INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, *CAMPUS* PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

JOÃO LUCAS GOERGEN

CSDC – UMA FERRAMENTA DE CONVERSÃO DE SCRIPT SQL EM DIAGRAMA DE CLASSES UML

Alexandre Tagliari Lazzaretti

PASSO FUNDO, 2012

JOÃO LUCAS GOERGEN

CSDC – UMA FERRAMENTA DE CONVERSÃO DE SCRIPT SQL EM DIAGRAMA DE CLASSES UML

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, *Campus* Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Alexandre Tagliari Lazzaretti

PASSO FUNDO, 2012

JOÃO LUCAS GOERGEN

CSDC - UMA FERRAMENTA DE CONVERSÃO DE SCRIPT SQL EM DIAGRAMA DE CLASSES UML

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Prof. Me. Alexandre Tagliari Lazzaretti

_____

Prof. Me. Evandro Miguel Kuszera

_____

Prof. Esp. Carmen Vera Scorsatto

_____

Evandro Miguel Kuszera

(Coordenador do Curso)

PASSO FUNDO, 2012

Ao único Deus,
que além de sempre estar comigo
permitiu sempre o apoio de minha família,
em especial minha mãe e meu irmão.

AGRADECIMENTOS

Agradeço a toda igreja do Senhor Jesus, em especial à congregação Batista Independente a qual congrego e me sinto feliz pelas orações destinadas a mim, que fizeram a diferença na realização deste trabalho, me mantiveram calmo e com a certeza que tudo daria certo; à minha mãe, Flávia Luisa Goergen, por me apoiar sempre nas dificuldades encontradas no decorrer desta caminhada; ao meu irmão José Anderson de Oliveira, que, mesmo não cooperando diretamente, é um grande exemplo de vida para mim; ao meu pai Édson Angelo Brito Bitscki, que, pelo caminho que a vida fez não pode me aconselhar neste trabalho, mas sem o que aprendi com ele jamais eu teria motivação para iniciar; à minha noiva Liliana Raphaela da Silva Brandim, pelo amor, pela fé, pelas orações, mesmo com tanta distância muito me auxiliou e sempre fortaleceu meu coração; ao meu orientador Alexandre Tagliari Lazzaretti, por, além do ensinamento como professor, ter me mostrado como ser uma boa pessoa, a manter os prazos, a seguir e insistir nos objetivos, mesmo nas adversidades; aos meus colegas, que por mais de dois anos me "aturaram". Ensinaram-me sempre com opiniões sinceras em todos os trabalhos, inclusive este; ao amigo Paulo, que por muitos momentos me ajudou de diversas maneiras e, sem tal ajuda, eu não poderia ficar tranquilo para estudar; ao CEUS de minha querida cidade Soledade, por conceder-me a bolsa integral de transporte, permitindo a tranqüilidade em todas as etapas desta caminhada; ao meu orientador de estágio Bibiano Filho, que além de demonstrar a realidade de uma empresa, agregou-me valores que foram utilizados para a construção deste trabalho; ao IFSUL, pela oportunidade de realizar pesquisas com a bolsa de iniciação científica. Por fim, a todos que pertencem à minha vida. Muito obrigado!

“Ninguém planeja fracassar,
mas fracassa por não planejar”.
(Jim Rohn)

RESUMO

Este trabalho apresenta uma ferramenta que realiza, utilizando engenharia reversa, a criação de um Diagrama de Classes baseado em comandos DDL contidos em um script SQL. São apresentadas as abstrações e transformações, além das comparações e criações de objetos Java juntamente com outras tecnologias envolvidas, demonstrando assim o processo de engenharia reversa. A principal contribuição deste trabalho baseia-se na apresentação do processo de mapeamento e engenharia reversa utilizada pela ferramenta, além da utilidade que o uso desta oferece, pois pode contribuir em termos de planejamento para sistemas informatizados em geral.

Palavras-chave: SQL; UML; Diagrama de Classes; Engenharia Reversa; Conversão;

ABSTRACT

This paper presents a tool that performs, using reverse engineering, the creation a class diagram based on commands DDL contained in a SQL script. Are presented the abstractions and transformations, comparisons and creations of Java objects along with other technologies involved, thus demonstrating the process of reverse engineering. The main contribution of this work is based on the presentation of the process mapping and reverse engineering used by tool and the utility of using this tool has, because it can contribute in terms of planning for information systems in general.

Keywords: SQL, UML, Class Diagram, reverse engineering; Conversion;

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

O Desenvolvimento de sistemas informatizados é uma atividade extremamente complexa que exige planejamento. Empresas que desenvolvem softwares e sistemas em geral procuram adotar padrões, métodos, tecnologias ou simplesmente seguem seus passos[1] para produzir produtos que atendam às necessidades de seus clientes.

O planejamento para a conclusão de um sistema informatizado ocorre independente do ciclo de desenvolvimento de uma empresa; todavia isso não é uma tarefa simples, merecendo cuidado na execução. Na maioria dos casos, para cada projeto é gerado uma espécie de documentação, seja com arquivos de texto ou diagramas e, estes podem representar o sistema exatamente como será, ou o mais próximo da realidade, como ressalta Braude (2005, p.28), "um projeto é produzido a partir de requisitos; Ele exclui o código. Uma notação extremamente útil para documentação de projeto é a Linguagem Unificada de Modelagem (UML - *Unified Modeling Language*)".

Portanto, "um projeto consiste em um conjunto de documentos." (BRAUDE, 2005, p.28). Sem requisitos não há projeto, porquanto, do simples ao mais complexo, o planejamento faz-se necessário. Assim, planejar é a única forma de realizar, com menos dificuldades, um software (BRAUDE, 2005, p.24).

Para qualquer empresa, perda de tempo no desenvolvimento é sinônimo de perda de lucros, por isso a importância de preservar cada minuto deste processo. Mesmo sem saber exatamente quando será a conclusão de um sistema, a projeção evita a plena certeza de estar desperdiçando tempo na criação, e, em relação à construção, assim explica Guedes:

> A estimativa de tempo é realmente um tópico extremamente complexo da engenharia de software. Na realidade, por melhor modelado que um sistema tenha sido, ainda assim fica difícil determinar com exatidão os prazos finais de entrega de um software. Uma boa modelagem auxilia a estimar a complexidade de desenvolvimento de um sistema, e isso, por sua vez, ajuda – e muito – a determinar o prazo final em que o software será entregue. (2011, p. 26).

Além do tempo, Yourdon (1990, p. 836) apresenta como sendo os principais problemas no desenvolvimento de sistemas: a produtividade, a confiabilidade e a manutenibilidade, ou seja, sabendo desses problemas, a construção completa de um sistema,

[1] Sequência de execução em partes e de forma organizada, predefinida e que sirva para concluir um objetivo específico.

em um período de tempo adequado a todos, torna-se possível somente com um bom planejamento.

A ferramenta tem por objetivo diminuir o tempo na hora de entender o funcionamento de um banco de dados, auxiliar na correção de um projeto de software, e poderá contribuir para aprendizagem de interessados em relação ao domínio trabalhado.

Embora existam diversas ferramentas com o funcionamento de engenharia reversa, há falta de softwares de acesso público e gratuito na Internet, que sejam de fácil manuseio e permitam realizar o processo de engenharia reversa de um banco relacional para um modelo conceitual. Assim sendo, a facilidade de entendimento de um sistema existente é comprometida, caso o mesmo não tenha sido bem documentado; uma possível criação de um novo sistema baseado em um banco de dados já existente pode-se tornar demorada e trabalhosa; o processo de remodelagem de um sistema que não atende às necessidades de uma determinada área torna-se complexo, e ainda, o controle e a geração de documentação podem até mesmo ser inexistentes. Considerando tais fatores, a principal motivação para realização deste trabalho baseou-se na criação de tal ferramenta, com o propósito de minimizar os problemas acima descritos.

Visando a conversão de um script SQL de um banco de dados relacional para um diagrama de classes, e ainda possibilitando a modularização das classes geradas, ou seja, uma organização por grupos/setores pré-definidos pelo usuário, e levando em conta a problemática acima descrita, a ferramenta desenvolvida torna-se essencial nesse contexto.

Assim, o presente estudo demonstrará a importância de uma ferramenta que sirva especificamente para auxiliar o entendimento de analistas/programadores na hora de criar, remodelar, corrigir ou verificar a estrutura de um sistema através da maneira com que o banco de dados está definido, após o levantamento de requisitos, parte essencial de um projeto. A ferramenta foi denominada CSDC (acrônimo de "**C**onversão de script **S**QL para um **D**iagrama de **C**lasses") e possui os seguintes objetivos:

Objetivo Geral

Realizar o processo de engenharia reversa de um script SQL para um diagrama de classes UML.

Objetivos Específicos

- Facilitar o entendimento de um sistema/software existente que se deseja analisar;

- Criar diagramas de classes através de regras de mapeamento em script SQL de um banco relacional, promovendo assim, a facilidade na criação de um novo sistema;
- Auxiliar no processo de remodelagem de um sistema que não atende às necessidades;
- Possibilitar a modularização (pacotes) do diagrama de classes com o uso da ferramenta;
- Estudar tecnologias de desenvolvimento.

Este trabalho foi organizado em capítulos: No capítulo 2 serão apresentados os conceitos relacionados ao trabalho: a linguagem UML, o banco de dados relacional, a linguagem Java e XML, a engenharia reversa, além de apresentar algumas ferramentas relacionadas.

O capítulo 3 descreverá detalhadamente todo o processo realizado pela ferramenta: Suas etapas, entradas e saídas, funcionalidades, mapeamento e tecnologias envolvidas no seu desenvolvimento, vinculado a um estudo de caso que ajuda esclarecer cada seção apresentada.

O capítulo 4 apresentará as considerações referentes ao desenvolvimento da ferramenta e de todo processo que esta realiza.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentadas as informações[2] necessárias para se obter o entendimento do funcionamento da aplicação que este trabalho propõe.

2.1 UML

Linguagem de Modelagem Unificada é a tradução de UML (Unified Modeling Language), que nada mais é do que uma linguagem de modelagem independente, ou seja, não está vinculada a nenhuma linguagem de programação ou ainda qualquer processo de desenvolvimento, que, de forma geral, procura representar e demonstrar em forma de diagramas o funcionamento de todo sistema, de parte de sistema ou até mesmo um processo específico no sistema. (GUEDES, 2011).

2.1.1 Representação UML

A UML tem por objetivo mostrar as características estruturais e comportamentais de um software para que, em sua construção, possam ser minimizados os imprevistos e erros que atrasam todo processo de desenvolvimento. Assim ressalta Guedes:

> Deve ficar bem claro, porém, que a UML não é uma linguagem de programação, e sim uma linguagem de modelagem, uma notação, cujo objetivo é auxiliar os engenheiros de software a definirem as características do sistema, tais como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado. (2011, p.19).

Para a indústria de software internacional, a Linguagem de Modelagem Unificada tornou-se um padrão em todos os modelos de aplicação, pois como a UML é baseada no paradigma de orientação a objetos e as principais empresas no ramo de desenvolvimento de sistemas usam esse mesmo "método" de desenvolvimento, a forma mais adequada de realizar as modelagens de seus sistemas é usando os diagramas UML. (GUEDES, 2011).

Imaginando uma bola a qual algumas pessoas estejam olhando de diferentes ângulos, cada diagrama UML pode ser comparado à visão dessas pessoas. Cada pessoa terá uma visão diferente da bola e uma descrição diferente que, se unida corretamente com a descrição das

[2] Estas informações referem-se, em síntese, às especificações de linguagens, padrões, e tecnologias relacionadas com a ferramenta.

outras pessoas, dará uma definição mais confiável, e com mais detalhes da bola, que simplesmente a visão e definição de uma pessoa. Usando essa analogia, a UML é dividida em vários diagramas, justamente para proporcionar múltiplas visões do sistema que será projetado. Visões estas, que acabam interligadas, fornecendo maior consistência do sistema, pois um engenheiro pode facilmente identificar e corrigir falhas que só são descobertas com uso desses diagramas ou com uma visão aprofundada do que está sendo projetado. Guedes discorre sobre o assunto com a seguinte questão:

> Por que a UML é composta por tantos diagramas? O objetivo disso é fornecer múltiplas visões do sistema a ser modelado, analisando-o e modelando-o sob diversos aspectos, procurando-se, assim, atingir a completitude da modelagem, permitindo que cada diagrama complemente os outros. (2011, p.30).

O autor apresenta os tipos de diagramas, sendo eles: diagrama de casos de uso, de classes, de objetos, de pacotes, de sequência, de comunicação, de máquina de estados, de atividade, de visão geral de interação, de componentes, de implantação, de estrutura composta e diagrama de tempo ou temporização. São os treze diagramas disponíveis pela UML, sendo divididos em diagramas estruturais e diagramas comportamentais. De forma geral, diagramas estruturais demonstram toda estrutura, tanto em ligações, dependências, relações; enfim, a forma de comunicação entre classes, objetos ou processos. Esses diagramas visam "o quê fazer" no sistema, diferentemente dos diagramas comportamentais, que focam no comportamento, em "como fazer" uma determinada tarefa, ou como realizar com sucesso certo procedimento. (GUEDES, 2011, p.30 à p.41). Na Figura 1, tem-se um exemplo de um diagrama UML.

Figura 1 - Exemplo de uma classe de um diagrama de classes

Fonte: GUEDES, 2011, p.107

2.1.2 Diagrama de Classes

Com o foco voltado em fornecer as informações das classes em um sistema e como elas comunicam-se entre si, de forma que apresente verdadeiramente a organização entre as

mesmas, o diagrama de classes torna-se um dos mais importantes dentre os demais diagramas UML, como já dito anteriormente.

Sabendo que o diagrama de classes faz a representação de um sistema orientado a objetos e que a ferramenta proposta irá gerar um diagrama de classes, pode-se entender, precipitadamente, que a ferramenta servirá apenas para projetos em que se pretende usar orientação a objetos, o quê não é verdade. No contexto de banco de dados relacionais, o diagrama de classes está associado a um nível de modelo conceitual, ou seja, serve para representar a forma que um sistema poderá ser construído, baseado em um banco de dados, mesmo que o projeto não possua o paradigma de orientação a objetos.

A seguir, os componentes de um diagrama de classes serão apresentados com mais detalhes, visando demonstrar o sentido da apresentação das características de um diagrama de classes para este trabalho. Cabe destacar que não são todas as relações existentes em um diagrama de classes que serão apresentadas nas próximas sessões, pois existem aquelas que não podem ser mapeadas[3] num primeiro momento, pois poderiam gerar diagramas que não representariam a realidade do sistema.

Assim, os componentes e relações de um diagrama de classes que a ferramenta deverá gerar, serão baseados seguindo as explicações retiradas do livro *UML 2*: Uma abordagem prática, de Gillenaes T. A. Guedes, seguindo confiabilidade na compreensão das saídas da ferramenta e conceitos abordados.

2.1.3 Classes

Classes possuem normalmente atributos e métodos que são as características e as funções que certo objeto instanciado deverá conter, a fim de utilizar quando necessário. Instanciar um objeto nada mais é do que, a partir de uma classe, criar um novo objeto, como por exemplo, a classe "pessoa", ela é apenas uma representação do que uma pessoa deve conter (seus atributos e métodos), agora, João, Maria e Pedro são exemplos de pessoas que pertencem à classe "pessoa", pois possuem os atributos da classe, e eles, por sua vez, foram instanciados, ou seja, foram criados com base em sua classe. O diagrama de classes não se preocupa em demonstrar instâncias, mas sim classes em si. Esta explicação foi dada apenas para que ficasse mais clara a diferença entre classe e objeto.

[3] Neste contexto, mapeamento está ligado à forma em que as tabelas do banco de dados serão representadas através de classes em um diagrama de classes UML.

Uma classe é representada em um diagrama de classes por um retângulo, o qual pode ser dividido com linhas horizontais nas definições de seus atributos e métodos, mas não necessariamente todas as classes terão essas divisões (Figura 2), muito embora essas divisões sejam vistas em quase todas as classes desse tipo de diagrama.

A forma com que uma classe é apresentada em um diagrama de classes UML está sendo informada desde já, pois a principal função da ferramenta proposta (CSDC) é gerar (de forma que se possa visualizar) as classes que podem ser formadas de um script SQL.

Figura 2 - Exemplo de uma classe abstrata 2

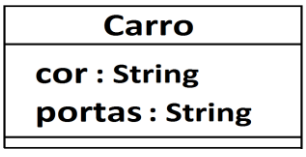
Fonte: GUEDES, 2011, p.45

2.1.4 Atributos em uma classe

Os atributos existem nas classes para representarem as "características" das mesmas, representam as "variáveis" nas linguagens de programação e por isso necessitam de um nome único e de um tipo de dado (cadeia de caracteres, inteiros, depende da sintaxe da linguagem). São declarados em um diagrama de classe com o nome desejado e separado por dois pontos o seu tipo (Figura 3).

"Os valores dos atributos podem variar", mostra Guedes (2011), para cada instância criada, com isso os objetos criados são diferenciados uns dos outros, pois imagine se todas as pessoas do mundo tivessem o mesmo peso, ou tamanho, ou ainda fossem do mesmo sexo? Justamente a variação dos valores dos atributos é o que "origina" objetos distintos.

Figura 3 – Declaração de atributos em uma classe de um diagrama de classes

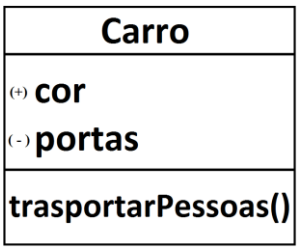


Fonte: Adaptado de GUEDES, 2011, p.47

2.1.5 Métodos

Não há muito o que destacar sobre métodos, pois, mesmo existentes em um diagrama de classes, não serão formados pela ferramenta, pois embora eles sejam necessários e estejam presentes, não é de fato o diagrama de classes que possui maior foco nos métodos, ou melhor, nas funções que eles farão. A Figura 4 demonstra a declaração de um método no diagrama de classes. No diagrama de classes eles são representados inicialmente, sem se preocupar de que forma ou em que momento o sistema fará uso dos mesmos, ou ainda se há tanta relevância no seu uso.

Figura 4 – Declaração de método em uma classe de um diagrama de classes



Fonte: Adaptado de GUEDES, 2011, p.47

2.1.6 Multiplicidade, navegabilidade e visibilidade

As relações em um diagrama de classes, não existem simplesmente para fazer ligações descoordenadas entre as classes que, por fim poderiam facilitar a visão do sistema, mas sim, são ligações específicas que visam controlar o nível de dependência dos objetos na relação. Para demonstrar a quantidade mínima e máxima dos objetos relacionados, usam-se duas formas de organização nas relações, a multiplicidade e navegabilidade. Destas duas formas de organização, é importante expor a visibilidade, que é responsável por demonstrar e caracterizar um método ou atributo quanto à forma de manipulação direta aos mesmos.

2.1.6.1 Conceito de multiplicidade

Antes de citar as relações existentes, é importante mencionar um fator existente nas relações em um diagrama de classes, que é chamado de multiplicidade. A multiplicidade procura determinar em uma relação qual a quantidade de objetos envolvidos, demonstrando com números, a quantidade mínima e máxima dos mesmos, sendo que se o número máximo for indeterminado usa-se o caractere “*” como forma de representar a expressão “muitos”. No

caso da falta dessa representação entende-se que a multiplicidade é “1..1”, ou seja, um e apenas um objeto de um lado, se relaciona com objetos do outro lado da relação.

As possíveis combinações de multiplicidade para um diagrama de classes são “0..1”, “1..1”, “0..*”, “*”, “1..*” e ainda “número inicial..número final”, sendo esse “número inicial” pode ser qualquer número e o “número final” qualquer número maior que o número inicial. Essas representações indicam o número mínimo e máximo de objetos, que se relacionam em uma associação (relação).

2.1.6.2 Conceito de navegabilidade

Em uma relação de classes, a navegabilidade indica a direção em que as informações são enviadas entre as classes, ou seja, o sentido que os métodos das classes serão disparados. A representação é simples, com uma seta na extremidade da linha da relação entre as classes. A ausência das setas de navegabilidade, é comum em um diagrama de classes, indica que ambos os lados da associação, poderão enviar e receber execuções de métodos dos objetos, das classes que estão se relacionando.

2.1.6.3 Conceito de visibilidade

Embora não esteja ligada a uma relação, a visibilidade, nada mais é do que a forma que um atributo da classe será tratado, quando inserido em um sistema, ou seja, será público (podendo receber atribuições diretas) ou privado (só poderá ser manipulado através de métodos. O símbolo de “(+)” apresentado na Figura 3, demonstra que o atributo “cor” é público, já o “(-)” do atributo “portas” especifica que ele é privado. A visibilidade é também aplicada nos métodos e, para a ferramenta proposta servirá apenas para os atributos, pois, como visto, os métodos não serão declarados no diagrama de classes gerado.

2.1.7 Relações das classes

Relações são necessárias para que o entendimento de um sistema, representado em um diagrama de classes, se torne mais claro. Uma relação é representada por linhas que ligam as classes envolvidas, essas linhas descreverão como será o vínculo de uma classe com outra (ou com outras). Como se pode perceber, este trabalho está sendo conduzido de maneira que conceitue o máximo das características dos itens, que serão usados para organizar e

demonstrar o funcionamento da ferramenta. As relações existentes em um diagrama de classes são: Associação Unária, Associação Binária, Associação Ternária ou N-ária, Agregação, Composição, Generalização, Classe Associativa, Dependência e Realização.

2.1.7.1 Associação unária ou reflexiva

Basicamente este relacionamento parte de uma classe e aponta para a mesma classe de onde partiu, ou seja, esta associação demonstra que haverá relacionamentos de objetos que partiram da mesma classe. Embora seja um dos relacionamentos mais simples de demonstrar, não é tão usado como uma associação binária ou n-ária. É um exemplo de associação unária, uma classe de nome “Funcionario” (que representa um padrão de características que um funcionário deverá ter para existir no sistema) com a seta chamada “chefia” apontando para a classe de onde partiu, ou seja, a classe “Funcionario”, pois isso demonstra que um funcionário pode ser chefe de outros funcionários, além disso, um chefe é um funcionário da empresa e por isso é uma instância da classe funcionário. Neste exemplo, essa associação reflexiva determina se um funcionário poderá ou não chefiar outros funcionários. Normalmente, em uma modelagem de diagrama de classes, esse tipo de associação representa algo como no exemplo dado acima.

2.1.7.2 Associação binária

A relação que provavelmente mais será formada pela ferramenta é esta, pois esse tipo de associação é a uma das mais comuns que existe num diagrama de classes, ocorre quando uma classe se relaciona com outra classe. Vale lembrar que em um diagrama de classes, uma classe não se relaciona de fato com outra classe, o que acontece é um “padrão” de relacionamento, que somente será válido para os objetos da classe, ou seja, uma classe é apenas um padrão de como um objeto deve ser para pertencer a ela, então quando se usa o termo “relacionamento entre classes”, isso significa na verdade “relacionamento entre os objetos que foram criados a partir de uma classe”, ou ainda “relacionamento entre instâncias de uma classe”, pois uma classe propriamente dita, não é usada em si em um sistema, o que é usado em um sistema são os objetos criados a partir de uma classe, existente no sistema. Voltando-se para o funcionamento da ferramenta, revisar ou destacar a importância das instâncias de uma classe torna-se desnecessário, pois instâncias são criadas por código de alguma linguagem de programação, e não são representadas no diagramas de classe em si.

2.1.7.3 Associação ternária ou N-ária

A associação ternária ou n-ária ocorre quando três ou mais classes estão relacionadas. Com um losango entre as classes e setas interligando-as assim é a representação desse tipo de associação. Bem como as outras relações, a multiplicidade e navegabilidade são introduzidas nas setas das relações.

A associação ternária é uma associação que demonstra complexidade nas relações das classes e por isso, embora úteis, o uso delas deve ser evitado, pois a leitura dessas relações pode ser difícil de ser interpretada. Assim, conclui-se que dificilmente uma relação desse tipo aparecerá como resultado, mesmo que planejada essa relação pode confundir um analista.

2.1.7.4 Agregação

Semelhantemente à binária, a associação de agregação existe quando uma classe agrega valores que outra classe utilizará, ou melhor, quando as informações de um objeto podem ser complementadas pelas informações de outros objetos (ou apenas outro), de alguma classe diferente. O objeto que será complementado recebe o nome de objeto-todo, enquanto o objeto da classe que complementa esse objeto-todo recebe o nome de objeto-parte. Para representar uma associação de agregação, usa-se o losango assim como a n-ária, porém este losango ficará na extremidade da classe que será complementada (do objeto-todo). Cada vez que for criado um objeto-todo, algumas informações podem ser complementadas por objetos-parte (no exemplo). Em muitos casos, a associação binária simples pode substituir a associação de agregação, pois a principal diferença entre elas é a obrigatoriedade, uma relação binária especifica que um objeto, pode ou não complementar outro que está relacionado, diferentemente da agregação, aonde demonstra uma possível complementação obrigatória do objeto-todo com os objetos-parte (ou objeto parte).

2.1.7.5 Composição

Um relacionamento do tipo composição é uma variação da agregação, na qual a principal diferença está no vínculo dos objetos-todo e os objetos-parte. Essa relação demonstra que os objetos-parte devem estar associados a um único objeto-todo (como o exemplo). A diferença entre o símbolo da agregação e o da composição é que o uso do losango com cor preenchida na composição. Em síntese, uma composição existirá sempre que um a classe deve estar associada à outra em sua representação no diagrama de classes.

2.1.7.6 Generalização/especialização

Usada para demonstrar a ocorrência de herança entre as classes, a generalização é normalmente usada quando existem classes com atributos muito semelhantes, visando representar isso para que não haja replicação de código no momento de programar, essa relação existe. O símbolo de uma especialização é uma seta fechada na extremidade da relação, sendo essa seta apontada da classe filha (classe que está herdando) para a classe pai (classe que contém os métodos e atributos a serem herdados). No uso de uma especialização, o uso da multiplicidade não é necessário, pois a especialização é um tipo especial de relacionamento específico, diferentemente dos anteriores.

2.1.7.7 Classe associativa

Diretamente ligada à multiplicidade na relação das classes envolvidas uma "classe associativa" não é uma relação, mas sim, uma classe que deve ser criada quando houver uma relação de multiplicidade "muito para muitos", aqui vista como "*" nas duas extremidades de uma relação. Obviamente essa classe associativa só deverá de fato ser criada, caso existam atributos relacionados à associação e que não podem ser armazenados em nenhuma das classes envolvidas na relação.

2.1.8 Considerações finais sobre UML

Todas estas informações foram organizadas para que ao se explicar logo mais sobre as características e formas de conversão da ferramenta proposta, de como ela gera um diagrama de classes a partir de um script SQL, fique mais simples o entendimento.

2.1 BANCO DE DADOS

Segundo Date (1983), o sistema de banco de dados consiste em um sistema de manutenção de informações por computador, que tem por objetivo manter as informações e disponibilizá-las aos seus usuários quando solicitadas. De forma mais simples, banco de dados é um conjunto de informações com uma estrutura regular, que podem ser recuperadas em algum momento. Existe uma enorme variedade de banco de dados, desde simples tabelas armazenadas em um único arquivo local, até gigantescos bancos de dados com muitos milhões de registros, armazenados em salas cheias de discos rígidos. A seguir serão abordadas algumas características referentes à banco de dados.

2.1.1 Banco de dados relacional

Os bancos de dados caracteristicamente modernos começaram a ser desenvolvidos na década de 1960, um dos pioneiros neste trabalho foi Charles Bachman. Desde sua criação, os bancos de dados passaram por evoluções consideráveis, pois como já dito, existe uma grande variedade de banco de dados, desde exemplos simples como uma simples coleção de tabelas até um modelo teoricamente definido, o relacional. Sobre o início do desenvolvimento dos bancos de dados, Heuser diz o seguinte:

> Nessa mesma época, houve um investimento considerável de pesquisa na área de banco de dados. Esse investimento resultou em um tipo de SGBD[4], o SGBD relacional. A partir da década de 80 e devido ao barateamento das plataformas de hardware/software para executar SGBD relacional, este tipo de SGBD passou a dominar o mercado, tendo se convertido em padrão internacional. O desenvolvimento de sistemas de informação ocorre hoje quase que exclusivamente sobre banco de dados, com uso de SGBD relacional. (HEUSER, 1998).

Desconsiderando a diferença entre os modelos de bancos, ou seja, Orientado a Objeto ou relacional ou os demais tipos existentes, o uso de bancos de dados é algo visível em qualquer site que contenha um cadastro do usuário, ou ainda em redes sociais, em programas de gerenciamento como supermercados, lojas etc. O que não se tem a certeza é como esses sites, ou programas foram modelados, como foram projetados ou ainda, qual o método inicial usado para uma remodelagem, caso necessitem uma atualização. A utilidade da ferramenta proposta não permitirá saber como um site ou um programa foi planejado, sem que se tenham os scripts do banco de dados dos mesmos, porém, ao possuir esses scripts, a ferramenta poderá ser de grande ajuda para quem a utilizará.

É necessário destacar que, quando usado o termo “script”, este se refere aos metadados das tabelas de um modelo de banco relacional. A ferramenta proposta, primeiramente, deve suportar scripts do banco Postgres, que é um banco relacional e também orientado a objetos, ou seja, é um banco objeto-relacional de acordo com as especificações em suas documentações[5]. Sabendo dessa diferença, e obtendo mais uma característica da ferramenta, é justo detalhar um pouco o porquê da ferramenta ser especificamente para um banco relacional.

---

[4] Sistema de gerenciamento de banco de dados.

[5] Fonte: http://www.postgresql.org/docs/9.1/static/intro-whatis.html Acesso em 23 de novembro de 2011.

Como já dito anteriormente, o uso dos bancos relacionais é grande, e assim, Takai, Italiano e Ferreira explicam que:

> 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[6]. [...] foi introduzido por Codd (1970). Entre os modelos de dados de implementação, o modelo relacional é o mais simples, com estrutura de dados uniforme, e também o mais formal. [...] O Modelo relacional revelou-se ser o mais flexível e adequado ao solucionar os vários problemas que se colocam no nível da concepção e implementação da base de dados. [...] O modelo relacional não tem caminhos pré-definidos para se fazer acesso aos dados como nos modelos que o precederam. (2005, p.8, p. 38).

Assim sendo, atualmente, devido ao grande uso e a certa facilidade de utilização de um banco de dados relacional, uma ferramenta focada na geração de diagramas de classes, facilitará a compreensão não só do banco de dados, mas do sistema em si.

2.1.2 Tabela

Uma tabela nada mais é do que um conjunto de tuplas ou linhas contendo atributos de uma entidade, de acordo com Heuser (2009, p. 120), "Uma tabela é um conjunto não ordenado de linhas (tuplas na terminologia acadêmica)". Assim sendo, quando descrito que uma entidade tem como característica permitir o armazenamento, na realidade é uma representação, do que ela poderá armazenar, quando for uma tabela em um banco de dados relacional.

Por fim, uma entidade pode representar objetos concretos (como pessoas, casas) como também objetos abstratos (finanças, departamentos), pois toda entidade, como já dito, representa um conjunto de objetos da realidade modelada. Neste ponto os tópicos já descritos passam a clarear as funcionalidades da ferramenta, pois anteriormente foram descritos os conceitos e declarações de uma classe em um diagrama de classes UML e agora se tem os conceitos de entidade, assim sendo, obviamente pode-se entender que através da conversão que a ferramenta realizará, uma entidade no modelo relacional passará a ser uma classe no modelo relacional.

---

[6] Processamento dedicado, exclusivo.

2.1.3 Atributos

Heuser (2009) explica que em um banco relacional cada tabela é formada por um conjunto de linhas, que, por sua vez, são formadas por uma série de campos. Um campo nada mais é do que uma das informações de uma linha. O atributo é mesmo semelhante ao atributo mostrado em um diagrama de classes, pois também possui um tipo, ou seja, uma declaração de que tipo de dado será gravado nele, como nas linguagens de programação existentes. Assim sendo, vários campos (com ou sem informações gravadas) formam uma linha, cada linha representará a informação completa de algo se deseja armazenar, e todas as linhas formam uma tabela.

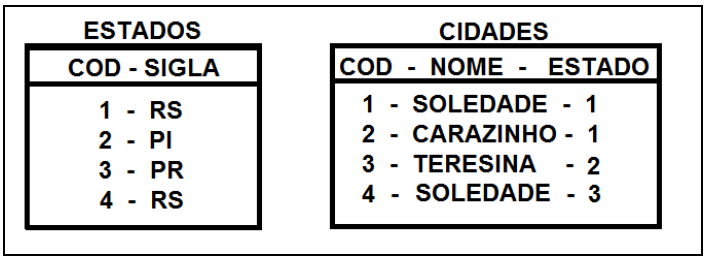
2.1.4 Chaves

Seguindo a definição de Heuser (2009, p.122) de que "o conceito básico para identificar linhas e estabelecer relações entre linhas de tabelas de um banco relacional é o de chave", pode-se entender chave como sendo um ou mais campos que terão, além do papel de simples campo, certa importância para a linha e consecutivamente para a tabela na qual está.

Uma chave pode ser primária, estrangeira ou alternativa. Como base na construção da ferramenta apenas duas chaves são vistas como fundamentais para o funcionamento. A chave primária, segundo Heuser (2009) é uma coluna, ou ainda uma combinação de colunas (campos), cujos valores distinguirão uma linha das demais da tabela, que de forma simplificada é como se fosse um identificador de uma linha, aonde a unicidade de valores desse campo, ou campos (neste caso seria uma chave primária composta), que é chave primária deverá ser mantida, e a chave estrangeira.

Uma chave estrangeira segundo Heuser (2009, p.123) é "coluna ou uma combinação de colunas, cujos valores aparecem necessariamente na chave primária de outra tabela. A chave estrangeira é o mecanismo que permite a implementação de relacionamentos em um banco de dados relacional". Uma das partes essenciais para a fundamentação desse trabalho é saber que a ferramenta possui a função de criar alguns dos relacionamentos vistos nos tópicos anteriores, logo mais, através das formas de mapeamento essa característica se tornará mais clara. Uma chave estrangeira é de extrema importância, pois ela forma as relações entre as tabelas. Desconsiderando códigos na linguagem SQL e as formas padrões de representação (UML ou MER), pode-se observar um exemplo simples do uso de uma chave estrangeira entendendo a Figura 5.

Figura 5 - Exemplo do uso de uma chave estrangeira



Fonte: Primária

O exemplo acima envolve todas as partes de uma tabela já descritas. Observa-se que em cada tabela existe um código (*COD*) que foi considerado como a chave primária, pois assim nenhuma cidade ou estado poderá ter o mesmo código, e cada cidade e estado será diferente do outro, mesmo que possa conter o mesmo nome, como é o caso do estado com *COD* 1 e 4 e da cidade com o *COD* 1 e 4.

Os campos da tabela *ESTADOS* foram definidos como *COD* e *SIGLA*, já os campos da tabela *CIDADES* como *COD*, *NOME* e *ESTADO*. Na tabela *ESTADOS* pode-se observar quatro linhas, ou seja, cada linha está representando um estado diferente do outro e no total têm-se as informações de quatro estados, intencionalmente existem dois *RS*, isso mostra que, mesmo contendo nomes iguais, o que mantém a diferença entre eles é seu código (chave primária). Na tabela de *CIDADES* existem também quatro linhas, mas obviamente poderiam ser mais linhas, cada uma representando uma cidade, assim como cada linha da tabela *ESTADOS.*

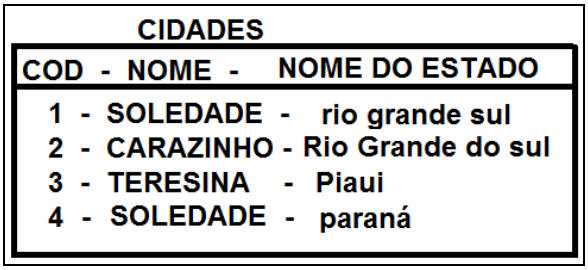
O uso de uma chave estrangeira existe na tabela de *CIDADES*, pois o campo *ESTADO* dessa tabela possui um valor correspondente ao da chave primária da tabela *ESTADOS*. Sabendo que a cidade de código 1 que possui o nome *SOLEDADE* pertence ao estado 1, e que o estado 1 refere-se à alguma chave primária da tabela *ESTADOS*, pelas informações da tabela *ESTADOS*, conclui-se que a cidade *SOLEDADE* de código 1 tem como estado, *RS*, bem como a cidade de nome *CARAZINHO*. Já a cidade *SOLEDADE* de código 4 possui o estado de número 3 e assim conclui-se que a cidade *SOLEDADE* de código 4 pertence ao estado *PR*. A cidade *TERESINA* que tem como estado o código 2, pertence ao estado *PI*, pois o *PI* é o único estado com o código 2 na tabela de *ESTADOS*.

Com isso, vê-se a importância das chaves em uma tabela, e são elas, as chaves, que farão a principal diferença para criação do diagrama de classes através da ferramenta proposta, pois através delas serão identificadas as relações entre as tabelas.

2.1.5 Relacionamentos

Em um banco de dados relacional, uma relação representa a associação entre uma chave estrangeira e uma primária, seja a primária de outra tabela ou da mesma. Uma relação permite maior integridade para o banco de dados, tendo como base o último exemplo, se solicitado (através de comando SQL) todas as cidades que pertencem ao estado 1, ter-se-ia rapidamente as cidades. Caso não existisse a relação entre as tabelas, outra maneira de representação seria conforme a Figura 6.

Figura 6 - Exemplo da falta de uma relação



Fonte: Primária

Supondo que em algum momento fossem solicitadas as cidades que pertencem ao estado do Rio Grande Do Sul, neste caso como não existe um padrão de nomenclatura nessa tabela para se referir ao estado solicitado, simplesmente não haveria como saber que as cidades “SOLEDADE” e “CARAZINHO” pertencem ao estado “Rio Grande do Sul”, pois, de acordo com o exemplo, “SOLEDADE” não está no estado “Rio Grande do Sul”, mas sim “rio grande sul”, e mesmo que uma pessoa consiga identificar isso, um banco de dados não conseguiria encontrar as cidades corretamente.

Trazendo isso para o contexto da ferramenta, se uma tabela for criada dessa forma, ou seja, sem relação, a ferramenta gerará apenas uma classe no diagrama de classes, e com isso um analista poderia facilmente identificar essa “falta de relação”, podendo repensar a forma de criar o sistema.

2.1.6 SQL

A linguagem SQL, do inglês *Structured Query Language*, é uma linguagem padrão para os bancos de dados relacionais (FERRARI, 2007), pode ser definida como uma linguagem de manipulação, definição e controle de dados de um banco de dados relacional. De acordo com Ferrari a definição da linguagem SQL é:

> A linguagem SQL é um conjunto de comandos (ou instruções) que permitem gerar enunciados, ou seja, linhas de comandos compostas por uma ou mais instruções. Alguns comandos permitem ou até mesmo exigem o uso de parâmetros adicionais, chamados de cláusulas e predicados. (2007, p. 12).

A linguagem SQL divide-se em grupos de comandos, ou seja, grupos para manipulação de tabelas (DDL - *Definition Data Language*), manipulação de dados nas tabelas (DML - *Data Manipulation Language*): e ainda privilégios aos usuários que possuem acesso ao banco de dados (DCL - *Data Control Language).*

Um script SQL (conjunto de comandos descritos em linguagem SQL) pode possuir diversos comandos destes grupos, porém relevante a este trabalho são os comandos do grupo DDL, que "são todos aqueles comandos usados para criar e alterar tabelas que compõem o banco de dados, ou seja, os comandos que definem a estrutura dos dados", segundo Ferrari (2007, p.12). Assim sendo, é através deste grupo que a ferramenta pode abstrair as tabelas, pela busca de comandos do tipo "CREATE".

2.2 ENGENHARIA REVERSA

É através do processo de engenharia reversa, que nada mais é do que uma abstração de um modelo de implementação para um modelo conceitual (HEUSER, 2009), um exemplo é transformar um script SQL em objetos da linguagem Java, sendo que estes objetos são manipulados por meio de métodos desenvolvidos, a fim de obter a conversão total e a geração do diagrama de classes.

O processo de engenharia reversa para um modelo conceitual serve, de acordo com Heuser:

> A engenharia reversa de modelos relacionais pode ser útil quando não se tem um modelo conceitual para um banco de dados existente. Isso pode acontecer quando o banco de dados foi desenvolvido de forma empírica, sem o uso de uma metodologia de desenvolvimento, ou quando o esquema do banco de dados sofreu modificações ao longo do tempo, sem que as mesmas tenham sido registradas no modelo conceitual. (2009, p.170).

Com a afirmação de Heuser fica clara neste ponto a importância que a engenharia reversa possui, considerando os casos descritos pelo autor, pois isso já promoveria grande ajuda para analistas.

2.3 JAVA

Projetada em 1991 por um grupo da Sun Microsystems, a linguagem que inicialmente tinha o nome de “Green”, possuía particularidades por ser simples e com a arquitetura neutra, para que pudesse ser executada em diversos tipos de *hardwares* (HORSTMANN, 2003).

Atualmente, uma aplicação desenvolvida na linguagem Java caracteriza-se por ser executada em uma máquina virtual específica, ou seja, independentemente do sistema operacional, a aplicação será executada. Serson discorre sobre o assunto da máquina virtual Java da seguinte maneira:

> A linguagem Java possui a singular característica de ser compilada e também interpretada. Primeiramente, o compilador Java transforma um programa-fonte em bytecodes e, posteriormente, na execução, os bytecodes são interpretados pela Máquina Virtual Java (JVM – Java Virtual Machine). Um programa Java nada mais é do que um conjunto de instruções para a Máquina Virtual Java (JVM), ou seja, o programa Java é implementado para ser interpretado por uma máquina virtual. (2007, p.05).

A JVM permite que uma aplicação Java seja executada em qualquer tipo de sistema operacional, caracterizando a portabilidade, (fruto do projeto inicial), como descreve Sampaio:

> A plataforma Java é bastante versátil e, por isto, multiplataforma. Reconhecendo as diferentes necessidades dos desenvolvedores, a Sun criou especificações separadas para o Java, de acordo com o tipo de plataforma: desktop (Standard), corporativa (Enterprise) e móvel (Micro Edition). (2007, p.08).

Existem diversas bibliotecas disponíveis em Java para auxiliar em tarefas, como manipular calendários, arquivos de imagem, sons, e, no caso da ferramenta proposta, manipular gráficos. Assim sendo, esta seção é concluída com a descrição de Horstmann sobre a linguagem Java:

> A linguagem Java em si é relativamente simples, mas Java contém um vasto conjunto de pacotes de biblioteca que são necessários para se escrever programas úteis. Há pacotes para gráficos, projeto de interfaces com o usuário, criptografia, redes, som, armazenamento em banco de dados e muitos outros propósitos. Nem mesmo programadores experientes conhecem o conteúdo de todos os pacotes – eles apenas usam aqueles que necessitam para determinados projetos. (2003, p.28)

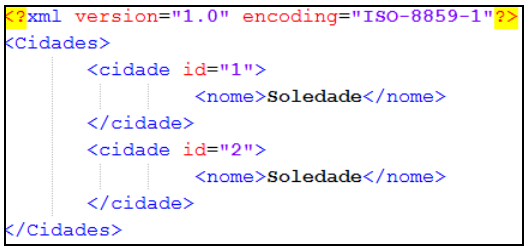
2.4 XML

Baseado em texto simples, *Extensible Markup Language* (XML) é uma metalinguagem de marcação que representa, com base em texto simples, informações diversas (W3C, 2012*)*.

Um documento XML é um arquivo que retém um grupo de dados da linguagem XML. Sabendo disso, a organização de um documento XML é de forma hierárquica, contento elementos pais e filhos, sendo que um elemento nada mais é do que uma *tag* que, além de delimitar o conteúdo que nela será inserido, especifica qual será a informação armazenada.

Para cada documento XML é necessário um "elemento raiz", ou seja, o elemento inicial da árvore de elementos, no exemplo da Figura 7, pode-se observar o elemento **Cidades** sendo raiz. A partir desde elemento os seus filhos, ou seja, os elementos que estão entre a *tag* inicial e final do elemento **Cidades** formam a estrutura que, conforme o planejamento representa as informações necessárias.

Figura 7 - Exemplo de um documento XML



Fonte: Primária

Além dos elementos, existem os atributos. Cada elemento pode conter atributos, no exemplo demonstrado na Figura 7, o elemento do tipo **cidade** possui um atributo chamado **id**, que, neste caso, pode diferenciar os dados dos dois elementos, pois o conteúdo deles é igual (Soledade), ou seja, um atributo é uma informação adicional sobre os elementos.

Com a definição da estrutura completa em XML, a manipulação da mesma pode ser realizada através de tecnologias específicas, como Xquery e Xpath, pois tais linguagens funcionam como linguagem de consulta para documentos XML. Sendo assim é possível facilitar o método de mapeamento, necessário para a construção da ferramenta que este trabalho apresenta.

A utilização do documento XML tem o propósito de armazenar temporariamente os dados, que foram extraídos do script SQL, para uma consulta dos mesmos, a fim de obter resultados pertinentes ao mapeamento desenvolvido, que será abordado no capítulo 3, seção 3.

Para a obtenção destes resultados, o uso da linguagem de consulta Xpath torna-se ideal, pelo fato que ela consegue identificar os nós que são formados pelas árvores de um documento XML, e ainda o tipo, nome e valores contidos nestes nós, podendo ainda comparar um nó com outros existentes (OFFICE, 2012).

Assim sendo, conforme o W3C, a tecnologia XML é muito utilizada atualmente por possuir diversas vantagens como detalhamento, legibilidade e processamento em relação a outros formatos. Para este trabalho apresenta-se como fundamental para a representação e manipulação, através de uma linguagem auxiliar das informações das classes existentes.

2.5 TRABALHOS RELACIONADOS

Existem diversas ferramentas com o propósito de auxiliar o desenvolvimento de um sistema, são conhecidas como ferramentas CASE (Computer Aided Software Engineering – Engenharia de Software Auxiliada por Computador) que, segundo Meireles pode ser definida da seguinte maneira:

> CASE é um termo genérico que se refere à automação do desenvolvimento de software. Segue todos os estágios do ciclo de vida do desenvolvimento de software. É baseado numa metodologia rigorosa, com ferramentas de software para automatizar a aplicação dessa metodologia pelos desenvolvedores e usuários. (2004, p.75).

Algumas ferramentas CASE, como Power Designer e Rational Rose se destacam no mercado por possuírem diversas funcionalidades e realizarem o processo completo de engenharia reversa.

O Power Designer foi desenvolvido pela Sybase, sendo uma ferramenta comercial que possibilita a modelagem e o gerenciamento de metadados, além da geração de *scripts* SQL para bancos de dados diversos através da definição do MER (SYBASE INC, 2012). De forma geral ela consegue realizar a engenharia reversa, também de scripts SQL para classes de linguagens de programação, como Java e C++.

Desenvolvida pela IBM (International Business Machines) a Rational Rose possui características semelhantes ao auxiliar na construção de softwares que possuem a orientação a objeto como método de desenvolvimento. A principal tarefa da Rational Rose é possibilitar a construção de diagramas UML, fornecendo como recurso adicional em seu funcionamento a possibilidade de engenharia reversa, para modelos não conceituais, como criação de códigos para linguagens de programação (IBM, 2012).

Assim sendo, além das ferramentas *Power Designer* e *Rational Rose* existem muitas outras, como *DBdesigner* e *DTM Data Modeler*, que realizam diversos processos de engenharia reversa, porém a maioria possui um custo para utilização, não possuem modularização ou ainda possuem funcionalidades complexas, dificultando a utilização para o usuário.

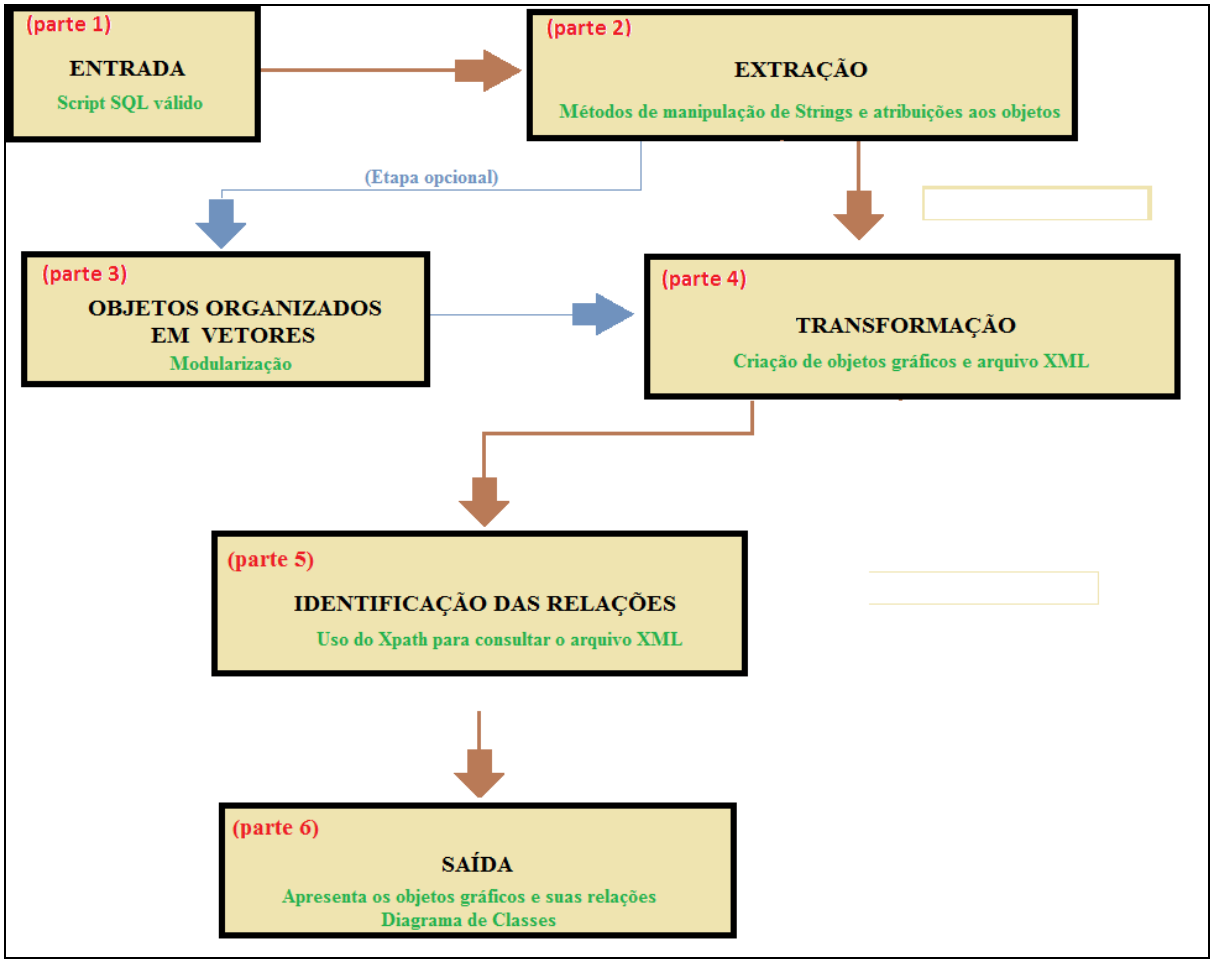
3 RESULTADOS E DISCUSSÃO

O objetivo deste capítulo visa à demonstração da arquitetura, do funcionamento, das tecnologias usadas para construção, do método de mapeamento desenvolvido e por fim na realização de um estudo de caso para que não só a ferramenta seja compreendida, mas para que os processos que formam a engenharia reversa sejam apresentados.

3.1 ARQUITETURA DA FERRAMENTA

Nesta seção será apresentada a arquitetura da ferramenta, desde a entrada do script SQL até a saída de um diagrama de classes, a fim de demonstrar, de maneira geral, como cada parte deste ciclo funciona. A Figura 8 representa a arquitetura da ferramenta.

Figura 8 – Arquitetura da aplicação CSDC



Fonte: Primária

3.1.1 Entrada

A entrada é um script SQL de um banco relacional, este deve ser composto por comandos DDL, ou seja, a definição das tabelas e seus componentes em um script SQL, e, além disso, deverá ser um “script SQL válido” aonde, além de corresponder a um banco relacional, não pode conter erros. Representado na parte 1 da Figura 8, entende-se por “script SQL válido” qualquer sequencia de comandos SQL, que esteja no padrão de interpretação SQL (apresentado na seção 3.4) imposto na ferramenta e que, considerando este padrão, não possua erros.

Os erros em si podem ser palavras inexistentes, caracteres especiais não interpretados por um banco de dados relacional, definição incorreta de tabelas, ou mesmo erros na estrutura das tabelas, como por exemplo, a declaração de uma chave primária de forma incorreta, ou uma chave estrangeira que referencia uma tabela inexistente no script.

Assim sendo, com script SQL não contendo erros e estando no padrão que a ferramenta interpreta, é possível realizar todos os ciclos desenvolvidos para que se tenha como resultado um diagrama de classes.

3.1.2 Extração

A extração é a etapa aonde todo conteúdo visto como importante dentro do script SQL, as tabelas com seus campos e chaves, são identificadas e atribuídas a objetos Java.

A extração é a parte inicial e essencial para que todo o ciclo da ferramenta seja formado e se obtenha o diagrama de classes como resultado, pois cada etapa depende diretamente do resultado da outra. É neste momento em que ocorre a identificação e criação de objetos correspondente aos itens encontrados no script SQL. Para que a extração ocorra, foram criados métodos que conseguem identificar, dentro de qualquer script SQL válido o que é uma tabela, o nome dela, quais os campos desta tabela e quais os tipos destes campos, além de identificar a obrigatoriedade deles. Por fim a extração realiza a identificação das chaves, tanto estrangeira como primária (simples ou composta), para cada tabela existente no script SQL.

Nesta etapa ocorre a atribuição dos resultados de uma extração para um objeto específico Java. Estes objetos são as instâncias de classes e são usados para armazenar as informações pertinentes a cada etapa que a ferramenta realiza.

3.1.3 Objetos organizados em vetores

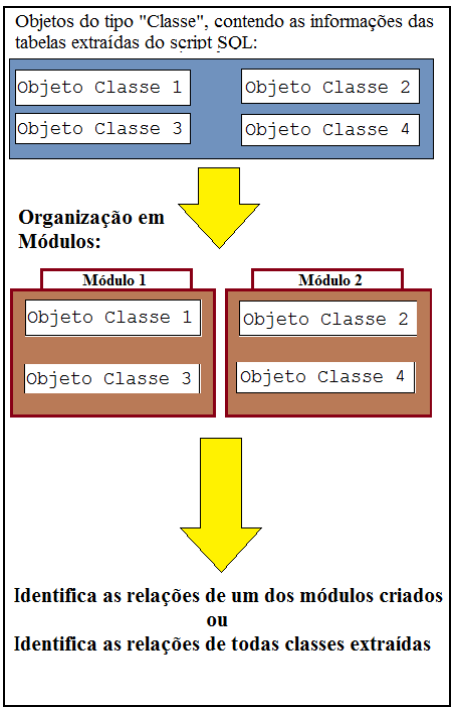
O resultado que a ferramenta gera é um diagrama de classes, ou seja, o script SQL de entrada é analisado e mapeado para um diagrama de classes, porém, considerando que neste script possa conter várias tabelas, é possível que para cada tabela existente nesse script tenha que se gerar uma classe no diagrama de classes, fazendo com que a representação final se torne muito grande e acabe com a "facilidade de entendimento". Por isso esta etapa é opcional, nem sempre usuário necessitará a modularização.

A modularização, caso seja necessária, foi definida para ocorrer nesta etapa (Figura 8, parte 3 da arquitetura), pois anteriormente as classes não poderiam ser organizadas visto que não havia uma maneira de manipular tais informações e, se essa modularização ocorresse mais tarde, ou seja, apenas na hora de visualizar, poderia comprometer não só a forma de identificar os relacionamentos entre as classes, mas também a velocidade de apresentar o resultado, pois desta forma a ferramenta gera o diagrama apenas de um módulo específico (que pode conter todas as classes) fazendo com que não se perca desempenho para montar um diagrama completo e apresentar apenas uma parte dele.

O usuário organiza os "módulos" conforme lhe convém, adicionando as classes em um único módulo, ou podendo também criar diversos módulos com diversas classes, tudo de acordo com sua necessidade. A forma conveniente para realizar esta modularização foi organizar as classes (objetos) em vetores, ou seja, dentre as classes encontradas, conforme o usuário as "adiciona" no "módulo" internamente ocorre a inserção desse objeto no vetor correspondente. Cabe lembrar que o vetor citado anteriormente não é simplesmente uma declaração de variável do tipo "Array" em Java, pelo contrário, é uma classe específica elaborada especificamente para gerenciar os possíveis módulos que serão criados, já que, se fosse uma variável do tipo "Array" não seria possível manipular e organizar tal situação devido a complexidade do processo.

Sendo assim, esta etapa é a que envolve a maior parte de codificação, pois ela organiza os objetos nas classes de acordo com os módulos criados. É importante destacar que após este agrupamento de classes, o usuário poderá escolher se deseja gerar o diagrama de classes de todas as classes extraídas, ou se deseja gerar o diagrama apenas de um destes possíveis módulos criados, e sabendo disso a "identificação das relações" pode ser será realizada. Todo este processo está representado na Figura 9.

Figura 9 – Processo geral da modularização



Fonte: Primária

3.1.4 Transformação

A forma encontrada para que a ferramenta possa obter as relações entre as classes é através de consultas específicas, ou seja, em um ciclo de repetição as classes passam por uma série de comparações com algumas condições aonde os resultados gerados serão posteriormente submetidos aos métodos específicos para criação gráfica.

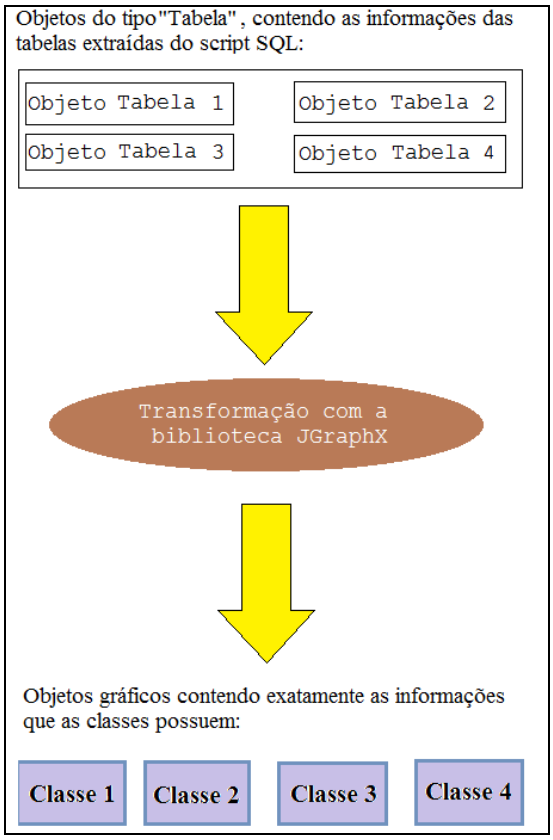
Para realizar tais comparações é necessário que as classes envolvidas passem por um processo de "transformação" (Figura 8, parte 4), aonde os objetos são mapeados a um arquivo XML para que possa ser consultado por uma linguagem específica, pois as consultas podem ser realizadas de forma simplificada.

De forma semelhante ocorre neste mesmo estado à transformação dos objetos Java que contém os dados das classes em novos objetos, sendo que estes novos objetos são objetos gráficos, como mostra a Figura 10, este processo tem o objetivo de formar objetos gráficos baseados nos objetos "Tabela"[7], após isso, os objetos "Tabela" são mapeados para um arquivo

[7] Classe desenvolvida em Java que realiza a representação de uma tabela extraída de um script SQL.

XML, assim, as comparações apresentadas são realizadas e os objetos gráficos são usados para criar as relações no diagrama de classes que será construído.

Figura 10 – Transformação de objetos Java em objetos gráficos Java



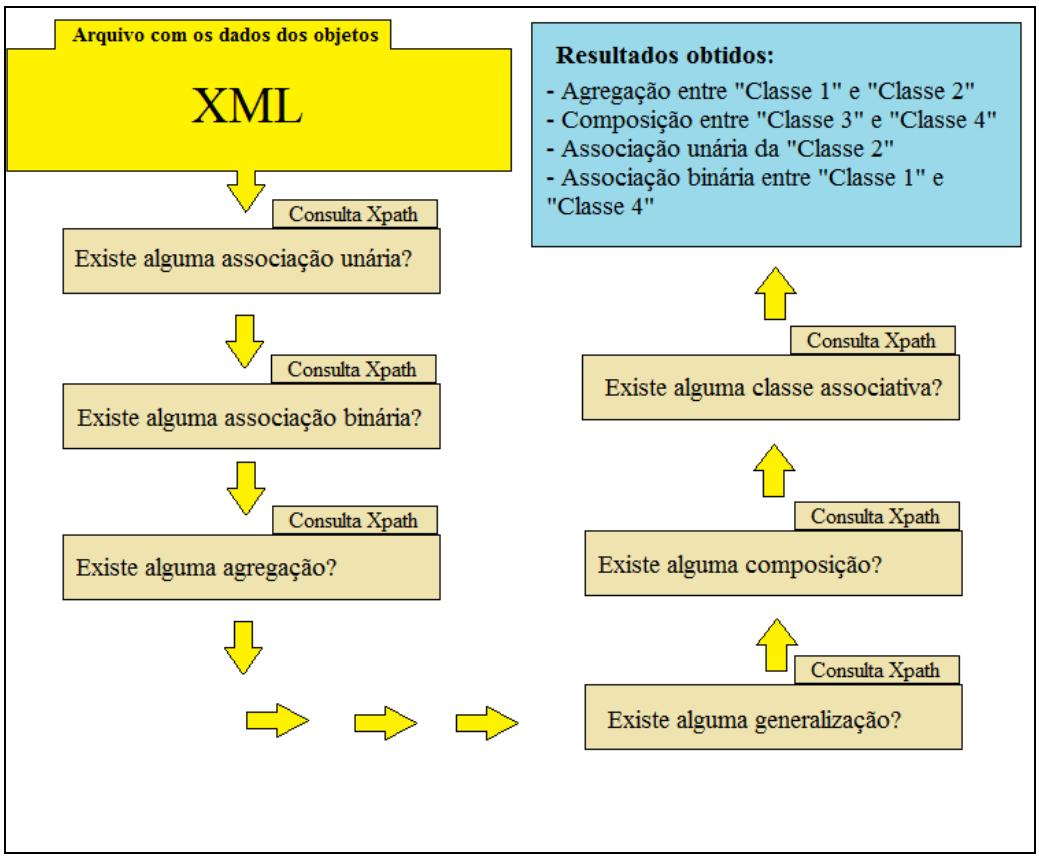
Fonte: Primária

Sabendo da complexidade para se identificar as relações entre as classes, a etapa de transformação tem o propósito de realizar as “conversões” e “representações” necessárias entre os objetos, através das tecnologias envolvidas na aplicação, para que o processo de obter as relações seja simplificado.

3.1.5 Identificação das relações

Esta etapa é responsável por realizar as comparações anteriormente citadas e possui como principal objetivo a obtenção das classes envolvidas em cada relação existente. O usuário determina se deseja gerar o diagrama de classes com todas as classes antes encontradas ou apenas de um módulo de classes específicas e, após a criação do arquivo XML correspondente, é executado o algoritmo presente na Figura 11.

Figura 11 – Algoritmo de comparações



Fonte: Primária

A Figura 11 representa a etapa de comparações, ou seja, quais são os testes realizados para obter as relações e as classes envolvidas em cada relação. A forma que as classes transitam dentro deste processo é unicamente para participar de testes, uma classe é comparada com as demais e, para cada teste é armazenado um resultado. Ao final de todos os testes entre todas as classes o resultado é um conjunto de vetores, cada um representando um tipo de relação, sendo que nestes vetores existem as classes que deverão ser mostradas na última etapa que compõe a arquitetura da ferramenta.

Estas comparações que identificam as relações poderiam ser realizadas de outra maneira, como por exemplo, o uso de banco de dados (realizando “selects” para obter as resultados específicos), ou mesmo com o desenvolvimento de métodos em Java (através de comandos internos como WHILE e FOR), porém a melhor forma de realizar as comparações necessárias foi através da linguagem XML principalmente por existir linguagens específicas de consulta como Xpath e Xquery.

É relevante destacar que com o uso do XML, além de ser facilitada a forma de consulta e identificação dos relacionamentos, pode-se utilizar este arquivo para outros fins. O

arquivo XML nesta etapa contém todas as tabelas extraídas do script SQL juntamente com seus campos e assim, embora seja usado exclusivamente para esta aplicação em um primeiro momento, poderá ser utilizado até mesmo por alguma outra ferramenta, desde que esta necessite a representação das tabelas já extraídas e tenha alguma forma de comunicação com o arquivo XML.

3.1.6 A Saída

Após as etapas anteriores a saída tem como principal objetivo realizar a construção das relações gráficas. Sabendo que o script SQL já foi extraído e transformado em classes Java na primeira e segunda etapa; as classes foram modularizadas (considerando a necessidade) na terceira etapa; após isto foram representadas em um arquivo XML e transformadas em objetos gráficos na quarta etapa; as relações foram identificadas na quinta etapa, a saída é responsável por "ligar" (através de relações gráficas) as classes envolvidas em uma determinada relação.

Não havendo muito para destacar, a saída é um diagrama de classes contendo as relações e classes que primeiramente eram tabelas de um script SQL.

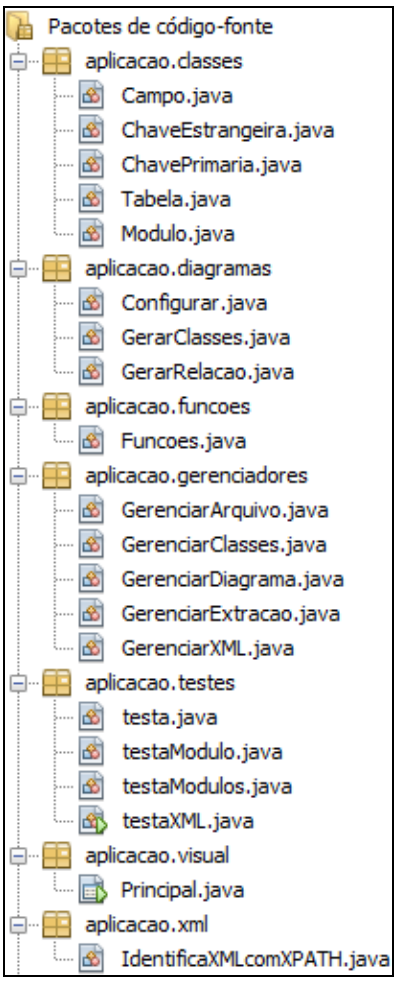
3.2 ORGANIZAÇÃO DAS CLASSES DA FERRAMENTA

Desenvolvida em Java a ferramenta é composta por várias classes e uma biblioteca gráfica (JGraphX[8]), com a organização realizada através de diversos pacotes. Pode-se observar na Figura 12 como esta organização acontece. A ferramenta foi projetada para realizar a execução de métodos em etapas de acordo com o que o usuário necessita, ou seja, com a aplicação em execução é possível identificar as classes, porém não gerar o diagrama de classes se assim desejar.

Além da funcionalidade de gerar o diagrama de classes, a ferramenta divide-se em etapas justamente para facilitar a usabilidade para o usuário.

[8] Biblioteca desenvolvida em Java especificamente para manipulação de grafos. Disponível em <http://www.jgraph.com/jgraph.html>

Figura 12 – Estrutura de arquivos da ferramenta



Fonte: Primária

De modo geral, os pacotes organizam as classes conforme suas funcionalidades ou mesmo de acordo com os tipos. Os pacotes observados na Figura 12 são estes:

- **aplicacao.classes**: Contém as classes desenvolvidas para representar os objetos de um diagrama de classes UML e ainda o objeto que representa um módulo do sistema.
- **aplicacao.diagramas**: Este pacote possui as classes responsáveis pela configuração e apresentação do diagrama de classes ao usuário, ou seja, estas classes possuem funções relacionadas com a parte gráfica.
- **aplicacao.funcoes**: Contém uma classe chamada “funcoes” que possui diversos métodos principalmente relacionados à manipulação de *Strings*.

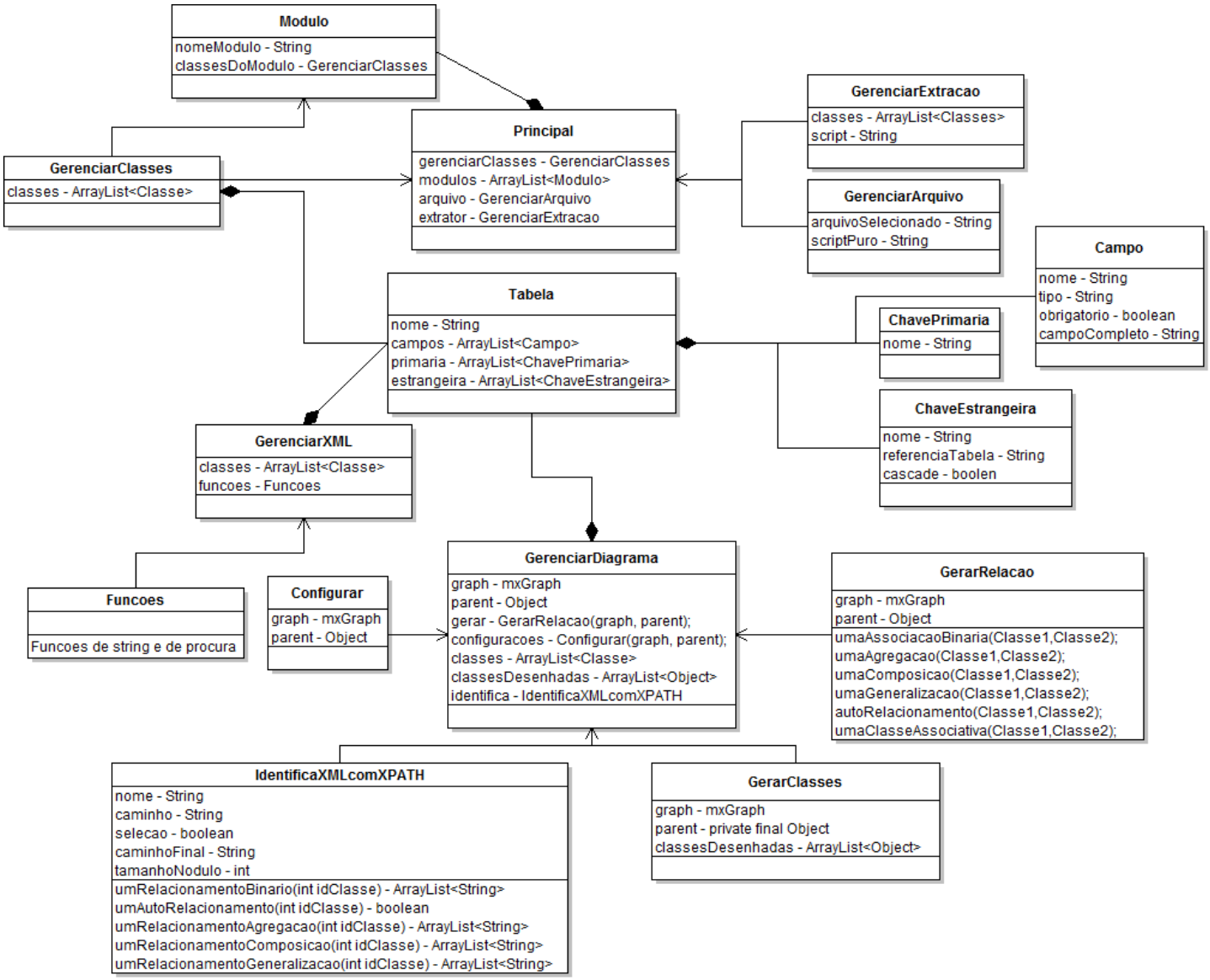
- **aplicacao.gerenciadores**: Embora não foi adotado nenhum padrão de projeto[9] no desenvolvimento da ferramenta, o uso de "pacotes" e de classes principais que possuam métodos específicos, para o gerenciamento relacionados ao seu grupo de classes, foi assim adotado para construir-se a ferramenta, pois, havendo a necessidade de uma possível edição de códigos, tal tarefa pode ser realizada semelhantemente como se houvesse um padrão de projeto, ou seja, de forma simplificada.
- **aplicacao.testes**: Pacote contendo classes que realizam testes em determinadas etapas da arquitetura da ferramenta.
- **aplicacao.visual**: Neste pacote existe a classe principal, um *Jframe* (tela) com os componentes criados para a interação com o usuário. É a interface principal.
- **aplicacao.xml**: Contém uma classe responsável pelas consultas em um arquivo XML, tal tarefa desta classe é essencial para identificar as relações entre as classes.

Com esta estrutura é possível acrescentar arquivos com determinadas características em futuras versões da ferramenta, visando assim o desenvolvimento de outras funcionalidades sem a alteração da organização ou ainda dano no funcionamento atual.

Não será detalhado o funcionamento de cada arquivo, porém é importante ter o conhecimento da estrutura da ferramenta, a fim de facilitar a visão sobre a engenharia reversa desenvolvida. Para isso, a representação em forma de diagrama de classes da ferramenta é observada na Figura 13, pois demonstra como as classes se comunicam entre si e qual o fluxo das informações que transitam entre elas.

[9] Representa uma solução geral reutilizável para resolver um determinado problema no desenvolvimento de sistemas.

Figura 13 – Diagrama de classes da aplicação desenvolvida



Fonte: Primária

3.3 REGRAS DE MAPEAMENTO

Esta seção tem o objetivo de especificar quais são as regras de mapeamento que, através de consultas Xpath, são utilizadas para obtenção das relações existentes em um diagrama de classes UML.

3.3.1 Considerações iniciais para o mapeamento

A abordagem a respeito dos códigos Xpath apresentados no decorrer desta seção será para complementar o entendimento, valendo-se mais na apresentação das regras de mapeamento em si.

Os relacionamentos entre tabelas estão diretamente ligados a composição das chaves estrangeiras, baseado nisso foram considerados os seguintes aspectos identificar as relações dentro do documento XML:

- A chave estrangeira é primária?
- A chave estrangeira é primária composta?
- A chave estrangeira é obrigatória?
- A chave estrangeira relaciona-se com a tabela que está presente?
- A chave estrangeira possui a cláusula *cascade?*

Para cada elemento do tipo "Tabela" (este representa uma tabela extraída do script SQL) mapeado no arquivo XML – mapeamento gerado através de um ciclo de repetição Java - é realizado uma pesquisa com a linguagem Xpath. Tal pesquisa (ao arquivo XML) possui o propósito de executar comparações entre as chaves estrangeiras da tabela em questão, a fim de obter o tipo de relacionamento que esta possui. Após cada pesquisa é armazenado o resultado em um vetor específico para mais tarde ser usado na criação das relações gráficas.

Assim sendo, é feita uma consulta por associações binárias e, caso sejam encontradas, os números identificadores (Id[10]) das tabelas envolvidas nestas relações são armazenados em um vetor específico de "associações binárias". Da mesma forma ocorre com os demais tipos de relações e com isso, ao final de todas as consultas, os vetores que armazenam as relações estarão preenchidos com os números das tabelas que, por sua vez já estão transformadas em objetos gráficos.

Por fim, com o uso da biblioteca gráfica e os vetores específicos de relações indicando quais são as relações existentes, é possível gerar o diagrama de classes.

Através destes pontos foi desenvolvido todo mapeamento das relações e, neste momento serão apresentadas todas as relações e as definições de mapeamento das mesmas. Cabe lembrar que, o termo "classe" é utilizado a partir deste momento semelhantemente à "tabela", pois, até então, o uso da palavra "tabela" significava a tabela extraída do script SQL, porém, durante a representação das tabelas no arquivo XML, o uso de "tabela" é transcrito para "classe".

---

[10] Este Id é um número único correspondente à posição do vetor que a tabela ocupa. Com este número é possível manipular as classes de forma simples, mesmo que todas possuam o mesmo nome por exemplo.

3.3.2 Mapeamento de uma associação unária

Uma associação unária é aquela que a classe UML relaciona-se consigo mesmo, e, neste caso, o script SQL possui um auto-relacionamento na tabela em questão, ou seja, uma chave estrangeira "apontando" para ela mesma. Assim sendo, se alguma chave estrangeira apontar para a tabela que pertence, automaticamente será interpretado uma relação unária. Para realizar este "teste", para cada classe existente no arquivo XML a linguagem Xpath utiliza o comando presente na Figura 14, substituindo o nome "idClasse" pelo número da classe atual:

Figura 14 – Consulta Xpath para obtenção de um relacionamento unário

```
/classes
/classe[nome=/classes/classe[idclasse="idClasse"]
/chaves
/estrangeiras
/estrangeira
    [referencia=
                /classes
                /classe
                    [idclasse="idClasse "]
                 /nome
                /text()
                 And
                cascade="false"
     ]
     /referencia
     /text()]
/idclasse
/text()
```

Fonte: Primária

Como exemplo, é possível considerar que existam 3 classes (antes tabelas e por isso ainda possuem chaves em sua composição) representadas no arquivo XML e que a última classe, de nome "classe 3", possua uma chave estrangeira que referencia ela mesma, ao ser executado o código presente na Figura 14, não serão identificados nenhuma relação unária nas duas primeiras classes desse exemplo, porém, quando a última classe passar por este processo, a consulta Xpath irá identificar que uma chave estrangeira da "classe 3" está referenciando a "classe 3" e assim, será retornado o número desta classe (o Id), para que posteriormente seja transformado em uma relação gráfica do tipo "unária".

3.3.3 Mapeamento de uma associação binária

A associação binária é identificada quando uma chave estrangeira possui uma relação com outra tabela, porém esta relação possui um vínculo fraco, ou seja, sem obrigatoriedade. Assim sendo, os requisitos para existir uma relação binária, são:

- A chave estrangeira não pode ser primária;
- A chave estrangeira não pode ser obrigatória;
- A chave estrangeira não deve referir-se a própria tabela;
- A chave estrangeira não pode conter a cláusula *cascade*.

Assim sendo, uma associação binária foi definida como uma relação de baixa influência, ou seja, além de não ser primária e nem possui a cláusula "*cascade*" (responsável por manter um vínculo maior entre as tabelas envolvidas) a chave estrangeira deve referenciar outra tabela, porém esta chave deve ser opcional.

Para realizar os testes descritos, seguindo a lógica definida nas considerações iniciais do mapeamento, a consulta realizada para obtenção de uma relação binária é demonstrado na Figura 15:

Figura 15 – Consulta Xpath para obtenção de um relacionamento binário

```
/classes
/classe
   [nome=
        /classes
        /classe
          [idclasse="idClasse"]
       /chaves
       /estrangeiras
       /estrangeira
          [nome=
            /classes
            /classe
                [idclasse="idClasse"]
            /campos
            /campo
                [obrigatorio="false"]
            /nome
            /text()  and cascade="false"
          ]
          /referencia
          /text() and idclasse!="idClasse"
    ]
/idclasse
/text ()
```

Fonte: Primária

Neste caso, a Figura 15 demostra uma consulta Xpath mais complexa, pois a chave estrangeira de uma determinada tabela deve ser opcional (observa-se isso na Figura 15 parte em destaque chamada “obrigatorio”), além disso, ela não poderá referenciar a mesma tabela (pois seria confundida com uma relação unária) e assim também são inseridas as demais condições na consulta.

O retorno esperado é o número de Id da classe que a chave estrangeira referencia, caso a chave estrangeira submetida a este processo esteja de acordo com todas estas condições. Vale lembrar que esta chave estrangeira pertence à classe que está sendo processada pelo ciclo no momento. Portanto, a realização deste processo é similar aos demais e, com estas pecularidades encontra-se as relações do tipo binária.

3.3.4 Mapeamento de uma agregação

A agregação é mapeada de forma semelhante a uma associação binária: a principal diferença de uma agregação para uma binária, - considerando a diferença de um banco relacional e uma representação orientada a objetos (diagrama de classes UML) - está na obrigatoriedade da chave estrangeira em uma tabela.

O mapeamento que mais adequa-se à realidade de uma agregação foi desenvolvido da seguinte maneira:

- A chave estrangeira não pode ser primária;
- A chave estrangeira deve ser obrigatória;
- A chave estrangeira não deve referir-se à própria tabela;
- A chave estrangeira não pode conter a cláusula *cascade*.

Uma agregação representa a relação aonde uma classe “possui” outra ou “agrega valores” de outra, ou seja: Esta relação possui maior importância. Esta importância é identificada pela obrigatoriedade de uma chave estrangeira.

Após a busca realizada com a expressão Xpath determinada, o resultado será também armazenado, desta vez em um vetor de relações do tipo agregação.

3.3.5 Mapeamento de uma composição

No diagrama de classes, uma composição significa que uma classe é composta por uma ou mais classes. Já em um banco de dados relacional, existe a cláusula “*cascade*”,

utilizada juntamente com uma chave estrangeira; esta expressão significa que se os dados da tabela A estão referenciando um dado específico da tabela B, e em algum momento este dado da tabela B for apagado, todas as informações presentes na tabela A que referenciavam este dado da tabela B serão também exluídos, ou seja, há um relacionamento de dependência entre as tabelas.

O mapeamento foi definido com base nessa perspectiva, quando se encontra uma chave estrangeira que possui esta cláusula, a relação possui maior vínculo, maior dependência e, num diagrama de classes não é diferente. Uma composição demonstra maior importância e um vínculo maior, pois um objeto pode ser composto por outros.

Um ponto importante a ser destacado é sobre a relação de dependência no diagrama de classes UML, que por sua vez, não é representado pela ferramenta. Este relacionamento de dependência existe quando um objeto de uma classe depende de métodos de um ou mais objetos de outra classe, portanto, a dependência entre classes indica que os objetos de uma classe usam serviços dos objetos de outra classe. Como não é possível identificar a funcionalidade de métodos das classes, pois este processo é detalhado em outra etapa, não é possível gerar uma relação de dependência UML baseada no script SQL.

Assim também ocorre com o relacionamento da realização, sendo um relacionamento entre os itens que implementam o comportamento especificado por outro, tal mapeamento não pode ser realizado com base nas chaves e atributos.

Em resumo, para que seja identificado um relacionamento do tipo composição, a consulta Xpath procura pela cláusula "*cascade*" em uma chave estrangeira e, após encontrá-la, identifica qual é a tabela que esta chave estrangeira referencia e retorna o número de *Id* da tabela referenciada e ainda da tabela atual.

3.3.6 Mapeamento de uma generalização/especialização

Quando existem tabelas genéricas e tabelas que se especializam, há um relacionamento de generalização. Uma generalização possui uma semelhança com uma herança em um diagrama de classes, ou seja, existe uma classe que "herda" características de outra classe, que é chamada de genérica.

Assim sendo, em alguns casos, quando existe uma chave primária que, ao mesmo tempo é estrangeira, isto indica que a relação entre as tabelas possui um tratamento diferenciado (de especialização), pois mesmo sendo um identificador de uma tabela, também é baseado em uma referência de outra tabela.

Este conceito pode também representar uma relação de cardinalidade 1 para 1, nesse sentido cabe ao analista identificar o tipo de relação, se é uma herança ou uma cardinalidade 1 para 1. A ferramenta proposta fará a geração como uma herança.

A forma de transformação e o mapeamento empregado a esta relação foi definido da seguinte forma:

- A chave estrangeira deve ser primária simples;
- A chave estrangeira deve ser obrigatória;
- A chave estrangeira não deve referir-se à própria tabela;
- A chave estrangeira não pode conter a cláusula *cascade*.

Definido as informações, uma consulta específica em Xpath consegue identificar quais são os *Id's* das tabelas (no XML definidas como classes) que possuem este tipo de relacionamento.

Figura 16 – Consulta Xpath para obtenção de uma generalização

```
1   /classes
2   /classe
3         [nome=
4          /classes
5         /classe[idclasse="idClasse"]
6        /chaves
7       /estrangeiras
8      /estrangeira
9         [nome=
10                /classes
11                 /classe  [idclasse="idClasse"]
12                 /campos/campo[obrigatorio="true"]
13                 /nome
14                 /text()  and  nome=/classes /classe [idclasse="idClasse"]
15                 /chaves
16                 /primaria
17                 /chave
18                /nome
19                /text() and cascade="false"]
20                /referencia
21             /text() and idclasse!=”idClasse"]
22            /idclasse
23            /text ()
```

Fonte: Primária

A Figura 16 apresenta como são realizadas as comparações em uma consulta Xpath para a busca de generalização. O trecho da linha 19 a 21 representa as condições de não conter a cláusula *cascade* (pois seria uma composição) e ainda não possuir um auto-

relacionamento (pois a chave estrangeira deve apontar para uma tabela diferente da atual). O trecho de código da linha 10 a 18 representa as comparações em uma tabela para verificar se uma chave estrangeira é também primária, ao modo que, se identificada uma ou mais chaves estrangeiras da tabela atual que entram em conformidade com esta comparação, serão retornados os números das classes que possuem este tipo de relação (código da linha 22 na Figura 16) e, assim como nos outros relacionamentos, este resultado será armazenado em um vetor e manipulado posteriormente para a formação gráfica desse relacionamento.

3.3.7 Mapeamento de uma classe associativa

Em um relacionamento muito para muitos em um banco de dados relacional é comum realizar a criação de uma nova tabela para armazenar as informações referentes às tabelas envolvidas, tal fato ocorre assim porque as informações pertinentes à relação não podem ser devidamente armazenadas em nenhuma das tabelas. Da mesma maneira, uma "classe associativa" existe, ou seja, para representar informações de uma relação entre classes que não podem ser armazenadas em nenhuma das classes envolvidas.

Assim sendo, quando identificada uma chave primária composta, que é também estrangeira, gera-se uma classe associativa.

3.3.8 Mapeamento de multiplicidade, navegabilidade e visibilidade

A visibilidade é mapeada conforme a obrigatoriedade dos atributos de uma tabela, ou seja, se um atributo é obrigatório, considerando a importância de um campo também obrigatório, este será transformado em um campo "não visível" em uma classe de um diagrama de classes, pois um campo "não visível" é aquele que não é acessado diretamente por outras classes; é acessado por métodos específicos.

Acerca da multiplicidade, esta é identificada pela obrigatoriedade das chaves estrangeiras, ou seja, se uma chave estrangeira não é obrigatória a relação será de "0 para N" (zero ou muitos), porém se for obrigatória a relação torna-se "1 para N" (um para muitos) pois ao menos uma ocorrência de uma classe deve existir na outra. A multiplicidade "muitos para muitos" está mapeada na classe associativa. A relação de "1 para 1" não foi mapeada, pois não foi identificado um caso específico para este tipo de relação.

3.3.9 Considerações finais sobre as regras de mapeamento

A ferramenta desenvolvida, em um primeiro momento não possui a funcionalidade de oferecer todas as possibilidades existentes perante determinadas situações, ou seja, se um analista realizar o diagrama de classes baseado em um banco de dados relacional, ele poderá formar diversos diagramas, dependendo o contexto e a forma que ele deseja manipular os dados em sua aplicação, porém, no caso da aplicação, considera-se apenas um contexto e, baseado neste contexto, é realizado o mapeamento.

Este contexto foi definido pelas chaves estrangeiras, como visto nas considerações iniciais desta seção.

Assim sendo, este mapeamento representa uma conversão em um único contexto, pois a ferramenta gera o diagrama de classes de forma conceitual e não considera a "lógica de negócio" que o sistema pode conter. Por isso é possível que em determinadas situações um script SQL não seja transformado em um diagrama de classes.

3.4 UTILIZAÇÃO DAS TECNOLOGIAS NA FERRAMENTA CSDC

Esta seção mostra como são utilizadas as tecnologias que contribuem à formação do resultado final, no contexto de funcionamento do CSDC, por isso, organiza-se em:

- Explicação do padrão SQL definido;
- Identificação das tabelas do script SQL;
- Identificação dos campos e chaves de uma tabela;
- Armazenamento em um documento XML;
- Criação de objetos gráficos utilizando JgraphX;
- Criação das relações gráficas.

3.4.1 Padrão SQL envolvido

É fundamental apresentar qual o padrão de script SQL (dialeto SQL) de um banco relacional que a ferramenta considera como "válido".

O padrão escolhido para a busca dos comandos DDL, foi definido com base em pequenos testes de buscas por *Strings* na linguagem Java, pois, considerando o tempo para o desenvolvimento da ferramenta e prioridades de eficiência na construção, o padrão foi assim definido por sua simplicidade de criação, definição dos campos e declaração das chaves. De outra maneira não seria possível englobar todas as interpretações das distintas formas de

criação de um SQL (comandos DDL), e, além disso, o padrão foi assim definido por ser um padrão simples de criação, de definição dos campos e declaração das chaves.

Figura 17 – Parte de um script SQL válido

```
create table ITENS(
codigo_it integer not null,
pedido integer not null,
foreign key(pedido) references
pedidos(codigo_ped) on delete cascade,
primary key(codigo_it)
);

create table avioes_aeroportos(
aviao integer not null,
aeroporto integer not null,
descricao varchar(250) not null,
foreign key(aviao) references
avioes(codigo),
foreign key(aeroporto) references
aeroportos(codigo),
primary key(aviao,aeroporto)
);
```

Fonte: Primária

Com base na Figura 17, pode-se observar um script de entrada válido, ou seja, um script que é interpretado pela ferramenta. Para um script ser válido para a ferramenta, deve-se considerar os seguintes pontos:

- Para cada tabela criada deve existir um "(" determinando o início da tabela e um ");" determinando o fim da mesma;
- A ferramenta não é *Case Sensitive*[11];
- A declaração da chave primária, tanto simples como composta, deve ser separado da declaração do campo, ou seja, primeiro cria-se o campo, e depois o especifica como primária;
- Os tipos de dados aceitos para os campos são: integer, varchar, char e date;
- Não podem conter espaços, as declarações entre parênteses, como: tamanho do char e varchar e, ou o nome da tabela que uma chave estrangeira referencia.
- Exceto a expressão "*on delete cascade*" em uma chave estrangeira outras expressões como "*check*", "*constraint*" e outros são desconsideradas pela ferramenta;

[11] Neste contexto, significa que não há problemas com a diferença entre maiúsculas ou minúsculas no script SQL.

- Palavras sem nenhum sentido não são extraídas, porém, um código com tais erros poderá prejudicar a formação do diagrama de classes.

A seguir serão apresentadas as principais formas de mapeamento de um script SQL que a ferramenta realiza, ou seja, como é realizada a extração do nome da tabela, tipo e nome dos atributos, obrigatoriedade, chave primária e estrangeira.

Vale lembrar que a principal tarefa da ferramenta é construir um diagrama de classes baseado em um código SQL, por isso expressões como "*check*" não foram consideradas pela aplicação por não ser vista como fundamental, em um primeiro momento para a construção do diagrama de classes.

Caso haja algum código SQL, incompreensível (entrada inválida), ou por estar fora do padrão que a ferramenta interpreta, ou por apresentar erros em sua estrutura, este será ignorado pela mesma, que dará continuidade à criação do diagrama de classes, mesmo que, de um script inválido, criando assim um diagrama defeituoso ou ficando impossibilitada à condição de fornecer saída; porém, estando o script dentro do padrão, o programa passará para segunda etapa: extração.

3.4.1.1 Identificação das tabelas do script SQL

Utilizando a classe que foi desenvolvida especificamente para manipular *Strings*, obter o nome de uma tabela torna-se maleável. Considerando o padrão adotado, todas as tabelas declaradas em um script SQL estão entre os comandos "CREATE TABLE" e o primeiro parênteses "(". Sendo assim, existindo uma função que identifica onde estas estão no documento, o "CREATE TABLE," e, em seguida, qual a posição do primeiro "(", obtém-se o nome da tabela.

Tal tarefa seria completamente "simples" se houvesse apenas uma tabela em um script. Um dos fatores que torna esta tarefa muito importante e, de certa forma, complicada, é que um script SQL pode conter inúmeras tabelas e, este método deve ser executado diversas vezes, sem afetar o funcionamento de extração dos outros elementos. Para que tudo isso ocorra de forma organizada, obtendo-se todas as informações necessárias, criou-se um ciclo, (comando WHILE), através do qual, todos os elementos são encontrados e atribuídos a objetos Java.

O modo usado para executar a extração de cada classe, foi através de uma lógica de programação, aonde, existe um ciclo que fica à procura do termo "CREATE TABLE" no

script SQL, porém, conforme as tabelas vão sendo identificadas, o termo “CREATE TABLE” é substituído, fazendo com que cada classe seja “encontrada” apenas uma vez e a extração seja feita de forma correta. Ao final o número de ciclos será correspondente ao número de tabelas e as abstrações podem ser realizadas com segurança.

Para que todas as tabelas sejam identificadas e armazenadas com seu conteúdo total, cria-se um vetor onde elas são armazenadas; assim, ao final deste ciclo de extrações, o resultado é um vetor com todas as tabelas armazenadas, sendo que estas estarão com todos seus atributos também devidamente armazenados. Pode-se observar, a forma que o algoritmo funciona, conforme a Figura 18 abaixo:

Figura 18 – Algoritmo que representa a extração e armazenamento das tabelas

```
Vetor de tabelas;

Enquanto encontrar um "Create table" no Script SQL
{

  Cria um objeto que armazena os dados de uma tabela;
  Adiciona nesse objeto o nome da tabela encontrada;

  (encontra os atributos)

  Adiciona nesse objeto os atributos;

  Adiciona no vetor a tabela que agora possui nome e atributos;

  Substitui o primeiro termo "Create table" no Script SQL


}
```

Fonte: Primária

3.4.1.2 Identificação dos campos e chaves de uma tabela

Para cada tabela identificada é necessário outro ciclo de identificação, ou seja, além do primeiro ciclo (que identifica o nome da tabela), existe um ciclo interno, responsável por identificar todos os outros elementos (campos e chaves) desta tabela. Neste ciclo identifica-se o conteúdo da tabela, ou seja: Nessa etapa, o código SQL, que está entre “CREATE TABLE NOME[12]” e “);” é o “centro” da tabela e neste centro estão os atributos e chaves correspondentes à tabela encontrada. Sabendo que a divisão de atributos em um script SQL é feita por uma vírgula, foi desenvolvido um método que delimita os campos através desta

[12] Este “NOME” representa o nome da tabela encontrada com o primeiro método de extração.

vírgula e armazena-os para manipulá-los depois. A figura 19 representa a lógica de extração de todo conteúdo de uma tabela.

Figura 19 – Algoritmo que demonstra a extração do conteúdo

```
Enquanto encontrar uma tabela no Script SQL faça
{
    Método que encontra a chave primária desta tabela ;
    Cria-se um objeto que armazena a chave primária encontrada;

        Método que encontra o conteúdo desta tabela ;
        Para cada "linha" do conteúdo encontrado faça
        {
            Se a linha é uma chave estrangeira
              - É criado um objeto que armazena esta informação
            Se a linha é um simples atributo
              - É criado um objeto que armazena esta informação
        }
    Adiciona o conteúdo dos objetos na tabela atual;
}
```

Fonte: Primária

Nota-se que todos elementos de um script SQL, além de ser identificados, são atribuídos (transformados) em objetos Java. Este processo ocorre quando, durante a extração das tabelas, todo conteúdo válido do script SQL é atribuído a objetos (instância de classes Java), tanto que, nas Figuras 18 e 19 a transformação é representada de forma "natural" através da frase "cria um objeto". É neste momento que a "tabela" no script SQL passa a ser tratada como um objeto do tipo "Classe" dentro da programação da ferramenta.

Com esta atribuição uma vez realizada, podem-se manipular os objetos do tipo "Classe". A Figura 20 demonstra qual é o método que encontra as tabelas de um script SQL.

Figura 20 – Trecho de código da transformação de tabelas em objetos

```
Classe c = new Classe();
c = encontrarUmaClasseComSeusCampos(getScript());
classesEncontradas.add(c);
```

Fonte: Primária

O método "encontrarUmaClasseComSeusCampos" é o responsável pelo processo anteriormente visto e demonstrado na Figura 18 e 19. Pode-se utilizar a melhor forma de extração, porém, se não há onde armazenar estes dados obtidos, ou se tais dados forem guardados de forma incorreta ou ainda de modo que não se possa manipular posteriormente,

de nada adianta a extração. Sabendo disso, as classes desenvolvidas em Java devem permitir que os dados extraídos do script SQL sejam representados dentro da programação, de forma similar, ou o mais próximo possível da forma original, quando eram apenas dados DDL.

Assim, com os objetos criados representando todas as tabelas do script SQL, esta primeira transformação que a ferramenta realiza é concluída, resultando em objetos Java que logo serão novamente transformados, em objetos gráficos e também em um arquivo XML.

3.4.2 Documento XML gerado

Nesta seção serão abordadas as transformações referentes ao documento XML, de forma que seja apresentada como é a criação de um documento e sua importância. A Figura 21 demonstra-o da seguinte maneira:

Figura 21 – XML que representa as tabelas extraídas do script SQL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<classes>
    <classe>
        <idclasse>3</idclasse>
        <nome>clientes</nome>
        <campos>
            <campo>
                <nome>codigo_cli</nome>
                <tipo>Integer</tipo>
                <obrigatorio>true</obrigatorio>
                <stringcampocompleto>codigo_cli integer not null</stringcampocompleto>
            </campo>
            <campo>
                <nome>cidade</nome>
                <tipo>integer</tipo>
                <obrigatorio>false</obrigatorio>
                <stringcampocompleto>cidade integer</stringcampocompleto>
            </campo>
        </campos>
        <chaves>
            <primaria>
                <chave>
                <nome>codigo_cli</nome>
                </chave>
            </primaria>
            <estrangeiras>
            <estrangeira>
                <nome>cidade</nome>
                <cascade>false</cascade>
                <referencia>cidades</referencia>
            </estrangeira>
            </estrangeiras>
        </chaves>
    </classe>
</classes>
```

Fonte: Primária

A fim de obter os relacionamentos que as classes possuem, foi desenvolvida a estrutura XML correspondente às informações dos objetos Java; este processo de desenvolvimento da estrutura do XML será abordado a seguir.

Observando a Figura 21, o arquivo XML foi desenvolvido com esta estrutura para representar as seguintes características:

- Representar todas as tabelas. Para isso existe o elemento raiz, “classes” que possui os filhos “classe”;
- Cada classe precisa ser identificada e tratada de forma única, portanto existe um *Id* para cada classe;
- As classes possuem nomes, assim, criou-se um elemento “nome” dentro do elemento “classe”;
- É necessária a representação de cada campo, por exemplo, para poder verificar obrigatoriedade, para tanto existe um elemento “campos” que possui como filhos os campos da classe, sendo que um campo deve possuir uma obrigatoriedade, um tipo e um nome.
- Assim como os campos, uma classe possui as chaves (considerando que uma “classe” é a fruto da transformação de uma tabela) e esta representação é feita através de um elemento “chaves”, que possua como filhos as chaves da classe (primária e estrangeira).
- Cada chave possui suas particularidades específicas, como “referencia” ou “*cascade*” no caso de uma estrangeira, somente o “nome” no caso de uma primária.

O elemento “stringcampocompleto” que é um filho de campo pode ser utilizado para ocasiões específicas, como verificação do tipo de dado, ou referência de alguma classe, porém, tal elemento apenas representa a linha a fim de facilitar a visão de um campo no XML.

É importante destacar que os elementos criados para mapear os objetos Java, conseguem representar exatamente os dados extraídos das tabelas, antes encontradas no script SQL. Com isso, através da consulta com Xpath neste arquivo XML, é possível obter dados como:

- O nome de todas as classes;
- O nome dos campos da classe de nome “cidades” (ou qualquer outro nome);
- O *Id* de uma classe que a chave estrangeira de outra classe a referencia;

Embora não seja necessário nesta etapa encontrar o nome de todas as classes ou o nome dos campos de uma determinada tabela, a estrutura definida e apresentada no documento XML consegue realizar o mapeamento de todos os dados dos objetos Java.

3.4.3 Biblioteca gráfica JgraphX

Esta seção apresentará os métodos responsáveis pela criação de objetos gráficos e relações gráficas que, através da biblioteca específica para a manipulação de grafos[13], realizam a apresentação visual do diagrama de classes.

3.4.3.1 Criação de objetos gráficos utilizando JgraphX

A criação dos objetos gráficos foi realizada unicamente pela biblioteca JgraphX, assim sendo nenhum método foi desenvolvido para tal função. Mesmo sem ter-se desenvolvido métodos para a criação dos gráficos, pois tais métodos estão embutidos na biblioteca, a necessidade principal foi o desenvolvimento de uma função que utiliza estes "métodos gráficos", assim sendo esta função é responsável pela criação das classes e das relações entre elas, de acordo com os objetos que possuem as informações de uma classe UML.

Utilizando os métodos disponíveis na biblioteca gráfica, com o objetivo de primeiramente construir um grafo que "aparentasse" uma classe UML os seguintes itens foram considerados:

- O objeto gráfico deve conter nome e atributos, conforme os objetos "Tabela" possuem.
- Para todo objeto classe é necessário um objeto gráfico correspondente;
- Cada classe pode conter diversos atributos e seu tamanho varia em cada caso;
- Uma classe UML possui divisões internas que devem ser correspondentes ao tamanho horizontal da classe;
- Os tipos de atributos devem ser transformados assim como a classe, pois um "varchar" no script SQL é na realidade representado por uma *String* em um diagrama de classes, bem como o "not null" pode ser representado como indicador de visibilidade de um atributo em diagrama de classes UML.

Baseado nestes pontos principais e com a realização de diversos testes a fim de obter o máximo das funcionalidades da biblioteca JgraphX, pois é uma biblioteca até certo ponto complexa quando se necessita personalizar, para a criação das classes (sem relações, apenas "classes gráficas[14]"), a ferramenta utiliza métodos específicos que não só transformam os objetos em grafos mas também calculam o tamanho adequado do grafo, estes métodos

[13] Conjunto de elementos que se relacionam de alguma maneira, um grafo é representado como um conjunto de pontos (vértices) ligados por retas (as arestas).
[14] Um grafo desenhado para representar uma classe de um diagrama de classes UML.

pertencem a uma classe Java e esta à ferramenta, tal classe é responsável pela transformação completa de todos objetos “Tabela” para objetos gráficos, e ao final deste procedimento, estas “classes gráficas” são armazenadas em um vetor específico para serem, em um segundo momento, ligadas umas as outras, conforme as relações encontradas pela etapa responsável.

Figura 22 – Cálculo do tamanho de uma classe gráfica

```
    //Algoritmo responsável pelo cálculo do tamnho da classe gráfico
    String nomeClasse = classeAtual.getNome();
    double altura = 100; //inicia com 100px
    double largura = 20; //inicia com 20px
    String linhaDivisao = "_"; //linha usada nas dvisões das classes
    int tamanho = 0; //usado para controle de caracteres de cada linha
    //de uma classe

    StringTokenizer st = new StringTokenizer(conteudoClasse);
    //o conteúdo da classe nada mais é do que os atributos

    //enquato houver linhas neste conteúdo
    while (st.hasMoreTokens()) {
        String linha = st.nextToken("\n");
        if (linha.length() > tamanho) {
            tamanho = linha.length(); //
        }
        altura += 20;
    }

    largura = tamanho * 8;
    for (int i = 0; i <= tamanho; i++) {
        linhaDivisao = linhaDivisao + "_";
    }
    linhaDivisao = linhaDivisao + "\n";

// para cada atributo a altura aumenta 20 (medida em pixels)
// calcula quantos caracteres existe em uma linha horizontal
// multiplicando por 8 (número definido por testes)encontra-se a largura
```

Fonte: Primária

Observa-se na Figura 22 como é realizado o cálculo do tamanho de uma classe gráfica, pois mesmo que a biblioteca JgraphX contenha diversos métodos específicos para situações semelhantes, como redimensionar, ou alinhar o conteúdo interno, nenhum se adequou o suficiente para este caso como o desenvolvido.

É importante saber que, ao final desta sequência de cálculos a ferramenta possui variáveis que serão usadas na formação de uma classe gráfica (as variáveis armazenam o nome da classe, conteúdo da classe, altura, largura e linha divisória).

Para se criar um objeto gráfico, considerando a classe Java desenvolvida especificamente para simplificar este processo, através do método da biblioteca JgraphX

chamado “insertVertex” é possível realizar a transformação desejada, a Figura 23 demonstra como é este procedimento.

Figura 23 – Criação de um objeto gráfico

```
//objeto criado para receber uma classe gráfica
Object v1 =
        graph.insertVertex(
        parent,
        null,
        "\n" + nomeClasse + "\n"
        + linhaDivisao +
        conteudoClasse + "\n" +
        linhaDivisao,
        20,
        20,
        largura,
        altura,
        "fillColor=white;"+
        "strokeColor=black");

//objeto gráfico adicionado no vetor de classes gráficas
classesDesenhadas.add(v1);
```

Fonte: Primária

Observando ainda a Figura 23 pode-se destacar que o método “insertVertex” utilizado para criação de um grafo (este semelhante a uma classe UML), só é possível ser utilizado se o objeto “graph” existir. Este objeto “graph” é na realidade do tipo “mxGraph” que é utilizado para executar diversos dos métodos que a biblioteca JgraphX possui, ou seja, quando relaciona-se a um objeto gráfico, é necessário instanciar um objeto do tipo “mxGraph” e, no caso da ferramenta, este objeto chama-se “graph”, sendo assim é possível criar a classe, porém esta criação deve respeitar os oito parâmetros presentes no método “insertVertex”, são eles:

- O primeiro parâmetro refere-se ao local que o objeto mxGraph está inserido, o contexto ou “tabuleiro” que ele está inserido, para isto ocorrer é usado a variável “parent” que é do tipo Objeto e foi criado pela biblioteca JgraphX;
- O segundo parâmetro é uma *String* que representa um comando a ser usado por métodos da biblioteca JgraphX, porém para criação deste tipo de objeto gráfico nenhum comando específico é dado, para isso o termo “null” é usado;

- O terceiro é um dos mais importantes, uma *String* que será apresentada internamente do objeto gráfico, neste caso a *String* contém o nome da classe com um linha divisória e os atributos correspondentes à classe atual seguida de outra linha divisória;
- O quarto e quinto parâmetro são inseridos números inteiros que representam a posição horizontal e vertical, sendo que primeiro refere-se à horizontal (quanto maior o número mais à direita) e o outro à vertical (quanto maior o número mais abaixo). É válido destacar que todas as classes são criadas com coordenadas iguais pela ferramenta (20 e 20), pois a organização do posicionamento é realizada de forma automática por um método específico da biblioteca gráfica;
- O sexto e sétimo parâmetros são a largura e altura, dois inteiros que anteriormente receberam os valores através do cálculo do tamanho adequado da classe;
- O último parâmetro é o que faz o objeto gráfico "parecer" com uma classe UML, estas configurações são especificadas em uma espécie de código CSS[15] disponibilizados pela biblioteca JgraphX;

Assim sendo, ao final de todo este processo o objeto "v1" (declarado no início da Figura 23) é adicionado a um vetor específico de classes gráficas e assim, podem ser usadas na hora da representação final (geração das relações e apresentação do diagrama de classes).

3.4.3.2 Criação das relações gráficas

Sabe-se que um diagrama de classes UML possui diversos relacionamentos, e que todos possuem diferenças em sua aparência para que o analista consiga identificá-los rapidamente. A seguir será apresentado o método da biblioteca JgraphX responsável pela criação de todos os relacionamentos UML.

Considerando que existe o padrão de representação de relacionamentos UML, as formas que as relações existentes em um diagrama de classe são apresentadas variam de ferramenta para ferramenta, e, neste caso não é diferente. Isso ocorre devido à biblioteca gráfica utilizada que permite diversos tipos de "ligações" entre grafos (através de vértices), porém, para realizar os relacionamentos específicos de um diagrama de classes UML foram utilizados estilos específicos a fim de que cada relação estivesse de acordo com o padrão UML existente.

[15] Folha de estilo usado normalmente para uma página WEB

Sabendo das diferenças entre a representação das relações, o modo com que a ferramenta constrói as relações é através de outro método que a biblioteca JgraphX disponibiliza, o “insertEdge”. Assim como o método anterior serão apresentados os parâmetros necessários para, neste caso, a exposição de como a ferramenta cria o gráfico das relações. A Figura 24 apresenta a criação de uma associação binária entre duas classes:

Figura 24 – Método para criação de uma associação binária

```
graph.insertEdge(
parent, //parâmetro 1
null,   //parâmetro 2

nome2 + " (0,N) " + //parâmetro 3
nome1 + "\n\n" +
nome1 +
"(0,1) " +
nome2,

classe1, //parâmetro 4
classe2, //parâmetro 5
"" //parâmetro 6
+ "dashed=0;"
//linha pontilhada? 1-sim 0-nao
+ "startArrow=none;"
//modo da seta inicial
+ "endArrow=none;"
//modo de seta final
+ "sourcePerimeterSpacing=1;"
//distancia da base da relação da 1ª para a 2ª
+ "startSize=0;"
//tamanho da seta inicial
+ "endSize=0;"
//tamanho da seta final
+ "strokeWidth=1;"
//tamanho do nivel de negrito da linha
+ "labelBackgroundColor=white;"
//cor de fundo do nome da relação
+ "fontStyle=2;strokeWidth=1;strokeColor=black"
);
```

Fonte: Primária

O método “insertEdge” é responsável pela relação entre os objetos gráficos. Sabendo que estas “relações entre objetos” devem assemelhar com os relacionamentos UML, o principal parâmetro que é diferenciado nos métodos de criação das relações é o parâmetro 6, apresentado na Figura 24. Vale destacar que para cada tipo de relação que a ferramenta consegue formar, ou seja, uma associação unária, binária, uma agregação, uma composição, uma generalização e uma classe associativa, o método acima é executado com a principal

diferença sendo o estilo da relação, ou seja, a forma de definição do parâmetro 6, pois é este parâmetro que faz com que haja a diferença visual entre uma relação e outra.

Os parâmetros necessários para a criação de uma relação utilizando o método "insertEdge" da biblioteca gráfica são:

- O primeiro e segundo parâmetro são idênticos aos parâmetros do método "insertVertex";
- O terceiro parâmetro é uma *String* que se refere ao nome da relação, ou seja, um texto que será apresentado na seta de ligação desta relação;
- O quarto e quinto parâmetros são objetos gráficos do tipo "mxGraph". Estes são os objetos que se relacionarão entre si, ou seja, eles são criados com o método "insertVertex" e relacionados com o método atual (insertEdge);
- O sexto parâmetro é o de estilo da relação, assim sendo, é este estilo de relação que faz com que as ligações entre as classes gráficas envolvidas pareçam com relações UML e possam ser apresentadas pela ferramenta na etapa final.

Com base no conteúdo demonstrado, para realizar a criação das diversas relações existentes e interpretadas pela ferramenta, o estilo da relação no método acima é editado e assim, sua aparência reflete a relação que ela representa.

3.5 ESTUDO DE CASO

Esta seção tem como objetivo demonstrar a aplicação em execução, para isso será apresentado um exemplo de uso da ferramenta perante uma determinada situação, com o foco voltado para o usuário que a utiliza.

3.5.1 Definição do script SQL de entrada

Para iniciar este estudo de caso foi definido um script SQL válido, este apresentado na Figura 25.

Este script define três tabelas (clientes, cidades e estados) sendo que a tabela "clientes" possui uma chave estrangeira que referencia a tabela de "cidades" e esta referencia a tabela "estados". Ao executar a aplicação e o script SQL ser analisado, espera-se obter um diagrama de classes que represente, de acordo com os conceitos observados ao longo deste trabalho, as tabelas e relações presentes nele.

Figura 25 – script SQL válido

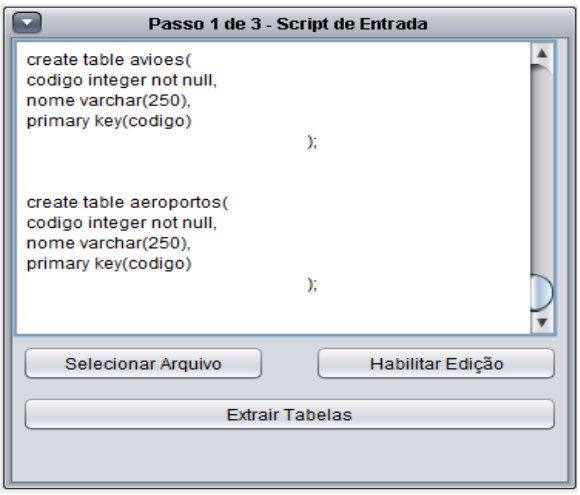
```
create table clientes(
codigo_cli integer not null,
nome_cli varchar(250) not null,
cpf_cli char(11) not null,
cidade integer not null,
foreign key(cidade) references
cidades(codigo_cid),
primary key(codigo_cli)
                );
create table cidades(
codigo_cid integer not null,
nome_cid varchar(250) not null,
estado integer not null,
foreign key(estado) references
estados(codigo_est),
primary key(codigo_cid)
                );
create table estados(
codigo_est integer not null,
nome_est varchar(250) not null,
sigla_est varchar(250) not null,
primary key(codigo_est)
                );
```

Fonte: Primária

3.5.2 Leitura do script SQL

A tela principal pode ser observada na Figura 26, esta tela está dividida em “telas internas” que representam as etapas a serem executadas. Ao inicializar a ferramenta, por padrão existem alguns exemplos de comandos SQL válidos na “área de texto”.

Figura 26 – Tela principal da ferramenta (passo 1 de 3)



Fonte: Primária

É possível observar na Figura 26 que existem os botões chamados “Selecionar Arquivo”, “Habilitar Edição” e “Extrair Tabelas”, estes botões são responsáveis respectivamente por proporcionar a importação de um arquivo que contenha o script SQL, habilitar a edição do conteúdo presente na área de texto da ferramenta e por iniciar o ciclo de extração de tabelas encontradas.

Para iniciar o processo de criação do diagrama de classes, é necessário um script SQL válido na área de texto, do contrário a segunda etapa não poderá ser realizada. A ferramenta foi projetada para disponibilizar duas formas de entrada para o script SQL, são elas:

- **Através do campo de texto:** o usuário habilita a edição do conteúdo presente na área de texto e insere o conteúdo manualmente;
- **Através do botão “selecionar arquivo”:** O usuário importa um arquivo que contenha o script SQL válido, lembrando que é possível mesmo nesta situação editar o conteúdo após a importação.

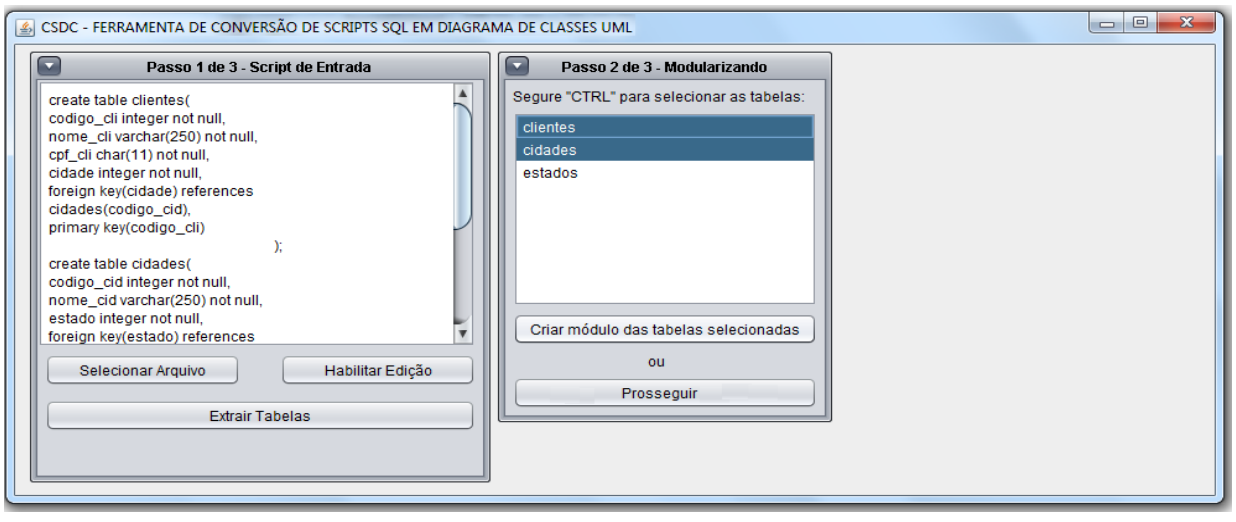
Assim sendo, após a definição de qual será o script analisado (neste exemplo será considerado o script da Figura 25) o usuário poderá seguir para a próxima etapa pressionando o botão “Extrair Tabelas” (presente na Figura 25).

3.5.3 Modularização

Após a extração das tabelas (tratadas neste momento como classes UML na lógica da programação), a segunda tela interna surge com uma lista e outros botões, nesta lista são apresentados os nomes das tabelas identificadas no script SQL inserido anteriormente.

Observa-se na Figura 27 a segunda tela interna (com o título “Passo 2 de 3 – Modularizando”) juntamente com as tabelas extraídas do script SQL base deste exemplo.

Figura 27 – Tela principal da ferramenta (passo 2 de 3)



Fonte: Primária

Com as tabelas identificadas e sendo representadas na lista, é possível a tomada de decisão referente à modularização, ou seja, pode-se modularizar as tabelas e organizá-las de acordo com uma necessidade específica ou optar por seguir para a última etapa sem a modularização, neste caso só poderá gerar o diagrama de classes de todas as tabelas extraídas.

Para dar continuidade ao exemplo, pode-se considerar a modularização das tabelas “cidades e clientes” (módulo 1) e também “cidades e estados” (modulo 2).

Independente da modularização realizada para prosseguir à última etapa, deve-se apertar o botão “Prosseguir”, observado na Figura 27.

3.5.4 Criação do diagrama de classes

A última etapa dentro da execução da ferramenta é a seleção de qual será o grupo de tabelas (módulos criados ou todas as tabelas), que serão apresentadas em forma de diagrama de classes. A Figura 28 apresenta esta etapa.

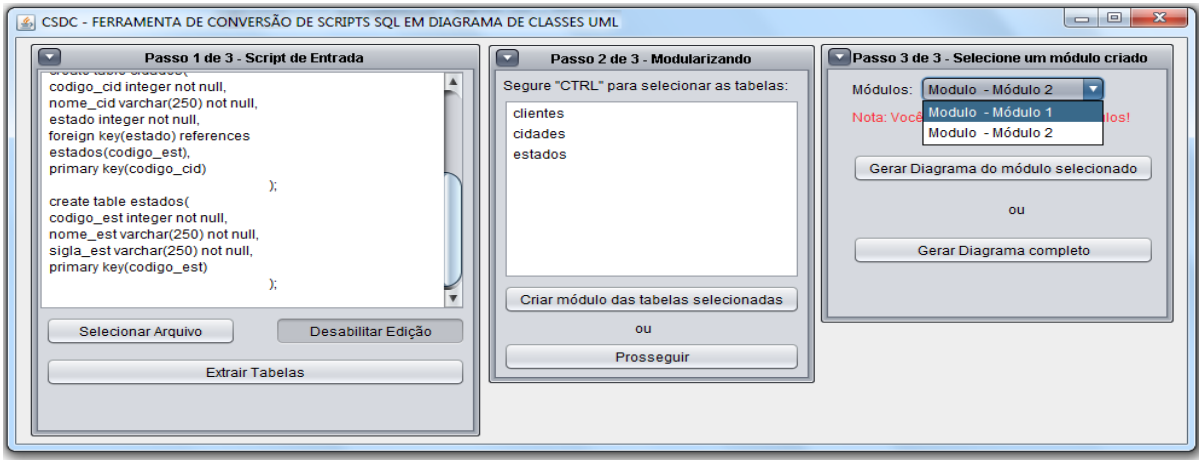


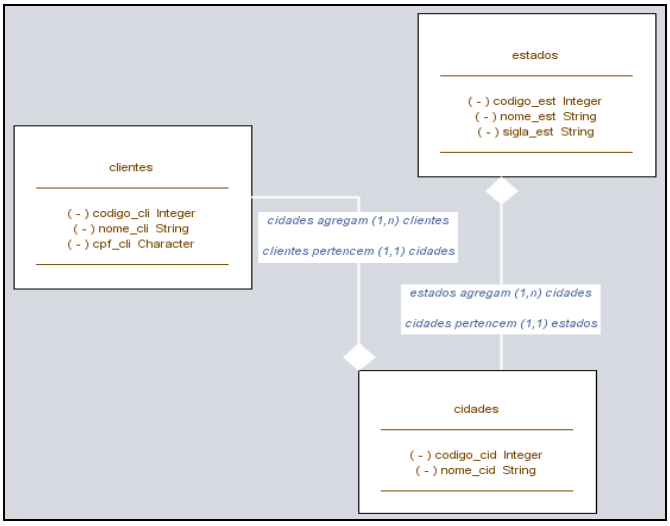
Figura 28 – Tela principal da ferramenta (passo 3 de 3)

Fonte: Primária

Após a definição de qual serão as tabelas envolvidas na criação do diagrama de classes, através da seleção de um módulo ou com o botão “Gerar Diagrama completo” (Figura 28), ocorre a criação do documento XML e todos os mapeamentos são realizados.

Observa-se o resultado, baseado no exemplo apresentado, na Figura 29, aonde o diagrama de classes é formado de acordo com todas as tabelas extraídas do script SQL.

Figura 29 – Diagrama de classes resultante



Fonte: Primária

Assim sendo, nesta seção foram apresentados os funcionamentos da ferramenta com a perspectiva do usuário, de forma que, com base no exemplo apresentado, pode-se demonstrar um dos possíveis resultados que a ferramenta promove.

4 CONSIDERAÇÕES FINAIS

O presente trabalho apresentou a ferramenta CSDC que realiza a conversão de um script SQL para um diagrama de classes. Os objetivos foram concluídos, embora a ferramenta necessite uma maior quantidade de testes.

Este trabalho englobou vários estudos, como a estrutura de um banco de dados relacional, alguns comandos SQL, a necessidade dos diagramas UML em um projeto de sistema, a importância do planejamento na área da informática, a utilidade de um documento XML em um processo de mapeamento, o desenvolvimento do processo de engenharia reversa através da lógica utilizada pela ferramenta apresentada.

Sendo assim as principais contribuições deste trabalho são: Os estudos envolvidos no planejamento da ferramenta; a definição das regras de mapeamento realizadas em diversas etapas e de diferentes formas na aplicação; o sistema de modularização criado; além da própria criação da ferramenta.

Este trabalho deixa lacunas para a realização de trabalhos futuros, como estudo para o aperfeiçoamento de cada processo apresentado, permitindo a conversão bidirecional, ou seja, criar um diagrama de classes (com modularização) e deste gerar um script SQL; a melhora das funcionalidades da CSDC, permitindo a edição do diagrama de classes; pesquisas para aperfeiçoar ou desenvolver processos de conversão mais simples e eficazes; promover uma ferramenta que identifique outros comandos que possam ser usados no mapeamento (*Triggers*, *Funcions*), com o intuito de gerar um diagrama de classes mais completo (com métodos) ou mesmo outros tipos de diagrama UML.

5 REFERÊNCIAS

BRAUDE, Eric. *Projeto de Software*. Porto Alegre. Bookman, 2005.

DATE, C. J. *Introdução a Sistemas de Bancos de Dados.* 8. ed. Rio de Janeiro: Campus, 1983.

FERRARI, A. F. *Crie banco de dados em MySQL* São Paulo: Digerati Books, 2007.

GUEDES, Gilleanes T. A. *UML 2 Uma Abordagem Prática*. 2. ed Novatec, 2011.

HEUSER, Carlor Alberto. *Projeto de Banco de Dados*. 4. ed. Porto Alegre: Sagra&Luzzatto, 1998.

_____. *Projeto de Banco de Dados*. 6. ed. Porto Alegre: Bookman, 2009.

HORSTMANN, Cay. *Conceitos de Computação com o Essencial de JAVA.* 3. ed. São Paulo: Bookman, 2003.

IBM, *Rational Rose*. Disponível em <www.ibm.com/software/awdtools/developer/rose/>. Acesso em: 22 jun. de 2012.

JGRAPH. Disponível em < http://www.jgraph.com/>. Acesso em: 25 jun. de 2012.

MEIRELES, Manuel. *Sistemas de informação: quesitos de excelência dos sistemas de informação operativos e estratégicos. Volume 1 da Série: Indicadores Gerenciais.* 2. ed. São Paulo: Arte & Ciência, 2004.

OFFICE. Microsoft InfoPath 2003. Disponível em <http://office.microsoft.com/pt-br/infopath-help/sobre-o-xpath-HP001096732.aspx>. Acesso em: 25 jun. de 2012.

SAMPAIO, Cleuton. *Guia do Java: Enterprise Edition 5: desenvolvendo aplicações corporativas.* Rio de Janeiro: Brasport, 2007.

SERSON, Roberto Rubistein. *Programação orientada a objetos Java.* Rio de Janeiro: Brasport, 2007.

SYBASE INC, *Power Designer*. Disponível em <http://www.sybase.com.br/products/modelingdevelopment/powerdesigner >. Acesso em: 22 jun. de 2012.

TAKAI, Osvaldo Kotaro; ITALIANO, Isabel Cristina; FERREIRA, João Eduardo. Introdução à Banco de Dados. Disponível em: <http://www.ime.usp.br/~jef/apostila.pdf>. Acesso em: 16 nov. 2011.

YOURDON, Edward. *Análise estruturada moderna*. 10. ed. Rio de Janeiro: Campus, 1990.

W3C, World Wide Web Consortium. Disponível em <http://www.w3.org/standards/xml/core>. Acesso em: 19 jun. de 2012.