FELIPE MONTEIRO LIBERAL

XMI2DB2XMI - UMA FERRAMENTA PARA GERAÇÃO DA ESTRUTURA DE UM BANCO DE DADOS USANDO ARQUIVOS XMI

FLORIANÓPOLIS

2011

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CURSO DE GRADUAÇÃO
EM CIÊNCIAS DA COMPUTAÇÃO

XMI2DB2XMI - UMA FERRAMENTA PARA GERAÇÃO DA ESTRUTURA DE UM BANCO DE DADOS USANDO ARQUIVOS XMI

Trabalho de Conclusão de Curso submetido à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Bacharel em Ciências da Computação.

FELIPE MONTEIRO LIBERAL

Florianópolis, Dezembro de 2011

XMI2DB2XMI - UMA FERRAMENTA PARA GERAÇÃO DA ESTRUTURA DE UM BANCO DE DADOS USANDO ARQUIVOS XMI

Felipe Monteiro Liberal

'Este trabalho de conclusão de curso foi julgado adequado para obtenção do Título de Bacharel em Ciências da Computação, e aprovado em sua forma final pelo Curso de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina.'

_____

Prof. Frank Augusto Siqueira, Dr. Ing.
Co-Orientador / Responsável

_____

Vinicius Moll, Me. Ing.
Orientador

_____

Prof. Vitório Bruno Mazzola, Dr.
Coordenador do Curso de Graduação em Ciências da Computação

Banca Examinadora:

_____

Prof. Renato Cislaghi, Dr. Ing.
Presidente

_____

Prof. Carina Friedrich Dorneles, Dr.

_____

Vinicius Moll, Me.

Agradecimentos

Gostaria de agradecer aos meus pais pelo apoio, à minha esposa - pessoa fundamental para chegar ao final desta jornada. Ao meu orientador Vinícius pelos incentivos e conversas. E aos mestres que me orientaram durante todo o período da graduação.

Resumo do Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciências da Computação.

XMI2DB2XMI - UMA FERRAMENTA PARA GERAÇÃO DA ESTRUTURA DE UM BANCO DE DADOS USANDO ARQUIVOS XMI

Felipe Monteiro Liberal

Dezembro/2011

Orientador: Vinícius Moll, Me. Ing.
Palavras-chave: XMI, Geração Automática de Código, Templates, Object Modeling
Número de Páginas: 48

Com ênfase na geração automática de código, a ferramenta que este trabalho implementa, tem a função de facilitar e agilizar o processo de geração automática da estrutura de dados de um sistema gerenciador de banco de dados. Considerando a entrada de um Arquivo XMI (versão 1.2) válido, produzido por uma ferramenta de modelagem com suporte a este padrão, o usuário tem a opção de escolher com qual SGBD deseja trabalhar (ORACLE ou Microsoft MSSQL Server). Realizando a etapa de configuração das informações necessárias à conexão com o SGBD, a ferramenta faz o processamento do arquivo de entrada dividindo-o em informações referentes às classes, associações e atributos do modelo convertidos pelo template de conversão em tabelas, relacionamentos e colunas do banco de dados respectivamente. A ferramenta permite a transformação da estrutura de um banco de dados relacional em um arquivo XMI que possa ser importado por uma ferramenta de modelagem com suporte ao padrão XML Metadata Interchange. Com este processo é possível partir da concepção de um sistema e gerar a parte estrutural dos dados, ou então, gerar a documentação de um sistema já existente.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Graduated in Computer Science.

XMI2DB2XMI - UMA FERRAMENTA PARA GERAÇÃO DA ESTRUTURA DE UM BANCO DE DADOS USANDO ARQUIVOS XMI

Felipe Monteiro Liberal

Dezembro/2011

Advisor: Vinícius Moll, Me. Ing.
Keywords: XMI, Code Generation, Templates,
Page count: 48

Using the automatic code generation, the function of this tool is to facilitate and agility the process of going from documentation and specification step to the structural part of the application data. Considering the input of a valid XMI file (version 1.2), produced by a modeling tool that supports this pattern, the user has the option to choose wich DBMS you want to work (ORACLE or Microsoft MSSQL Server). Performing the configuration step and informing necessary information for connection with the SGBD, the tool processes the input file by splitting it into classes, associations and attributes of the desing model and are converted by the template to tables, relationships and columns of the database respectively. The tool will allow transforming the structure of a relational database in an XMI file that can be imported by a modeling tool that supports the XML Metadata Interchange standard. With this process it will be possible to go from the conception of a system and generate the database structure, or generate the documentation for an existing system.

Sumário

1. Introdução

Analisando a forma de desenvolvimento dos softwares atuais, os desenvolvedores e analistas pecam no planejamento e concepção dos seus sistemas. Muitos gerentes de projetos deixam por último a etapa de documentação do sistema no processo de desenvolvimento, porém podemos abordar os projetos atuais na ordem inversa e iniciar um desenvolvimento pela documentação. A documentação é uma etapa do processo desenvolvimento que consome uma, considerável, quantidade de horas de desenvolvimento. A vantagem deste tempo gasto é que estas horas utilizadas beneficiam no entendimento do software e na possibilidade de geração automatizada de código.

A criação dos algoritmos de programação que são elaborados manualmente são sucetíveis a erros de sintaxe e lógica. Como exemplo podemos citar a confecção dos scripts que criam a estrutura de dados de um Sistema Gerenciador de Banco de Dados (SGBD), como uma tarefa repetitiva e com grande chance de conter erros de sintaxe. Os scripts confeccionados manualmente por uma equipe, pode não seguir um padrão nos tipos e formatos das colunas e nomenclatura. Estes pontos dificultam o entendimento do código e sua posterior manutenção, além disso, tem-se o tempo gasto no desenvolvimento destes scripts. Além deste ponto ao iniciar o desenvolvimento pela documentação, tem-se outro ponto positivo, como a facilidade de compreensão e abstração do sistema por utilizar linguagens de altíssimo nível, comparáveis à linguagem natural.

As ferramentas existentes no mercado que possuem um objetivo semelhante à ferramenta desenvolvida por este trabalho prezam pelo desenvolvimento do código fonte das aplicações. Porém, existe um exemplo semelhante, a ferramenta AxGen que faz a geração da estrutura do SGBD a partir de um arquivo XMI. As outras ferramentas que manipulam arquivos no padrão XMI são voltadas a geração automática de definições, métodos e procedimentos do código fonte.

Para a elaboração desta ferramenta é necessário o estudo do padrão XMI para conhecer as tags utilizadas, bem como definir uma forma de leitura e mapeamento dos

dados contidos no arquivo para que se possa gerar a estrutura do banco de dados a partir do modelo. Segundo a OMG o padrão XMI foi desenvolvido para facilitar a troca de metadados entre diferentes ferramentas de modelagem de dados que seguem os padrões da UML. Também é necessário o estudo aprofundado dos conceitos de bancos de dados relacionais e sua estrutura. A ferramenta deve ter os arquivos necessários para prover a conexão aos dois sistemas gerenciadores de bancos de dados (SGBD), após esta etapa é executada a criação da estrutura do banco de dados relacional a partir do modelo documentado no padrão XMI. Realiza-se - também - a criação do modelo documentado em um arquivo XMI a partir da estrutura de um banco de dados existente. Para realizar estes pontos a ferramenta utiliza templates para criar os arquivos de saída, logo, para trabalhos futuros, pretende-se expandir esta ferramenta para que permita utilizar os templates especificados para padrão XMI, que gere o código-fonte da estrutura básica do sistema projetado.

1.1: Objetivos

Baseado no contexto especificado na seção anterior resume-se o objetivo geral do trabalho em:

Desenvolver uma ferramenta que atue em ambos os sentidos da geração da estrutura de um banco de dados relacional e da documentação de um sistema através de um documento XMI gerado por uma ferramenta de modelagem de sistemas que utiliza o padrão UML.

Mais especificamente, deseja-se alcançar ainda os seguintes objetivos:

- Armazenar em memória as tags padronizadas do arquivo XMI. O padrão XMI desenvolvido pela OMG, possui tags específicas voltadas a linguagem UML. A ferramenta desenvolvida realiza o armazenamento destas tags e obtém informações que geram a estrutura de dados do SGBD.
- Montar a estrutura de dados necessária com as informações do arquivo XMI. Com posse das informações obtidas no item acima, a ferramenta relaciona os dados do arquivo XMI e monta a estrutura de dados que é aplicada sobre o template para o

banco de dados escolhido.

- Elaborar a ferramenta que trabalhe com a leitura de templates. A ferramenta gera as informações que são armazenadas no SGBD aplicando as informações dos dois itens acima sobre templates. Os templates contém a parte fixa de código e variáveis que são substituídas em tempo de execução.

- Estabelecer a conexão com o banco de dados. A ferramenta possui uma alternativa de escolha de qual SGBD é utilizado (Oracle ou MSSQL Server).

- Criar a estrutura do banco de dados. Através dos scripts gerados a ferramenta executa-os e gera as colunas, tabelas e os relacionamentos da estrutura de dados.

- Gerar um arquivo XMI de saída. O usuário tem a opção de gerar um arquivo XMI a partir da conexão com um banco de dados. A ferramenta armazena as informações das colunas, tabelas e seus relacionamentos e monta a estrutura de dados no padrão das tags XMI.

- Definir as regras de mapeamento utilizadas para criar o modelo de dados de uma estrutura de dados de um SGBD já existente.

- Verificar a eficiência da ferramenta em casos de teste.

1.2: Organização do documento e convenções

No Capítulo 2 deste documento é apresentada uma visão geral dos conceitos abordados neste projeto como Modelos Orientados a Objetos e Modelos UML. Também é abordado a visão geral dos modelos relacionais de dados. Este capítulo contém como os modelos relacionais e o modelo orientado a objetos são representados na UML. Por fim, tem-se as definições formais e uso de arquivos do padrão XML e XMI.

O Capítulo 3 descreve as ferramentas relacionadas ao tema deste trabalho existentes no mercado. É realizada uma análise comparativa dos trabalhos relacionados e descrito o principal motivo do desenvolvimento desta ferramenta que não é encontrada nos trabalhos mencionados.

Na leitura do Capítulo 4 pode-se observar a forma de desenvolvimento da ferramenta, bem como a tecnologia utilizada e o método de transformação dos dados do arquivo XMI. Neste ponto são descritos os fluxos da ferramenta e a forma de utilização.

Por fim, o Capítulo 5 apresenta as considerações finais, incluindo uma breve análise do que foi alcançado com este trabalho e algumas possibilidades para estudos futuros.

2. Fundamentação Teórica

O usuário que utiliza esta ferramenta deve ter um prévio conhecimento de alguns conceitos importantes. Neste capítulo é abordado a relação entre o modelo orientado a objetos e a UML, com o domínio desta relação, passa-se a explanar os conteúdos necessários sobre o modelo relacional pertinentes a este trabalho. A representação, tanto do modelo relacional, quanto do modelo orientado a objetos, através da UML é exemplificado na seção 2.3. Por fim, as definições dos arquivos XML e XMI são abordadas na seção 2.4.

2.1: Modelagem Orientada a Objetos e a UML

De acordo com a Object Management Group (OMG), a Linguagem de Modelagem Unificada (UML) é uma linguagem visual para especificar, construir e documentar os artefatos dos sistemas. (LARMAN,2007)

Fazendo uma analogia com a construção civil pode-se considerar a fase de projetos como a mais importante para o bom andamento das obras e futuras manutenções de uma edificação. Para a realização de uma reforma em uma das salas deste edifício, é de extrema importância, que o responsável pela obra saiba onde encontram-se os eletrodutos e a tubulação da parte hidráulica antes que se faça um furo em uma das paredes, para evitar complicações futuras.

Assim como é necessário o estudo da planta elétrica, hidráulica e da planta baixa de um edifício antes da realização de uma reforma, as correções e melhorias em um software desenvolvido por um analista, também precisa de um estudo prévio da sua documentação. Caso seja necessária a implementação destas alterações por um outro analista que desconheça o código-fonte. Isto tudo, para que a alteração em um ponto do código não reflita em implicações futuras em outras funcionalidades do sistema.

Para que não ocorra este tipo de imprevisto descrito anteriormente, a OMG desenvolveu o padrão UML, que foi conceituado no início deste capítulo, para a modelagem e documentação de sistemas.

2.1.1: Visão Geral da UML

Em (FOWLER,2003), a UML pode ser utilizada de três formas diferentes:

- usar a UML como rascunho, ou seja, possibilita o desenho da estrutura de um sistema através de desenhos não formalizados, para dar uma noção geral do que abrange o sistema.
- usar a UML como a planta de um software, que neste caso, necessita de diagramas detalhados.
- utilizar a UML como uma linguagem de programação, o que possibilitaria, a partir de uma especificação muito detalhada, gerar o corpo do sistema em questão automaticamente através das ferramentas Case de modelagem de sistemas.

Alguns aspectos, no decorrer da concepção dos sistemas, são mais fáceis de compreender quando se lê algumas linhas de código, pois segundo (BOOCH,2005), o código em formato de texto é uma forma maravilhosamente mínima e direta para escrever expressões e algoritmos. Mesmo que o programador adote esta forma de desenvolvimento, em algum momento ele realizou uma modelagem mental ou rabiscou em alguns pedaços de papéis, quadros ou guardanapos as idéias que servirão para conceber este novo sistema.

Como cada sistema tem suas particularidades, poderá ser melhor uma modelagem em forma de texto para uns; para outros, poderá ser mais conveniente uma modelagem através de diagramas e gráficos. Logo, um desenvolvedor pode usar a UML para escrever ou desenhar o seu modelo, e com isso qualquer outro desenvolvedor que tenha conhecimento dos padrões adotados pela UML poderá interpretar as funcionalidades e prestar uma manutenção do código com maior facilidade.

A UML permite especificar e construir um modelo preciso de um determinado sistema. Isto permite que não haja ambiguidades e que os modelos sejam completos, ou seja, contempla as decisões importantes em termos de análise, modelagem e implementação que devem ser tomadas para a implantação e desenvolvimento de softwares complexos.

2.1.2: Modelo orientado a objetos

A visão atual no desenvolvimento de sistemas, adota uma perspectiva orientada a objetos. Ou seja, nessa visão os elementos necessários para o desenvolvimento de sistemas é um conjunto de objetos ou classes. Em (BOOCH,2005), existe uma definição simplificada para objetos e classes, segundo o autor, *um objeto é alguma coisa geralmente estruturada a partir do vocabulário do espaço do problema ou do espaço da solução; uma classe é a descrição de um conjunto de objetos comuns.* Em função deste conceito, os objetos possuem uma identidade, um estado e um comportamento.

O desenvolvimento de software, utilizando um método orientado a objetos, consolidou-se na construção de sistemas. Esta certeza se deve ao fato da orientação a objetos estar presente em todos os tipos de domínios de problemas, abrangendo pequenos e grandes sistemas de baixa e alta complexidade. Além da gama de sistemas, a visualização, a especificação, a construção e a documentação de sistemas orientados a objetos é exatamente o objetivo da UML.

2.1.3: Utilizando Diagramas de Classes

Segundo (LARMAN,2007), a UML possui os diagramas de classes para ilustrar classes, interfaces e suas associações. Este tipo de diagrama também é utilizado para exibir a modelagem estática de objetos, bem como modelar o domínio e aplicar o diagrama de classe em uma perspectiva conceitual. Esta última característica nos permite criar um relacionamento direto entre um diagrama de classe e a modelagem e estrutura de um banco de dados relacional que é mais detalhada no próximo capítulo.

Conforme (BOOCH,2005), os diagramas de classes da UML podem ser considerados como um superconjunto dos diagramas voltados para modelagem conceitual de bancos de dados, mais conhecidos como diagramas de entidade-relacionamento (ER). A principal diferença entre os diagramas de classes e os diagramas de entidade-relacionameto é que o último tem seu foco apenas nos dados, já o primeiro, abrange a modelagem de comportamentos. Estes modelos de comportamentos traduzem-se no banco de dados físico em procedimentos armazenados. Além destas características, os

diagramas de classes mostram - graficamente - uma coleção de vértices e arcos.

Como principal uso dos diagramas de classes podemos apontar a função de modelar a parte estática de um sistema, oferecendo suporte para os requisitos funcionais do sistema, ou seja, determinar os serviços e funções que estarão a disposição do usuário final. (BOOCH,2005) define três usos básicos para os diagramas de classe:

- Para fazer a modelagem do vocabulário de um sistema.
- Para fazer a modelagem de colaboração simples.
- Para fazer a modelagem do esquema lógico de um banco de dados.

A Figura 1 mostra os aspectos de um diagrama de classe.

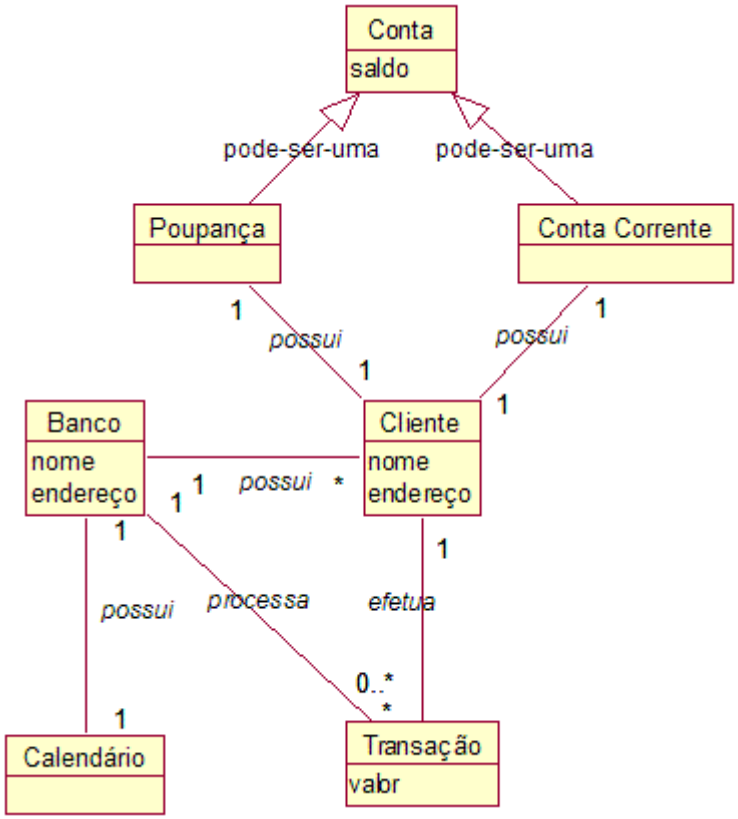

Figura 1: Exemplo de um Diagrama de Classes do domínio de uma agência bancária

2.2: Modelo Relacional

Resgatando o histórico, (RIORDAN,1999), o modelo relacional é baseado em um conjunto de princípios matemáticos elaborado, principalmente, a partir da teoria dos conjuntos e lógica de predicados. Estes princípios foram primeiramente aplicados ao campo da modelagem de dados em 1960 pelo Dr. EF Codd em seguida mantido e aprimorado por Chris Date e Hugh Darwen como um modelo geral de dados.

Para tal organização os bancos de dados relacionais definem:

- Como os dados podem ser representados (estrutura de dados).
- A forma que os dados podem ser garantidos (integridade dos dados).
- E as operações sobre os dados que podem ser realizadas (manipulação dos dados).

Como explana o documento de introdução ao Oracle, (GRAVINA, 2000), o banco de dados relacional utiliza as relações ou as tabelas bidimensionais para armazenar suas informações. Como exemplo, pode-se citar o armazenamento e informações de todos os funcionários de uma empresa. No caso do banco de dados ser relacional permite-se criar várias tabelas que possuem relacionamento com a tabela de funcionário, como por exemplo, as tabelas de departamentos e dependentes.

Pode-se considerar o modelo de dados como o nível mais alto de abstração de um banco de dados. Os modelos de dados são expressos em forma de tabelas, colunas e relacionamentos que serão abordados abaixo:

2.2.1: Tabelas

Qualquer coisa sobre a qual o sistema precisa para armazenar informações é um conceito simples do termo tabela. Durante as reuniões de definição de escopo do projeto e de modelagem do sistema, os analistas e clientes trocam informações utilizando substantivos e verbos. A maioria dos substantivos descritos podem tornar-se

tabelas do modelo de dados. Ao utilizarem frases como: ”Os clientes compram produtos”, ”Os Funcionários vendem produtos”, ou, ”Os fornecedores nos vendem produtos”. Os substantivos ”Clientes”, ”Produtos” e ”Fornecedores” são classificados como tabelas do modelo.

A maioria das tabelas e eventos do mundo físico: clientes, produtos ou vendas. Podem ser caracterizados como exemplos concretos de tabelas. Mas as tabelas também podem modelar conceitos abstratos. Como exemplo pode-se citar uma tabela abstrata que modela a relação entre outras tabelas. O fato de que um certo representante de vendas ser responsável por um determinado cliente, é um exemplo da questão levantada.

2.2.2: Colunas

Após a definição das tabelas do modelo precisa-se avaliar que tipo de informação deseja-se armazenar da tabela em questão. Para isso podemos enumerar as colunas daquela tabela. Ao utilizar a tabela ”Cliente” tem-se a opção de armazenar informações como: nome, endereço, telefone e outras informações pertinentes à tabela mencionada. Para determinar as colunas a serem incluídas é preciso uma análise criteriosa para atender todas as requisições que o cliente solicitou, além disso precisa-se oferecer a flexibilidade para lidar com as perguntas que o sistema do cliente irá solicitar agora, mas também prever as solicitações que o sistema pode vir a pedir no futuro.

2.2.3: Relacionamentos

Um modelo de dados relacional não é especificado utilizando apenas as tabelas soltas e as respectivas colunas. Existe relacionamentos entre as tabelas do modelo. No nível conceitual, as relações são simplesmente associações entre tabelas. A afirmação utilizada ao localizar as tabelas: ”Os clientes compram produtos” indica a existência de uma relação entre a tabela de clientes e a tabela de produtos. O número de tabelas envolvidas no relacionamento é o grau do relacionamento.

2.3: Impedância entre os modelos

Impedância é um conceito utilizado na engenharia elétrica, porém, ao utilizá-lo no desenvolvimento de software, condiz com a diferença existente entre o modelo de dados orientado a objetos e o modelo de dados relacional.

Como abordado no tópico 2.2, o modelo relacional organiza todos os dados em tabelas separando suas informações em colunas. Cada registro é representado em uma linha, e as colunas facilitam a divisão e a interpretação dos dados de uma determinada tabela. Se os dados forem complexos demais para serem representados em uma grade bidimensional, outras tabelas são criadas para conter as informações *relacionadas*.

O modelo de dados orientado a objeto não está limitado a manter as informações em tabelas e colunas. Em vez disso, o desenvolvedor cria uma definição (um modelo), que descreve completamente uma determinada classe de informação. Cada registro (objeto) é uma instância específica daquela classe. Assim, cada objeto contém todos os itens de informação para um, e apenas um, registro. Mas isso não é tudo. As definições de classe também podem incluir trechos de programação, denominados métodos, que agem sobre os dados descritos pela classe. Não há uma concepção análoga no modelo relacional.

2.4: XML e XMI

XML (Extensible Markup Language) é uma linguagem para marcar documentos que contêm informações estruturadas. Ao contrário do HTML, que foi projetado para marcar documentos com estruturas fixas, o XML foi projetado para marcar o documento de forma arbitrária, ou seja, o XML é uma meta-linguagem que permite descrever uma linguagem de marcação, definindo marcadores e a relação estrutural entre eles. Pelo fato de ser uma linguagem de marcação arbitrária, o XML entrou no mercado como uma linguagem forte para a questão de interoperabilidade de sistemas, facilitando assim, a troca de informação entre sistemas que funcionam em plataformas distintas.

A definição de tipo de documento, ou simplesmente DTD, contém as regras que definem quais os marcadores (tags) que podem ser usados e quais os valores que são

válidos em um documento XML. O DTD era usado antes de surgir o padrão XML. Desde o padrão SGML a DTD era utilizada como forma padrão de validação de documentos dessa linguagem, porém desde 2001 que ele vem sendo substituído aos poucos pelo XML Schema(XSD). O XML Schema Definition é uma alternativa baseada em XML ao DTD. A função do XSD é descrever a estrutura de um documento XML.

Conforme descrito no portal W3Schools (W3SCHOOLS,2011) o XML Schema possui as seguintes propriedades:

- Define os elementos que poderão aparecer no documento XML.
- Define os atributos que poderão aparecer no documento XML.
- Define quais são os elementos filhos.
- Define a quantidade de elementos filhos.
- Define a ordem dos elementos filhos.
- Define quando um elemento é vazio ou pode incluir textos.
- Define os tipos de dados dos elementos e atributos.
- Define valores padrões ou fixos para elementos e atributos.

O XML foi utilizado como base para outras tecnologias. O XMI (*XML Metadata Interchange*)é um exemplo de padrão, com base no XML, voltado para a área de modelagem de sistemas. Uma vez que os arquivos XMI utilizam o padrão XML como base, todos os esforços produzidos para a evolução do XML afetam indiretamente as abordagens adotadas aos arquivos XMI. Ou seja, toda ferramenta que facilita a utilização da linguagem XML facilita, também, a utilização dos arquivos XMI. Todas as tecnologias abordadas pela W3C para a evolução do XML são estendidas ao XMI.

O XMI não estende o XML, na verdade o XMI é construído com base no XML. Por sua vez o XML é construído com base no Unicode, que é a especificação padrão de vários idiomas e que representa dados através de caracteres. Da mesma forma o XMI é construído com base no XML para representar objetos. A Figura 2 ilustra este conceito.

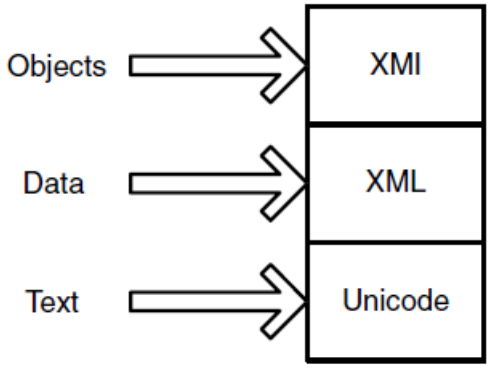

Figura 2: XMI construído com base no XML

Todo documento XMI é um documento XML e todo esquema XMI é um esquema XML, isto porque o XMI não é estendido do XML e sim baseia-se na sua especificação. A Figura 3 ilustra esta relação.

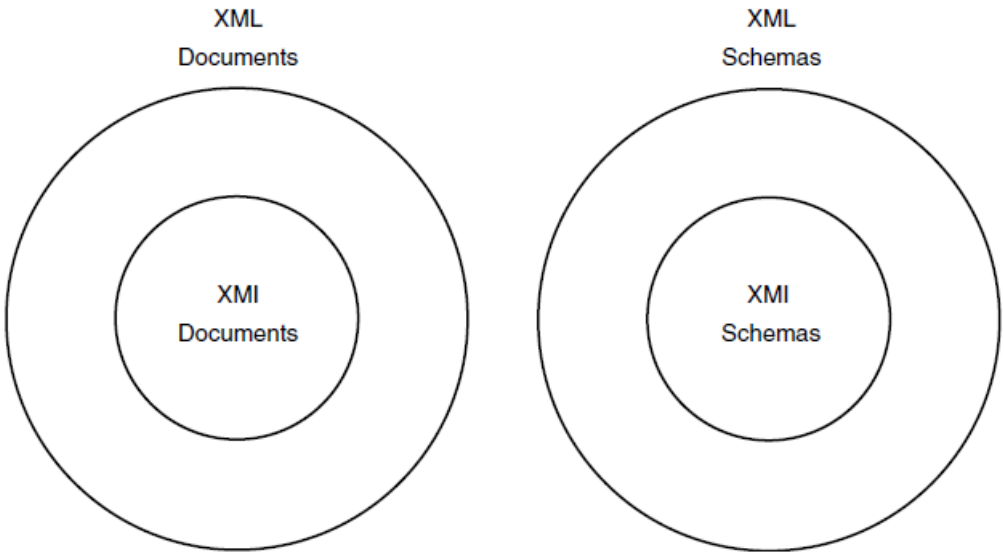

Figura 3: Documentos e esquemas XMI pertencem ao universo de documentos e esquemas XML

2.4.1: Propriedades e Benefícios do XMI

O XMI teve como objetivo principal preencher uma lacuna deixada entre o uso da linguagem XML e a Orientação a Objetos. O XMI fornece um padrão para mapear objetos definidos na UML para o formato XML. Também permite mapear padrões de modelo em esquemas XML. Com isso o XMI permite a troca de informações entre softwares que dão suporte a este padrão.

Podemos explanar alguns outros benefícios do XMI como:

- XMI ajuda na produção de documentos XML que podem ser facilmente trocados.
- XMI utiliza tecnologias da linguagem XML.
- XMI permite criar documentos simples e aperfeiçoá-los com a evolução da aplicação.
- XMI permite adequar as representações em XML em objetos e modelos.
- XMI permite trabalhar com os dados e os metadados.

2.4.2: XMI e os Metadados

O padrão XMI permite trabalhar com os dados e metadados. Tudo dependerá do ponto de vista. Por exemplo, por um lado o modelo dos objetos pode ser considerado como um meta-dado, mas pelo ponto de vista de uma ferramenta de modelagem podemos dizer que o modelo de objetos é apenas os dados de entrada.

Para utilizar as vantagens do XMI sobre o modelo de objetos, deve-se definir um meta-modelo. Como exemplo podemos citar um modelo UML que é uma instância do meta-modelo UML. O XMI permite a troca de informações entre estes meta-modelos. Quando se cria modelos UML para serem usados com XMI, não precisa se preocupar com a representação XML dos objetos definidos. De forma semelhante, um sistema que implemente o padrão XMI possibilita o trabalho direto com seus objetos, melhor do que trabalhar com elementos e atributos XML.

A Figura 4 mostra um trecho de código de um arquivo XMI que contém as informações de uma classe *"Customer"*. No início do arquivo constam informações referentes ao padrão estabelecido pela OMG, como versão do XML, versão do XMI, os namespaces do XML e as especificações do padrão XMI. Em seguida o elemento *"uml:Model"* identifica as informações do modelo de dados. O elemento *"packageElement"* armazena as informações da classe *"Customer"* e informa o tipo, o identificador e o nome da classe. Em seguida, existe uma lista de atributos que pertencem a classe *"Customer"* e armazenam as informações pertinentes aos atributos da classe.

```
<?xml version="1.0" encoding="utf-8"?>
<xmi:XMI xmi:version="2.1" xmlns:uml="http://schema.omg.org/spec/UML/2.1.1/uml.xml"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:L2=
"http://schema.omg.org/spec/UML/2.1.1/schemas/StandardProfileL2.xmi" xmlns:L3=
"http://schema.omg.org/spec/UML/2.1.1/schemas/StandardProfileL3.xmi" xmlns:Cx_Profile=
"http://schemas.microsoft.com/UML2.1.2/CSharpProfile.xmi">
  <uml:Model xmi:id="64fe5057-3500-4ef6-8fa7-1841a3d6deaa" name="Modelo de Dados">
    <profileApplication>
      <appliedProfile xmi:type="uml:Profile" xmi:idref="Cx_Profile" />
    </profileApplication>
    <packagedElement xmi:type="uml:Class" xmi:id="5678b355-ad89-49fb-b681-0be57bf4c2ca"
    name="Customer">
      <ownedAttribute xmi:type="uml:Property" xmi:id="75eeef2f-34a7-481a-b532-5a25a06334ef"
       name="City" visibility="public" isUnique="false" />
      <ownedAttribute xmi:type="uml:Property" xmi:id="5ce6265c-2d1a-4976-be09-4b45f612dfd6"
       name="FirstName" visibility="public" isUnique="false" />
      <ownedAttribute xmi:type="uml:Property" xmi:id="33767cbd-1d12-47ec-82a3-66ab3a09e83e"
       name="LastName" visibility="public" isUnique="false" />
      <ownedAttribute xmi:type="uml:Property" xmi:id="34a574ae-5b2c-42ad-86c8-83e70844be9c"
       name="PostalCode" visibility="public" isUnique="false" />
      <ownedAttribute xmi:type="uml:Property" xmi:id="eaee0fa9-6016-4426-b175-a21f80a8ec0a"
       name="State" visibility="public" isUnique="false" />
      <ownedAttribute xmi:type="uml:Property" xmi:id="3180a449-4258-454d-b551-16a5b0d4c2c3"
       name="StreetAddress" visibility="public" isUnique="false" />
    </packagedElement>
```

Figura 4: Exemplo de um arquivo XMI na versão 2.1

2.4.3: XSLT

O XSLT (*eXtensible Stylesheet Language for Transformation*), é uma linguagem que permite transformar um arquivo XML em outro arquivo XML. O XSLT permite organizar e ordenar elementos, realizar testes e decidir quais elementos são exibidos ou ocultados nos dados de um arquivo XML. No processo de transformação o XSLT utiliza o XPath(usado para navegar nos elementos de um arquivo XML) para definir as infor-

mações do documento XML que combinam com um template pré-definido. Quando um trecho de informação do arquivo XML combina com o template, o XSLT transforma a parte coincidente em um resultado do arquivo destino, ou seja no documento XML de saída.

2.4.4: Parser XML

Como citado anteriormente, o documento XMI tem a mesma estrutura de um arquivo XML o que torna possível a utilização de um Parser XML para a leitura de um arquivo XMI. Um Parser XML, é um pacote, biblioteca ou um módulo que é usado na leitura de documentos XML. O parser XML torna possível que um motor de leitura ou um visualizador acesse a estrutura e o conteúdo de um documento XML. Logo, podemos utilizar este tipo de ferramenta em conjunto com a linguagem de transformação XSLT para fazer a leitura dos arquivos XMI. No processo de transformação de um arquivo XML, a linguagem XSLT usa XPath para definir partes do documento de origem que combinam com um ou mais template. Quando uma combinação é encontrada, a linguagem XSLT transformará a parte que combinou no documento de origem ao documento de resultado. As partes do documento de origem que não combinam com um template não serão modificadas no documento XML de saída.

2.4.5: Elementos do XMI

Abaixo serão listadas os elementos XMI mais relevantes para o desenvolvimento deste projeto.

- **xmiName:** utilizada para especificar o nome a ser usado para uma classe em um determinado esquema.
- **idName:** o valor do elemento idName, indica a classe que está sendo descrita.
- **contentType:** o XMI define um número de elementos que afetam o conteúdo das declarações da classe em questão. Muitos dos elementos para atributos e associações UML afetam o conteúdo.

- **serialize:** é utilizada em situações em que não se deseja serializar o valor dos atributos. Caso este elemento seja definida como falsa, o XMI não define um atributo ou elemento em XML. Por padrão este elemento é definido como verdadeira.

- **element/attribute:** para alguns atributos UML o padrão XMI cria tanto uma declaração de atributo XML quanto uma declaração de elemento XML. A criação destes elementos pode ser obtida ao adicionar o valor de verdadeiro para a tag *elemento* ou verdadeiro para o elemento *atributo*, mas existe uma restrição do XMI que permite apenas um destes elementos habilitados para cada atributo UML.

- **multiplicity:** este elemento indica a multiplicidade de um atributo UML, porém o XMI só cria um atributo XML para o atributo UML se os seguintes critérios forem obedecidos.

  - Multiplicidade é exatamente 1.
  - O seu tipo é um DataType.
  - A tag do atributo está com o valor verdadeiro.

  Neste caso o XMI cria um atributo XML com um atributo ***use*** que é definido como *required*.

- **defaultValue:** se o elemento *defaultValue* possui valor, este valor é usado como valor padrão para o atributo XML do atributo UML.

3. Trabalhos Relacionados

Há diversas ferramentas voltadas para a modelagem de dados. Estas ferramentas não trabalham com um padrão de arquivo que permita a troca de informações de forma fácil e ágil. No entanto, alguma destas ferramentas importam e exportam arquivos no padrão XMI, um padrão abordado no tópico 2.4, que contempla a troca de informações entre ferramentas case. As principais ferramentas que dão suporte a este tipo de funcionalidade são pagas como: Enterprise Architected, ARGOS UML, Together, Rational Rose entre outras, o que torna sua utilização inviável, para projetos de pequeno porte.

As ferramentas listadas a seguir possuem uma certa similaridade com a ferramenta XMI2DB2XMI.

3.1: AXGen

Esta ferramenta tem o objetivo de gerar código fonte. A ferramenta utiliza como parâmetros de entrada um arquivo XMI e aplica suas transformações aos templates Velocity pré-definidos, realizando a geração das classes em JAVA e scripts em linguagem SQL. A principal diferença da ferramenta em questão à ferramenta XMI2DB2XMI, é o fato da AXGen ser voltada a geração de código fonte e não permitir a geração de arquivos XMI a partir de um SGBD existente. A Figura 5 ilustra o processo de funcionamento:

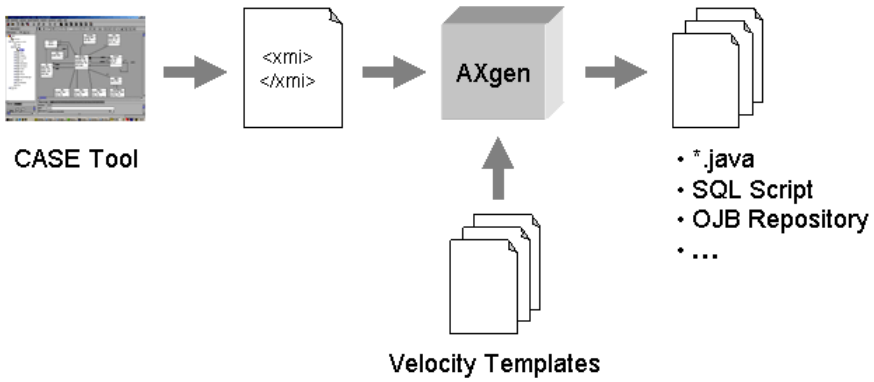

Figura 5: Funcionamento da ferramenta AXGen

3.2: Butterfly Code Generator

A ferramenta Butterfly realiza a geração de código que utiliza um arquivo XMI de entrada e templates XSLT para gerar qualquer tipo de código. Muitos destes templates são disponibilizados em comunidades de desenvolvimento incluindo EJB, JDO e Struts. O usuário pode utilizar estes templates fornecidos ou alterá-los pessoalmente para atender as suas necessidades.

Esta ferramenta possui uma vantagem sobre as demais, pois permite a geração de qualquer tipo de código-fonte, porém, esta flexibilidade reflete em uma dificuldade, pois o usuário tem que desenvolver ou realizar o download do XSLT específico para a geração do código que deseja obter.

4. XMI2DB2XMI - A descrição da ferramenta

Neste capítulo é abordado os passos de criação da ferramenta deste trabalho. A ferramenta XMI2DB2XMI é gratuita e auxilia o processo de desenvolvimento de software.

4.1: Visão Geral

Para iniciar o desenvolvimento deste sistema foram realizados alguns estudos sobre quais tecnologias seriam empregadas. A IDE escolhida (Visual Studio 2010) foi feita pelo grande suporte que a comunidade MSDN disponibiliza aos desenvolvedores que utilizam esta ferramenta de desenvolvimento, principalmente para a linguagem C-Sharp. Para agilizar o processo de leitura dos arquivos XMI foi utilizada a biblioteca XMICodeDomLib (XMICODEDOMLIB,2011) disponibilizada por Dustin Metzgar, um dos desenvolvedores do portal Code Project que implementa e otimiza um parser XML para realizar a leitura dos arquivos XMI.

Outra ferramenta escolhida para auxiliar o desenvolvimento foram os templates NVelocity. Esta tecnologia é uma parte do projeto Apache Jakarta Velocity mantida pelo portal CASTLEPROJECT (NVELOCITY,2011). Para trabalhar com o NVelocity é necessária uma instância do motor e um ou mais templates. O resultado do processo é conhecido como a junção do template com um contexto(dados).

A ferramenta XMI2DB2XMI auxilia o processo de geração automática de código, possibilitando ao usuário gerar a estrutura de dados do SGBD a partir de um arquivo XMI(gerado através das ferramentas Case do mercado). A ferramenta serve para usuários que necessitam gerar a documentação de um sistema legado, bem como, para usuários que desejam criar - a partir do seu diagrama de classes - a estrutura do SGBD.

4.2: XMI2DB

O sistema XMI2DB2XMI possui duas funcionalidades, a primeira delas, abordada nesta seção, cria a estrutura de dados de um SGBD a partir de um arquivo XMI. Para

desenvolver esta funcionalidade adotou-se a arquitetura da Figura 6.

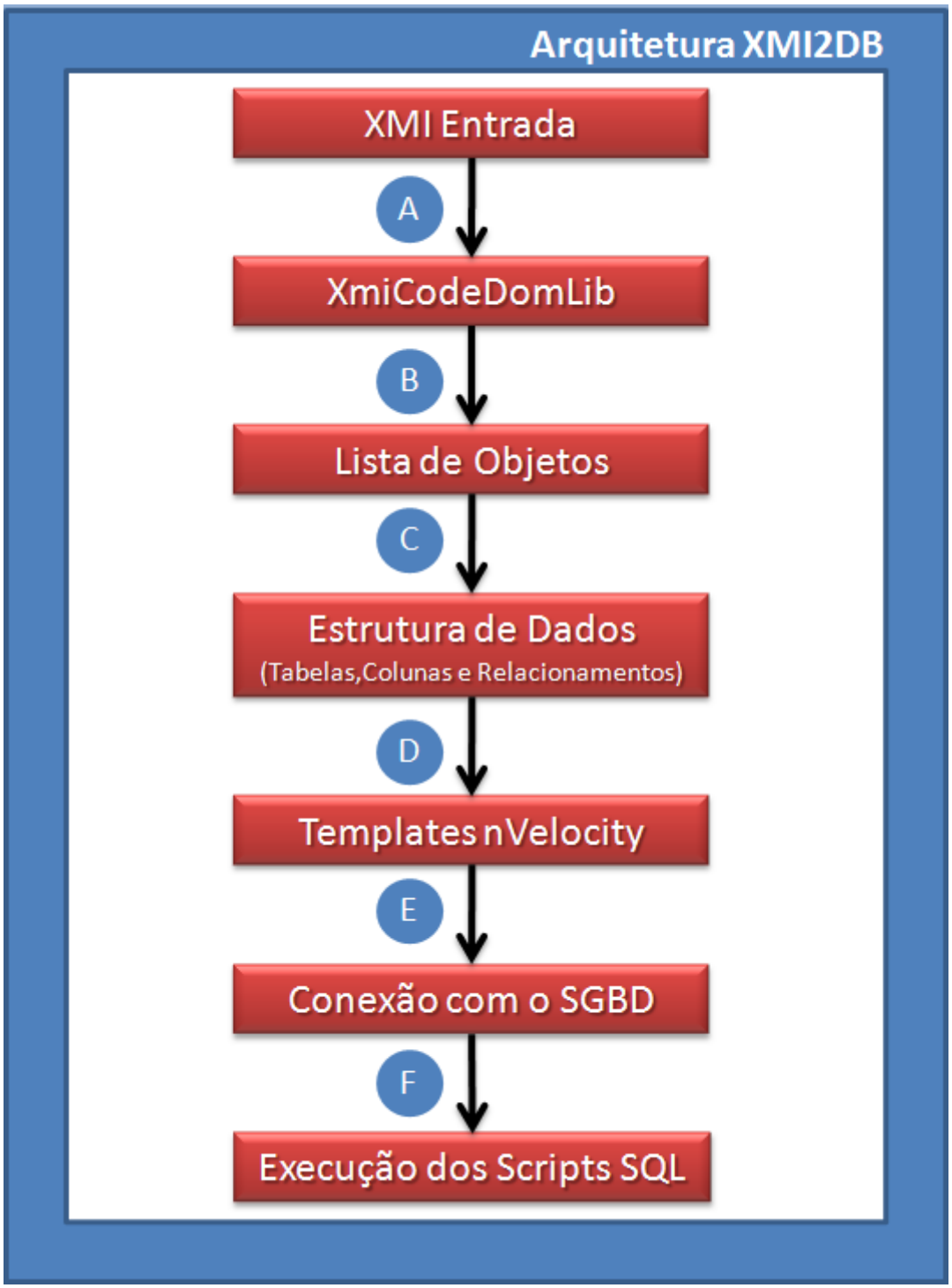

Figura 6: Arquitetura da ferramenta no caminho XMI->DB

A Figura 6 possui as etapas de A-F, estas são descritas nas sub-seções abaixo.

4.2.1: Escolha e validação do arquivo XMI - Etapa A, B e C

Para iniciar o processo de geração do banco de dados. O usuário deve selecionar a primeira opção da tela do sistema, conforme a Figura 7, que indica a geração de um banco de dados a partir de um arquivo XMI.

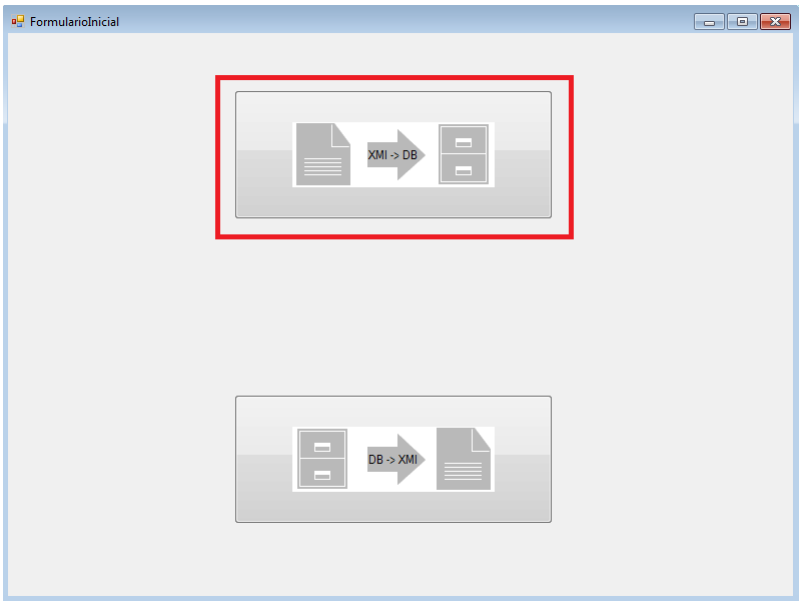

Figura 7: Tela Inicial do Sistema XMI->DB

A próxima tela do sistema, exibida na Figura 8, solicita ao usuário para que selecione o arquivo XMI contendo o modelo de dados. É importante ressaltar que esta ferramenta dá suporte a versão 2.1 do XMI conforme especificação encontrada no portal da OMG.

Após selecionar o arquivo XMI o usuário tem a opção de validar o arquivo (no botão "*Validar Arquivo*" exibido na Figura 8), ou seja, o sistema retorna uma mensagem informando se a versão do arquivo XMI é compatível com a ferramenta (versão 2.1).

Ao escolher o arquivo XMI, a aplicação invoca o método "***Parse(XmlReader xr, XmiParserType parserType)***" do parser XML contido na biblioteca XmiCodeDomLib. O parser realiza a leitura do arquivo XMI e armazena os objetos obtidos em classes específicas do XMI. As classes são mapeadas para os objetos da classe C-Sharp *"Xmi-Class.cs"*, os atributos das classes são mapeados para os objetos da classe C-Sharp

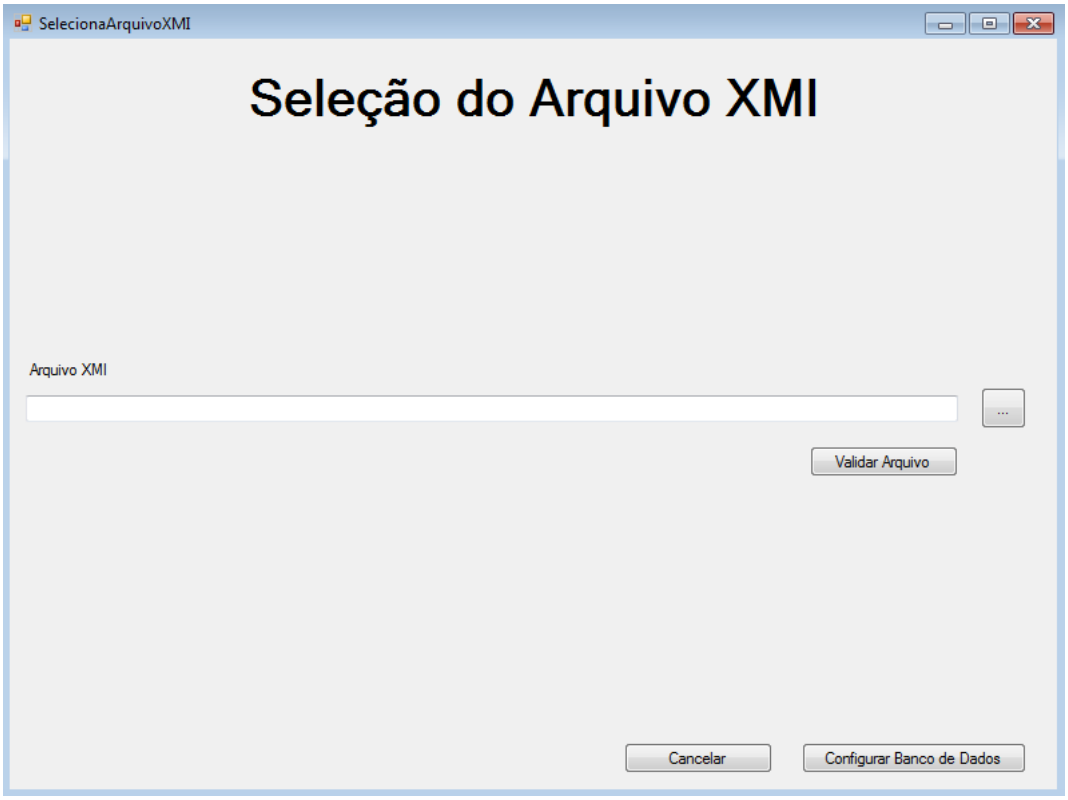

Figura 8: Tela de Seleção e Validação do arquivo XMI

"XmiAttribute.cs" e por fim, os relacionamentos são mapeados para as classes *"XmiAssociation.cs"* e *"XmiAssociationEnd.cs"*.

Ao armezanar as informações obtidas do arquivo XMI, através das adaptações feitas à biblioteca XmiCodeDomLib, os métodos contidos na Listagem 1 retornam uma lista de objetos. Esta lista é utilizada ao montar a estrutura de dados contendo as informações do arquivo XMI.

Listagem 1 Métodos de Retorno da biblioteca XmiCodeDomLib

```
public List<object> GetElements()
{
    CodeCompileUnit ccu = new CodeCompileUnit();
    ccu = GetCodeCompileUnit();
    List<object> list = new List<object>();
    if (_Model != null)
```

```
        list = _Model.GerarElementos(ccu);

    return list;
}
```

4.2.2: Estrutura de dados

Para armazenar as tabelas, os atributos e as associações modelados no arquivo XMI, o sistema armazena suas informações na classe da Listagem 2.

Listagem 2 Estrutura de Dados para as tabelas

```
public class Tabela
    {
        public string nome { get; set; }
        public List<Atributo> listaAtributos { get; set; }
    }

public class Atributo
    {
        public string nome { get; set; }
        public TipoBD tipo { get; set; }
        public bool flg_nulo { get; set; }
        public bool flg_PK { get; set; }
        public int tamanho { get; set; }
        public int precisao { get; set; }
        public int decimais { get; set; }
    }

public class Associacao
    {
        public Tabela tabelaorigem { get; set; }
```

```
        public Tabela tabeladestino { get; set; }
        public Atributo atributoorigem { get; set; }
        public Atributo atributodestino { get; set; }
        public Mult multiplicidadeorigem { get; set; }
        public Mult multiplicidadedestino { get; set; }
    }
```

A classe ***"Tabela"*** é preenchida com informações das classes contidas no arquivo XMI:

A classe ***"Atributo"*** armazena as seguintes informações:

Esta estrutura é preenchida com os valores definidos pelo usuário através do diagrama de classes da ferramenta de modelagem e que passou pelo processo de exportação para o padrão XMI 2.1. Para preencher este modelo aplica-se a listagem de objetos que retorna da biblioteca XmiCodeDomLib ao método da Listagem 3.

Listagem 3 Cria a estrutura de dados para as informações do arquivo XMI

```
private void RecuperaObjetos(object obj)
  {
      XmlReader xr = obj as XmlReader;
      XmiRoot root = new XmiRoot();
      root.Parse(xr, XmiParserType.Reflection);

      List<object> lista = new List<object>();
      lista = root.GetElements();

      XMI = new EstruturaXMI();

      XMI.listaClasses = new List<XmiClass>();
      XMI.listaAtributos = new List<XmiAttribute>();
      XMI.listaAssociacoes = new List<XmiAssociation>();
      XMI.listaAssocDestinos = new List<XmiAssociationEnd>();
```

```
        foreach (object elementos in lista)
        {
            if (elementos is XmiClass)
            {
                XMI.listaClasses.Add(elementos as XmiClass);
            }

            if (elementos is XmiAttribute)
            {
                XMI.listaAtributos.Add(elementos as XmiAttribute);
            }

            if (elementos is XmiAssociation)
            {
                XMI.listaAssociacoes.Add(elementos as XmiAssociation);
            }

            if (elementos is XmiAssociationEnd)
            {
                XMI.listaAssocDestinos.Add(elementos as XmiAssociationEnd);
            }
        }
    }
```

4.2.3: Aplicação dos templates NVelocity

Os dados armazenados na estrutra de dados da Listagem 2 são aplicados sobre templates NVelocity para gerar os scripts SQL temporários que serão executados pelo aplicativo XMI2DB2XMI. O conceito de um template NVelocity é manter em um arquivo partes fixas de código e aplicar os valores passados como parâmetro em var-

iáveis pré-definidas. O NVelocity permite além da substituição de variáveis por valores parametrizados, algumas estruturas lógicas, como: comparações, laços e atribuição de valores em tempo de execução.

A Listagem 4 exibe um Template NVelocity que percorre um laço na lista de tabelas do sistema. O comando *"foreach"* realiza esse laço e é finalizado pelo comando *"end"*. Todas as variáveis iniciadas pelo caractere *"$"* são as partes dinâmicas do template e são substituídas pelo valor armazenado na estrutura de dados em tempo de execução.

Listagem 4 Template NVelocity para gerar as tabelas de um BD Oracle

```
#foreach( $tabela in $tabelas )
Prompt CREATE TABLE $classe.Tabela.nome
CREATE TABLE $tabela.nome (
        #set($separador = '')
        #foreach( $atributo in $tabela.listaatributos )
        $separador$atributo.nome $atributo.tipo  $atributo.permitenulo
        #set($separador = ',')
        #end
)
#end
COMMIT;

#foreach( $tabela in $tabelas )
#foreach( $atributo in $tabela.listaatributos )
#if ($atributo.flg_PK)
#if ($atributo.tipo == 'Integer')
prompt SEQUENCE $atributo.nome
CREATE SEQUENCE $atributo.nome
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 999999
```

```
 NOCYCLE
 NOCACHE;

#end
#end
#end
#end
COMMIT;
```

4.2.4: Definição dos parâmetros de configuração com o banco de dados

Cumprida as etapas de A-D da Figura 6, inicia-se o processo de conexão com a instância de destino do SGBD que é configurada pelo usuário. Para tal configuração o usuário deverá selecionar qual o tipo de banco de dados que deseja se conectar, Oracle ou Microsoft SQL Server.

No caso da escolha ser um banco de dados Oracle, o usuário deverá informar o nome da instância, um usuário e uma senha, além destas informações, é importante lembrar que o usuário deve ter os devidos privilégios para criação das tabelas e objetos no SGBD informado. Após o preenchimento destas informações o usuário tem a opção de testar se a conexão foi bem sucedida.

Na escolha de um banco de dados Microsoft SQL Server, a instância é substituída pelo endereço da conexão e o nome do banco de dados. Também é necessário informar um usuário e senha com privilégios para criação das tabelas.

4.2.5: Execução dos Scripts SQL

O código gerado através deste script pode ser encontrado no apêndice A do trabalho. Dando sequência ao processo de geração, são criadas as chaves primárias e estrangeiras. Esta etapa é realizada após a criação das tabelas em função dos atributos que relacionam estas chaves já estarem presentes no banco de dados.

Com os arquivos gerados, o software utiliza a conexão com o banco de dados e

executa os Scripts em linguagem SQL, retornando uma validação se o processo ocorreu sem a presença de erros ou não. Com esta etapa finalizamos um dos fluxos do sistema que vai do arquivo XMI a geração do banco de dados. Na próxima seção é descrito o fluxo inverso que fará a leitura da estrutura do banco de dados e gerará um arquivo XMI que poderá ser importado por uma ferramenta case para exibir um diagrama de classe.

4.3: DB2XMI

Para construir o arquivo XMI a partir do banco de dados é necessário realizar uma conexão com o banco de dados. Para isso, o usuário deve definir os parâmetros para conexão com o banco de dados. No caso do banco de dados Oracle são necessárias as definições como: a entrada do banco no TNS, o Owner das tabelas, o usuário com as permissões de conexão ao owner definido e da senha de acesso. Após estas configurações o usuário deve definir um diretório de destino, onde é armazenado o arquivo XMI gerado pelo aplicativo. Ao clicar em gerar o arquivo XMI, o fluxo banco de dados para o arquivo XMI do aplicativo realiza a leitura das tabelas e colunas do banco de dados.

4.3.1: Consultas em SQL

Esta etapa é feita aplicando consultas em linguagem SQL. As consultas realizadas tem o objetivo de obter as tabelas do owner definido. Com a lista de tabelas em memória é realizada a busca das colunas de cada tabela listada. E para finalizar, as chaves primárias e estrangeiras são lidas para criar os relacionamentos entre as tabelas obtidas do banco de dados.

Abaixo podemos analisar na Listagem 5, a consulta que retorna a lista de tabelas do usuário em um banco de dados Microsoft SQL Server.

Listagem 5 Script em SQL para recuperar as tabelas de um banco de dados SQL Server

```
SELECT T.TABLE_NAME AS TABELA,
                    C.COLUMN_NAME AS COLUNA_PK
FROM INFORMATION_SCHEMA.TABLES T,
            INFORMATION_SCHEMA.TABLE_CONSTRAINTS PK,
```

```
                INFORMATION_SCHEMA.KEY_COLUMN_USAGE C
WHERE PK.TABLE_NAME = T.TABLE_NAME
                AND CONSTRAINT_TYPE = ''PRIMARY KEY''
                AND C.TABLE_NAME = PK.TABLE_NAME
                AND C.CONSTRAINT_NAME = PK.CONSTRAINT_NAME
                AND T.TABLE_NAME <> ''sysdiagrams''
```

O retorno desta consulta exibe o nome da tabela e sua chave primária, no caso do retorno de mais de um registro com o mesmo nome de tabela, considera-se que a tabela em questão possui uma chave primária composta. Com as tabelas armazenadas em memória é realizado através de um laço que controla as tabelas obtidas, uma nova consulta que passa como um dos parâmentros o nome da tabela para obter suas colunas. Além do nome da coluna são retornados alguns atributos como: tipo da coluna, tamanho da coluna, precisão da coluna, escala da coluna, se a coluna permite nulo e a ordem das colunas. A consulta realizada para obter estes dados está descrita na Listagem 6:

Listagem 6 Script em SQL para recuperar as propriedades da coluna de uma tabela

```
SELECT UT.TABLE_NAME AS TABELA,
      UTC.COLUMN_NAME AS COLUNA,
      UTC.DATA_TYPE AS TIPO,
      UTC.DATA_LENGTH AS TAMANHO,
      UTC.DATA_PRECISION AS PRECISAO,
      UTC.DATA_SCALE AS ESCALA,
      UTC.NULLABLE AS NULO,
      UTC.COLUMN_ID AS ORDEM
FROM USER_TABLES UT, USER_TAB_COLS UTC
WHERE UT.TABLE_NAME = UTC.TABLE_NAME
AND UT.TABLE_NAME = ''tabela_obtida''
```

4.3.2: Retorno da Aplicação

O arquivo XMI gerado pode ser importado por ferramentas de modelagem de dados que suportem o padrão de arquivos XMI 2.1 da OMG. Ao importar este arquivo o usuário terá uma visão gráfica, através de um diagrama de classe, da estrutura que existe no banco de dados conectado. Neste ponto do processo, o arquivo XMI está disponível na pasta de destino configurada na primeira etapa deste fluxo.

5. Conclusão e Trabalhos Futuros

É de interesse do ramo dos desenvolvedores de software, indentificar trechos de código de uma arquitetura que permitam seu desenvolvimento através da automatização de código. Além de proporcionar uma forma ágil de desenvolvimento, trechos de código gerados automaticamente diminuem as estatísticas de erros de sintaxe e de lógica, pois são revisados com maior empenho para gerar o código corretamente. As equipes de desenvolvimento utilizam em projetos recentes arquiteturas complexas e divididas em diversas camadas que permitem a troca de tecnologia, ao passar do tempo, sem muito impacto e modificações no projeto como um todo. Salvo em casos específicos onde a divisão em várias camadas pode impactar na questão de performance do sistema. A possibilidade de gerar a estrutura de banco de dados a partir de uma documentação no padrão UML facilita projetos concebidos a partir da modelagem de dados. Bem como a possibilidade de gerar os diagramas das classes contidas em um banco de dados pré-existente, neste caso criando a cultura de possuir uma documentação atualizada dos sistemas de uma empresa.

A principal motivação para elaborar esta ferramenta, deve-se ao fato dos sistemas de modelagem de dados disponibilizarem apenas em suas versões completas, funcionalidades como a geração do banco de dados a partir dos diagramas de classe. Como a leitura dos objetos contidos no diagrama de classe já está pronta, e a estrutura para o armazenamento destes objetos também, existe a possibilidade de implementar futuramente novos templates que gerem código para outras partes da arquitetura de um software. Por exemplo, pode-se utilizar esta estrutura para criar os XML de mapeamento do nHibernate, ou então definir uma camada DAO (Data Access Object) ou ainda gerar o esqueleto das classes em CSharp.

Ainda não existe uma ferramenta que gere o software como um todo, pois existe as regras de negócio que são mais complexas e necessitam de um estudo detalhado, mas a abordagem de geração automática de algumas partes da arquitetura permite que o desenvolvedor concentre seus esforços em criar regras de negócio otimizadas e com qualidade, deixando a parte massante de desenvolvimento por conta de ferramentas

como esta.

Outro problema que pode ser encontrado é a utilização de ferramentas de modelagem que anunciam ter suporte ao padrão XMI 2.1, mas na verdade criam tags próprias para facilitar a geração e importação apenas da sua ferramenta, dificultando assim a interoperabilidade entre os sistemas de modelagem existentes no mercado.

A ferramenta ajuda o desenvolvedor a criar a cultura de manter uma documentação atualizada, pois com isso, permite a geração da estrutura de tabelas do banco de dados a qualquer momento. A utilização de um padrão é importante para dar suporte a diversas ferramentas utilizadas no mercado e, consequentemente, facilitar a troca de informação entre equipes ou empresas que desejam trocar idéias de desenvolvimento, mas que não trabalham com as mesmas ferramentas de modelagem de dados.

Apêndices

A. Código Fonte

A.1: Telas do Sistema

Na tela da Figura 9 o usuário deve selecionar qual é o seu fluxo de trabalho: XMI2DB - Este botão permite a geração da estrutura do banco de dados com a entrada de um arquivo XMI fornecido pelo usuário. DB2XMI - Este botão gera um arquivo XMI a partir de um banco de dados existente.

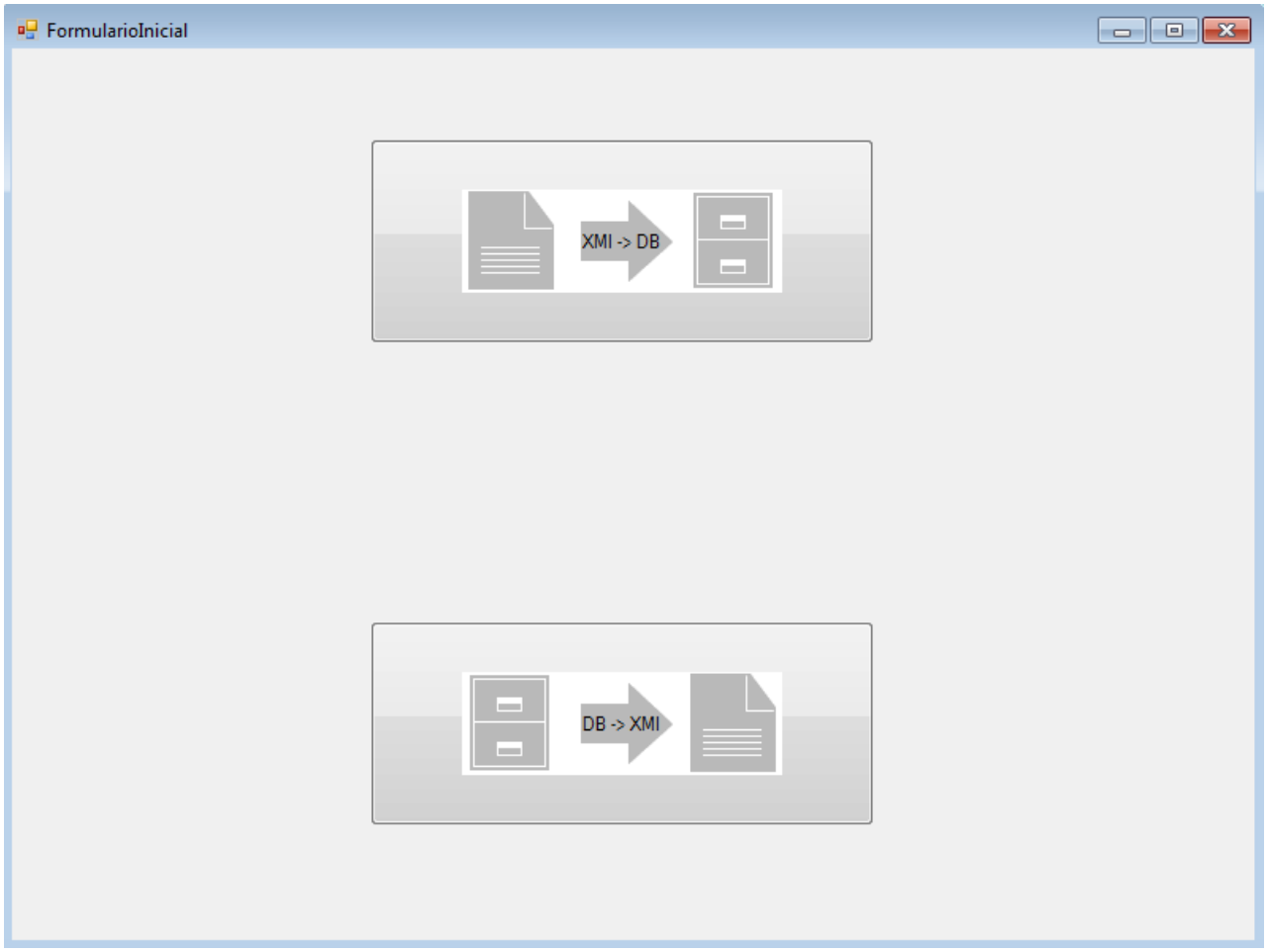

Figura 9: Tela Inicial para selecionar o Fluxo do Sistema

Na tela da Figura 10, o usuário seleciona um arquivo XMI para implementar o fluxo XMI2DB.

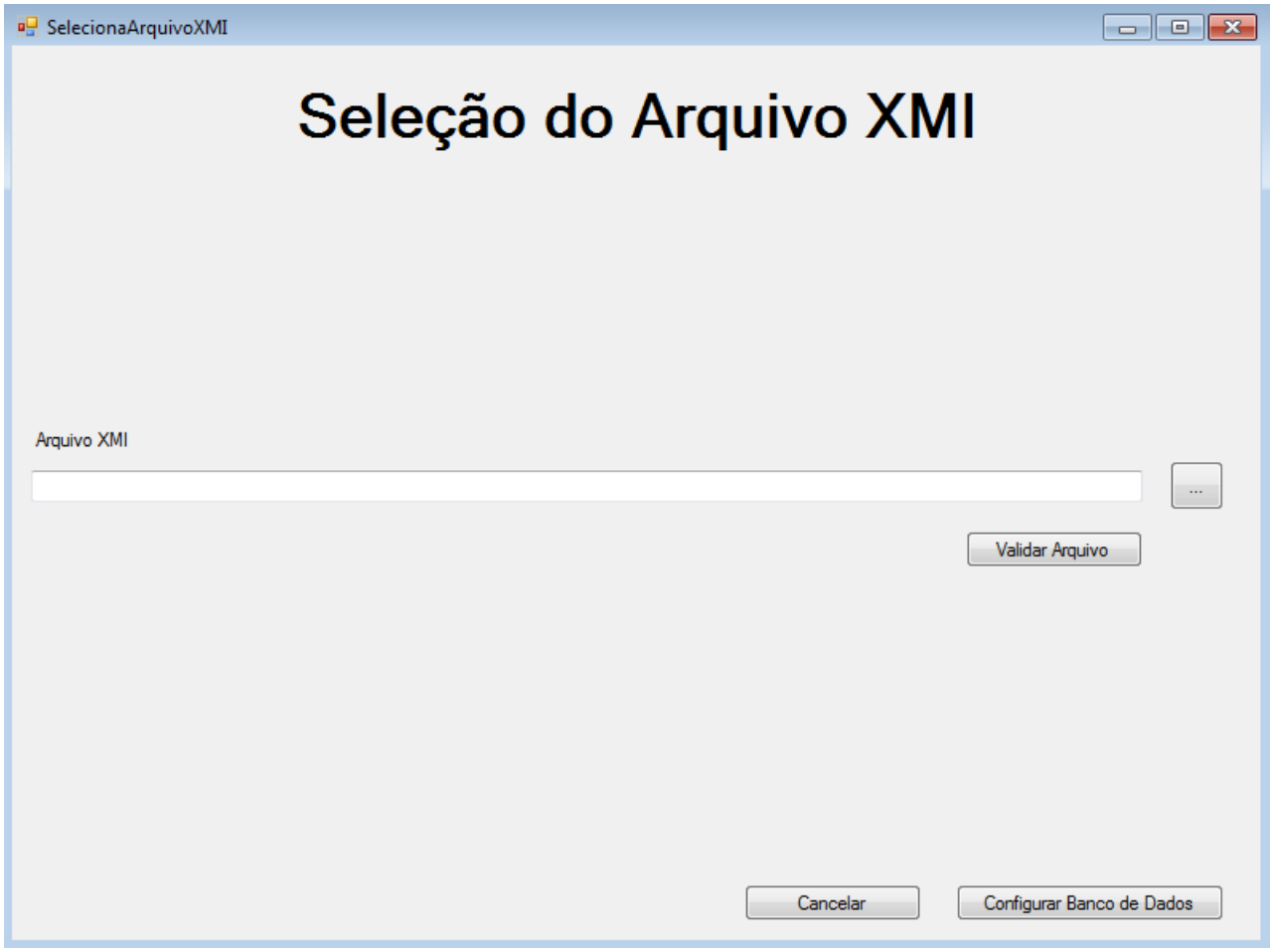

Figura 10: Tela de seleção do Arquivo XMI de entrada ou saída dependendo do fluxo escolhido

A tela de Configuração do Banco de Dados (Figura 11) permite ao usuário especificar a configuração necessária para gerar a estrutura de tabelas. Esta tela também é utilizada no segundo fluxo do sistema, ou seja, quando conecta ao SGBD de onde o usuário deseja recuperar a estrutura de tabelas existentes e criar um arquivo XMI de saída.

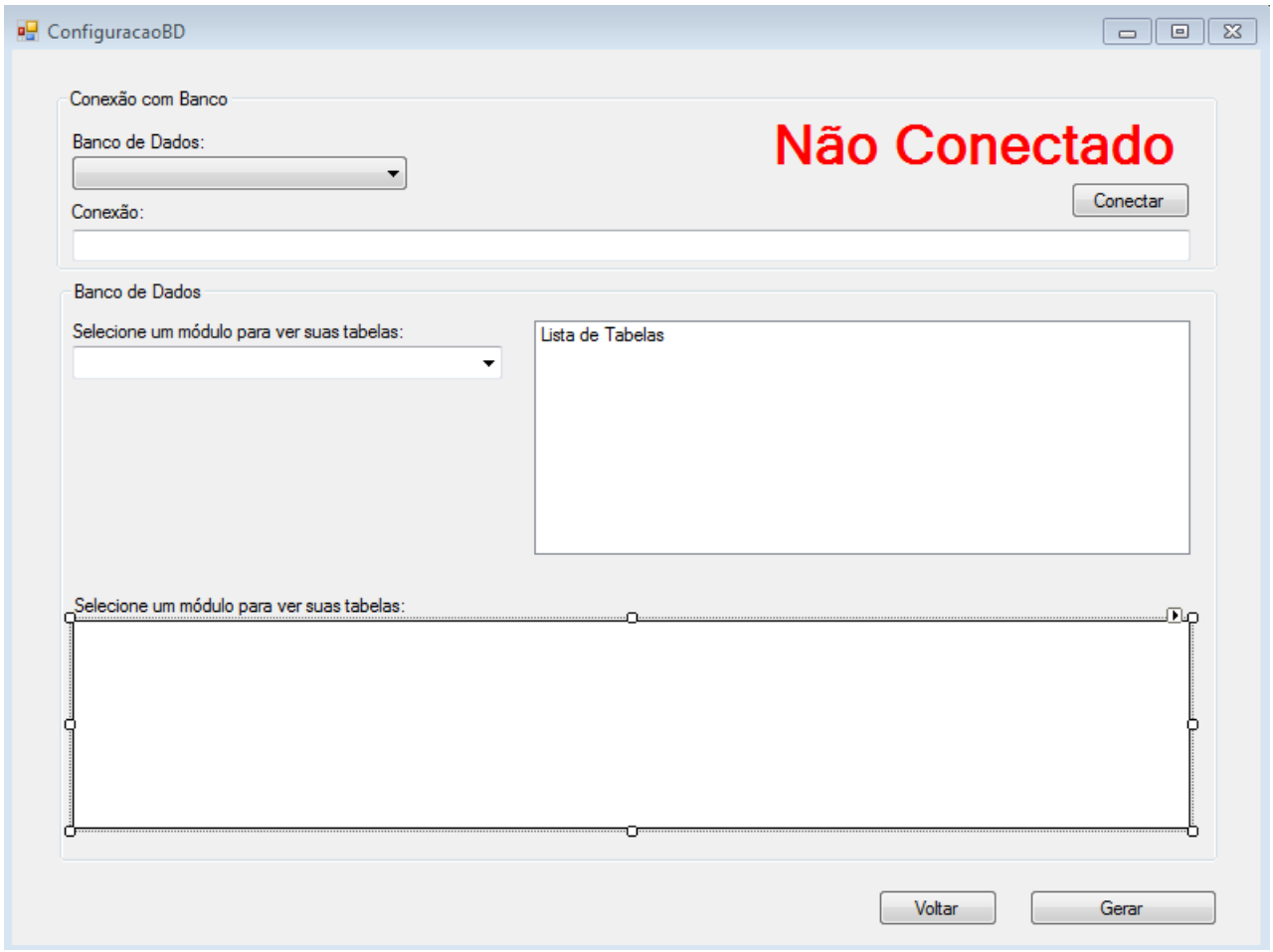

Figura 11: Tela de Configuração para conexão com o Banco de Dados

A.2: Arquivo XMI

A Figura 12 exibe um arquivo XMI no padrão da OMG que é válido para utilizar na ferramenta XMI2DB2XMI.

```
<?xml version="1.0" encoding="utf-8"?>
<xmi:XMI xmi:version="2.1" xmlns:uml="http://schema.omg.org/spec/UML/2.1.1/uml.xml" xmlns:xmi=
"http://schema.omg.org/spec/XMI/2.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:L2=
"http://schema.omg.org/spec/UML/2.1.1/schemas/StandardProfileL2.xmi" xmlns:L3=
"http://schema.omg.org/spec/UML/2.1.1/schemas/StandardProfileL3.xmi">
  <uml:Model xmi:id="d67df018-1079-45a5-ba01-0781a170e773" name="ModelingProject2">
    <profileApplication>
      <appliedProfile xmi:type="uml:Profile" href=
      "http://schema.omg.org/spec/UML/2.1.1/schemas/StandardProfileL2.xmi#_0" />
    </profileApplication>
    <profileApplication>
      <appliedProfile xmi:type="uml:Profile" href=
      "http://schema.omg.org/spec/UML/2.1.1/schemas/StandardProfileL3.xmi#_0" />
    </profileApplication>
    <packagedElement xmi:type="uml:Package" xmi:id="78f3cebf-860f-4196-83de-db325c52cc8c" name="Modelo">
      <packagedElement xmi:type="uml:Class" xmi:id="ffc4842e-08e1-4c9d-8044-788ea80b113a" name="Banco">
        <ownedAttribute xmi:type="uml:Property" xmi:id="66281f93-dcf6-4df9-969b-6536fb1fd359" name="Codigo"
        visibility="public" isUnique="false" />
        <ownedAttribute xmi:type="uml:Property" xmi:id="879b7518-ab20-4a61-9097-d169a0154245" name="NumeroAgencia"
        visibility="public" isUnique="false" />
      </packagedElement>
      <packagedElement xmi:type="uml:Class" xmi:id="a17a2e48-04b0-4c52-b43e-ec0cc489f535" name="Cliente">
        <ownedAttribute xmi:type="uml:Property" xmi:id="2f40c05b-3336-4cff-98b5-7b49a34b5f81" name="Nome" visibility=
        "public" isUnique="false" />
        <ownedAttribute xmi:type="uml:Property" xmi:id="7858c7b2-cb11-40eb-84fd-400cb42a984a" name="Endereço"
        visibility="public" isUnique="false" />
        <ownedAttribute xmi:type="uml:Property" xmi:id="4471d909-b800-4803-bbd7-12b7beb06026" name="Habilitado"
        visibility="public" isUnique="false">
          <type xmi:type="uml:PrimitiveType" href="http://schema.omg.org/spec/UML/2.1.1/uml.xml#Boolean" />
          <defaultValue xmi:type="uml:LiteralBoolean" xmi:id="1e6d42f8-0ca0-4f15-9ef7-07df660761d0" name=
          "DefaultValue" value="true" />
        </ownedAttribute>
      </packagedElement>
      <packagedElement xmi:type="uml:Association" xmi:id="a01f7734-eafd-484c-871e-42f86c1a25d9" memberEnd=
      "1e7fff73-2a50-4ba2-8d28-f1991d3e676e ab23172f-d01c-4c5e-95f1-13441717cbd1" navigableOwnedEnd=
      "ab23172f-d01c-4c5e-95f1-13441717cbd1">
        <ownedEnd xmi:type="uml:Property" xmi:id="1e7fff73-2a50-4ba2-8d28-f1991d3e676e" name="Banco" visibility=
        "public" type="ffc4842e-08e1-4c9d-8044-788ea80b113a" association="a01f7734-eafd-484c-871e-42f86c1a25d9" />
        <ownedEnd xmi:type="uml:Property" xmi:id="ab23172f-d01c-4c5e-95f1-13441717cbd1" name="Cliente" visibility=
        "public" type="a17a2e48-04b0-4c52-b43e-ec0cc489f535" association="a01f7734-eafd-484c-871e-42f86c1a25d9">
          <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="6d4a5193-b13f-437b-9843-9bd1000b19a9" value="*"
          />
          <lowerValue xmi:type="uml:LiteralInteger" xmi:id="552c4c0e-fc9c-44b6-be3c-557c28496f65" value="*" />
        </ownedEnd>
      </packagedElement>
    </packagedElement>
  </uml:Model>
</xmi:XMI>
```

Figura 12: Tela de Configuração para conexão com o Banco de Dados

A.3: NVelocity

A Listagem 7 exibe um método do sistema que cria os objetos necessários para preencher o template NVelocity.

Listagem 7 Método que cria a estrutura de contexto para os templates NVelocity

```
#region ### Gerar Multiplos Arquivos ###
    private void GerarMultiplosArquivos(StringWriter resultado,
                                        string pNomeTemplate,
                                        string extensao,
                                        string caminhoPastas,
                                        string prefixo,
                                        string sufixo)
    {
        VelocityContext parametros = new VelocityContext();
        Velocity.SetProperty(FILE_RESOURCE_LOADER_PATH,
                             Application.StartupPath +
                             "\\Templates\\");
        Velocity.Init();

        Template template;

        foreach (Tabela tabela in lstTabelas)
        {
            parametros.Put("tabela", tabela);

            template = Velocity.GetTemplate(pNomeTemplate);

            template.Merge(parametros, resultado);

            if (!Directory.Exists(caminhoPastas))
                Directory.CreateDirectory(caminhoPastas);
```

```
                GravarArquivo(resultado.ToString(), caminhoPastas +
                                "\\", prefixo + tabela.nome + sufixo +
                                extensao);

                resultado = new StringWriter();
            }
        }
#endregion
```

Referências

Sítio do portal code project (http://www.codeproject.com/kb/codegen/xmi_codedom.aspx). Acessado em 17 de outubro de 2011.

E. GRAVINA, N. KOCHHAR, P. N. *Introdução ao Oracle: SQL e PL/SQL*. Oracle Corporation, 2000.

FOWLER, M. *UML Distilled, 3rd edition*. Addison Wesley, 2003.

G. BOOCH, J. RUMBAUGH, I. J. *UML Guia do Usuário, 2a edition*. Elsevier, 2005.

LARMAN, C. *Utilizando UML e Padrões*. Bookman Companhia Editora., Porto Alegre, RS, Brasil, 2007.

Sítio do nvelocity no portal castle project (http://www.castleproject.org/others/nvelocity/index.html). Acessado em 02 de dezembro de 2011.

Sítio da Object Management Group (http://doc.omg.org/formal/20050704.pdf). Acessado em 17 de outubro de 2011.

RIORDAN, R. *Designing Relational Database Systems*. Microsoft Press, 1999.

Sítio da w3c school (http://www.w3cschool.com). Acessado em 17 de outubro de 2011.