META-PROGRAMAÇÃO EM SISTEMAS CORPORATIVOS UTILIZANDO A API REFLECTION DA PLATAFORMA JAVA

Bruno ARAÚJO (1); Lierbet MEDEIROS (2), Diego SOUZA (3), Fellipe ALEIXO (4)
(1) CEFET-RN, Av. Sen. Salgado Filho, 1559, Tirol, Natal-RN – CEP: 59015-000, telefone: (84) 4005 2637, e-mail: brunogomes3000@gmail.com
(2) CEFET-RN, e-mail: lierbetnietzsche@gmail.com,
(3) CEFET-RN, e-mail: diegouern@gmail.com,
(4) CEFET-RN, e-mail: fellipe@cefetrn.br

RESUMO

Com o avanço e aparecimento de novas tecnologias, a exigência no mercado de software vem crescendo a cada dia, tornando-se assim num ramo altamente competitivo. Devido a este fator, a busca por sistemas robustos, seguros e confiáveis está sendo cada vez maior. Em contrapartida, o nível de complexidade no desenvolvimento desses sistemas é bastante alto, dificultando a manipulação dos mesmos. Outro fator importante a ser observado, é o gerenciamento da informação envolvida, que constantemente sofre modificações, agregando um custo maior e prejudicando a reusabilidade. Uma solução para essas problemáticas é a utilização da Meta-programação, que sugere um modelo de desenvolvimento de geração e leitura de código em tempo de execução, possibilitando desenvolver um sistema dinâmico que se adapta automaticamente às modificações que vierem a ser feitas. A partir disto, o presente artigo apresenta uma forma de realizar esse tipo de programação utilizando a API de desenvolvimento *Reflection*, que faz parte da plataforma Java. A proposta da utilização desta API baseia-se num estudo de caso, onde serão apresentadas todas as implicações da escolha desta tecnologia no desenvolvimento do sistema Extrator do projeto SIEP Gerencial, iniciativa da Secretaria de Educação Profissional e Tecnológica (SETEC/MEC) e conta com a participação de nove CEFETs.

Palavras-chave: Reflection, Java, Sistemas Corporativos, Meta-Programação.

1. INTRODUÇÃO

Em meio a um mercado que se torna mais exigente, a busca por sistemas que sejam eficientes, confiáveis e que possam agregar vantagens competitivas às empresas é cada vez maior. Devido à concorrência e ao próprio crescimento interno, a necessidade por mudanças nas empresas é constante, resultando em alterações nos seus diversos ramos. Isto influi diretamente nos sistemas utilizados por ela, sendo necessários ajustes para se adaptarem.

Estas constantes modificações contribuem para o desenvolvimento de sistemas cada vez mais complexos e de difícil manutenção, o que resulta em altos valores gastos. Uma solução para este problema é a utilização da Meta-Programação no desenvolvimento.

A Meta-Programação é uma técnica que adiciona flexibilidade ao sistema, permitindo que ele se adapte a determinadas modificações que precisem ser feitas. Uma das maneiras de se utilizar a Meta-Programação é utilizando a linguagem de programação Java, bastante utilizada no desenvolvimento de Sistemas Corporativos. Ela disponibiliza um conjunto de interfaces (*Application Programming Interface* - API) chamado de *Reflection*, que proporciona a utilização desta técnica (MICROSYSTEMS, 2008).

Um estudo de caso do projeto SIEP é apresentado neste artigo, mostrando como foi utilizada a API de desenvolvimento *Reflection* para adicionar a meta-programação ao projeto. O processo de como realizar esta adaptação é descrito, assim como os resultados obtidos.

2. META-PROGRAMAÇÃO

A Meta-Programação permite que programas sejam escritos ou modificados em tempo de compilação (TRATT, 2005). É um conceito bastante importante nos dias atuais, já que existe uma grande exigência no mercado por sistemas que sejam flexíveis e se adaptem às constantes mudanças as quais estão sujeitos.

Segundo Pace e Valente (2006), as vantagens de se utilizar a Meta-Programação no desenvolvimento de sistemas são:

- Maior escalabilidade e extensibilidade do sistema;
- Redução do volume de código necessário;
- Redução do esforço de manutenção;
- Possibilidades de otimização;
- Possibilidade de pré-executar tarefas.

A linguagem que um programa é escrito através da meta-programação é chamada de Metalinguagem, e a habilidade de uma linguagem de programação ser sua própria metalinguagem é chamada de Reflexão, ou em inglês, como é mais conhecida, *Reflection* (BARTLETT, 2008). Existem diversas linguagens que permitem a Meta-Programação, entre elas Java, através da API de desenvolvimento *Reflection* (TRATT, 2005).

3. REFLECTION

Reflection é o nome que se dá a habilidade de acessar informações internas de Classes, localizadas dentro de uma determinada Máquina Virtual, em tempo de compilação (SOSNOSKI, 2008). Sendo assim, um programa é capaz de invocar métodos de objetos sem conhecer precisamente de qual classe ele pertence, permitindo a melhor escolha de execução de tarefas para uma determinada ação. (SOBEL, 1996)

A aplicabilidade do *Reflection* se dá em diversos cenários, dentre os quais sistemas que necessitem de flexibilidade, dinamicidade e que sofram alterações constantes. De acordo com Forman et al. (2005), *Reflection* se torna uma solução simples e elegante nestes cenários, evitando que desenvolvedores constantemente reimplementes interfaces, modifiquem métodos e fiquem sempre gerando os executáveis do projeto após as devidas modificações. Isto ajuda a evitar imprevistos e atrasos no cronograma de desenvolvimento do sistema.

Em Java, a programação utilizando *Reflection* trabalha com arquivos conhecidos como *metadata*, que são dados que descrevem outros dados. Sendo assim, esta API se torna uma ferramenta poderosa,

disponibilizando diversos métodos que dão acesso a uma variedade de informações relacionadas às Classes, métodos e atributos em tempo de execução de uma aplicação (SOSNOSKI, 2008).

4. META-PROGRAMAÇÃO NO PROJETO SIEP

4.1. Projeto SIEP Gerencial

O projeto SIEP (Sistema de Informação da Educação Tecnológica e Profissional) Gerencial é uma iniciativa da Secretaria de Educação Profissional e Tecnológica (SETEC/MEC) e consiste no desenvolvimento de sistemas responsáveis por centralizar as informações relacionadas às várias unidades acadêmicas que oferecem educação profissional no país. O projeto é dividido em cinco módulos independentes e complementares: Extrator, Atualizador, Seletor, Relator e Sincronizador (AZEVEDO et al., 2007).

O primeiro módulo, o Extrator, é responsável pela extração dos dados das várias bases utilizadas pelas unidades acadêmicas de Educação Profissional e Tecnológica (EPT), e disponibilizá-los para uma futura coleta e armazenamento no banco de dados de destino, localizada no MEC.

Os passos referentes à extração dos dados são (ver Figura 1):

- Reconhecimento do Banco de Dados de Origem;
- Extração dos dados;
- Padronização dos dados extraídos;
- Armazenamento dos dados em um banco intermediário para disponibilizá-los para a coleta;

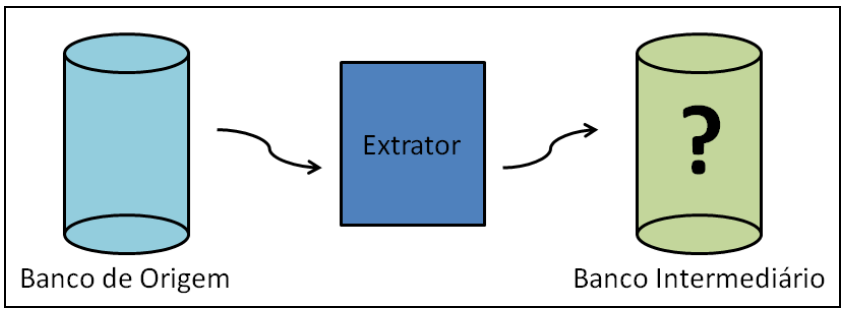

Figura 1 – Representação do processo da extração dos dados

4.2. Problemática

A extração é baseada em um modelo de dados, que consiste numa representação do banco de destino que irá receber os dados extraídos. Como o projeto é desenvolvido em Java, é utilizada a *Java Persistence API* (JPA) para realizar a extração dos dados. Seguindo o modelo de programação desta API, é necessário criar uma classe para cada tabela do banco, chamadas de entidades, e uma classe conhecida como *Data Access Object (*DAO) para a manipulação das conexões com o banco e também dos métodos responsáveis pela extração e armazenamento de cada Entidade. Para que se realize a extração, é necessário fazer um mapeamento de todos os campos do banco no código, para que o sistema possa realizar a padronização das informações e consiga posteriormente armazená-los no banco de dados intermediário.

O problema deste modelo de desenvolvimento é que, ao sofrer alterações no banco, as classes das entidades, a classe DAO e as da padronização das informações devem ser modificadas para se adequar as modificações. Isto custa tempo e dificulta a reusabilidade futura do sistema, ocorrendo cada vez mais atrasos no cronograma. Também é gerado muito código devido à quantidade de entidades envolvidas, o que desorganiza o projeto e atrapalha as devidas modificações.

4.3. Utilização da Meta-Programação

Devido a uma constante modificação no modelo de dados do sub-sistema Extrator, foi necessário criar um novo modelo de desenvolvimento que evitasse a paralisação das tarefas para a correção das classes e métodos do sistema. A solução para este problema foi a utilização da Meta-Programação, com o objetivo de tornar o sistema independente do modelo de dados e capaz de se adequar a qualquer alteração que venha a ser feita.

Para que esta independência se torne possível, primeiramente é necessário separar o projeto do modelo dos dados. Esta separação é realizada durante a geração do projeto. No diretório do sistema, irão ficar dois arquivos, um referente ao executável do projeto de nome *extrator.jar*, que irá conter toda a implementação responsável pela extração e a Meta-Programação, e outro referente ao modelo de dados do banco,chamado de *modelo.jar*. No mesmo diretório localiza-se uma pasta com o nome *lib* que serve para armazenar toda a biblioteca necessária para a execução do projeto (ver Figura 2).

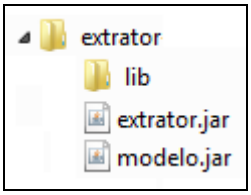

Figura 2 – Sistema de diretórios e arquivos do projeto

O objetivo desta separação é que, a cada modificação solicitada no banco de dados, o único arquivo que precisará ser alterado é o do modelo dos dados, sendo somente ele atualizado no diretório do projeto.

Após a separação, é necessário fazer com que o sistema Extrator reconheça este modelo localizado no mesmo diretório. Para isto, é necessário que haja uma leitura em tempo de execução das classes e atributos que representam as entidades do banco. Esta habilidade é possível através da API *Reflection*, que disponibiliza todos os métodos necessários para que se possa fazer esta operação de identificação das informações referentes ao modelo.

O próximo passo é realizar a extração das informações do banco de dados de origem e padronizá-las seguindo o modelo do banco de dados final. O sistema então irá executar uma consulta no banco e receber um *ResultSet*, que consiste numa interface utilizada pra guardar os dados resultado de uma consulta no banco (MICROSYSTEMS, 2008). Para que o sistema possa identificar e manipular as informações deste *ResultSet*, é necessário que ele saiba todas as entidades que ele contém. Sendo assim, o arquivo do modelo adicionado ao projeto deve ser uma cópia do banco, devendo ter as mesmas nomeações dos atributos, que são a representação dos campos do banco. Através da leitura dessas informações, o sistema irá identificar o modelo correspondente, realizar uma leitura sobre os nomes das entidades e atributos dos mesmos, e irá gerar as consultas em tempo de execução de retornos dos dados localizados no *ResultSet*. Estas informações são armazenadas em um objeto, que posteriormente serão armazenadas no banco de dados através do JPA.

O processo utilizado no Extrator é possível devido aos pacotes da API, o *java.lang.Class* e *java.lang.reflect*. Cada pacote possui uma variedade de métodos com funcionalidades que atendem as diversas exigências relacionadas à leitura de códigos em tempo de execução (MICROSYSTEMS, 2008).

Tabela 1 – Descrição dos métodos da API Reflection utilizados no desenvolvimento (MICROSYSTEMS, 2008).

Pacote	Método	Descrição
java.lang.Class	*getFields()*	Retorna um array contendo todos os atributos da classe ou interface requisitada
	getMethods()	Retorna um array contendo todos os métodos públicos da classe ou interface requisitada
	getName()	Retorna o nome da entidade
	getSuperclass()	Retorna a Classe que representa a superclasse da entidade
	newInstance()	Cria uma nova instancia de uma determinada classe
java.lang.reflect	*invoke(Object obj, Object... args)*	Invoca o método representado pelo objeto passado
	getName()	Retorna o nome do método representado pelo objeto passado

Para possibilitar a manipulação dos dados extraídos, é necessário criar um objeto da Entidade correspondente, para que ele possa armazenar os valores recebidos do *ResultSe*t. Isto é feito através do método *newInstance()*, que em tempo de execução, permite o sistema criar um objeto representando qualquer entidade localizada no modelo.

O método *getName()* é bastante utilizado durante a execução do sistema para recuperar os nomes das entidades que irão ser utilizadas no processo. O método *getFields()* é usado para recuperar os nomes dos atributos que correspondem aos campos das tabelas do banco de dados. O *getMethods()* irá listar os métodos localizados das entidades, possibilitando assim adicionar os valores extraídos nos atributos correspondentes. Por fim, é executado o método *invoke(Object obj, Object... args)*, adicionando o valor recebido do *ResultSet* ao atributo do objeto da Entidade relacionada.

4.4. Resultados

O sistema se tornou capaz de realizar extrações seguindo qualquer modelo de dados que seja adicionado em tempo de execução no mesmo diretório do projeto. Com isto, o desenvolvimento das Classes responsáveis pela Extração foi realizado em menos tempo, e o projeto teve um ganho de produtividade, já que a cada modificação no banco de destino, as únicas Classes que devem ser modificadas são as do modelo.

Com a utilização desta técnica, o número de códigos reduziu bruscamente, pois ao invés de criar uma classe para cada entidade contendo a implementação da padronização das informações envolvidas, agora só é necessário a criação de apenas uma, que irá realizar este processamento de acordo com o modelo de dados cadastrado.

5. CONCLUSÃO

A Meta-Programação é uma técnica que se mostra cada vez mais eficiente para o desenvolvimento de sistemas de grande porte que necessitam mudar constantemente. Ela provê um meio versátil de tornar sistemas dinâmicos sem atrapalhar na reusabilidade e robustes dos mesmos.

A API Reflection da linguagem Java demonstrou ter diversas aplicabilidades, mostrando ser uma ferramenta bastante poderosa no ramo. A sua utilização no sistema responsável pela extração dos dados do projeto SIEP foi de grande importância, já que resultou na diminuição do tempo de desenvolvimento, numa melhor reusabilidade do sistema e também na diminuição do código. Tudo isto aliado ao dinamismo adicionado durante a operação de Extração dos dados, diminuindo também as manutenções que antes eram necessárias ao ocorrerem modificações no modelo de dados.

REFERÊNCIAS

AZEVEDO, G. S.; ALEIXO, F. A.; BORGES, R. P.; CÂMARA, W. S.; ARAÙJO, B. G. MODELO DE DESENVOLVIMENTO DE APLICAÇÕES CORPORATIVAS COM JAVA ENTERPRISE EDITION 5. Em: "II Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica", João Pessoa, PB, 2007.

BARTLETT, J. The art of metaprogramming, Part 1: Introduction to metaprogramming. Disponível em: <http://www-128.ibm.com/developerworks/linux/library/l-metaprog1.html?ca=dgr-lnxw07Meta-PRG1> Acesso em: 10 ago 2008.

FORMAN, R. I.; FORMAN, N. Java Reflection In Action. Greenwich, Estados Unidos da América: Manning Publications CO., 2005.

MICROSYSTEMS, S. Java™ 2 Platform Standard Edition 5.0 API Specification. Disponível em: <http://java.sun.com/j2se/1.5.0/docs/api/> Acesso em: 13 ago 2008.

MICROSYSTEMS, S. Trail: The Reflection API. Disponível em: <http://java.sun.com/docs/books/tutorial/reflect/index.html> Acesso em: 11 ago 2008.

PACE, S. V.; VALENTE, M. T. O. Catálogo de Técnicas de Meta-Programação por Templates em C++. Em: Revista Eletrônica de Iniciação Científica (REIC), Sociedade Brasileira de Computação, Número IV, dezembro 2006.

SOBEL, J. M.; FRIEDMAN, D. P. An Introduction to Reflection-Oriented Programming. Em: "Proceedings of Reflection'96", San Francisco, CA, USA, 1996

SOSNOSKI, D. Java programming dynamics, Part 2: Introducing reflection. Disponível em: <http://www.ibm.com/developerworks/java/library/j-dyn0603/> Acesso em: 10 ago 2008.

TRATT, L. Compile-time meta-programming in a dynamically typed OO language. Em: "Proceedings of the 2005 symposium on Dynamic languages table of contents", San Diego, CA, USA, 2005.