JAW – Uma Ferramenta para a automação do desenvolvimento de sistemas de informação orientada a modelos e estruturas de bancos de dados

Fabiana de Lima, Flávio Luiz Schiavoni, Rafael Maltempe da Vanso, Robson Juliano Beterincosto

FAFIMAN – Faculdade de Filosofia, Ciências e Letras de Mandaguari

fabiana@fafiman.br, fls@rendera.com.br, rafaelmaltempe@yahoo.com.br, robson.beterincosto@gmail.com

Abstract. *In a software development environment we can identify repetitive tasks that may be automatic. This paper presents a tool that extract metadata through a JBDC connection from a database. These metadata are patterned into an owner metamodel such as Entities and Attributes. Entities and Attributes represents objects in a computational view. The tool proposed in this paper is able to generate files with extensions java, form and sql. The code generation into JAW is done using code and templates from Jakarta Velocity generated using the Enterprise JavaBeans (EJB) model.*

Resumo. *Em um ambiente de desenvolvimento de software podemos identificar tarefas repetitivas e possíveis de serem automatizadas. Este trabalho apresenta uma ferramenta que, por meio de conexões com a API JDBC, extrai metadados de bases de dados. Esses metadados são padronizados em um metamodelo, próprio da ferramenta, na forma de Entidades e Atributos. Estas Entidades e Atributos representam objetos na visão computacional. Partindo dessas Entidades, a ferramenta proposta neste artigo é capaz de gerar arquivos de padrão java, form e sql. A geração de código na JAW é feita utilizando padrões de código e templates da API Jakarta Velocity gerados seguindo o modelo Enterprise JavaBeans (EJB).*

1. Introdução

Atualmente, os custos com desenvolvimento de projetos, documentações e padronizações de código são muito altos e de grande influência na qualidade e no tempo de desenvolvimento de sistemas de informação, devido ao tamanho, sofisticação e tecnologias utilizadas [Leon, 2000].

Partindo das dificuldades citadas e com o aumento da complexibilidade de desenvolvimento, o grau de previsiblidade dos sistemas a serem desenvolvidos pode ser totalmente distinto do previsto, caso não haja um controle de padrão de código. Segundo [Pressman 2001] a não existência de um controle de projeto no sistema a ser desenvolvido poderá causar sérias conseqüências, levando até mesmo ao cancelamento do projeto e ao caos.

Seguindo padrões da *Unified Modeling Language (UML)* pode-se identificar situações de reusabilidade e flexibilade no desenvolvimento de projetos com alta complexiblidade [Schmidt, Fayad e Johnson, 1999]. Com estas situações fica claro a

existência de etapas que podem ser automatizadas por uma ferramenta que execute as tarefas repetitivas, aumentando assim a produtividade, qualidade de código, facilidade de manutenção e interpretação de código [Rocha, Nogueira, Prudêncio, Andrade e Souza, 2006].

Este artigo propõe uma ferramenta, denominada JAW, que tem como funções a conexão com qualquer banco de dados que possua *interface* JDBC; a extração do metadado do banco; a conversão destes metadados em um metamodelo próprio. A ferramenta realiza um merge do metamodelo produzido com a API (*Application Programming Interface*) *Jakarta Velocity* e seus *templates*, resultando na geração do código das classes, incluindo interfaces gráficas e o esquema SQL do banco de dados no qual a ferramenta foi conectada.

A JAW possui uma *interface* de importação e exportação de arquivos XML (*eXtensible Markup Language*) seguindo os padrões determinados para a ferramenta. Possui também uma extensão própria de arquivos para salvar e reutilizar trabalhos processados por ela.

O artigo está dividido da seguinte forma: seção 2 - trabalhos e ferramentas relacionadas; seção 3 - padrões utilizados; seção 4 - arquitetura da JAW; seção 5 – conclusão, resultados obtidos e possíveis sugestões de melhorias para a ferramenta.

2. Trabalhos Relacionados

A geração de código e reusabilidade tem sido foco em empresas e fundações que utilizam Programação Orientada a Objetos (POO). Diante disso, muitos trabalhos e pesquisas tem sido voltados para a implementação de ferramentas capazes de automatizar as tarefas de desenvolvimento de sistemas, de forma simples, padronizada e funcional.

DAP-EJB (*Distributed Adapters Pattern with EJB*) e PDC-EJB (*Persistent Data Collections with EJB*) são padrões utilizados na estruturação de aplicações já existentes, que possuam banco de dados e *intefaces* gráficas em outra linguagem de programação não Orientada a Objetos (OO) [Dias, Borba, 2002].

Um exemplo da utilização destes padrões é apresentado pela ferramenta Xspeed. A Xspeed, utilizando-se de padrões, tem como objetivo aumentar a produtividade no desenvolvimento de aplicações para ambientes distribuídos. Ela recebe como entrada um arquivo XMI contendo um modelo UML de determinada aplicação. A partir do modelo ela realiza a geração automática do código para uma plataforma específica [Rocha, Nogueira, Prudêncio, Andrade, Souza, 2006].

Há ainda várias pesquisas acadêmicas que envolvem refatoração e engenharia reversa para automatização de desenvolvimento como a ferramenta MADE [Schiavoni, 2004] que trabalha reengenharia a partir de classes Java e a ferramenta GeCA [Steinmacher, Amorim, Schiavoni, Huzita, 2006] que utiliza como entrada pacotes jar.

3. Padrões Utilizados

Dentre vários padrões existentes, foram escolhidos os mais compatíveis com as nossas necessidades e aplicações. A justificativa para a escolha deles será apresentada na próxima seção. Para representação do modelo de dados em diagramas foi escolhida a UML e para o código a ser gerado o EJB.

3.1. EJB - Enterprise JavaBeans

Até a versão J2EE 1.4, a plataforma Java não possuía uma forma simples para mapear objetos em um banco de dados [Gonçalves, 2007]. De forma muito complexa havia a possibilidade de mapear entidades para o banco de dados, porém, isto exigia, obrigatoriamente, um *container* EJB. Com a JPA (*Java Persistence API*) definida na JSR-220 EJB 3.0, o mapeamento objeto/relacional na plataforma Java passou a ser padronizado.

Antes desta definição para objetos persistentes, o programador não precisava atentar-se às interfaces ou classes especiais as quais seu objeto implementaria para ser persistido. Com isto, o uso de objetos Java regulares se popularizou devido sua facilidade de implementação e compreensão de código na manutenção. Esse modelo de persistência ficou conhecido como POJO (*Plain Old Java Object*), algo como "Bom e Velho Objeto Java" [Gonçalves, 2007].

Novas tecnologias para persistência surgiram. Entre elas a JPA que é baseada no conceito POJO. Na JPA, os objetos persistentes são denominados Entidades (*Entities*). Para a JPA, uma entidade é um objeto simples (POJO), que representa um conjunto de dados persistindo no banco [Gonçalves, 2007].

Como Entidades são definidas por simples classes Java sem relação com *frameworks* ou bibliotecas, elas podem ser abstratas ou herdar de outras classes sem nenhuma restrição. As figuras 1 e 2 apresentam o exemplo de uma classe Java com características e denominação de um POJO. Na figura 1, apresentamos uma representação da classe Pessoa em forma de um diagrama UML.

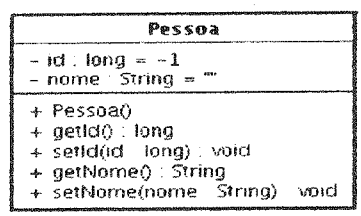

Figura 1 – Diagrama da classe Pessoa

Na figura 2, temos a representação da mesma classe na forma de seu código fonte, no padrão EJB 3.0.

```
public class Pessoa {
   private long id = -1;
   private String nome = "";
   public Pessoa() {
   }
   public long getId() {
return id;
   }
   public void setId(long id) {
      this.id = id;
   }
   public String getNome() {
      return nome;
   }
   public void setNome(String nome) {
      this.nome = nome;
   }
}
```

Figura 2 – código-fonte da classe Pessoa

O padrão divide os objetos a serem persistidos em classes, representadas por um nome, um conjunto de atributos e de métodos de acesso para estes atributos. Os atributos possuem um nome e um tipo. Por exemplo: a classe Pessoa possui seu nome

(Pessoa) e um conjunto de atributos (nome e id). Os atributos também possuem tipos sendo que o atributo nome é do tipo String e o atributo id é do tipo long.

4. A Ferramenta JAW

Em princípio, havia uma proposta de implementar uma conexão genérica para a importação de metadados, nele haveria interface de conexão com qualquer banco de dados por meio do *driver* JDBC. A partir daí seria feita a geração de código. Porém, após uma análise de requisitos mais profunda, surgiu a necessidade de tratamento e armazenamento dos dados para a posterior geração de código.

A arquitetura da ferramenta proposta passou a ser dividida em módulos, seguindo a MDA (*Model Driven Architecture*). O módulo inicial trata a importação dos dados, a partir dela são feitos o tratamento e o armazenamento, e a seguir a geração de código a partir dos dados extraídos. A arquitetura é representada na Figura 3.

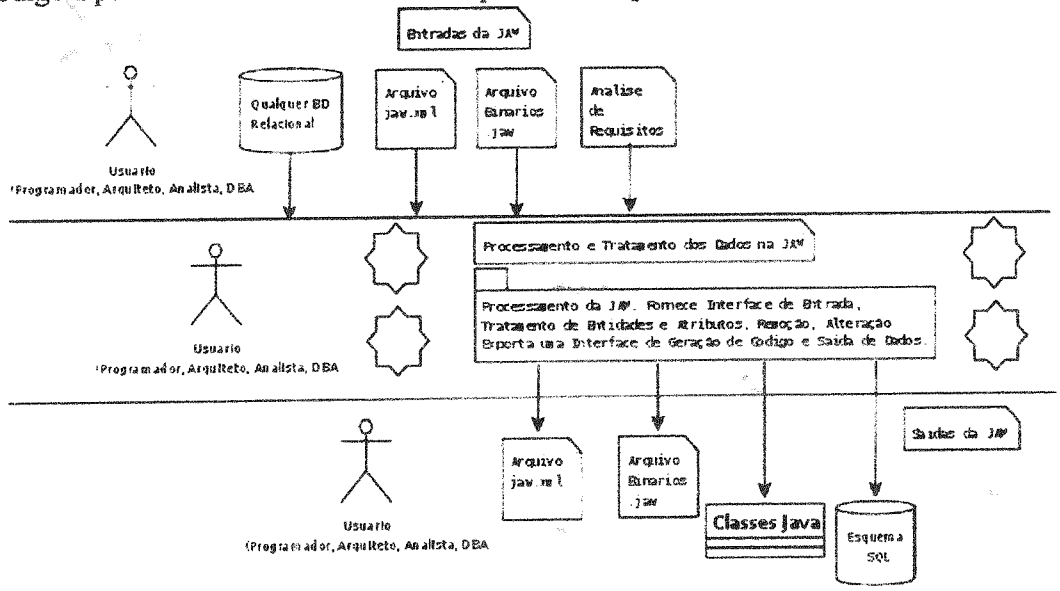

Figura 3 – Arquitetura da ferramenta

A implementação da ferramenta é subdividida conforme a Figura 4. Ela representa os pacotes implementados. O armazenamento de dados é tratado pelo pacote de entrada e saída uma vez que, para a ferramenta, a geração de código pode ser feita a partir da importação de dados por meio de conexão JDBC, por arquivos XML ou por arquivos binários da própria ferramenta.

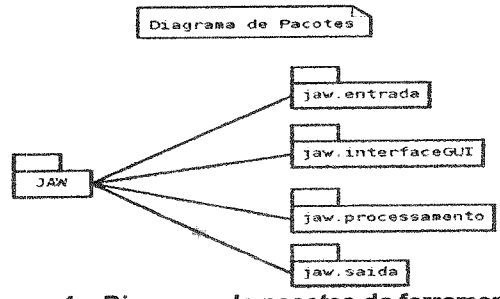

Figura 4 – Diagrama de pacotes da ferramenta

As próximas subseções apresentam o metamodelo da ferramenta; o processo de

tratamento e manipulação do metamodelo em memória; importação e exportação de arquivos XML; geração de código; e a definição da extensão do arquivo que foi padronizado para salvar e abrir os mesmos na JAW, com o intuito de possibilitar a manipulação dos mesmos de acordo com a necessidade do usuário.

4.1. – Metamodelo

Uma vez que o domínio da aplicação de um determinado sistema possua um modelo de banco de dados definido ou um banco seja criado para suprir as necessidades de alguma regra de negócio, existe a possibilidade de conectarmos a ferramenta para auxiliar a criação das classes do sistema ou até mesmo a extração do esquema SQL do banco e a geração de classes.

Para a conversão de dados de entrada e a padronização dos mesmos para a geração da saída, constatou-se a necessidade da criação de um tipo comum a ser tratado pela ferramenta. Isso permite maior flexibilidade da ferramenta podendo ela ser adaptada para outras funcionalidades além das previstas neste trabalho. A definição deste tipo foi baseada na proposta do EJB e divide-se em dois metamodelos: Entidade e Atributo. A figura 5 ilustra o diagrama da classe Entidade.

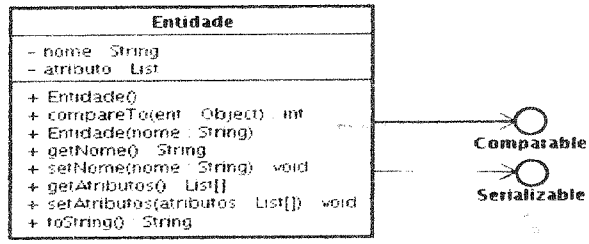

Figura 5 – Diagrama da classe Entidade

Podemos observar na figura 5 que a classe Entidade possui um atributo do tipo java.util.List. Isto significa que uma Entidade pode possuir uma coleção de atributos. A definição do Atributo para a JAW é apresentada na Figura 6.

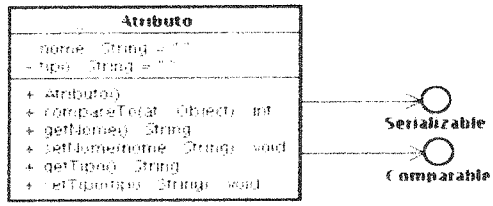

Figura 6 – Diagrama da classe Atributo

Este metamodelo utilizado pela ferramenta servirá como uma linguagem comum a ser utilizada para a troca de informações entre todos os módulos da JAW. Todos os módulos trabalham sobre o mesmo modelo garantindo assim interoperabilidade e flexibilidade.

4.2. – Importação dos dados

A partir de alguns *drivers* JDBC é possível extrair metadados, essa afirmação vem em prol da identificação de alguns *drivers* que não implementam essa funcionalidade.

A conexão JDBC é um padrão especificado pela *Sun Microsystem*. Ele possui métodos idênticos para todos as conexões, com isso, possibilita implementar uma conexão genérica e funcional para a extração de metadados, para a futura conversão ao

modelo próprio da ferramenta. Na Figura 7 é apresentada a interface GUI (*Graphic User Interface)* da JAW, para a conexão com banco de dados.

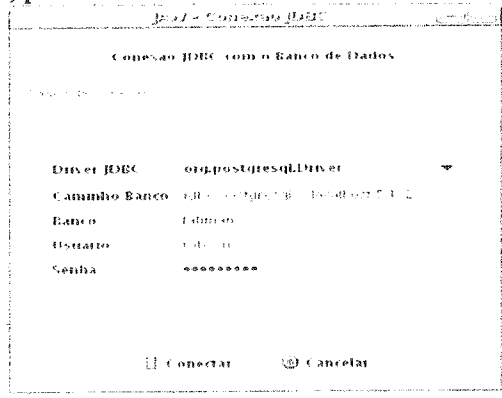

Figura 7 – Tela de Conexão via JDBC genérica.

Os dados apresentados nesta tela são compatíveis com as conexões JDBC de qualquer banco. A utilização da ferramenta JAW em um determinado banco depende dos dados de conexão do mesmo, da existência de um *driver* JDBC e da possibilidade dele exportar metadados.

A ferramenta trata uma tabela em um banco de dados como uma Entidade. Seguindo o metamodelo pré-definido, a Entidade possui um nome e um conjunto de Atributos com nomes e tipos. Com a extração destas informações a partir de uma tabela em um banco de dados há a possibilidade da criação de uma classe; ela irá representar um objeto compatível à arquitetura lógica da tabela no momento da implementação do sistema. A Figura 8 apresenta um exemplo do esquema de tabela do banco de dados.

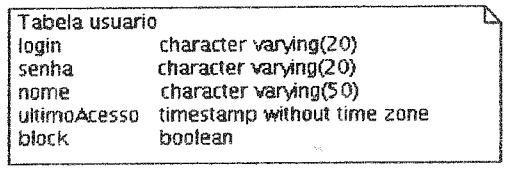

Figura 8 – Esquema da tabela Usuário

Os dados que podem ser extraídos desta tabela são: nome da tabela, nome dos campos e tipo de dados dos campos. Diante disto, há um mapeamento direto entre as informações extraídas do banco de dados e o metamodelo apresentado para a ferramenta.

Devido à existência de vários tipos para atributos em banco de dados, a ferramenta aqui apresentada utiliza uma tabela de conversão de tipos para normalizar as possibilidades em um formato padrão. Esta normalização é mais restrita que as possibilidades de tipos em BD e assemelha-se com a IDL (*Interface Definition Language*) da tecnologia CORBA (*Common Object Request Broker Architecture).* A Tabela 1 apresenta alguns exemplos desta normalização.

Tabela 1 – Conversão dos dados de entrada na JAW

tipo de entrada	***Conversão para a JAW***
Varchar	String
Text	String
byte[]	String

4.3. – Tratamento e Armazenamento

Mais do que tratar o mapeamento de atributos entre uma tabela e uma Entidade, notou-se a necessidade do tratamento de várias Tabelas/Entidades em sistemas convencionais. Além disso, há a possibilidade de nem todas as tabelas pré-existentes em um sistema computacional ser utilizada pela ferramenta. Para o tratamento destas entidades, a JAW possui uma interface com o usuário que permite selecionar as entidades desejadas antes do processamento final.

Além de tratar as Entidades importadas via JDBC a ferramenta permite a criação de novas Entidades para completar o modelo de dados. Além disso, é permitido ao usuário alterar nomes e tipos de entidades e atributos. A ferramenta apresenta também a possibilidade de criar um projeto sem nenhuma referência a um banco de dados pronto ou a um arquivo XML. Na JAW podemos criar entidades e adicionar atributos e seus tipos diretamente a partir de sua GUI.

Para simplificar a continuidade de um trabalho iniciado, a JAW oferece a possibilidade de armazenar o metamodelo em arquivos. Para isso, há a possibilidade de utilizar dois modelos de arquivos: o XML e o binário. Os arquivos gerados na ferramenta e exportados em jaw.xml podem ser abertos em navegadores e modificados em qualquer software que tenha suporte a arquivos XML. Isso torna o trabalho manipulável em qualquer outra ferramenta com essa interface de importação.

Os arquivos binários possuem extensão .jaw que sofrem a restrição de só serem tratados pela própria ferramenta, pois, os mesmos utilizam uma interface de serialização implementada pelo metamodelo. A figura 9 abaixo apresenta a interface gráfica com o usuário citada.

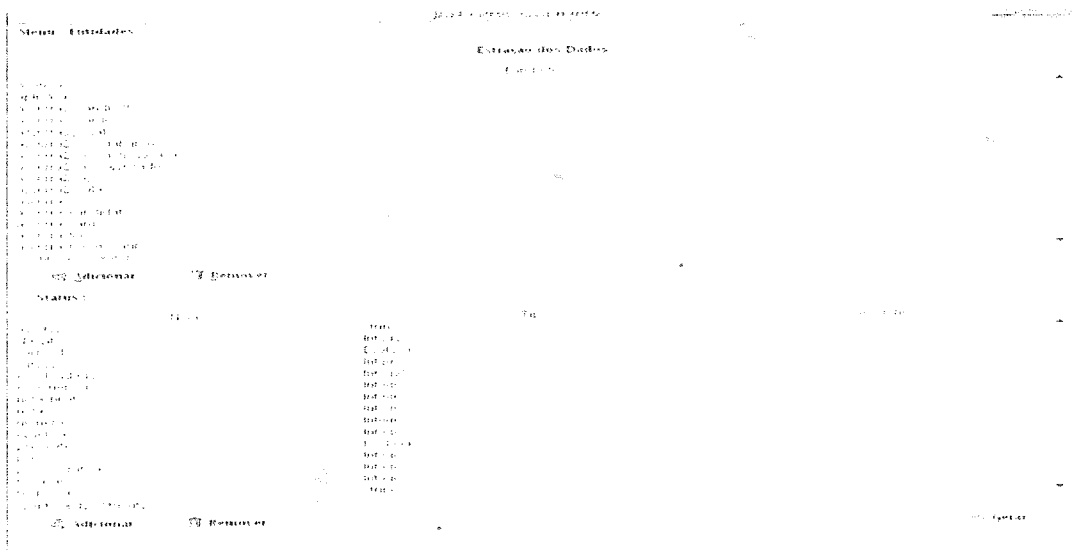

Figura 9 – Tela para a manipulação de dados

Esta tela permite todo o tratamento dentro da JAW: remoção de Entidades e Atributos, exportação e importação XML, além da possibilidade de armazenamento em arquivos binários serializados no metamodelo da ferramenta e, por fim, possui um botão para a geração de código em Java, Form XML ou SQL. Ao pressionar o botão *"gerar"* pode-se escolher um *template* dentre os existentes ou criar um novo, conforme a necessidade.

4.4. – Geração de código

Hoje em dia, pode-se encontrar diversas ferramentas baseadas em *templates*. Tem-se como exemplos desses softwares o Netbeans 5.0 com sua possibilidade de refatorar classes automaticamente e encapsular as propriedades de uma classe selecionada criando métodos de acesso *getters* e *setters* da mesma.

Templates são utilizados para a geração de documentos que seguem um determinado padrão. Todo tipo de documento com uma determinada padronização de formatação é capaz de ser gerado a partir de um *template*. Assim, pode-se gerar códigos-fonte de XML, HTML, documentos RTF e muitos outros.

Entre as várias possibilidades de geração de código utilizando *templates* o *framework Jakarta Velocity* é apresentado como uma alternativa viável devido a sua simplicidade de geração e integração a aplicações. Um exemplo de programa que utiliza *o framework Jakarta Velocity* é o Poseidon for UML, ferramenta *CASE* desenvolvida em Java. O Posseidon utiliza os templates para a geração de código Java a partir de diagramas UML [Silveira, Steil, 2006]. Assim, a JAW utiliza o *framework Velocity* para a geração de código de saída.

Os *templates* são arquivos com grande parte de sua estrutura e formatação prontas contendo diretivas para receber parâmetros que serão inseridos e ações a serem realizadas de acordo com o sistema que os utiliza. Com diretivas pode-se imprimir na tela ou em um arquivo os valores de um objeto tipo java.util.*Vector* , um *array* comum e também invocar métodos de objetos [Silveira, Steil, 2006].

Diretivas é a parte principal da *Velocity Template Language* (VTL). Essa forma é usada para incorporar conteúdo dinâmico no arquivo de *template*. Toda diretiva é precedida pelo caracter "#", conforme ilustrado na figura 10.

```
1 #foreach ($atributo in entidade.getAtributos())
2 #set (\$ferramenta = "JAW")
```

Figura 10 – Exemplo de VTL

A linha de código 1 é responsável por fazer um laço em uma diretiva vinda de um merge feito entre o *template* e uma classe Java. Ele se estenderá até que o objeto seja "descarregado". A linha 2 cria uma variável chamada $ferramenta com o conteúdo JAW que lhe foi atribuído.

O funcionamento básico em uma aplicação utilizando o *Velocity* executa basicamente os seguintes passos: inicializa o contexto, adiciona a ele objetos, carrega um *template* e realiza o merge (mistura entre o, conteúdo do *template* com o objeto passado para o mesmo). Os contextos são as formas usadas no *Velocity* para fornecer dados para o mecanismo de *templates*. Um contexto é uma instância da classe org.apache.velocit.*Velocity*Context que utiliza internamente um *HashMap* para manter um conjunto de pares String-objeto. Possui dois métodos importantes: put (String chave, Object valor) e get(String chave).

Uma vez preenchido o contexto prossegue-se para o *merge*. Durante o *merge* ("concatenação" ou "mistura"), o *Velocity* interpreta as diretivas e extrai os dados necessários do contexto gerando a saída desejada.

Para a JAW serão elaborados *templates* compatíveis com sua necessidade de acordo com o que prevê o projeto descrito, no entanto, é importante citar que com o

Velocity é possível produzir e facilitar em grande escala o trabalho repetitivo de algumas situações rotineiras de um programador.

A geração de códigos na JAW é feita para cada Entidade selecionada na tela principal. A tela principal instancia um Gerador e chama um método que trabalha cada entidade individualmente. O diagrama de classes do Gerados é apresentado na Figura 11.

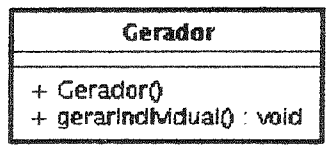

Figura 11 – diagrama da classe Gerador

Com essa classe e um *template* construído para uma determinada aplicação faz-se possível gerar arquivos com a extensão de acordo com as aptidões do desenvolvedor. Com criatividade, um desenvolvedor é capaz de desenvolver uma ferramenta para geração de código e *scripts* SQL entre outros sem maiores dificuldades.

Com Entidades e Atributos tratados em nossa ferramenta, somente devemos escolher as extensões de arquivos a ser gerado indicando um caminho onde os mesmos serão armazenados e disparar o método *gerar*. Com os arquivos prontos, o desenvolvedor deve fazer as alterações necessárias de acordo com as necessidades do *software* que está a desenvolver.

5. Conclusão

Com a pesquisa desenvolvida no decorrer deste projeto, trabalhos e ferramentas foram visto e analisados, podemos identificar que nossa idéia não é particular no horizonte do desenvolvimento de *software*. Analisando um projeto de *software* percebemos que há sempre códigos semelhantes que são empregados para cumprir determinadas tarefas em um domínio de aplicação. Uma vez encontrado estes padrões de códigos repetidos há a possibilidade de automatizar esta atividade de desenvolvimento dando, assim, uma margem de vantagem para a equipe de desenvolvimento com ganhos significativos no cumprimento do cronograma e na padronização do código fonte.

Estas atividades repetitivas com possibilidade de automatização possuem um padrão definido, porem dependem de dados específicos para cada entidade no processo de desenvolvimento. A possibilidade de alimentar esta automatização utilizando os metadados extraídos do modelo Entidade-relacionamento via interface JDBC ampliam ainda mais as possibilidades da utilização de tal ferramenta.

Para que os dados extraídos do banco sejam tratados de uma forma homogênea foi necessário criar um metamodelo consistente e adaptável conforme especificações existentes em outros modelos genéricos como POJO e EJB. Mais do que padronizar os dados distintos em um formato próprio, o metamodelo interno da ferramenta facilita a exportação dos dados importados pois a padronização da entrada implica em uma saída formatada e conhecida.

Para a integração da ferramenta em um ambiente de desenvolvimento de software adotou-se ainda a exportação de dados no formato XML que permite a interoperabilidade entre a ferramenta proposta neste artigo e outras ferramentas como planilhas eletrônicas ou *web services*.

A geração de artefatos de *software* depende de padrões e decisões quanto a escolha de tecnologias. A possibilidade de alterar padrões sem a alteração do código fonte da ferramenta fez-nos optar pela utilização do *framework Jakarta Velocity* para permitir este baixo acoplamento entre a ferramenta e os modelos gerados.

Este trabalho possui algumas melhorias que já foram identificados ou que constam em nosso cronograma de trabalhos futuros, como:

- A implementação de um editor de *templates* incorporado a ferramenta, e criação de novos *templates;*
- Um módulo com interface visual para adicionar *drivers* JDBC de acordo com o banco a ser conectado;
- Melhorias no módulo de tratamento de Entidades e Atributos;
- Importação de metadados a partir de classes, pacotes e diagramas UML;
- Representação gráfica da estrutura relacional do metamodelo importado.

6. Referências

APACHE-b. **Velocity – Users Guide.** Disponível em <http://jakarta.apache.org/velocity/docs/user-guide.html>. Acessado em Jul. 2006.

DEITEL, H. M. **Java, como programar.** Bookman. 2003.

DIAS, K., BORBA, P. **Padrões de projeto para estruturação de aplicações distribuídas.** Enterprise JavaBeans. In Second Latin American Conference on Pattern Languages of Programming, SugarLoafPLoP'2002, pages 55-86, Itaipava, Brazil, 5th-7th August 2002.

GONÇALVES 1, Edson. **Desenvolvendo Aplicações WEB com Netbeans.** Ciência Moderna, 2007

GONÇALVES, Edson. **Dominando o Netbeans.** Ciência Moderna 2006.

LEON, A.. **A Guide to Software Configuration Management.** Norwood, MA Artech HousePublishers. 2000.

PRESSMAN, R. S.. **Software Engineering, A Practitioner's Approach.** McGraw-Hill. 2001.

ROCHA, Lincoln S., NOGUEIRA, Rute, PRUDÊNCIO, João Gustavo, ANDRADE, Rossana Maria de Castro, SOUZA, Jerffeson Teixeira de. **XSpeed: Uma ferramenta para geração de aplicações distribuídas baseadas em padrões.** 2006

SCHIAVONI, Flávio Luiz. **Made - Uma ferramenta para automação de desenvolvimento de sistemas de informação para a Web**. Monografia de Especialização em Especialização Em Desenvolvimento Para a Web. Universidade Estadual de Maringá. 2004.

SCHMIDT, D.C., FAYAD, M.E., JOHNSON, R.E.. **Building Application Frameworks: Object-Oriented Foundations of Framework Design.** Wiley Computing Publisher.1999.

STEINMACHER, Igor ; AMORIM, Éderson ; SCHIAVONE, Flávio Luiz ; HUZITA, Elisa Hatsue Moriya . **GeCA: Uma Ferramenta de Engenharia Reversa e Geração Automática de Código.** In: Simpósio Brasileiro de Sistemas de Informação (SBSI), 2006, Curitiba. III Simpósio Brasileiro de Sistemas de Informação, 2006.