

Metastorage

Acelerando o desenvolvimento de aplicações com
banco de dados em PHP

Geração automática de código com a ferramenta
Metastorage

<http://www.meta-language.net/>

Manuel Lemos

mlemos@acm.org

Novembro de 2005

Projetos de software

RUP - Desenvolvimento iterativo em 3 fases:

1. Análise

Requisitos, riscos, especificação

2. Planejamento

Sistemas, sub-sistemas, casos de uso, diagramas UML

3. Execução

Implementação, teste, correção, documentação →

Implementação em 3 camadas

Padrão MVC: **M**odel – **V**iew – **C**ontroller

- **Modelo**

Acesso à informação manipulada

- **Controle**

Interação com o usuário e sistemas externos

- **Vista**

Apresentação da informação e controles da interface com o usuário →

Motivação da criação de Metastorage

O desenvolvimento da camada de acesso a dados:

1. Segue padrões de desenho bem conhecidos: DAO (ActiveRecord), Factory, etc.
2. Pode consumir demasiado tempo de codificação, dependendo da complexidade do projeto
3. Toda a codificação manual está sujeita a erros do programador, cuja correção requer mais tempo
 - Uma ferramenta de geração automática de código pode reduzir drasticamente o tempo e custo de desenvolvimento e aumentar a sua qualidade
 - **Outubro de 2002:** Criação do projeto Metastorage →

Escopo de Metastorage

Ferramenta de geração de código de classes:

- Acesso a objetos de dados
- Fábrica de objetos
- Instalação do esquema
- Formulários Web
- Geração de relatórios
- Diagramas UML →

Acesso a objetos de dados

Classes DAO – Data Access Objects

- Armazenar e recuperar objetos de dados
- Operações básicas CRUD: Create, Read, Update and Delete
- Validar objetos de acordo com regras
- Manipular objetos relacionados ↓

[Exemplo de classe de dados](#) [Exemplo de uso de classe de dados](#)

Fábrica de objetos

Classe para gerenciar o ciclo de vida dos objetos de dados

- Gerencia conexões e transações
- Cria novos objetos de dados
- Evita múltiplas cópias de objetos de dados em memória
- Remove da memória objetos eliminados ↓

Exemplo de classe fábrica de objetos

Instalação do esquema

Classe de instalação do banco de dados

- Instalação inicial a partir duma definição de esquema em XML independente do banco de dados - formato do Metabase
- Atualização do esquema sem afetar dados adicionados depois da primeira instalação ou desde a última atualização do esquema ↓

[Exemplo de classe de esquema](#) [Exemplo de esquema do Metabase](#)

Formulários Web

Classes de interface com o usuário

- Inicialização, validação, processamento e apresentação de formulários
- Formulários para criação de novos objetos de dados
- Outros tipo de formulários ↓

Exemplos de formulários: [Criação de objeto](#) [Tema Gnome Sawfish](#) [Tema 9X](#) [Tema XP](#)

Geração de relatórios

Classes para extração de dados em massa

- Geração de relatórios e execução tarefas de processamento em massa
- Consultas envolvendo múltiplas classes
- Resultados em arrays, em vez de objetos
- Suporta condições de filtragem, ordenação, agrupamento e delimitação de resultados ↓

Exemplo de classe de relatório

Diagramas UML

Diagramas em UML de todas as classes geradas

- Útil para documentação ou para explicar a um cliente ou patrocinador do projeto
- Geração em formato GraphViz da AT&T
- Conversão possível para vários tipos de formatos: GIF, JPEG, PNG, Postscript, etc.
- Fácil integração em documentação do projeto ↓

[Exemplo de diagrama de classes em UML](#)

Definição de componentes

CPML – Component Persistence Markup Language

- **Síntaxe extensível usando XML simples**
- **Classes de dados com variáveis, condições de validação, relacionamentos entre classes, funções indicadas pelo desenvolvedor**
- **Classe de fábrica e instalação do esquema**
- **Classes de formulários ou relatórios ↓**

[Exemplo de definição de componente](#)

Linguagem de busca de objetos

OQL – Object Query Language

- Síntaxe em XML integrada na definição de componentes em CPML
- Capacidade semelhante a SQL, mas com sintaxe orientada a objetos
- Capaz de expressar condições complexas ↓

Exemplo de expressão de consulta de objetos

Compilador

Geração de código e outros tipos de arquivos

- Geração por meta-programação usando MetaL
- Interface Web ou de linha de comando
- Código independente do banco de dados através da biblioteca Metabase
- Código comentado para melhor compreensão
- Qualidade do código idêntica à programação manual, melhorando a cada nova versão ↓

Exemplo de código gerado

MetaL

Linguagem de meta-programação

- **Julho de 1999**: Criação do projeto MetaL
Proteger o projeto Web-ERP da evolução das linguagens
- **Geração de código em várias linguagens-alvo**
PHP, Java, Perl, etc..
- **Código fonte em XML**
- **Compilador com módulos para vários fins**
Controle de fluxo, tipos de dados, objetos, documentação, etc.
- **Meta-meta-programação**
Meta-programação em cascata: Ex. módulo de persistência ↓

Exemplos de código em MetaL: [MetaL](#) [PHP](#) [Java](#) [Template do Metastorage](#)

Curiosidades sobre MetaL e Metastorage

Números de Novembro de 2005

- Licença tipo BSD
- Totalmente escrito em PHP
- Mais de 50 classes de objetos
- 44.000 linhas de código PHP (1.3MB - 6 anos)
- 20.000 linhas (570KB - 3 anos) são do módulo de persistência que é a base do Metastorage
- 1.000 linhas (23KB - 1 ano) de templates de MetaL (XML) para o código que o Metastorage gera →

Metabase

Acesso a bancos de dados relacionais independente do SGBD

- **Dezembro de 1998:** Criação do projeto Metabase
Tornar o projeto Web-ERP independente do SGBD
- **API comum para muitos SGBD**
MySQL, PostgreSQL, Oracle, MS SQL, Access, Interbase, etc..
- **Mapeamento de tipos entre PHP e o SGBD**
integer, text, boolean, float, decimal, date, time, blob
- **Abstrai capacidades para aplicações Web**
Paginação de resultados, auto-increment, transações, etc..
- **Instalação e atualização de esquema a partir de definição em XML independente do SGBD**
- **Qualidade com testes de unidade extensivos** →

Forms Generation and Validation

Geração e validação de formulários Web

- **Janeiro de 1999:** Criação da classe de *forms*
- Separação de camadas
Definição de propriedades e regras, filtragem e validação, geração do HTML do formulário e Javascript de validação
- Apresentação definida em PHP e HTML ou por sistema de templates (plug-in Smarty)
- Conectividade entre controles após eventos
- Tratamento de eventos do cliente no servidor
- Extensível através de classes plug-in ↓

[CAPTCHA](#), [linked select](#), [calendar date](#), [etc.](#)

Futuro

Desenvolvimentos previstos

- ✓ Customização de classes na linguagem-alvo
- ✓ Gerenciamento integrado de transações
- Suporte a variáveis de grande capacidade (BLOB)
- Melhorar a linguagem de busca de objetos
- Relatórios em formato em HTML, XML, Excel e PDF
- Cache de objetos e resultados de relatórios
- Gerar scripts de teste de unidade
- Classes otimizadas para PHP 5 e PEAR::MDB2
- Documentação automática
- **2006**: Repositório MetaCodigo.com →

Oportunidades

Oportunidades de contribuição para outros

- Reportar bugs e sugerir novas capacidades
- ✓ Ferramentas visuais de edição de CPML
- ✓ Conversão entre UML-XMI e CPML
- Ferramentas de migração de bancos de dados instalados usando engenharia reversa
- Desenvolvimento de projetos acadêmicos ou de código aberto baseados em Metastorage, recebendo suporte de um mentor
- Aproveitar oportunidades de emprego criadas pela aumento da procura de profissionais capacitados no uso do Metastorage →

Fim

Obrigado pela atenção

Questões?

Referências

- **Metastorage**

<http://www.meta-language.net/metastorage.html>

- **Lista de discussão de Metastorage para quem fala Português**

<http://br.groups.yahoo.com/group/metastorage-pt/>

- **Metabase**

<http://www.phpclasses.org/metabase>

- **MetaL**

<http://www.meta-language.net/>

- **Forms generation and validation**

<http://www.phpclasses.org/formsgeneration>

- **GraphViz**

<http://www.graphviz.org/>

- **UML XMI**

<http://www.omg.org/technology/documents/formal/xmi.htm>