# Uma Metodologia de Desenvolvimento de Sistemas Web com a Utilização de Software Livre

**Ériko Moreira, Máiquel Melo, Simão Gurguel, Luana Ramos, Marcos Camurça, William Sales, Sandra Rodrigues, Javam Machado**[1]

[1]Universidade Federal do Ceará (UFC)
Campus do Pici. Bloco 901
60455-760 Fortaleza – CE – Brasil

{eriko,maiquel,simao,sandra,camurca,william,luana,javam}@npd.ufc.br

***Abstract.*** *In the last years one of the challenges of the Software Engineering is a way to reduce the cost of system evolution. This paper presents a methodology for Web systems developtment based on MDA. By using this process, it has a bigger coherence between the project and its implementation, facilitating the tasks of maintenance and inclusion of new functionalities. The case study it shows the applied methodology to an academic control system project (SOFIA), using free software.*

***Resumo.*** *Nos últimos anos, um dos desafios da Engenharia de Software é reduzir o alto custo da evolução natural de sistemas. Este artigo apresenta uma metodologia para desenvolvimento de sistemas Web, baseada no paradigma MDA. Com a utilização desse processo, há uma maior coerência entre o projeto e sua implementação, facilitando as tarefas de manutenção e inclusão de novas funcionalidades. O estudo de caso mostra a metodologia aplicada a um projeto de sistema de controle acadêmico (SOFIA), com a utilização de software livre.*

## 1. Introdução

A Engenharia de Software vem ao longo dos anos buscando uma forma de melhorar o processo de desenvolvimento e manutenção de sistemas [16]. Essa necessidade é ocasionada pela ausência de uma forma completa e sistemática de organizar a sua elaboração, condução e evolução. Em particular, o último aspecto representa cerca de 90% do custo total de um software [21].

Um dos principais problemas que implicam no alto custo da evolução (manutenção e inclusão de novas funcionalidades) é a incoerência entre o projeto e sua implementação. Essas disparidades ocorrem, na maioria das vezes, devido a documentação do sistema não existir ou não refletir exatamente o que foi codificado. Estima-se que 40% a 60% do esforço de manutenção seja consumido na compreensão do código e planejamento de mudanças [22].

Em relação a documentação, muitos dos processos de desenvolvimento de software adotam alguma forma de representação do projeto, seja ela textual e/ou gráfica. Este trabalho apresenta uma metodologia de desenvolvimento de sistemas que visa manter uma maior dependência entre o modelo elaborado e o código resultante da implementação, por meio de uma arquitetura dirigida a modelo: *Model-Driven Architecture* (MDA) [9].

Os resultados desse trabalho foram obtidos com a utilização de ferramentas *open source*, tanto na etapa de projeto como na implementação do sistema. Embora o estudo de caso seja aplicado em um ambiente web, a metodologia em questão aplica-se a outros ambientes, sem perda de generalidade.

A seção 2 descreve a arquitetura MDA e as ferramentas que seguem este paradigma. Na seção 3 é descrita a metodologia exposta neste trabalho. As últimas seções apresentam um estudo de caso e as conclusões obtidas com o experimento até o presente momento.

## 2. Arquitetura Dirigida por Modelo

A Arquitetura Dirigida por Modelo (MDA) é uma metodologia de projeto de sistemas proposta pelo Object Management Group (OMG). Seu principal conceito é a separação entre a etapa de projeto e definição das lógicas de negócio e a etapa de definição da arquitetura e sua implementação. Assim, o desenvolvimento se concentra na definição de um modelo abstrato, que serve de base para a implementação do sistema. Esse modelo é geralmente representado utilizando UML [17], mas outros modelos detalhados podem ser adotados, como o MOF [18] por exemplo.

O processo de desenvolvimento utilizando MDA é focado na transformação de modelos. Sua especificação define quatro modelos básicos e um item auxiliar:

- Computacional Independent Model (CIM): modelo inicial e representa os requisitos do domínio do sistema de uma forma textual;
- Platform Independent Model (PIM): modelo estruturado, independente de plataforma, representado geralmente em UML;
- Platform Specific Model (PSM): modelo específico para uma determinada plataforma, criado para cada ambiente de implementação;
- Platform Model (PM): modelo auxiliar para a implementação de cada plataforma;
- Implementation Specific Model (ISM): modelo específico para a criação de cada codificação, a partir de um PM.

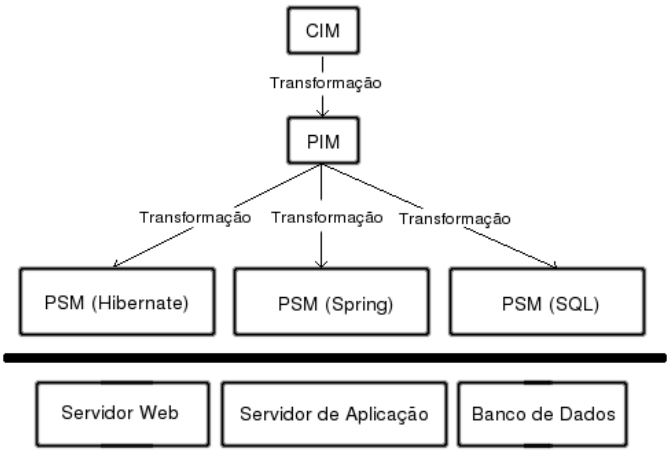


**Figura 1. Transição entre modelos do MDA**

O processo de desenvolvimento utilizando MDA pode ser dividido em cinco etapas. A primeira consiste em levantar os requisitos do domínio do sistema. Em seguida, é

feita a análise dos requisitos para definir o CIM, que serve como base para a criação do PIM na etapa seguinte. A partir dessa modelagem, um ou mais PSM são gerados. Esses, por sua vez, são combinados com o PM e ISM, realizando a geração do código base para o sistema. A figura 1 sintetiza esse processo.

Várias ferramentas adotam esse paradigma para a geração de código fonte com base em um modelo UML. Dentre elas, o AndroMDA [1] é uma ferramenta *open source* feita em *Java* que possui características ideais para a execução da metodologia proposta. Existem outras ferramentas que geram código de forma dinâmica, mas não seguem o paradigma MDA [23].

## 3. Metodologia de Desenvolvimento

Nossa metodologia propõe um processo de desenvolvimento dividido em três etapas: análise, projeto e implementação. Na primeira etapa, são elaborados documentos textuais com levantamento dos requisitos referentes a lógica de negócio. Na etapa seguinte, o sistema é projetado utilizando a linguagem UML. Finalmente, é realizada a implementação, tendo como base o código fonte gerado a partir da modelagem, seguindo o paradigma MDA.

O modelo abstrato é desenvolvido utilizando apenas três tipos de diagramas UML: caso de uso, sequência e de classe. Seguindo o paradigma MDA, a modelagem deve considerar, além das informações levantadas pelos requisitos, um mecanismo adicional que forneça informações detalhadas que auxiliem na geração do código fonte. Para isso, o AndroMDA utiliza basicamente dois recursos da linguagem UML: *tag-values* e estereótipos.

Após a elaboração, o modelo abstrato é salvo em um arquivo no formato XMI (XML Metadata Interchange) [19], que corresponde a uma linguagem XML criada pela OMG para representar uma modelagem. A ferramenta AndroMDA, transforma esse arquivo em um modelo interno. E a partir daí, é gerado o código fonte da aplicação.

O código gerado é baseado em boas práticas de programação, pois utiliza padrões de projeto conhecidos e segue a programação baseada em interfaces. Isso facilita a implementação e a manutenção das regras de negócio específicas da aplicação.

A metodologia segue um processo de desenvolvimento de software interativo e incremental. A inclusão, alteração ou exclusão de um componente inicia-se pela documentação, segue pela modificação do modelo UML e finalmente, reflete-se no código após a execução da ferramenta AndroMDA.

## 4. Estudo de Caso: Projeto SOFIA

O projeto SOFIA (Software de Informação Acadêmica) é um projeto *open source* cujo objetivo é desenvolver um Sistema de Controle Acadêmico (SCA) que atenda às necessidades gerais de uma universidade de médio ou grande porte. Esse projeto está sendo desenvolvido na Universidade Federal do Ceará (UFC) seguindo a metodologia apresentada neste trabalho.

Durante a fase inicial do projeto, avaliamos a viabilidade da utilização da metodologia MDA e verificamos que essa arquitetura seria melhor empregada na camada de

acesso aos dados e na camanda de serviços, onde encontram-se as definições das regras de negócios.

O componente de visão é desenvolvido seguindo o padrão de projeto *Model-View Controller* (MVC). Essa parte do sistema é feita sem a utilização do MDA, portanto, não existe nenhuma geração de código para esta camada. Também utilizamos o conjunto de tecnologias conhecido por AJAX (*Assincronous Javascript and XML*) [20]. Com isso, busca-se uma melhoria da usabilidade, já que a interface provê maior dinamismo do conteúdo e amplia as possibilidades de interação com o usuário.

### 4.1. Uso do Software Livre

Podemos categorizar os softwares livres utilizados em quatro grupos: apoio ao processo de análise e projeto, apoio ao desenvolvimento, implementação e ambiente de implantação.

Como ferramenta de modelagem UML utilizou-se o Poseidon for UML [10]. Escolheu-se como ferramenta para a geração de código, seguindo o paradigma MDA, o AndroMDA [1].

O IDE de desenvolvimento adotado foi o Eclipse [5]. Também utilizamos uma ferramenta de automação de compilação, empacotamento e distribuição de *software*: Ant [2].

Para a implementação do sistema, optou-se pela plataforma *Java Enterprise Edition* (JEE), associada a um conjunto de APIs. As principais foram: o Hibernate [6] como camanda de persistência; Spring [13] como um *container* de objetos; e como *framework* Web escolhemos o Struts [14]. Também foram utilizadas o Junit [8], Jcaptcha [7], Sitemesh [12], dentre outras.

O ambiente de produção compreende o sistema operacional Debian Linux 3.1 [4] e o banco de dados utilizado é o PostgreSQL 8.1 [11]. Como servidor de aplicação, tem-se o Tomcat 5.5.15 [3] e o como servidor Web o Apache 2.2 [15].

## 5. Conclusões

A utilização do paradigma MDA aliada ao uso de *software* livre no processo de desenvolvimento de sistemas, mostra-se uma alternativa viável para resolver o desafio da evolução da aplicação a um baixo custo. Além disso, sendo a modelagem um passo essencial neste processo de desenvolvimento, a documentação mantem-se atualizada, refletindo melhor os requisitos de negócio.

| **CATEGORIAS** | **LINHAS DE CÓDIGO (%)** | |
|---|---|---|
| | **Gerado** | **Implementado** |
| Scripts | 90 | 10 |
| Persistência | 100 | 0 |
| Serviços | 40 | 60 |
| Web | 0 | 100 |

**Tabela 1. Estimativa do percentual de linhs de código geradas e implementadas**

Medições empíricas (ver tabela 1) mostram um ganho significativo na produtividade de maneira geral. Um menor tempo de implementação é exigido, uma vez que o AndroMDA gera código a partir do modelo detalhado. Dessa forma, o foco da etapa de implementação se concentra principalmente na lógica do negócio.

Os *softwares* livres escolhidos ofereceram um ambiente satisfatório para a aplicação da metodologia. Embora tenha sido necessária a realização de alguns ajustes no código fonte do AndroMDA, isso não impactou fortemente no projeto devido ao grau de extensibilidade da ferramenta. Quanto as outros *softwares*, a documentação e os fóruns disponíveis forneceram o suporte necessário para a execução do projeto.

A principal desvantagem dessa metodologia é que sendo ela fortemente vinculada ao modelo, se decidir-se por descontinuar a atualização do modelo, acarretaria uma maior complexidade no processo de manutenção, pois haveria a necessidade da codificação manual de classes e métodos que antes gerados automaticamente. Outra limitação da metodologia é a falha na utilização concorrente do modelo, o que gera um gargalo na distribuição do trabalho em equipe. A não aplicação da metodologia MDA à camada Web também implica num esforço maior para o seu desenvolvimento.

## Referências


[1] AndroMDA, Março 2006. Disponivel em http://www.andromda.org.

[2] Ant, Fevereiro 2006. Disponivel em http://ant.apache.org.

[3] Apache Tomcat version 5.5, Fevereiro 2006. Disponivel em http://tomcat.apache.org.

[4] Debian Linux version 3.1, Fevereiro 2006. Disponivel em http://www.debian.org.

[5] Eclipse, Fevereiro 2006. Disponivel em http://www.eclipse.org.

[6] Hibernate, Fevereiro 2006. Disponivel em http://www.hibernate.org.

[7] JCaptcha, Março 2006. Disponivel em http://jcaptcha.sourceforge.net/.

[8] JUnit, Março 2006. Disponivel em http://www.junit.org/.

[9] Model-Driven Architecture, Março 2006. Disponível em http://www.omg.org/mda.

[10] Poseidon for UML, Março 2006. Disponivel em http://gentleware.com.

[11] PostgreSQL version 8.1, Fevereiro 2006. Disponivel em http://www.postgresql.org.

[12] Sitemesh, Março 2006. Disponivel em http://www.opensymphony.com/sitemesh/.

[13] Spring, Fevereiro 2006. Disponivel em http://www.springframework.org.

[14] Struts, Fevereiro 2006. Disponivel em http://struts.apache.org.

[15] The Apache HTTP Server version 2.2, Fevereiro 2006. Disponivel em http://httpd.apache.org.

[16] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. A study of the documentation essential to software maintenance. In *SIGDOC '05: Proceedings of the 23rd annual international conference on Design of communication*, pages 68–75, New York, NY, USA, 2005. ACM Press.

[17] Martin Fowler and Kendall Scott. *UML Distilled: Applying the Standard Object Modeling Language*. Addison-Wesley, 1997.

[18] OMG. Meta-Object Facility, Fevereiro 2006. Disponivel em http://www.omg.org/technology/documents/formal/mof.htm.

[19] OMG. XML Metadata Interchange Specification, version 2.1, Fevereiro 2006. Disponivel em http://www.omg.org/technology/documents/formal/xmi.htm.

[20] Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, 2005.

[21] TM Pigoski. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. John Wiley Sons, 2000.

[22] Ian Sommerville. *Software Engineering*. Addison Wesley, 6th edition, 2000.

[23] Ériko J. R. Moreira, Maiquel S. Melo, Simão M. Gurguel, and Javam C. Machado. Geração de código utilizando o paradigma orientado a modelo. In *6º Fórum Internacional de Software Livre*, pages 96–100, 2005.