UMA PROPOSTA DE UM METAMODELO PARA IMPLEMENTAÇÃO DE SOFTWARE ORIENTADO A OBJETOS ATRAVÉS DA GERAÇÃO AUTOMÁTICA DE CÓDIGO-FONTE

Jonnathan dos Santos Carvalho[1,2], Aline Gomes Cordeiro[1], Marcelo M. Feres[2]

[1]Centro de Informática Hospital Geral de Guarus (HGG)
Campos dos Goytacazes RJ Brasil

[2]Centro Federal de Educação Tecnológica de Campos (CEFET-Campos)
Campos dos Goytacazes RJ Brasil

{aline,joncarv}@hgg.rj.gov.br, mferes@cefetcampos.br

Resumo. *No decorrer do desenvolvimento de um software um dos riscos mais visíveis e talvez o maior obstáculo na implementação está relacionado ao gerenciamento do tempo. Todos os prazos de entrega das versões do software devem ser respeitados, mas nem sempre isso é possível, muitas vezes por atraso na codificação. Este artigo apresenta um metamodelo para implementação de software, que futuramente dará origem a uma ferramenta de geração automática de código-fonte com o objetivo de tornar qualquer padrão de desenvolvimento transparente ao programador, diminuindo consideravelmente o tempo de codificação dos artefatos que compõem o software.*

Palavras-chave: projeto de software, metamodelo, implementação

Abstract. *During the development of software one of the most visible risks and perhaps the biggest implementation obstacle relates to the time management. All delivery deadlines software versions must be followed, but it is not always possible, sometimes due to delay in coding. This paper presents a metamodel for software implementation, which will rise to a development tool for automatic generation of source code, in order to make any development pattern transparent to the programmer, significantly reducing the time spent in coding artifacts that make up the software.*

Key-words: *software project, metamodel, implementation*

1. INTRODUÇÃO

Com o avanço cada vez mais notável das tecnologias e da crescente demanda por sistemas de informação para suprir as necessidades dos mais variados tipos de negócios, o tempo tem sido um fator crítico e determinante no desenvolvimento de tais soluções. Em termos de gerência de projeto, a baixa produtividade de uma equipe de desenvolvimento de software pode colocar em risco o cumprimento dos prazos e levar, com isso, à falha de projetos. Um dos principais problemas relacionados ao tempo e a

produtividade da equipe está situado na fase de implementação em um processo de desenvolvimento de software. Durante essa fase muito tempo é gasto na codificação e escrever código-fonte muitas vezes pode se tornar uma atividade repetitiva, tornando o tempo disponível um grande obstáculo no cumprimento do cronograma.

Além do tempo e da produtividade, um outro aspecto que também merece destaque é a rotatividade que existe dentro das equipes de desenvolvimento. Toda equipe passa por mudanças e há sempre uma grande rotatividade de recursos humanos, já que as empresas estão sempre buscando profissionais cada vez mais qualificados. Com essa rotatividade, muito tempo é perdido para que um novo desenvolvedor aprenda um padrão de desenvolvimento e codificação adotados, caso exista algum a ser seguido, o que é fortemente recomendado.

Nesse sentido, o objetivo deste artigo é apresentar um metamodelo para possibilitar a implementação de classes de negócio através de uma ferramenta que permita a geração automática de código-fonte, baseado em um padrão de desenvolvimento qualquer, com o intuito de amenizar, ou até mesmo eliminar os problemas acima descritos.

Apesar de ser abordada de uma forma superficial, a ferramenta para geração automática de código-fonte, baseada no metamodelo apresentado, ainda está em fase de desenvolvimento. Assim, este artigo visa principalmente, compartilhar a experiência de desenvolvimento da equipe do PROSAH (Projeto Open-Source de Administração Hospitalar), com o objetivo de ajudar outras comunidades de desenvolvimento para que não passem pelos mesmos problemas.

1.1. TRABALHOS RELACIONADOS

Há muitos trabalhos sendo desenvolvidos com o objetivo de superar a dificuldade da fase de implementação através da geração automatizada de código-fonte. Apesar disso, este trabalho tem como principal objetivo não somente abordar o desenvolvimento de uma ferramenta para geração de códigos, apresentada aqui em forma de protótipo, mas principalmente de solidificar uma estrutura de reutilização, onde através dela outras ferramentas possam ser desenvolvidas. Para [Oliveira 2006], entende-se por reutilização uma atividade que já vem sendo realizada na comunidade de desenvolvimento de software, embora muitas vezes de forma *ad hoc*. Uma vez que idéias, objetos, argumentos e abstrações sempre foram reutilizados por desenvolvedores de software, pretende-se que o metamodelo apresentado neste artigo trabalhe em cooperação com outras comunidades de desenvolvimento, de forma a atingir o principal objetivo da reutilização de software, que em outras palavras, é o desenvolvimento de artefatos de software baseados em um determinado domínio[1] (por exemplo, na área de saúde) e disponibilizá-los em uma base de componentes reutilizáveis.

Embora, como mencionado, existam outros trabalhos que tem como objetivo a implementação de uma ferramenta para geração automática de código, este trabalho não tem a intenção de comparar tais ferramentas, uma vez que o objetivo aqui apresentado é de propor um metamodelo genérico, que possa ser reutilizado e aplicado independente

[1] Domínio, na engenharia de software, pode ser entendido como uma coleção de aplicações, existentes ou futuras, que compartilhem características comuns, isto é, uma classe de sistemas que apresentam funcionalidades similares [Werner e Braga 2005].

do domínio abordado. Nas ferramentas encontradas na literatura, havia uma maior preocupação em disponibilizar a ferramenta, e não os conceitos envolvidos em seu desenvolvimento.

2. OS RISCOS DA FASE DE IMPLEMENTAÇÃO

O ciclo de vida genérico de um software é composto pelas fases de definição, desenvolvimento e manutenção [Pressman 2002]. Um bom relacionamento entre essas fases, independente do processo de software adotado, torna possível a construção de um software com qualidade e que atenda a todos os requisitos dos principais interessados no projeto.

Requisitos mal compreendidos são a maior causa de falha nos projetos de software [Hofmann e Lehner 2001]. Os requisitos devem ser respeitados e limitados, para que o cliente/usuário não fique frustrado com alguma funcionalidade que não tenha sido exigida, ou ainda, para que não haja omissão por parte da equipe de desenvolvimento de qualquer dessas funcionalidades. Além disso, um outro aspecto que merece destaque e que também pode ser considerado uma das causas de falha nos projetos de software está relacionado ao tempo.

Sempre que um projeto é colocado em prática (nesse caso, um projeto de software), um cronograma deve ser criado a fim de estipular prazos e metas a serem alcançadas, mas nem sempre esses prazos conseguem ser cumpridos, dificultando assim, o andamento do projeto. Baseado na experiência dos autores deste artigo, pode-se afirmar que um dos principais motivos que levam a equipe a um estado de bloqueio está na fase de implementação, mais especificamente no que diz respeito à codificação dos artefatos[2] de software. Codificar um artefato de software, como por exemplo, um diagrama de classes, pode se tornar uma tarefa repetitiva e cansativa, levando em consideração que pessoas não se sentem a vontade na realização de atividades robotizadas. Com isso, o tempo que poderia ser melhor distribuído na busca de soluções de implementação mais eficazes durante o desenvolvimento, passa a ter um foco maior na repetição de código-fonte.

Como exemplo, a figura 1 ilustra um tipo de padrão criado pelos autores deste artigo durante o desenvolvimento de um software no domínio hospitalar. Como pode ser observado, a classe PessoaAssistidaNegocio possui uma grande quantidade de atributos. Nesse padrão todas as classes de negócio possuem um método chamado setHelper (figura 2), que transforma todos os objetos do tipo Helper[3]. A implementação desse método acontece em todas as classes de negócio e dependendo da quantidade de atributos da classe, escrever um método desse tipo pode ser uma tarefa exaustiva e sujeita a erros de digitação, já que cada atributo da classe deve ser referenciado.

---

[2] Artefato é o termo usado para qualquer produto do trabalho de desenvolvimento de software, a saber: código, gráficos para a *Web*, esquemas de banco de dados, documentos em texto, diagramas, modelos e assim por diante [Larman 2002].

[3] Helper, no padrão de desenvolvimento utilizado como exemplo neste artigo, é uma classe que faz a comunicação entre as interfaces gráficas e as classes de negócio.

```
1  package prosah.negocio;
2
3  import java.sql.Date;
14
15 public abstract class PessoaAssistidaNegocio extends Negocio {
16
17     private Date dataCadastro;
18     private Integer situacaoCadastral;
19     private String nome;
20     private Date dataNascimento;
21     private Boolean dataNascimentoAparente;
22     private EnderecosNegocio endereco;
23     private Integer numResidencia;
24     private Double altura;
25     private String email;
26     private Integer estadoCivil;
27     private Character fatorRh;
28     private ProfissaoNegocio profissao;
29     private String nomeMae;
30     private String nomePai;
```

Figura 1. Trecho de uma classe de negócio do padrão de desenvolvimento utilizado como exemplo atributos de classe.

```
484     public void setHelper(Helper helper) {
485         if (!(helper instanceof PessoaAssistidaHelper))
486             return;
487         PessoaAssistidaHelper h = (PessoaAssistidaHelper) helper;
488         setIdPessoaAssistida(h.getId());
489         setNome(h.getNome());
490         setDataCadastro(h.getDataCadastro());
491         setSituacaoCadastral(h.getSituacaoCadastral());
492         setDataNascimento(h.getDataNascimento());
493         setDataNascimentoAparente(h.getDataNascimentoAparente());
494         setDataNascimento(h.getDataNascimento());
495         setNumResidencia(h.getNumResidencia());
496         setAltura(h.getAltura());
497         setEmail(h.getEmail());
```

Figura 2. Trecho de uma classe de negócio do padrão de desenvolvimento utilizado como exemplo método setHelper.

Um outro risco associado à fase de implementação está relacionado com a rotatividade de recursos humanos dentro das equipes de desenvolvimento. Atualmente, as empresas encontram-se cada vez mais competitivas, investindo cada vez mais na expansão de seus negócios e, conseqüentemente, há sempre uma grande demanda por recursos humanos qualificados. Essa rotatividade, apesar de também trazer benefícios para a equipe, como a experiência adquirida por um desenvolvedor em outros projetos, pode também causar atraso na execução do projeto, já que um novo integrante antes de começar a codificar deve aprender o padrão de desenvolvimento e codificação. Para que esse aprendizado aconteça de forma efetiva, é necessário que um tempo seja gasto até que ele esteja confiante e apto a começar a escrever suas primeiras linhas de código.

Mas, quando o tempo disponível é curto, esse aprendizado pode se tornar um problema, muitas vezes sem solução, acarretando na falência do projeto.

Para tentar solucionar essas questões, uma ferramenta está sendo desenvolvida com o objetivo de tornar qualquer padrão de desenvolvimento transparente aos programadores, possibilitando que se preocupem apenas com a implementação dos requisitos funcionais identificados durante a análise. Assim, mais tempo pode ser gasto para tratar a parte específica do domínio que estiver sendo informatizado, como por exemplo, em um domínio hospitalar, fazer o controle de vagas para uma determinada especialidade médica.

3. A EXPERIÊNCIA DO PROSAH

O metamodelo apresentado neste artigo foi obtido durante o desenvolvimento do software de gestão hospitalar PROSAH (Projeto Open-Source de Administração Hospitalar). O PROSAH é um projeto de desenvolvimento de software iniciado no Hospital Geral de Guarus, cujo objetivo é desenvolver um software hospitalar de forma livre, permitindo que outras unidades de saúde também o utilizem. Para isso, a maior preocupação da equipe consiste em obter uma solução reutilizável, que fosse capaz de abranger os diferentes processos existentes nas diferentes realidades de cada unidade de saúde. O modelo de negócio alcançado até o momento não será ilustrado aqui, uma vez que não faz parte do foco deste artigo.

O principal objetivo ao se buscar um metamodelo para geração automática de código-fonte está relacionado ao esforço gasto para codificar os artefatos da análise e principalmente na manutenção necessária nos códigos à medida que os diagramas fossem atualizados, considerando que o projeto é baseado em iterações, onde cada iteração apresenta uma nova estratégia para coleta de requisitos, análise, documentação, implementação, etc.

Apesar de já existirem outros projetos referentes à geração automática de código-fonte, o que se propõe aqui é uma estrutura que possa ser adaptada a qualquer projeto de desenvolvimento de software orientado a objetos, independente da linguagem de programação adotada na implementação e do padrão de desenvolvimento utilizado.

4. METODOLOGIA

O metamodelo proposto foi criado baseado na experiência da equipe de desenvolvimento do PROSAH, que passou por todas as dificuldades descritas nas seções anteriores.

Para desenvolver a ferramenta de geração automática de código-fonte, foi proposto um modelo genérico a ser seguido (figura 3), com intuito de promover e adotar a reutilização de software, nesse caso na construção de uma infra-estrutura reutilizável. De acordo com [Guizzard 2000] a reutilização pode acontecer em um nível mais alto de abstração do que somente através de componentes de código, como, por exemplo, a

experiência embutida em padrões de solução, arquiteturas de estruturação de elementos do domínio e, idealmente, estruturas capazes de capturar, de forma explícita, o conhecimento disponível em uma classe de problemas. Assim, o modelo apresentado, baseado em uma estrutura de programação orientada a objetos, oferece uma visão dos componentes mais comumente utilizados no desenvolvimento de sistemas orientados a objetos e os relacionamentos entre tais componentes.

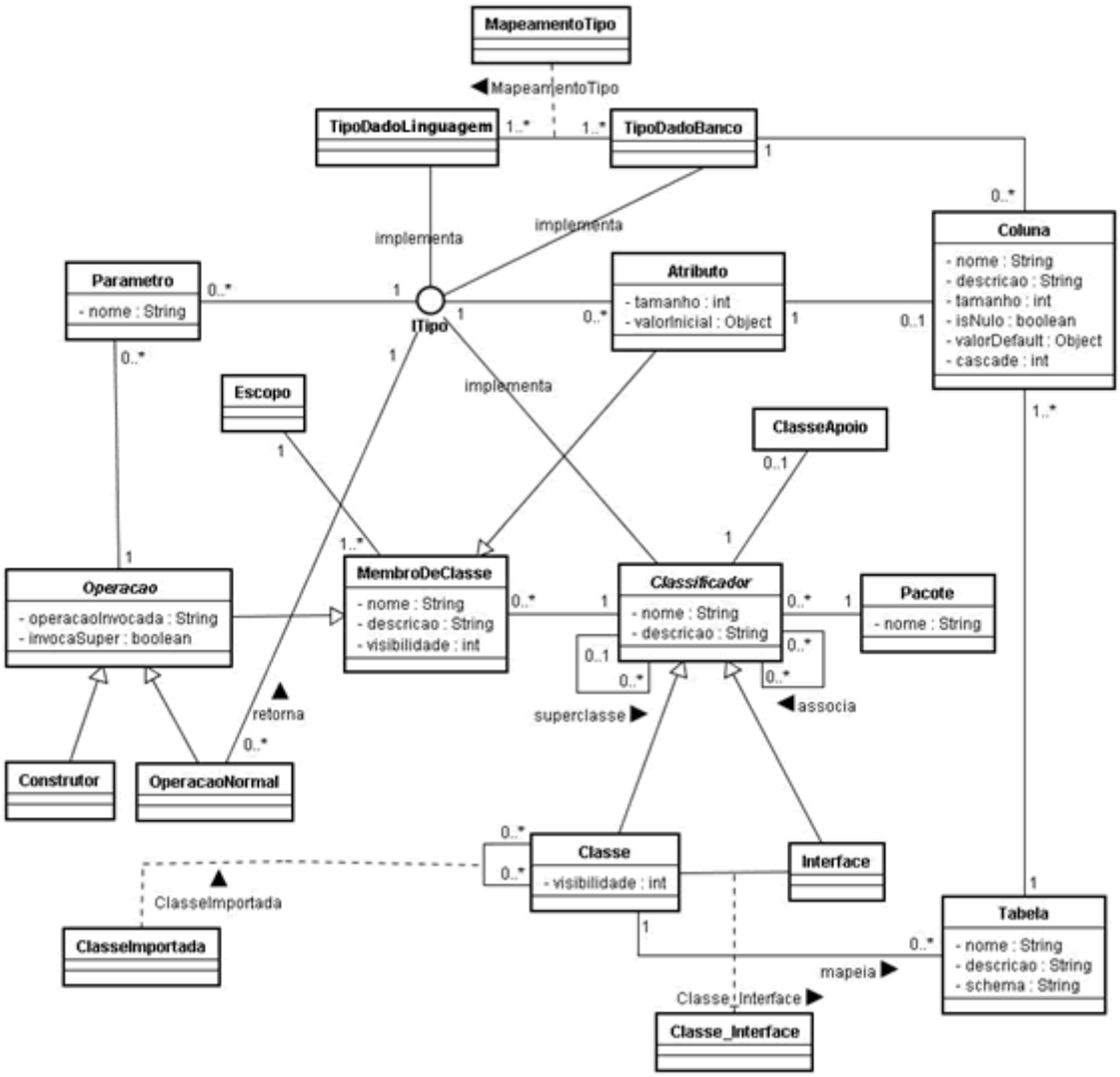

Figura 3. O metamodelo de programação.

Devido ao alto grau de abstração, o modelo apresentado está sendo tratado como um metamodelo. Como o objetivo deste artigo é apresentar o metamodelo, alguns conceitos adotados são abordados de forma mais clara nas sub-seções a seguir, para uma melhor compreensão.

4.1 CLASSIFICADOR (*CLASSIFICADOR*)

Um Classificador pode ser uma classe, uma interface ou um tipo primitivo de dados [Veronese *et al.* 2002]. Como os tipos primitivos não possuem a mesma representação dos outros tipos de classificadores, não foi considerado no metamodelo abordado nesse artigo. Baseado no metamodelo apresentado, um classificador possui então duas subclasses, Classe e Interface . Cada uma dessas subclasses herda as características e o comportamento que um objeto Classificador possui, além de suas características e comportamento próprio. Essas características e comportamentos que um Classificador pode ter está representado no metamodelo através da classe MembroDeClasse que se relaciona com a classe Classificador e por sua vez, possui como subclasses Atributo e Operacao . Em termos de orientação a objetos, as classes Atributo e Operacao representam as características e o comportamento mencionados anteriormente.

Podem ser observados ainda, dois auto-relacionamentos na classe Classificador . O primeiro, identificado como superclasse , refere-se à extensão, já que na orientação a objetos uma classe ou uma interface pode estender funcionalidades de um outro objeto, enquanto o auto-relacionamento associa está relacionado com as possíveis associações entre objetos do tipo Classificador.

4.2 CLASSE DE APOIO (CLASSE APOIO)

Ao gerar uma classe ou uma interface[4] através da ferramenta de geração de códigos citado nesse artigo e implementado a partir do metamodelo apresentado, o programador deve implementar a parte que não é genérica, ou seja, a parte que não tem como prever, como por exemplo, particularidades de cada domínio, ou em outras palavras, os métodos de negócio. Esse código extra referente ao domínio da aplicação, se escrito diretamente nas classes geradas a partir do metamodelo pode ser perdido, caso necessite que a classe seja gerada novamente. Para evitar esse problema, surge o conceito de classe de apoio, representado pela classe ClasseApoio e que tem o objetivo de servir de um repositório para tratar especificamente as particularidades de cada domínio. Assim, quando houver a necessidade da geração de uma classe novamente, seja por mudanças no padrão de desenvolvimento e codificação, ou por qualquer outro motivo, a parte específica não será sobrescrita, já que está armazenada em outra classe.

Para uma melhor compreensão sobre os demais conceitos abordados, a tabela 1 a seguir lista algumas classes que compõem o metamodelo, e suas respectivas descrições.

Tabela 1. Classes do metamodelo

Classe	***Descrição***
ClasseImportada	Considerando que algumas classes importam outras, o relacionamento de um objeto do tipo Classe com ele mesmo deu origem a classe ClasseImportada, que mantém informações sobre as importações necessárias em uma determinada classe.

[4] Uma interface é um contrato entre a classe e o mundo externo. Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface [Sun Microsystems 2008].

Coluna	Classe responsável pela criação de objetos do tipo Coluna que representam o mapeamento de um determinado atributo para um campo referente a uma tabela do banco de dados.
MapeamentoTipo	Classe responsável pelo relacionamento entre TipoDadoLinguagem e TipoDadoBanco . O tipo de dado String em Java é referente ao tipo Varchar no PostgreSQL, por exemplo.
MembroDeClasse	Um membro de classe, na orientação a objetos, pode ser descrito como um atributo ou uma operação. Cada atributo e operação possui, além de nome e uma descrição, um nível de visibilidade que pode ser public, private, protected ou default.
Atributo	A classe Atributo é responsável por armazenar informações sobre um determinado atributo que pertença a uma classe ou interface qualquer, representada no metamodelo pela classe Classificador.
Metodo	A classe Metodo armazena informações sobre objetos do tipo Metodo, referente a uma operação qualquer de determinada classe ou interface.
Parametro	Uma operação pode estar vinculada a 0 (zero) ou mais parâmetros, ou seja, uma operação, em outra palavras, um método, pode ou não possuir um ou mais valores passados como parâmetro. Assim, essa classe mantém informações sobre todos os parâmetros que um método pode ter.
ITipo	Como vários conceitos abordados utilizam o conceito de tipo (parâmetro, operação, atributo, etc), foi criada uma interface que provê serviços para essas classes.
Tabela	Classe responsável pela criação de objetos do tipo Tabela que representam o mapeamento de uma determinada classe para uma tabela do banco de dados.
TipoDadoBanco	Classe responsável pela criação de objetos do tipo TipoDadoBanco que representam os tipos de dados que o banco de dados utilizado no projeto de software possui.
TipoDadoLinguagem	Classe responsável pela criação de objetos do tipo TipoDadoLinguagem que representam os tipos de dados que a linguagem de programação adotada para codificação possui.

5. DESENVOLVIMENTO DO PROTÓTIPO

Para desenvolver o protótipo inicial do gerador de código, foi escolhida a linguagem Java por se tratar de uma linguagem orientada a objetos bastante difundida, tanto no mercado, como no meio acadêmico [Veronese *et al.* 2002]. Além disso, o Java é uma linguagem que está disponível para diferentes sistemas operacionais, tornando a ferramenta apresentada nesse artigo independente da plataforma utilizada.

A fim de alcançar uma infra-estrutura de implementação que pudesse ser reutilizada em qualquer outro domínio de aplicação, foram buscadas tecnologias capazes de se comunicar umas com as outras. Se o código referente à criação das classes da aplicação fosse uma parte fixa do gerador de código, não seria possível a criação de código-fonte em qualquer linguagem, restringindo apenas a uma linguagem específica. Por exemplo, para gerar uma classe em Java, em algum momento deve-se especificar que

o escopo de uma classe pública em Java é public class <<nome da classe>> , que é diferente do escopo de uma classe em PHP. Essa definição do escopo de uma classe Java não deve estar amarrada diretamente no gerador, caso contrário, o gerador escreveria apenas código Java. Como solução, foi adotado o uso de documentos XML para criação de uma estrutura que especifica a sintaxe da linguagem a ser utilizada na geração dos códigos-fonte. De acordo com o W3C[5], o padrão XML permite descrever uma classe de dados denominada documentos XML. A XML tem como principal objetivo a descrição de informações, além da recuperação e transmissão dessas informações e possibilita armazenar em um mesmo lugar os dados e os chamados metadados, estabelecendo um formato textual que pode ser facilmente entendido quando olhado diretamente. Em outras palavras, a XML descreve dados estruturados que são passíveis de serem interpretados [Martins 2004].

A figura 4 apresenta a estrutura do gerador de código. A partir de uma biblioteca de documentos XML, é possível a criação dos arquivos contendo o código-fonte de acordo com a linguagem utilizada no desenvolvimento de um software.

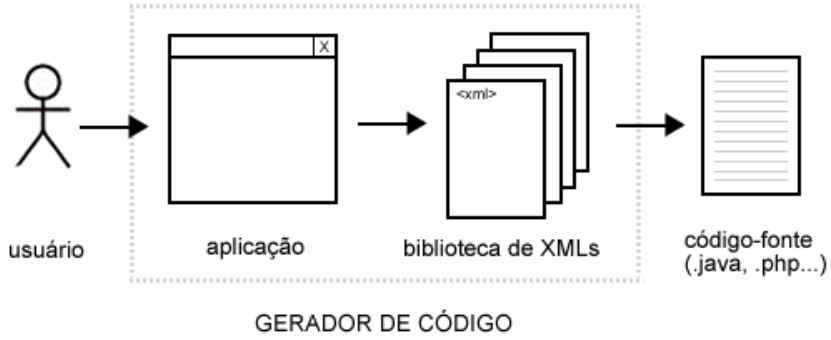

Figura 4. Estrutura do gerador de código.

[5] O W3C (World Wide Web Consortium) é o órgão internacional responsável pelos padrões utilizados na Internet.

```
<?xml version="1.0" ?>
<classe tipo="Negocio" declara_atributos="sim" pacote="package prosah.negocio">
  <escopo>$visibilidade class $nome_classe{</escopo>
  <construtor insere="sim" />
  <metodo nome="get">
    <estrutura_dinamica>public $tipo$ get$atributo$(){ return this.$atributo$; }
      </estrutura_dinamica>
  </metodo>
  <metodo nome="set">
    <estrutura_dinamica>public void set$atributo$($tipo$ $atributo$)
      { this.$atributo$ = $atributo$; }</estrutura_dinamica>
  </metodo>
  <metodo nome="toHelper" assinatura="public Helper toHelper()">
    <estrutura_fixa>{ $classe$Helper helper = new $classe$Helper();
      helper.setId(getID());</estrutura_fixa>
    <estrutura_fixa_classe_abstrata>{ $classe$Helper helper = instanciaHelper
      (); helper.setId(getID());</estrutura_fixa_classe_abstrata>
    <estrutura_fixa_subclasse_abstrata>{ $classe$Helper helper =
      instanciaHelper(); helper = ($classe$Helper) super.toHelper
      ();</estrutura_fixa_subclasse_abstrata>
    <estrutura_fixa_subclasse_concreta>{ $classe$Helper helper = new
      $classe$Helper(); helper = ($classe$Helper) super.toHelper
      ();</estrutura_fixa_subclasse_concreta>
```

Figura 5. Estrutura de um documento XML para geração de classes de negócio em Java.

A criação dos documentos XMLs não é abordada neste artigo, mas para efeito de visualização, a figura 5 ilustra um trecho da estrutura de um documento XML para criação de classes de negócio, baseado no padrão de desenvolvimento e codificação seguidos pelos autores deste artigo e utilizando a linguagem Java como exemplo.

Cada documento XML deve conter a estrutura de um tipo de classe (negócio ou persistência, por exemplo), assim como os métodos referentes ao padrão de desenvolvimento adotado pela equipe. Assim, a aplicação vai interpretar os documentos XMLs e saber exatamente o tipo de classe que deve ser criada.

6. PROTÓTIPO DO GERADOR DE CÓDIGO-FONTE

O gerador de códigos, como mencionado anteriormente, é capaz de gerar arquivos de código-fonte em qualquer linguagem de programação orientada a objetos a partir de uma biblioteca de XMLs que contém a estrutura das classes a serem geradas, além da sintaxe da linguagem. Como não é possível abordar neste artigo todos os aspectos que envolvem a ferramenta, a figura 6 apresenta uma das interfaces que compõem o gerador.

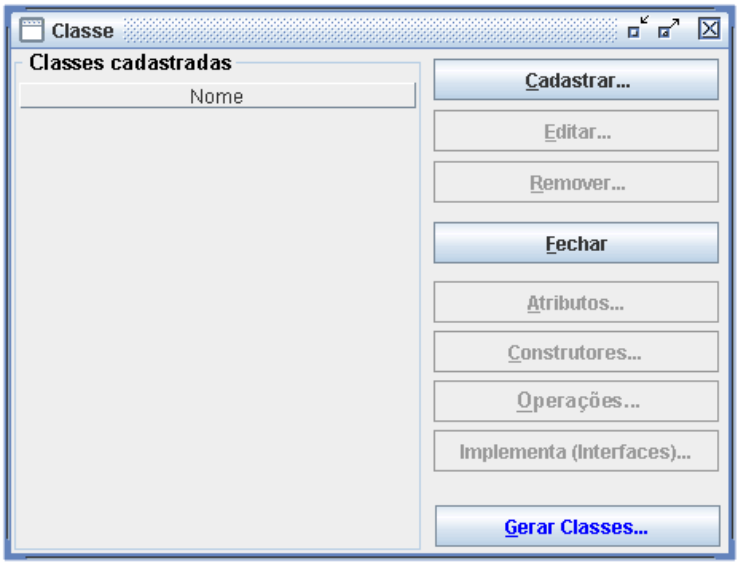

Figura 6. Interface gráfica para geração de classes.

Para gerar as classes, o desenvolvedor deve apenas cadastrar a classe, informando o nome da classe. Todas as informações sobre as classes geradas são mantidas em uma base de dados. Assim, é possível alterar os dados de alguma classe já criada anteriormente, caso seja necessário adicionar um novo atributo ou um novo método. Os botões Atributos , Construtores e Operações possibilitam informar os membros de classe que farão parte da classe. Ao clicar em Cadastrar... , uma nova interface é chamada com os dados a serem preenchidos sobre a classe a ser criada, onde é possível informar, por exemplo, a visibilidade da classe, se é abstrata, se possui superclasse, entre outros. Depois de cadastrar as classes, a geração dos arquivos contendo os códigos-fonte é realizada ao se clicar no botão Gerar Classes... . Feito isso, todas as classes serão criadas (para cada classe será criada uma classe de persistência, negócio, etc.), baseadas na biblioteca de XMLs disponíveis, de acordo com o padrão de desenvolvimento especificado em cada XML. Dessa forma, qualquer padrão de desenvolvimento se torna transparente ao desenvolvedor, já que toda sua estrutura está escrita em um XML específico.

A ferramenta apresentada neste artigo está em constante desenvolvimento, sempre visando seu aperfeiçoamento, uma vez que novas tecnologias estão surgindo para suprir as necessidades de um mercado de desenvolvimento de software cada vez mais exigente.

7. RESULTADOS OBTIDOS

Com a criação do metamodelo proposto, que possibilitou o desenvolvimento de um protótipo inicial da ferramenta de geração de código, foi possível verificar um melhor aproveitamento das atividades do projeto em relação ao tempo, que até então ficavam focadas no esforço para codificar. Assim, foi possível focar em outras atividades que antes existiam apenas na teoria, como por exemplo, testes de unidade.

Ao tornar dinâmica a geração do código-fonte, quando uma nova funcionalidade de um módulo já desenvolvido fosse identificada durante a validação dos requisitos nas

freqüentes reuniões com clientes, a manutenção do código visando tal adaptação se tornou uma tarefa razoavelmente rápida, dependendo somente da complexidade envolvida no desenvolvimento da nova função. Com isso, foi verificado um aumento na produtividade da equipe, uma vez que o esforço da equipe de desenvolvimento se concentrou na solução dos problemas identificados, ao invés da repetição de código.

8. CONCLUSÃO

Este artigo apresentou um metamodelo para implementação de software orientado a objetos e abordou a nível de ilustração a ferramenta para geração automática de código-fonte que está em fase de desenvolvimento, visando apoiar aos profissionais envolvidos na implementação de sistemas orientados a objetos, uma vez que muito tempo é desperdiçado durante a codificação dos artefatos que compõem o software.

Embora existam outros projetos focados no mesmo objetivo, este artigo pode ser de grande utilidade em discussões na busca de novas soluções de implementação de softwares cada vez mais complexos, onde o tempo de desenvolvimento está cada vez mais escasso.

Outro ponto a ser considerado como objetivo deste artigo, é levar a estrutura para geração automática de código-fonte aqui apresentada a outras culturas de desenvolvimento, observando como essa estrutura se adapta nos mais variados tipos de ambientes de desenvolvimento de software.

9. REFERÊNCIAS

GUIZZARD, G. (2000), **"Desenvolvimento Para e Com Reuso: Um Estudo de Caso no Domínio de Vídeo Sob Demanda",** Dissertação de M.Sc, UFES, ES, Brasil.

HOFMANN, H.F. e Lehner, F. (2001), **Requirements engineering as a success factor I**n software projects, IEEE Software, pp. 58-66 vol. 18, n. 4.

LARMAN, C. (2002), ***Utilizando UML e Padrões – Uma Introdução à análise e ao Projeto Orientados a Objetos***, Bookman.

MARTINS, S.R. (2004), ***Geração Automática de Textos em Plataformas de Governo Eletrônico: Um Estudo de Caso na Plataforma Lattes*****,** Dissertação de M.Sc, UFSC, SC, Brasil.

OLIVEIRA, R. F. (2006), ***Formalização e Verificação de Consistência na Representação de Variabilidades*****,** Dissertação de M.Sc, Programa de Engenharia de Sistemas e Computação, UFRJ/COPPE, RJ, Brasil.

PRESSMAN, R.S. (2002), ***Engenharia de Software,*** Trad., McGraw Hill, 5° edição.

VERONESE, G.; JEZINI Netto, F.; CORREA, A. L., WERNER, C. M. L. (2002), ***Uma Ferramenta de Auxílio à Recuperação de Modelos UML de Projeto a partir de Código Java***. REIC. Revista Eletrônica de Iniciação Científica, v. 2, n. 4.

WERNER, C.M.L., BRAGA, R.M.M. (2005), ***A Engenharia de Domínio e o Desenvolvimento Baseado em Componentes***. In: GIMENES, I.M.S., HUZITA, E.H.M. (eds), *Desenvolvimento Baseado em Componentes: Conceitos e Técnicas*, Rio de Janeiro, Ciência Moderna.

SUN MICROSYSTEMS, Inc. (2008), "**The Java Tutorial"**, http://java.sun.com/docs/books/tutorial/java/concepts/interface.html