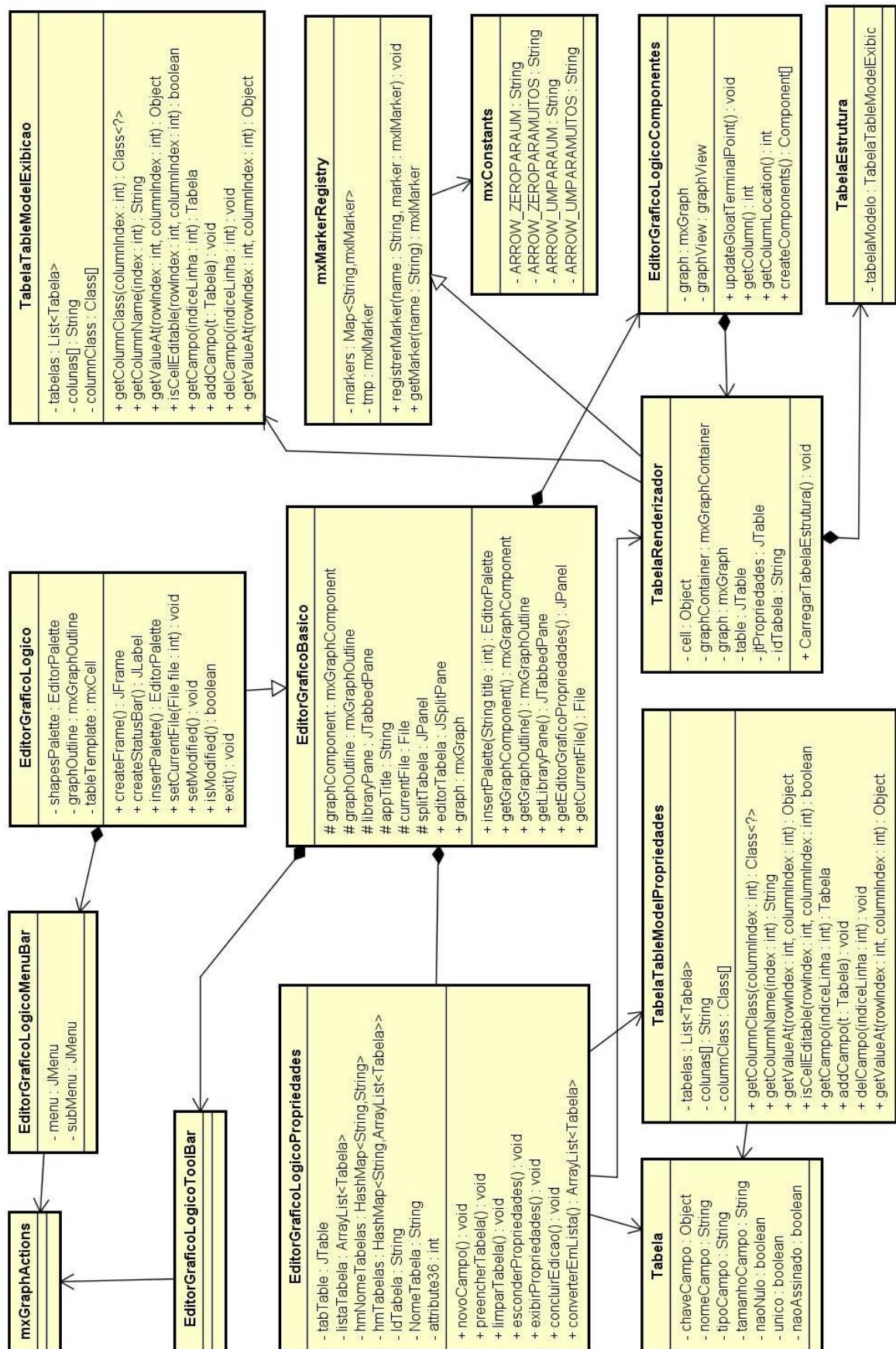
As descrições dos casos de uso estão presentes no apêndice deste trabalho.

## 5.4 Diagrama de Classes

Em programação, um diagrama de classes é uma representação da estrutura e relações das classes que servem de modelo para objetos. (DEVMEDIA, 2019)

A figura 23 apresenta o diagrama de classes do módulo gerador de modelos relacionais da ferramenta ConceptER 3.0, contendo as classes, atributos, métodos, associações, heranças e composições.

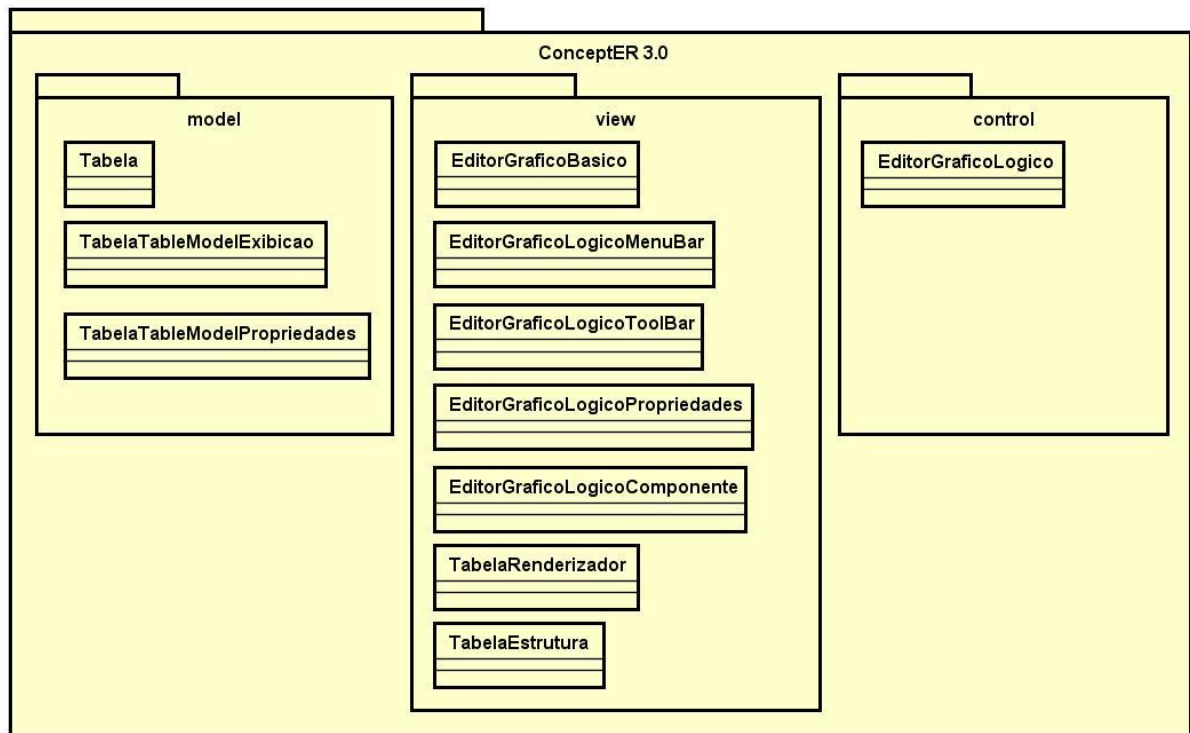
**Figura 23** – Diagrama Classes do Módulo Gerador de Modelos Relacionais.



## 5.5 Diagrama de Pacotes

A figura 24 representa o diagrama de pacotes do módulo gerador de modelos relacionais da ferramenta ConceptER 3.0, apresentando os pacotes que englobam as principais classes.

**Figura 24** – Diagrama de Pacotes do Módulo Gerador de Modelos Relacionais.



powered by Astah

Fonte: Elaboração do autor (2019).

## 5.6 Diagramas de Sequência

Para este trabalho, foram elaborados os principais diagramas de sequência do módulo gerador de modelos relacionais do ConceptER 3.0. De acordo com Guedes (2011), o diagrama de sequência “é um diagrama comportamental que se preocupa com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo”.

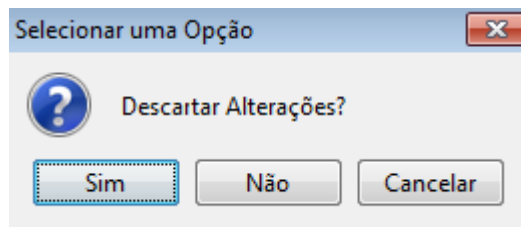
Neste diagrama pode-se identificar os atores responsáveis por um evento acionado, bem como demonstrar o comportamento do processo em meio as chamadas de métodos por mensagens enviadas pelos objetos.

A seguir, serão apresentados os diagramas de sequência elaborados para elucidar as principais funcionalidades citadas no diagrama de casos de uso já abordados na figura 20.



Após o acionamento do submenu Novo, a ação `newAction` é disparada através da classe `EditorAction`. Esta ação solicita a classe `EditorGraficoBasico` informações sobre um componente possivelmente já instanciado. Após receber esta informação, a ação `newAction` continua sua rotina e realiza uma verificação, afim de descobrir se a informação sobre o componente retornada não está vazia. Caso a verificação retorne `true`, uma nova verificação é realizada para saber se o possível componente instanciado sofreu modificações. Se esta nova verificação retornar `true`, uma mensagem é enviada ao Usuário perguntando se deseja descartar as alterações no arquivo atual em edição, como é demonstrado na figura 27.

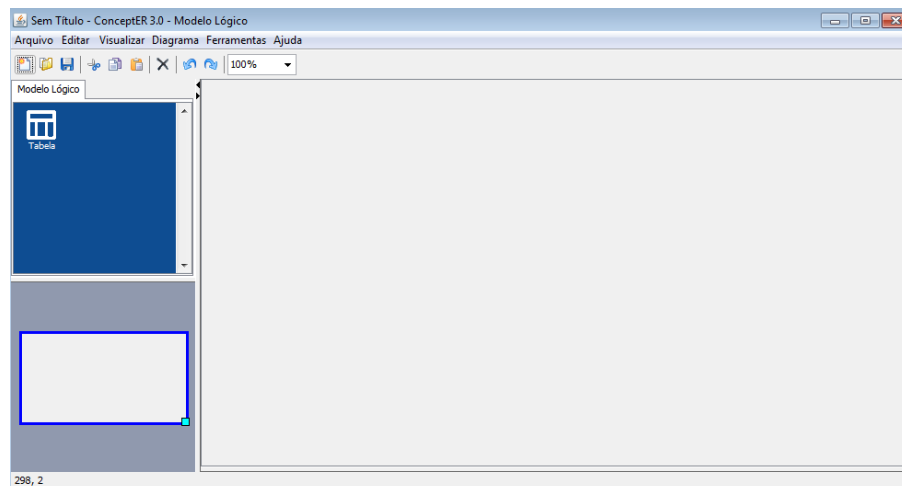
**Figura 27** – JOptionPane – Descartar Alterações.



Fonte: Elaboração do autor (2019).

Se a segunda verificação retornar `false`, a sequência restante do fluxo de ações deste diagrama de sequência é cancelada. Nos casos em que a primeira verificação retorne `false` e a segunda verificação retorne `true`, é dada continuidade ao fluxo. Para estes casos de continuidade do fluxo, logo após a verificação, um novo `EditorGraficoLogicoComponente` é instanciado, fornecendo para classe `EditorGraficoBasico` a visão de um modelo novo, garantindo ao usuário um modelo limpo a ser editado, como é ilustrado na figura 28.

**Figura 28** – Ambiente de Edição com Novo Modelo.



Fonte: Elaboração do autor (2019).

A figura 29 revela um fragmento do código fonte da classe EditorActions responsável por disparar a rotina de criação de um novo modelo.

**Figura 29** – Script actionPerformed para criar novo modelo.

```
public void actionPerformed(ActionEvent e) {
    EditorGraficoBasico editor = getEditor(e);
    if (editor != null) {
        if (!editor.isModified())
            || JOptionPane.showConfirmDialog(editor,
                mxResources.get("loseChanges")) == JOptionPane.YES_OPTION)
        {
            mxGraph graph = editor.getGraphComponent().getGraph();
            mxCell root = new mxCell();
            root.insert(new mxCell());
            graph.getModel().setRoot(root);

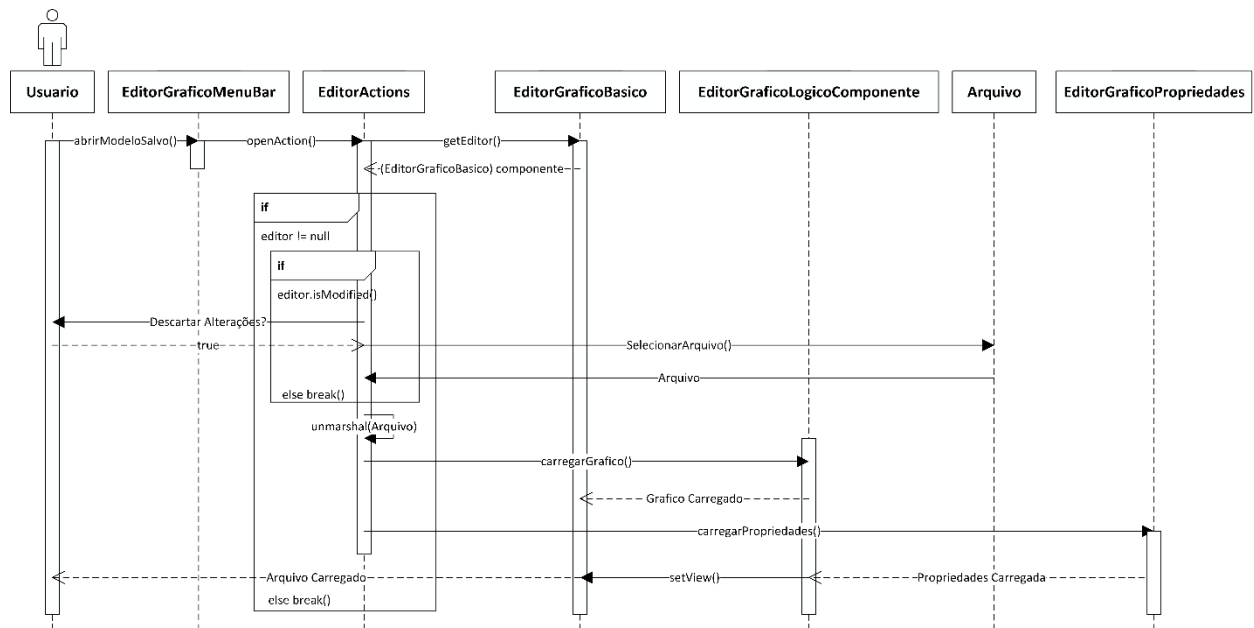
            editor.setModified(false);
            editor.setCurrentFile(null);
            editor.getGraphComponent().zoomAndCenter();
        }
    }
}
```

Fonte: Elaboração do autor (2019).

### 5.6.2 Diagrama de Sequência Abrir Modelo Salvo

A figura 30 apresenta o diagrama de sequência referente a funcionalidade “Criar novo modelo”.

**Figura 30** – Diagrama de Sequência: Abrir Modelo Salvo.

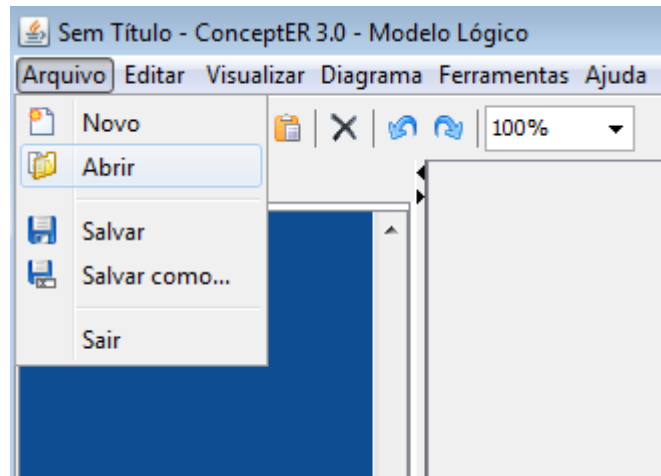


Fonte: Elaboração do autor (2019).



O diagrama de sequência da figura 30 demonstra o comportamento do fluxo de métodos e mensagens que se iniciam quando o Usuário deseja abrir um modelo lógico salvo acionando o submenu “Abrir” do menu “Arquivo” que está presente na Classe “EditorGraficoMenuBar” já instanciada na classe “EditorGraficoLogico”, como é ilustrado na figura 31.

**Figura 31** – Submenu Abrir.



Fonte: Elaboração do autor (2019).

Após o acionamento do submenu Abrir, a ação `openAction` é disparada através da classe `EditorAction`. Esta ação solicita a classe `EditorGraficoBasico` informações sobre um componente possivelmente já instanciado. Após receber esta informação, a ação `newAction` continua sua rotina e realiza uma verificação, afim de descobrir se a informação sobre o componente retornada não está vazia. Caso a verificação retorne `true`, uma nova verificação é realizada para saber se o possível componente instanciado sofreu modificações. Se esta nova verificação retornar `true`, uma mensagem é enviada ao Usuário perguntando se deseja descartar as alterações no arquivo atual em edição, conforme demonstrado anteriormente na figura X.

Se a segunda verificação retornar `false`, a sequência restante do fluxo de ações deste diagrama de sequência é cancelada. Nos casos em que a primeira verificação retorne `false` e a segunda verificação retorne `true`, é dada continuidade ao fluxo. Para estes casos de continuidade do fluxo, logo após a verificação é disparado o evento para selecionar o arquivo desejado, este arquivo é carregado em uma variável do tipo *File* que sofrerá a desserialização, *unmarshal*, onde os metadados XML são convertidos para objetos Java, que são armazenados em uma variável do tipo `SavedObjectLogico`, que é uma classe modelo com anotações XML responsável pela estrutura de serialização e desserialização para objetos Java criados através do modulo gerador de modelo lógico do ConceptER 3.0.

**Figura 32** – Script de desserialização do arquivo com modelo salvo.

```
JAXBContext context = JAXBContext.newInstance(SavedObjectLogico.class);
Unmarshaller um = context.createUnmarshaller();
SavedObjectLogico svol = (SavedObjectLogico) um.unmarshal(fc.getSelectedFile());
grafico = svol.getGrafico();
tabMapAux = svol.getTabelaMap();
```

Fonte: Elaboração do autor (2019).

Podemos verificar na figura 32, que após o processo de desserialização dos dados do arquivo, a variável que armazena os objetos Java é usada para definir o valor de outras duas variáveis. Neste caso, a variável `grafico` é do tipo `String` e receberá o objeto que possui as informações para gerar os elementos gráficos da classe `EditorGraficoComponente`. Já a variável `tabMapAux`, receberá os objetos que fazem referência ao mapeamento das tabelas.

A variável `grafico` sofrerá um parseamento do tipo XML para `Document`, sendo armazenadas em uma nova variável deste segundo tipo, que será decodificada para gerar o `grafico` no ambiente de edição da ferramenta, conforme é demonstrado na figura 33.

**Figura 33** – Script de parseamento e decodificação dos objetos de criação do gráfico.

```
Document document = mxXmlUtils.parseXml(grafico);
mxCodec codec = new mxCodec(document);
codec.decode(document.getDocumentElement(), graph.getModel());
```

Fonte: Elaboração do autor (2019).

A figura 34 demonstra que a variável `tabMapAux` é percorrida, de modo que a cada *loop*, cria um novo `ArrayList` do tipo `Tabela` e seta os dados de cada campo. Ao final de cada *loop*, a tabela criada e seu respectivo id são armazenados no `HashMap`, de nome `hmTabelas`, estático da classe `EditorGraficoLogicoPropriedades`.

**Figura 34** – Script de alimentação do `HashMap` da classe de propriedades de tabelas.

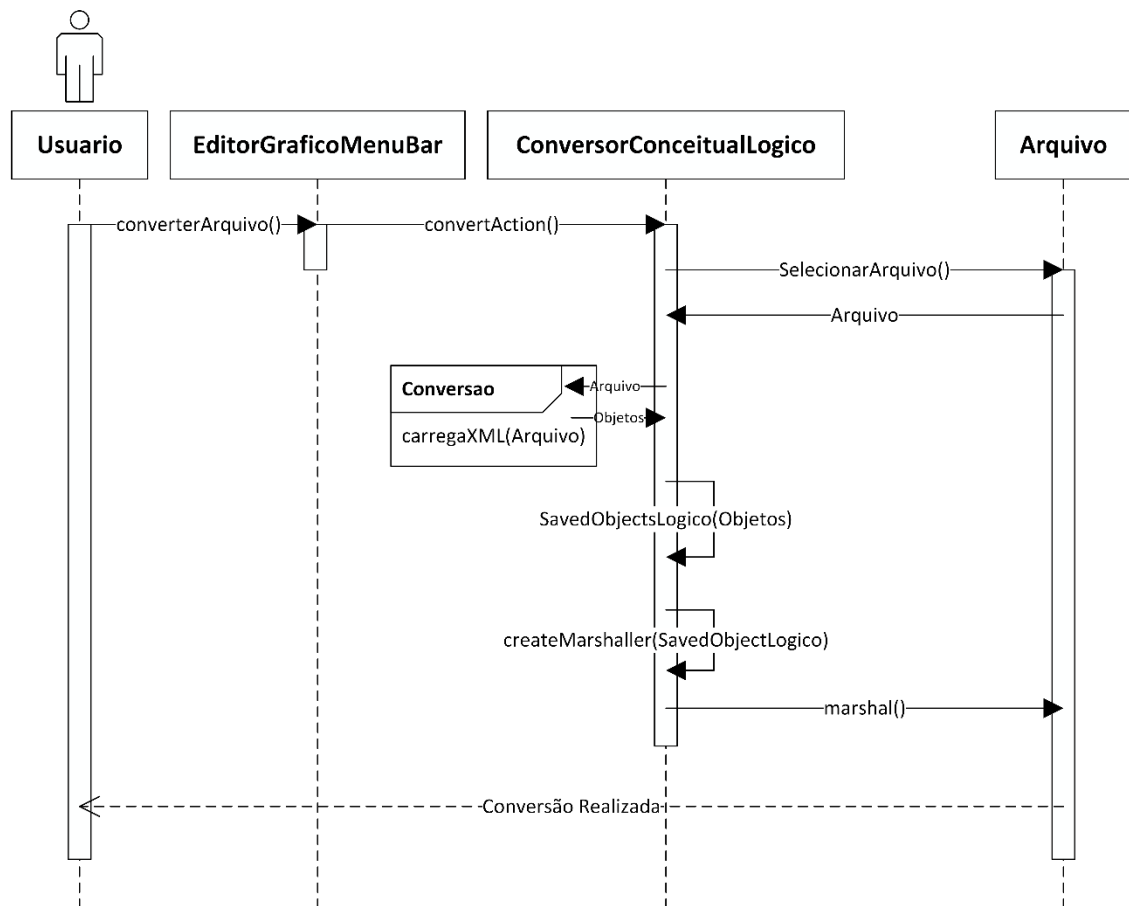
```
for(String s : tabMapAux.getTabelaMap().keySet()) {
    ArrayList<Tabela> arrTabela = new ArrayList<Tabela>();
    for(int i = 0; i < tabMapAux.getTabelaMap().get(s).getTabelasList().size(); i++) {
        Tabela tbTabela = new Tabela();
        tbTabela.setChaveCampo(tabMapAux.getTabelaMap().get(s).getTabelasList().get(i).getChaveCampo());
        tbTabela.setNomeCampo(tabMapAux.getTabelaMap().get(s).getTabelasList().get(i).getNomeCampo());
        tbTabela.setTipoCampo(tabMapAux.getTabelaMap().get(s).getTabelasList().get(i).getTipoCampo());
        tbTabela.setTamanhoCampo(tabMapAux.getTabelaMap().get(s).getTabelasList().get(i).getTamanhoCampo());
        tbTabela.setNaoNulo(tabMapAux.getTabelaMap().get(s).getTabelasList().get(i).getNaoNulo());
        tbTabela.setUnico(tabMapAux.getTabelaMap().get(s).getTabelasList().get(i).getUnico());
        tbTabela.setNaoAssinado(tabMapAux.getTabelaMap().get(s).getTabelasList().get(i).getNaoAssinado());
        arrTabela.add(tbTabela);
    }
    EditorGraficoLogicoPropriedades.hmTabelas.put(s, arrTabela);
}
```

Fonte: Elaboração do autor (2019).

### 5.6.3 Diagrama de Sequência Converter XML Conceitual em XML Lógico

A figura 35 apresenta o diagrama de sequência referente a funcionalidade “Converter XML Conceitual em XML Lógico”.

**Figura 35** – Diagrama de Sequência: Converter XML Conceitual em XML Lógico.

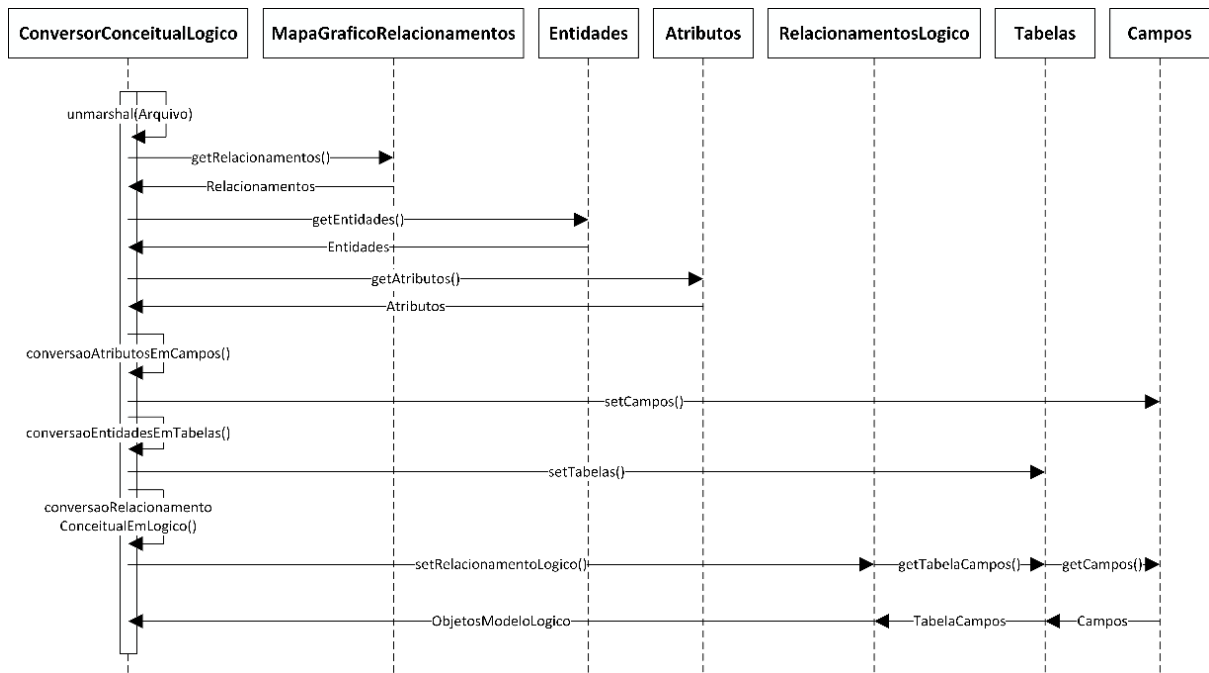


Fonte: Elaboração do autor (2019).

O fluxo do diagrama de sequência apresentado na figura 35 inicia seu fluxo quando o usuário deseja obter um arquivo XML baseado em modelo lógico a partir de um XML baseado em modelo conceitual preexistente.

O usuário aciona a opção “Converter XML Conceitual em XML Lógico” que se encontra no menu “Ferramentas” da classe EditorGraficoMenuBar e a ação de conversão é acionada na classe ConversorConceitualLogico.

O fluxo para a operação de conversão será destrinchado em um outro diagrama de sequência, apresentado na figura 36.

**Figura 36** – Diagrama de Sequência: Fluxo de Conversão Conceitual para Lógico.

Fonte: Elaboração do autor (2019).

O fluxo de conversão inicia quando o usuário seleciona o arquivo que contém os dados XML do modelo conceitual criado no módulo gerador de modelo conceitual do ConceptER. Este arquivo será carregado na Classe `ConversorConceitualLogico`, logo sofrerá uma desserialização pelo método `unmarshal` do JAXB e será armazenado em uma variável, denominada `svoXML`, do tipo `SavedObject`, que é uma classe modelo com anotações XML responsável pela estrutura de serialização e desserialização para objetos Java criados através do módulo gerador de modelo conceitual do ConceptER 3.0, como apresenta a figura 37.

**Figura 37** – Script de desserialização do arquivo XML.

```

public static SavedObject carregaXML() throws JAXBException{
    JAXBContext context = JAXBContext.newInstance(SavedObject.class);
    Unmarshaller um = context.createUnmarshaller();
    Object obj = um.unmarshal(arquivo);
    SavedObject svoXML = (SavedObject) obj;
    return svoXML;
}
  
```

Fonte: Elaboração do autor (2019).

Com a variável carregada, iniciamos o fluxo de mapeamento começando pelos relacionamentos, como é ilustrado na figura 38.

**Figura 38** – Script de mapeamento de relacionamentos do modelo conceitual.

```

for (Integer iRC : svo.getMapaGraficoRelacionamentos().keySet()) {
    List listEntidades = new ArrayList();
    rl.setId(svo.getMapaGraficoRelacionamentos().get(iRC).getId());
    rl.setNome(svo.getMapaGraficoRelacionamentos().get(iRC).getNome());
}

```

Fonte: Elaboração do autor (2019).

Através do mapeamento dos relacionamentos é possível acessar as Entidades e seus respectivos atributos, permitindo aplicar as regras de conversão. A primeira regra será para cada tipo de entidade regular (forte), criar uma relação que inclua todos os atributos simples. Para isso é necessário também capturar as informações sobre a cardinalidade e obrigatoriedade das Entidades em seus relacionamentos, estas informações serão úteis para definir a cardinalidade e participação de cada tabela para o relacionamento lógico. As entidades são carregadas em um ArrayList, de forma que seja possível percorrer sua estrutura e a partir de cada variável carregada com seu respectivo Id, realizar o mapeamento de seus atributos que é demonstrado na figura 39.

**Figura 39** – Script de mapeamento de entidades dos relacionamentos mapeados

```

for (Entidade en : svo.getMapaGraficoRelacionamentos().get(iRC).getEntidades().keySet()) {
    boolean possuiChave = false;
    List listaAtributos1 = new ArrayList();
    Tabela t = new Tabela();
    String cardTemp = svo.getMapaGraficoRelacionamentos().get(iRC).getEntidades().get(en).getCardinalidade();
    TipoObrigatoriedadeEnum obrTemp = svo.getMapaGraficoRelacionamentos().get(iRC).getEntidades().get(en).getObrigatoriedade();
    t.setId(en.getId());
    t.setNome(en.getNome());
    t.setCardinalidade(defineCardinalidade(cardTemp, obrTemp));
    t.setParticipacao(defineParticipacao(obrTemp));
}

```

Fonte: Elaboração do autor (2019).

A figura 40 apresenta o fluxo de mapeamento dos atributos, novos campos são criados atribuindo para cada campo as informações preliminares, respectivas dos atributos, como Id e Nome. Em seguida é coletado dados referentes aos tipos de atributos que definem os tipos de campos, como: campo simples, chave-primária e chave-estrangeira.

Neste mesmo fluxo ocorre a atribuição de mais uma regra. Esta regra define a inclusão de apenas atributos de componente simples de um atributo composto como novo campo. Os atributos do tipo derivado, por regra, são ignorados.

**Figura 40** – Script de mapeamento de atributos das entidades mapeadas.

```

for(int at = 0; at < en.getAtributos().size(); at++){
    Campo c = new Campo();
    c.setId(en.getAtributos().get(at).getId());
    c.setNome(en.getAtributos().get(at).getNome());
    tipoAtributoEnum = en.getAtributos().get(at).getTipoAtributo();
    if(tipoAtributoEnum != DERIVADO) switch (tipoAtributoEnum){
        case SIMPLES:
            c.setTipo(TipoCampoEnum.SIMPLES);
            listaAtributos1.add(c);
            break;
        case CHAVE:
            c.setTipo(TipoCampoEnum.CHAVEPRIMARIA);
            listaAtributos1.add(c);
            possuiChave = true;
            break;
        case COMPOSTO:
            for (int atc = 0; atc < en.getAtributos().get(at).getAtributos().size(); atc++){
                Campo cc = new Campo();
                cc.setId(en.getAtributos().get(at).getAtributos().get(atc).getId());
                cc.setNome(en.getAtributos().get(at).getAtributos().get(atc).getNome());
                cc.setTipo(TipoCampoEnum.SIMPLES);
                listaAtributos1.add(cc);
            }
            break;
    }
}

```

Fonte: Elaboração do autor (2019).

Caso o tipo de atributo seja multivalorado é atribuída a seguinte regra: para cada atributo multivalorado, uma nova tabela é criada e incluirá como campos de chave primaria a chave primária do atributo multivalorado e mais uma chave com o nome do atributo multivalorado, formando uma chave composta, conforme demonstrado na figura 41.

**Figura 41** – Script de mapeamento de atributos multivalorados.

```

case MULTIVALORADO:
    Tabela tt = new Tabela();
    List listaAtributos2 = new ArrayList();
    for (int atc = 0; atc < en.getAtributos().get(at).getAtributos().size(); atc++){
        Campo cc = new Campo();
        if(tipoAtributoEnum == CHAVE){
            cc.setId("m"+en.getAtributos().get(at).getAtributos().get(atc).getId());
            cc.setNome(en.getAtributos().get(at).getAtributos().get(atc).getNome());
            cc.setTipo(TipoCampoEnum.CHAVEPRIMARIA);
            listaAtributos2.add(cc);
        }
        if(tipoAtributoEnum == MULTIVALORADO){
            cc.setId("m"+en.getAtributos().get(at).getAtributos().get(atc).getId());
            cc.setNome(en.getAtributos().get(at).getAtributos().get(atc).getNome());
            cc.setTipo(TipoCampoEnum.CHAVEPRIMARIA);
            listaAtributos2.add(cc);
        }
    }
    t.setCampos(listaAtributos2);
    listEntidades.add(t);
    break;

```

Fonte: Elaboração do autor (2019).

Ainda no mesmo fluxo, para garantir a integridade referencial das tabelas geradas, é realizado uma verificação para saber se a tabela possui algum campo do tipo chave-primária. Caso não possua, é criada uma chave-primária genérica para tabela, conforme exibe a figura 42.

**Figura 42** – Script de criação de chave-primária genérica.

```
if(possuiChave == false){
    listaAtributos2.add(criaCampoChavePrimaria(t.getId(),t.getNome()));
    t.setCampos(listaAtributos2);
}

listEntidades.add(t);
```

Fonte: Elaboração do autor (2019).

Ao final do fluxo, as tabelas são adicionadas a lista de tabelas e é criado um novo relacionamento logico contendo as respectivas tabelas relacionadas, como demonstra a figura 43.

**Figura 43** – Script de atribuição para um novo relacionamento lógico.

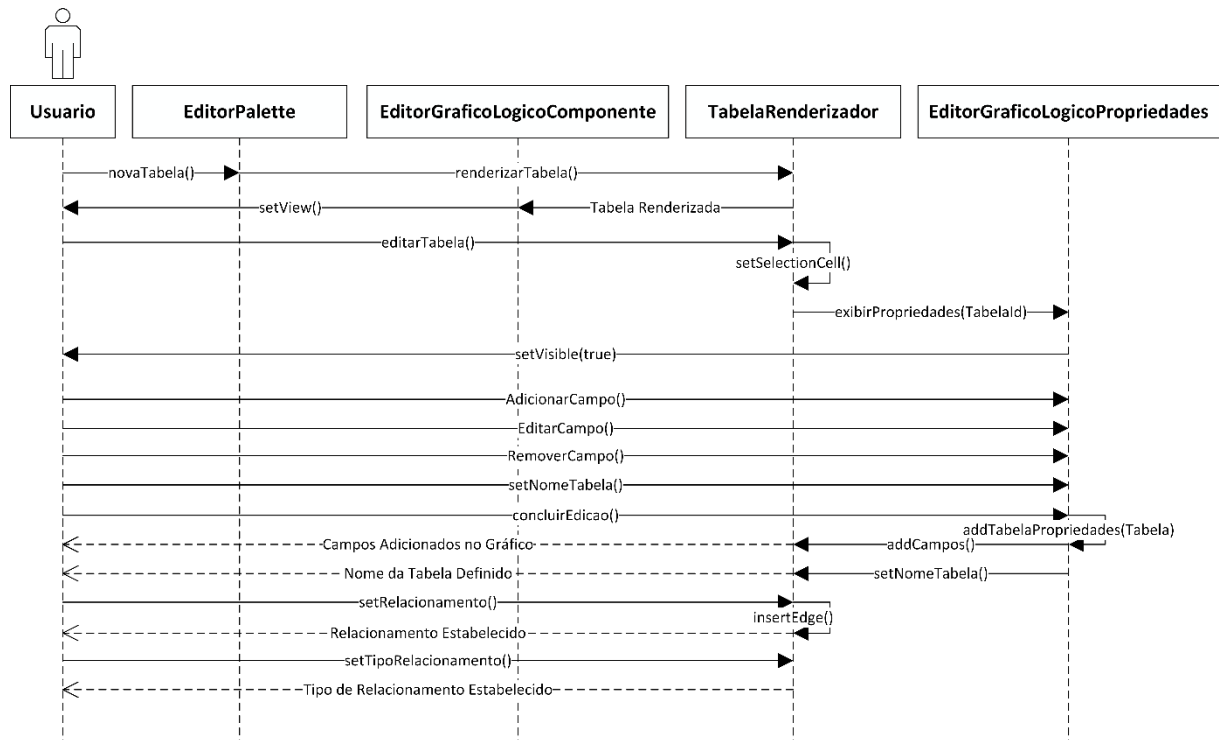
```
rl.setTabelas(listEntidades);
RelacionamentoLogico rlhs = new RelacionamentoLogico();
rlhs.setId(rl.getId());
rlhs.setNome(rl.getNome());
rlhs.setTabelas(rl.getTabelas());
mapRelacionamentoLogico.put(rl.getId(), rlhs);
```

Fonte: Elaboração do autor (2019).

O resultado desta conversão é inserido em um arquivo de estrutura XML, que é apresentado com detalhes no capítulo 6.

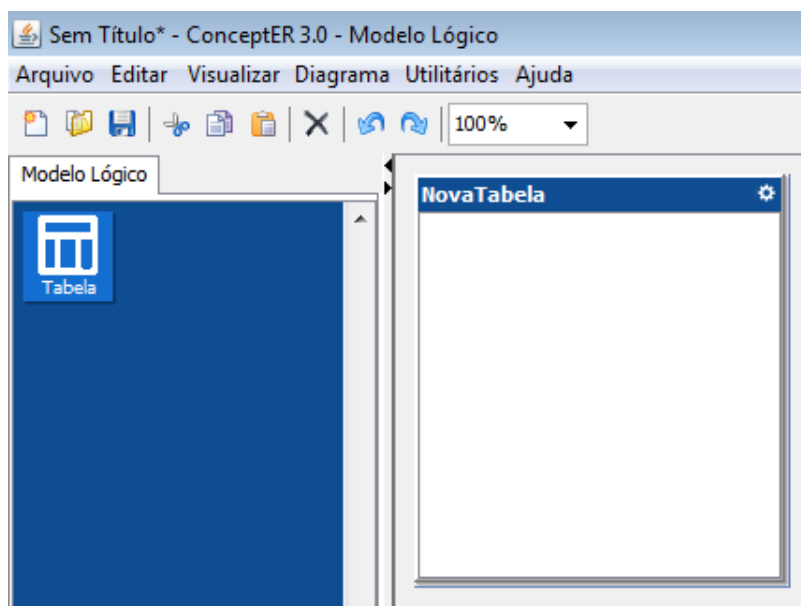
#### 5.6.4 Diagrama de Sequência Editar Modelo

A figura 44 representa o diagrama de sequência “Editar Modelo”.

**Figura 44** – Diagrama de Sequência: Editar Modelo

Fonte: Elaboração do autor (2019).

O fluxo do diagrama de sequência inicia quando o usuário arrasta o ícone da tabela que está na paleta de opções definida na classe EditorPalette para a área de modelagem definida no EditorGraficoComponente. Esta ação serve para adicionar novas tabelas ao modelo em edição, conforme ilustra a tabela 45.

**Figura 45** – Nova Tabela Criada.

Fonte: Elaboração do autor (2019).



Uma nova instância da classe `TabelaRenderizador` é adicionada à área gráfica de edição, baseada em um template predefinido, contendo uma tabela vazia de nome “NovaTabela” e dimensões padronizadas, de acordo com a figura 46.

**Figura 46** – Script de template predefinido para novas tabelas.

```
mxCell tableTemplate = new mxCell("NovaTabela", new mxGeometry(0, 0, 200, 220), null);
tableTemplate.getGeometry().setAlternateBounds(new mxRectangle(0, 0, 140, 25));
tableTemplate.setVertex(true);
```

Fonte: Elaboração do autor (2019).

Após as tabelas serem adicionadas ao componente de edição, elas podem ser editadas acionando o botão `ExibirPropriedades` que está localizado ao lado do título da tabela com um ícone de uma engrenagem, como demonstrado na figura 47.

**Figura 47** – Botão `ExibirPropriedades`.



Fonte: Elaboração do autor (2019).

Ao pressionar o botão `ExibirPropriedades`, uma nova ação é disparada, onde a tabela em questão será atribuída como a célula selecionada para edição. Conforme demonstra a figura 48, o id e nome da tabela são capturados e enviados para a classe `EditorGraficoLogicoPropriedades`, o painel de propriedades é exibido e caso a tabela já contenha dados, estes dados serão atribuídos aos componentes do painel de propriedades.

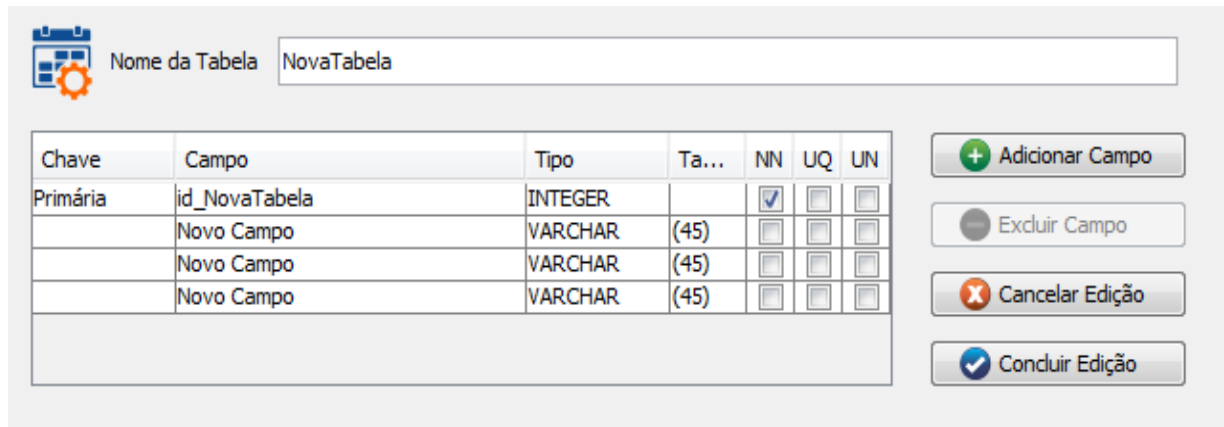
**Figura 48** – Script de ação para clique no botão `ExibirPropriedades`.

```
graph.setSelectionCell(cell);
idTabela = ((mxCell)graph.getSelectionCell()).getId();
String nomeTabela = ((mxCell)graph.getSelectionCell()).getValue().toString();
EditorGraficoLogicoPropriedades.exibirPropriedades();
EditorGraficoLogicoPropriedades.getIdTabela(idTabela);
EditorGraficoLogicoPropriedades.getNomeTabela(nomeTabela);
EditorGraficoLogicoPropriedades.limparTabela();
EditorGraficoLogicoPropriedades.preencherTabela2(idTabela, nomeTabela);;
```

Fonte: Elaboração do autor (2019).

Com o painel de propriedades ativo é possível realizar várias ações, como: editar o nome da tabela, adicionar campos, editar campos, remover campos, cancelar a edição e concluir a edição, como é exibido na figura 49.

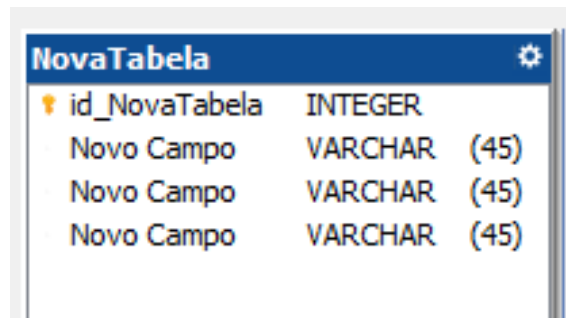
**Figura 49** – Painel de Propriedades de Tabelas.



Fonte: Elaboração do autor (2019).

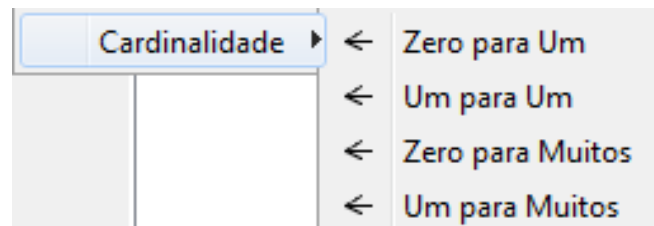
Após concluir a edição a tabela instanciada receberá os respectivos dados editados no painel de propriedades, conforme demonstra a figura 50.

**Figura 50** – Tabela Carregada.



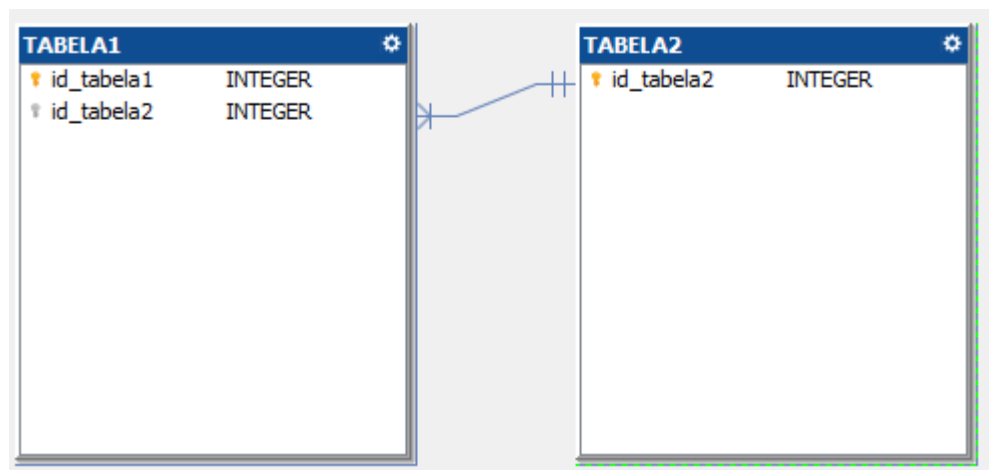
Fonte: Elaboração do autor (2019).

Quando o componente gráfico de edição possuir duas ou mais tabelas, será possível criar relacionamento entre as tabelas e atribuir os tipos de relacionamentos, conforme a figura 51, que são baseados em cardinalidade e participação, através do clique direito do botão do mouse no relacionamento.

**Figura 51** – Tipos de Relacionamentos.

Fonte: Elaboração do autor (2019).

Selecionando o tipo de relacionamento, o gráfico para o lado escolhido da relação será modificado. O resultado é apresentado na figura 52.

**Figura 52** – Tabelas Relacionadas.

Fonte: Elaboração do autor (2019).

Os dados das tabelas exibidas nos gráficos são armazenados em um HashMap, que contém o modelo de tabela baseado nos atributos para tipo de chave, nome do campo, tipo de campo e tamanho de campo.

Os dados de propriedades da tabela que estão associados as respectivas tabelas exibidas no componente gráfico de edição, são armazenadas em um outro HashMap que contém um modelo de tabela baseado nos atributos para tipo de chave, nome do campo, tipo de campo, tamanho de campo, não nulo, único e não assinado.

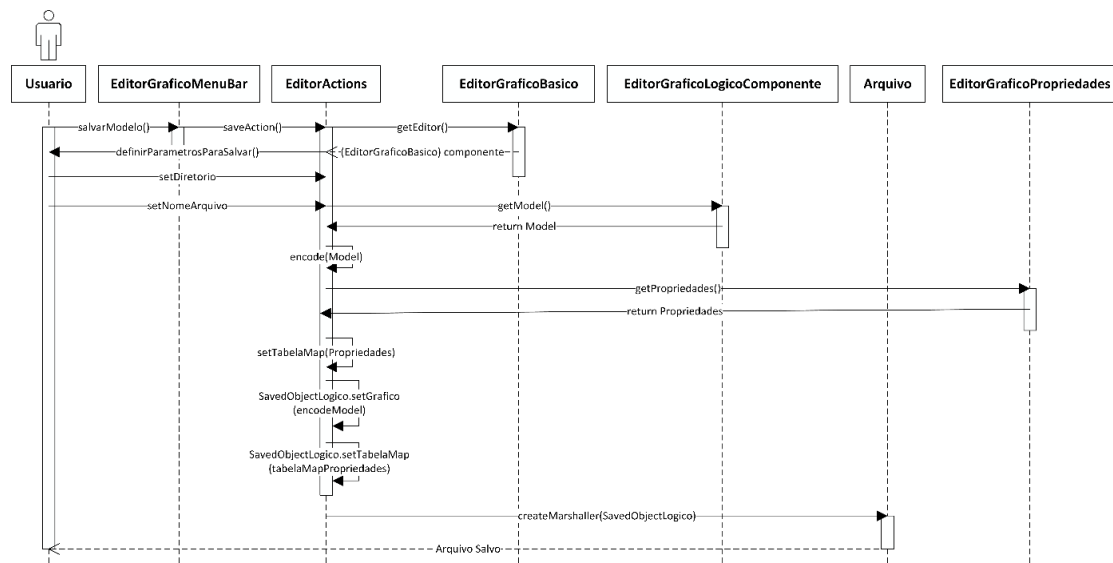
As alterações nos dados da tabela baseado no HashMap de propriedades, refletem automaticamente nos dados armazenados nas tabelas baseadas no HashMap de tabelas do gráfico.

O fluxo se encerra quando o usuário inicia qualquer outro fluxo de outro diagrama de sequência, como: novo modelo, abrir modelo salvo, converter modelo conceitual para lógico e/ou salvar modelo.

### 5.6.5 Diagrama de Sequência Salvar Modelo

A figura 53 apresenta o diagrama de sequência para a funcionalidade “Salvar Modelo”.

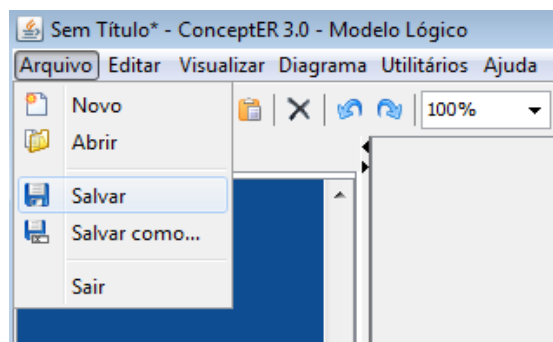
**Figura 53** – Diagrama de Sequência: Salvar Modelo.



Fonte: Elaboração do autor (2019).

O diagrama de sequência da figura 53 demonstra o comportamento do fluxo de métodos e mensagens que se iniciam quando o Usuário deseja salvar um modelo lógico acionando o submenu “Salvar” do menu “Arquivo” que está presente na Classe “EditorGraficoMenuBar” já instanciada na classe “EditorGraficoLogico”, como é apresentado na figura 54.

**Figura 54** – Submenu Salvar.



Fonte: Elaboração do autor (2019).

Após acionamento do submenu Salvar a ação `saveAction` é disparada através da classe `EditorAction`. Esta ação solicita a classe `EditorGraficoBasico` informações sobre um componente possivelmente já instanciado. Após receber esta informação, a ação `saveAction` solicita ao usuário informações referentes ao nome que deseja dar ao arquivo e também o diretório que deseja salva-lo. Após receber estas informações, é solicitado ao `EditorGraficoComponente` o modelo gráfico atual que está instanciado em sua classe que prontamente retorna o componente para ação `saveAction`, que por sua vez insere este componente gráfico em uma variável local do tipo `mxGraphComponent` para que seja obtido informações do gráfico que serão armazenados em outra variável local do tipo `mxGraph`, conforme demonstrado na figura 55.

**Figura 55** – Script de captura de informações do gráfico.

```
EditorGraficoBasico editor = getEditor(e);
    if (editor != null) {
        mxGraphComponent graphComponent = editor.getGraphComponent();
        mxGraph graph = graphComponent.getGraph();
```

Fonte: Elaboração do autor (2019).

Com estes dados obtidos sobre o gráfico, é realizado a codificação do gráfico para o formato de metadados XML e armazenados em uma variável do tipo `String`. Em sequência é solicitado a classe `EditorgraficoPropriedades` as propriedades, das tabelas, que estão armazenadas em uma variável do tipo `HashMap`. Os dados do gráfico e os dados de propriedades são atribuídos a uma variável que contém o modelo estrutural de salvamento predefinido na classe `SavedObjectLogico`. O fluxo tem sua continuidade com a serialização do conteúdo da variável de tipo `SavedObjectLogico` para um arquivo, definido pelo usuário, com estrutura XML e extensão “.er3”. O fluxo descrito neste parágrafo é representado na figura 56.

**Figura 56** – Codificação e serialização do gráfico e propriedades das tabelas.

```
String xml = mxXmlUtils.getXml(codec.encode(graph.getModel()));
TabelaMap tabMapa = new TabelaMap();
tabMapa.setTabelaMap(EditorGraficoLogicoPropriedades.hmTabelas2);
SavedObjectLogico svol = new SavedObjectLogico();
svol.setGrafico(xml);
svol.setTabelaMap(tabMapa);
JAXBContext jaxbContext = JAXBContext.newInstance(SavedObjectLogico.class);
Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
OutputStream os = new FileOutputStream(filename);
jaxbMarshaller.marshal(svol, os);
```

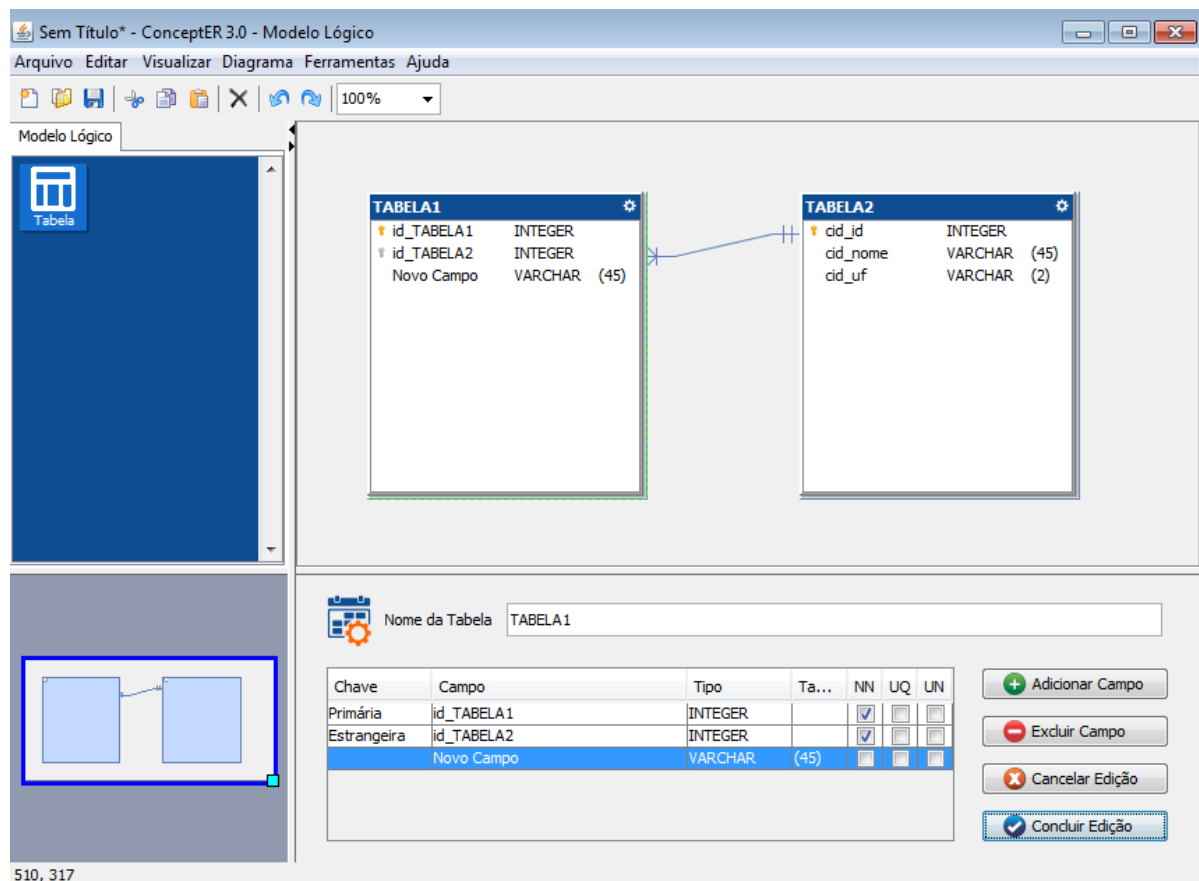
Fonte: Elaboração do autor (2019).

## 6 DETALHAMENTO DA FERRAMENTA

Os novos módulos do conceptER 3.0 tem como propósito permitir a criação de modelo relacionais e também de converter arquivos XML baseados em modelos conceituais em arquivos XML baseados em modelos lógicos.

A figura 57 apresenta a interface principal do módulo gerador de modelos relacionais, que permite a criação e edição de modelos relacionais. Esta mesma figura apresenta um modelo salvo que foi aberto, e que está em edição. Este modelo apresentado faz referência a duas tabelas, TABELA1 e TABELA2, que estão instanciadas no componente de edição. A TABELA1 demonstra um componente da classe TabelaRenderizador que possui uma JTable carregada com três linhas e quatro colunas, sendo cada linha a representação de um campo da tabela. Quanto as colunas, temos que: a primeira coluna representa se o campo possui uma chave, e caso sim, o tipo de chave; a segunda coluna representa o nome do campo; a terceira coluna representa o tipo do campo e; a quarta coluna representa o tamanho do campo.

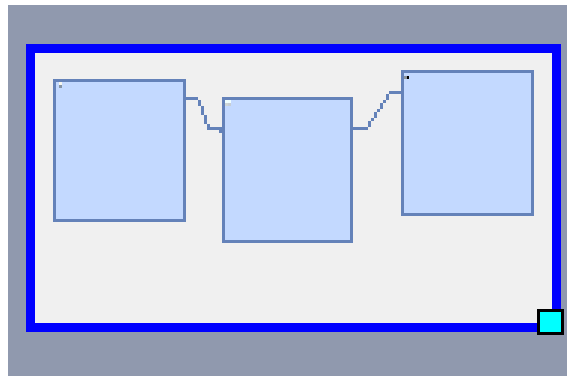
**Figura 57** – Interface principal do módulo gerador de modelos relacionais.



Fonte: Elaboração do autor (2019).

A figura 58 representa um componente gráfico do tipo `mxGraphOutline`, que interage com a classe `EditorGraficoLogicoComponente` a fim de exibir uma visualização de todos os componentes inseridos comportados em toda extensão do componente de edição. Através deste componente é possível também acionar a ferramenta de *zoom*, para aumentar ou diminuir a visualização.

**Figura 58** – Componente gráfico `mxGraphOutline`.



Fonte: Elaboração do autor (2019).

Na interface principal do módulo de criação de modelos relacionais, ao acionar o botão de propriedades de quaisquer tabela, inserida no componente de edição, um painel de propriedades será exibido com o conteúdo desta tabela. Este painel de propriedades permite ao usuário definir as propriedades da tabela, como: alterar nome da tabela, adicionar campos, editar campos, excluir campos, cancelar a edição e concluir a edição. O painel de propriedades da tabela é exibido na figura 59.

**Figura 59** – Painel de propriedades ativo.

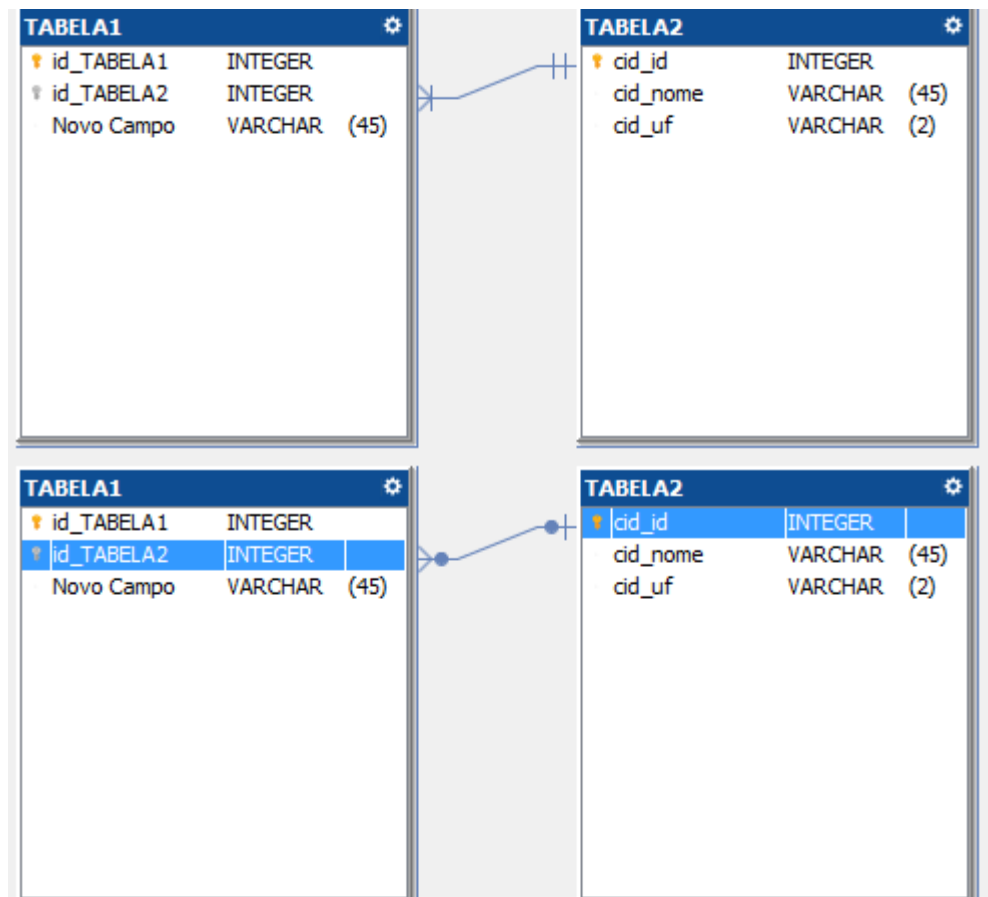
Chave	Campo	Tipo	Ta...	NN	UQ	UN
Primária	id_TABELA1	INTEGER		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Estrangeira	id_TABELA2	INTEGER		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Novo Campo	INTEGER (45)		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		VARCHAR				
		BOOLEAN				
		FLOAT				
		CHAR				
		DATE				

Nome da Tabela: TABELA1

Fonte: Elaboração do autor (2019).

Durante a construção do modelo relacional, o usuário após instanciar as tabelas e relaciona-las, poderá definir os tipos de relacionamentos. Conforme demonstrado na figura 60.

**Figura 60** – Relacionamentos entre tabelas.



Fonte: Elaboração do autor (2019).

Ao finalizar a modelagem, o usuário poderá salvar sua modelagem em um arquivo e posteriormente abrir novamente a modelagem realizada a partir deste arquivo. O arquivo salvo possui estrutura de metadados XML e extensão “.er3”, que permite sua edição em qualquer editor XML.

Os dados contidos no arquivo salvo fazem referência a dois principais objetos para carregamento da modelagem salva, os dados do gráfico e os dados de propriedades da tabela. Os dados gráficos são responsáveis pelas informações que vão gerar os gráficos na área de edição gráfica, e os dados de propriedades da tabela são o conjunto de informações que serão carregados no painel de propriedades que fazem referência a cada tabela do gráfico. O resultado do XML adicionado no arquivo salvo é apresentado na figura 61.



**Figura 61** – Arquivo XML com modelo logico salvo.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<savedObjectLogico>
  <grafico>&lt;mxGraphModel&gt;&lt;&lt;root&gt;&lt;&lt;mxCell id="0"/&gt;&lt;&lt;mxCell id="1" parei
  <tabelaMap>
    <Tabelas>
      <entry>
        <key>2</key>
        <value>
          <Tabela>TABELA1</Tabela>
          <Campos>
            <Campo>
              <chave xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xsi:type="xs:string">Primária</chave>
              <nomeCampo>id_TABELA1</nomeCampo>
              <tipoCampo>INTEGER</tipoCampo>
              <tamanhoCampo> </tamanhoCampo>
              <naoNulo>true</naoNulo>
              <unico>false</unico>
              <naoAssinado>false</naoAssinado>
            </Campo>
            <Campo>
              <chave xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xsi:type="xs:string">Estrangeira</chave>
              <nomeCampo>id_TABELA2</nomeCampo>
              <tipoCampo>INTEGER</tipoCampo>
              <tamanhoCampo></tamanhoCampo>
              <naoNulo>true</naoNulo>
              <unico>false</unico>
              <naoAssinado>false</naoAssinado>
            </Campo>
            <Campo>
              <chave xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xsi:type="xs:string"> </chave>
              <nomeCampo>Novo Campo</nomeCampo>
            </Campo>
          </Campos>
        </value>
      </entry>
    </Tabelas>
  </tabelaMap>
</grafico>
</savedObjectLogico>
```

Fonte: Elaboração do autor (2019).

O ConceptER 3.0 integra o módulo de criação de modelos conceituais, que resulta em um diagrama de entidade relacionamento, com o módulo de criação de modelos relacionais, que resulta em um modelo lógico de banco de dados. Quando o usuário salva uma modelagem realizada no módulo de criação de modelos conceituais, o resultado do salvamento é um arquivo com estrutura XML e extensão “.er0”. Este arquivo possui metadados estruturados que possibilitam sua manipulação em qualquer editor XML. Partindo deste princípio, o ConceptER 3.0 integrou menu Ferramentas do módulo de criação de modelos relacionais a opção de converter o arquivo XML que contém a modelo conceitual em um arquivo XML contendo um modelo lógico. As regras de conversão estão baseadas nos conceitos de mapeamento ER e EER para relacional definidos por Elmasri e Natathe (2011). Um exemplo de arquivo XML contendo um modelo conceitual salvo é apresentado na figura 62.

**Figura 62** – Arquivo XML com modelo conceitual salvo.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<savedObject>
  <cont_entidade>2</cont_entidade>
  <cont_relacionamento>1</cont_relacionamento>
  <cont_atributo>4</cont_atributo>
  <graph>&lt;mxGraphModel&gt;&lt;root&gt;&lt;mxCell id="0"/
  <mapaGraficoEntidades>
    <entry>
      <key>2</key>
      <value>
        <atributos uuid="4">
          <id>4</id>
          <nome>Atributo1</nome>
          <tamanhoAltura>25</tamanhoAltura>
          <tamanhoLargura>100</tamanhoLargura>
          <tipoAtributo>SIMPLES</tipoAtributo>
          <pX>170.0</pX>
          <pY>220.0</pY>
        </atributos>
        <atributos uuid="6">
          <id>6</id>
          <nome>Atributo2</nome>
          <tamanhoAltura>25</tamanhoAltura>
          <tamanhoLargura>100</tamanhoLargura>
          <tipoAtributo>CHAVE</tipoAtributo>
          <pX>170.0</pX>
          <pY>220.0</pY>
        </atributos>
        <id>2</id>
        <nome>Entidade1</nome>
        <tamanhoAltura>50</tamanhoAltura>
        <tamanhoLargura>100</tamanhoLargura>
        <tipo>FORTE</tipo>
        <pX>202.0</pX>
        <pY>208.0</pY>
      </value>
    </entry>
```

Fonte: Elaboração do autor (2019).

Após o processo de conversão do arquivo XML que contém a modelo conceitual em um arquivo XML contendo um modelo lógico, é possível acessar e editar o resultado da conversão através de qualquer editor XML. Um exemplo do resultado desta conversão é apresentado na figura 63.

**Figura 63** – Arquivo XML com modelo logico convertido.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<savedObjectLogico>
  <mapaGraficoRelacionamentos>
    <entry>
      <key>4</key>
      <value>
        <id>4</id>
        <nome>Relacionamento1</nome>
        <tabelas>
          <id>2</id>
          <nome>Entidade1</nome>
          <cardinalidade>MUITOS</cardinalidade>
          <participacao>OPCIONAL</participacao>
          <campos>
            <id>9</id>
            <nome>Atributo2</nome>
            <tipo>CHAVEPRIMARIA</tipo>
          </campos>
        </tabelas>
        <tabelas>
          <id>3</id>
          <nome>Entidade2</nome>
          <cardinalidade>MUITOS</cardinalidade>
          <participacao>OPCIONAL</participacao>
          <campos>
            <id>7</id>
            <nome>Atributo1</nome>
            <tipo>SIMPLES</tipo>
          </campos>
          <campos>
            <id>11</id>
            <nome>Atributo3</nome>
            <tipo>CHAVEPRIMARIA</tipo>
          </campos>
        </tabelas>
      </value>
    </entry>
  </mapaGraficoRelacionamentos>
</savedObjectLogico>
```

Fonte: Elaboração do autor (2019).

## 7 EXPERIMENTOS E RESULTADOS

A proposta deste trabalho foi a criação de um módulo gerador de modelos relacionais utilizando tecnologias como Java, JAXB e JGraphX. E também a criação de um módulo integrado que permita a conversão de um arquivo XML Conceitual para XML Lógico.

As dificuldades encontradas para evolução da ferramenta, com ênfase na criação de um módulo gerador de modelos lógicos e conversão entre arquivos XML do modelo conceitual para modelo lógico estiveram concentradas na curva de aprendizagem na tecnologia da API JGraphX, citada na seção 5.3 do capítulo 5.

A biblioteca java do JGraphX é bem extensa e são inúmeras as possibilidades de criação através dela. Porém, o manual é pobre e oferece apenas informações básicas de uso. A oferta de informações e materiais para pesquisa é bem escassa, e solução encontrada foi baseada em tentativa e erro.

Algumas classes da biblioteca do JGraphX foram extraídas e usadas para a criação de novas classes do projeto, incluído algumas modificações, como é o caso da classe `mxMarkerRegistry`, onde foram incluídos novos *registerMarkers* para sinalizar os tipos de cardinalidade e participação em cada ponta do gráfico, *edge*, que associa uma tabela a outra em uma relação, conforme apresenta a figura 64.

**Figura 64** – Fragmento do script incluído em uma nova classe `mxMarkerRegistry`.

```
registerMarker(mxConstants.ARROW_ZEROPARAUM, new mxIMarker() {
    public mxPoint paintMarker(mxGraphics2DCanvas canvas,
        mxCellState state, String type, mxPoint pe, double nx,
        double ny, double size, boolean source) {
        canvas.getGraphics().draw(new Line2D.Float((int) Math.round(pe.getX()-nx),
            (int) Math.round(pe.getY()-nx),
            (int) Math.round(pe.getX()-nx),
            (int) Math.round(pe.getY()+nx)));
        double cx = pe.getX() - nx * 2;
        double cy = pe.getY() - ny * 2;
        double a = size / 2;
        Shape shape = new Ellipse2D.Double(cx - a, cy - a, size, size);
        if (mxUtils.isTrue(state.getStyle(), (source) ? "startFill" : "endFill", true)) {
            canvas.fillShape(shape);
        }
        canvas.getGraphics().draw(shape);
        return new mxPoint(0, -ny);
    }
});

registerMarker(mxConstants.ARROW_UMPARAUM, new mxIMarker()
...

registerMarker(mxConstants.ARROW_ZEROPARAMUITOS, new mxIMarker()
...

registerMarker(mxConstants.ARROW_UMPARAMUITOS, new mxIMarker()
...
```

Fonte: Elaboração do autor (2019).

Ao realizar a integração do módulo gerador de modelo lógico ao ConceptER que já possui o módulo gerador de modelo conceitual, a nova classe baseada na classe `mxMarkerRegistry` não pode ser acessada pelas outras classes da biblioteca `JGraphX`, não permitindo a inserção modificação dos tipos de cardinalidade e participação que foram customizados. Porém as customizações podem ser realizadas a partir de uma ação disparada na classe `TabelaEstrutura`.

Os elementos gráficos que geram e modificam as tabelas possuem atributos e métodos não estáticos, de forma a garantir que cada tabela e relacionamento possua uma única instância. Para isso foi necessário criar ações em botões para realizar diversas operações nestes elementos não estáticos. Para acionar os botões de forma estática, foram criados métodos estáticos que acionam as ações destes botões.

O arquivo salvo que faz referência a um modelo logico criado na ferramenta possui estrutura de metadados XML. O arquivo é salvo com a extensão `.er3`, porem pode ser aberto em qualquer ferramenta de edição de arquivo XML.

A ferramenta realiza a conversão do XML baseada no modelo conceitual para um XML baseado em modelo lógico, salvando o resultado desta conversão em um arquivo. Porém a ferramenta ainda não é capaz de carregar um gráfico baseado neste arquivo convertido, esta funcionalidade foi atribuída a trabalhos futuros.

A exportação para o tipo de arquivo de imagem PNG não teve o resultado esperado, devido ao fato da integração entre o `JGraphx` e o `Canvas` do java garantir apenas o envio do gráfico interno do componente e não dos elementos Java Swing a ele associados, logo são gerados apenas o frame das tabelas, sem seu conteúdo que está inserido em uma `JTable`.

O ConceptER 3.0 foi validado em ambientes Windows e Linux com arquiteturas x86 e x64. O código fonte da ferramenta está disponibilizado no endereço <https://github.com/sergiorobertomao/ConceptER3.0>. Esta ferramenta foi publicada sob a licença GNU, permitindo a distribuição do código fonte.

## 8 CRONOGRAMA DE ATIVIDADES

Neste capítulo é apresentado o cronograma que foi definido para concretização dos objetivos do projeto.

**Quadro 4 – Cronograma de Atividades**

ATIVIDADES		2018					2019					
		ago	set	out	nov	dez	jan	fev	mar	abr	mai	jun
1	Definição do Tema											
2	Delimitação do Escopo											
3	Levantamento de Requisitos											
4	Análise de Trabalhos Relacionados											
5	Fundamentação Teórica											
6	Correlação de Tecnologias											
7	Proposta de Solução											
8	Apresentação para Banca (TCC I)											
9	Correção e ajustes na Documentação											
10	Implementação											
11	Testes práticos em sala de aula em turmas da disciplina de banco de dados.											
12	Apresentação para Banca (TCC II)											

Fonte: Elaboração do autor (2018).

O cronograma apresentado foi integralmente cumprido, ficando a apresentação para banca do TCCII como última etapa a ser executada.

## 9 CONSIDERAÇÕES FINAIS

O objetivo geral e os objetivos específicos definidos para este projeto foram cumpridos. Apesar das dificuldades e possibilidades de melhoria como a integração de outras funcionalidades, a ferramenta cumpre bem o seu papel e garante ao usuário uma excelente experiência.

O software ConceptER foi evoluído para a versão 3.0, e os módulos de geração de modelos conceitual e lógico foram integrados. Os recursos da nova interface ofertada nesta ferramenta, permitem a criação de modelos relacionais baseados no padrão gráfico utilizado em Elmasri e Navathe (2011). As estratégias de mapeamento do modelo conceitual para lógico, também conceituadas por Elmasri e Navathe (2011), foram implementadas em um módulo de conversão, que permite ao usuário gerar um arquivo XML com modelo relacional a partir de um arquivo XML com modelo conceitual criado no ConceptER. A técnica de inspeção de software através de checklist, garantiu a correta verificação dos artefatos gerados durante o desenvolvimento desta ferramenta, minimizando e corrigindo falhas.

A fundamentação teórica levantada foi de grande importância para determinar o conjunto de conceitos e conversões que estão agregadas ao ConceptER 3.0. Todas as tecnologias abordadas no capítulo 4 foram utilizadas e determinantes para conclusão do projeto. O planejamento deste projeto foi baseado no cronograma de atividades apresentado no capítulo 7. A fase definida para a implementação contemplou as etapas de testes de forma gradativa, onde as funcionalidades atenderam positivamente aos cenários submetidos.

A principal contribuição desta ferramenta é a garantia de uma visão adequada entre os níveis de modelos de dados conceitual e lógico, que apoiará estudantes e profissionais na concepção e análise de projetos de banco de dados.

### 9.1 Trabalhos Futuros

Diversos trabalhos futuros podem ser elaborados a partir do estado atual do ConceptER, como: melhoramento da interface de edição; adição de novos componentes gráficos; carregamento do modelo relacional gerado pela conversão do modelo conceitual para lógico; melhoramento do módulo de conversão do modelo conceitual para o lógico, adicionando mais regras; criação de um módulo de conversão de modelo lógico para físico; módulo de criação para modelo físico; administração de banco de dados; depuração de scripts sql; exportar imagem do modelo lógico criado; editar mais de um modelo no mesmo executável com

navegação em abas e até mesmo uma versão de avaliação de alunos, onde o aluno cria os gráficos livremente sem quaisquer impedimento baseado em regras de modelagem e depois a ferramenta gera um feedback informando os erros.

O ConceptER é um software livre e possibilita que qualquer pessoa tenha acesso e modifique seu código fonte, desde que mantenha as novas contribuições também livres para alterações e incorporações em novos projetos, possibilitando novas evoluções e modificações de forma a atender a necessidades profissionais e acadêmicas.



## REFERÊNCIAS BIBLIOGRÁFICAS

1KEYDATA. **Data Modeling - Conceptual, Logical, And Physical Data Models**. Disponível em: <<https://www.1keydata.com/datawarehousing/data-modeling-levels.html>>. Acesso em 8 de setembro de 2018.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 10520: informação e documentação – Citações em documentos – Apresentação**. Rio de Janeiro, 2002.

ASTAH. **Astah Professional**. Disponível em: <<http://astah.net/editions/professional>>. Acesso em 20 de novembro de 2018.

ASTAH BLOG. **How to add ER Data Type**. Disponível em: <<https://astahblog.com/2013/08/16/how-to-add-er-data-type/>>. Acesso em 20 de novembro de 2018.

CHEN, P. The Entity-Relationship Model: Toward a Unified View of Data. EUA: ACM Transactions on DataBase Systems, v. 1, n. 1, pp. 9-36, 1976.

COUGO, Paulo. Modelagem conceitual e projeto de banco de dados. Elsevier Brasil, 2013.

DAUM, Berthold. **Modelagem de objetos de negócios com XML: Abordagem com base em XML schema**. Tradução de Edson Furmankiewicz. Rio de Janeiro: Elsevier, 2014.

DEITEL, H.M. **XML, como programar**. Tradução de Luiz Augusto Salgado e Edson Furmankiewicz. Porto Alegre: Bookman, 2003.

DEVMEDIA. **Artigo Engenharia de Software 3 - Requisitos Não Funcionais**. Disponível em: <<https://www.devmedia.com.br/artigo-engenharia-de-software-3-requisitos-nao-funcionais/9525>>. Acesso em 01 de junho de 2019.

DEVMEDIA. **Java Swing: Conheça os componentes JTextField e JFormattedTextField.** Disponível em: <<https://www.devmedia.com.br/java-swing-conheca-os-componentes-jtextfield-e-jformattedtextfield/30981>>. Acesso em 11 de outubro de 2018.

DEVMEDIA. **O que é UML e Diagramas de Caso de Uso: Introdução Prática à UML.** Disponível em: <<https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>>. Acesso em 16 de outubro de 2018.

DEVMEDIA. **Orientações básicas na elaboração de um diagrama de classes.** Disponível em: <<https://www.devmedia.com.br/orientacoes-basicas-na-elaboracao-de-um-diagrama-de-classes/37224>>. Acesso em 01 de junho de 2019.

DPI-UFV. **Ferramentas CASE.** Departamento de Informática. UFV. GeopProfile Disponível em: <[http://www.dpi.ufv.br/projetos/geoprofile/ferramentas\\_case.html](http://www.dpi.ufv.br/projetos/geoprofile/ferramentas_case.html)>. Acesso em 11 de outubro de 2018.

ELMASRI, Ramez; NAVATHE, Shamkant B.. **Sistemas de banco de dados.** 6ª ed. São Paulo: Pearson Addison Wesley, 2011.

FURGERI, Sérgio. **Java 8 – Ensino didático: desenvolvimento e implementação de aplicações.** São Paulo. Érica, 2015.

GUEDES, Gilleanes T. A. **UML 2: Uma abordagem prática.** São Paulo. Novatec Editora LTDA, 2011.

JACOBSON. **Object Oriented Software Engineering: A Use Case Driven Approach.** Addison Wesley, 1992.

JGRAFH. **JGraphX (JGraph 6) User Manual.** JGraphX Version 3.9.11 – 06. Disponível em: <[https://jgraph.github.io/mxgraph/docs/manual\\_javavis.html](https://jgraph.github.io/mxgraph/docs/manual_javavis.html)>. Acesso em 10 de Outubro de 2018.

HEUSER, Carlos Alberto. **Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS**. Bookman Editora, 2009.

LARMAN, Craig. **Utilizando UML e padrões: Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo**. Tradução de Rosana Vaccare Braga. 3ª ed. Porto Alegre: Bookman, 2007.

LUCIDCHART. **O que é um diagrama de classe UML?**. Disponível em: <<https://www.lucidchart.com/pages/pt/o-que-e-diagrama-de-classe-uml>>. Acesso em 10 de fevereiro de 2019.

MACHADO, Felipe Nery Rodrigues. **Banco de Dados - Projeto e Implementação**. 3ª ed. São Paulo: Érica, 2014.

MICROSOFT. **SQL Server 2017. Tabelas**. Disponível em: <<https://docs.microsoft.com/pt-br/sql/relational-databases/tables/tables?view=sql-server-2017>>. Acesso em: 28 de outubro de 2018.

NASCIMENTO. F.S. **Evolução da Ferramenta ConceptER**. Trabalho de Conclusão de Curso. Manaus: IFAM, 2015.

ORACLE. **Java – Documentação. Os tutoriais Java™. Lição: Introdução ao JAXB**. Disponível em: <<https://docs.oracle.com/javase/tutorial/jaxb/intro/>>. Acesso em: 05 de outubro de 2018.

PFLEEGER, S.L., **Engenharia de Software: Teoria e Prática**. 2ª Ed. São Paulo: Prentice Hall, 2004.

SAP. **SAP Power Designer: Data Architecture**. SAP Help Portal. Disponível em: <<https://help.sap.com/viewer/856348b84a7c479489d5172a630f014d/16.6.4/en-US>>. Acesso em: 09 de Setembro de 2018.

SCHROEDER, R.; MELLO, R. d. S. **Uma abordagem para projeto lógico de banco de dados xml baseada em informações de carga de dados**. Dissertação de mestrado. Universidade Federal de Santa Catarina, 2008.

SILBERSCHATZ, Abraham; SUNDARSHAN, S.; KORTH, Henry F. **Sistema de banco de dados**. Elsevier Brasil, 2016.

SOMMERVILLE, I., **Engenharia de Software**. 8ª Ed. São Paulo: Pearson – Addison Wesley, 2007.

VISUAL PARADIGM. **Design Database with Professional ERD Software**. Disponível em: <<https://www.visual-paradigm.com/features/database-design-with-erd-tools>>. Acesso em 11 de Setembro de 2018.

W3C. **Extensible Markup Language (XML)**. Disponível em: <<https://www.w3.org/XML>>. Acesso em 09 de outubro de 2018.

## APÊNDICE – DESCRIÇÃO DOS CASOS DE USO

A seguir são apresentados os quadros referentes a descrição dos casos de uso elaborados para este projeto.

**Quadro 5** – Descrição do caso de uso para criar um novo modelo.

<b>ID</b>	UC01
<b>Caso de Uso</b>	Criar novo modelo
<b>Ator(es)</b>	Usuário
<b>Descrição</b>	Usuário deseja criar um modelo a partir de um documento novo.
<b>Pré-condições</b>	N/A
<b>Fluxo Principal</b>	1. Usuário clica na opção Arquivo que está no menu principal e escolhe a opção Novo Modelo. 2. O sistema exibe na interface um novo documento, em branco, com os componentes referentes ao modelo lógico.
<b>Fluxo Alternativo</b>	<b>A. Uso do menu rápido de ações</b> 1. Usuário clica no ícone Novo Modelo no menu rápido de ações.
<b>Fluxo de Exceção</b>	N/A

Fonte: Elaboração do autor (2018).

**Quadro 6** – Descrição do caso de uso para abrir um modelo salvo.

<b>ID</b>	UC02
<b>Caso de Uso</b>	Abrir modelo salvo
<b>Ator(es)</b>	Usuário
<b>Descrição</b>	Usuário deseja abrir um modelo a partir de um documento salvo.
<b>Pré-condições</b>	Possuir um modelo salvo.
<b>Fluxo Principal</b>	1. Usuário clica na opção Arquivo que está no menu principal e escolhe a opção Abrir Modelo. 2. Usuário procura e seleciona o arquivo desejado. 3. O sistema exibe na interface o documento referente ao arquivo selecionado, contendo as informações que foram previamente gravadas.
<b>Fluxo Alternativo</b>	N/A

<b>Fluxo de Exceção</b>	<b>A3. Arquivo com formato não compatível</b> 3. Exibe a mensagem: “ Formato de arquivo não suportado. ”  <b>B3. Arquivo com estrutura incompatível</b> 3. Exibe a mensagem: “ Arquivo corrompido. ”
-------------------------	--

Fonte: Elaboração do autor (2018).

**Quadro 7** – Descrição do caso de uso para editar um modelo.

<b>ID</b>	UC03
<b>Caso de Uso</b>	Editar modelo
<b>Ator(es)</b>	Usuário
<b>Descrição</b>	Usuário deseja editar um modelo.
<b>Pré-condições</b>	UC1 ou UC2.
<b>Fluxo Principal</b>	1. Usuário adiciona, remove ou move os componentes no documento.
<b>Fluxo Alternativo</b>	N/A
<b>Fluxo de Exceção</b>	N/A

Fonte: Elaboração do autor(2018).

**Quadro 8** – Descrição do caso de uso para salvar um modelo.

<b>ID</b>	UC04
<b>Caso de Uso</b>	Salvar modelo
<b>Ator(es)</b>	Usuário
<b>Descrição</b>	Usuário deseja salvar um modelo editado.
<b>Pré-condições</b>	UC03
<b>Fluxo Principal</b>	1. Usuário clica na opção Arquivo que está no menu principal e escolhe a opção Salvar. 2. O sistema grava o arquivo, com as informações da última alteração, no disco.
<b>Fluxo Alternativo</b>	<b>A1. Uso do menu rápido de ações</b> 1. Usuário clica no ícone Salvar no menu rápido de ações.  <b>A2. Arquivo originado a partir de um novo documento, e que ainda não foi salvo.</b> 2. Sistema exibe uma tela para escolher o caminho e nomear o arquivo.

	2.1 O sistema grava o arquivo, com as informações da última alteração, no disco.
<b>Fluxo de Exceção</b>	N/A

Fonte: Elaboração do autor (2018).