ARQUITETURA ORIENTADA A MODELOS (MDA) E SUA APLICAÇÃO NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

ARTIGO ORIGINAL

SANTOS, Gilmar Melo dos[1]

SANTOS, Gilmar Melo dos. **Arquitetura orientada a modelos (MDA) e sua aplicação no processo de desenvolvimento de *software*.** Revista Científica Multidisciplinar Núcleo do Conhecimento. Ano 06, Ed. 07, Vol. 12, pp. 121-131. Julho de 2021. ISSN: 2448-0959, Link de acesso: https://www.nucleodoconhecimento.com.br/ciencia-da-computacao/aplicacao-no-processo, DOI: 10.32749/nucleodoconhecimento.com.br/ciencia-da-computacao/aplicacao-no-processo

RESUMO

Existem diversas tecnologias que são usadas para o desenvolvimento de *software*. No entanto, é preciso tornar o processo mais rápido possibilitando ganho de produtividade. O artigo busca responder a seguinte questão: Que ferramenta pode ser utilizada para melhorar o processo de desenvolvimento de forma a deixar a arquitetura do projeto mais flexível ao ponto de ser adaptável para qualquer linguagem de programação? Existe uma abordagem estabelecida pelo *Object Management Group* (OMG) que torna o processo de desenvolvimento mais rápido possibilitando o ganho de produtividade chamado *Model Driven Architecture* (MDA), cujo objetivo é a criação de um conjunto de diretrizes que são aplicados em modelos, ou seja, a partir de um modelo abstrato são gerados modelos mais

[1] Graduação em Sistemas de Informação. MBA em Engenharia de Software. Pós-graduação em Desenvolvimento de Sistemas com Java. Pós-graduação em Desenvolvimento em aplicações web. Pós-graduação em Banco de Dados.

específicos. Contudo, essa arquitetura é pouco utilizada, e o propósito desse estudo é mostrar a funcionamento dessa ferramenta e os benefícios que ela proporciona. Dessa forma, com a disseminação desse conhecimento o processo de desenvolvimento poderá ser realizado de forma mais flexível resultando em uma arquitetura com baixo acoplamento.

Palavras-chave: *Model Driven Architecture*, *Unified Modeling Language*, AndroMDA.

1. INTRODUÇÃO

A inclusão de novas funcionalidades em um *software* acaba resultando em alterações nos requisitos fazendo com que boa parte do trabalho seja refeita gastando mais tempo na geração e atualização da documentação do sistema. Além disso, quando é preciso modificar o sistema, seja para incluir uma nova funcionalidade ou para realizar uma simples correção, dependendo da situação, o programador realiza as alterações diretamente no código deixando toda a documentação desatualizada. Como consequência, devido ao surgimento de novas tecnologias, os sistemas acabam sendo refeitos aproveitando pouco da documentação existente, no qual se torna necessário realizar um novo processo de licitação e especificação de requisitos novamente.

Em função disso, foi criado o *Model Driven Architecture* (MDA) com o objetivo de diminuir o acoplamento entre a arquitetura do *software* e sua implementação deixando o modelo mais flexível para receber qualquer tipo de modificação, além de forçar atualização da documentação do sistema. (BROERING, s.d)

1.1 PROBLEMAS NO DESENVOLVIMENTO DE SOFTWARE TRADICIONAL

Produtividade*:* processos tradicionais de desenvolvimento contemplam diversas fases e exigem uma vasta coleção de artefatos antes da implementação do produto.

O problema de produtividade está no fato de que as equipes de desenvolvimento se concentram na tarefa de implementação visando entregar uma versão do produto o mais rápido possível ao cliente, não dando a importância necessária aos artefatos que antecedem a tarefa de codificação. O reuso, nesse cenário, está em nível de linhas de código o que não traz grandes ganhos de produtividade, principalmente, quando se trata de sistemas em linguagens distintas (KLEPPE, 2003).

Portabilidade: é necessário que os sistemas sejam construídos de forma mais independente de tecnologia para favorecer a portabilidade de uma plataforma para outra sem grandes problemas de adaptação.

Interoperabilidade: sistemas de *software* raramente vivem isolados e a maioria das vezes precisam se comunicar com outras aplicações. Devido à diversidade de tecnologias existentes torna-se necessário que tais tecnologias sejam interoperáveis. (KLEPPE, 2003).

Manutenção e documentação: com as mudanças do sistema o código fica cada vez mais diferente da documentação, pois, não há compromisso das equipes de desenvolvimento com a atualização da documentação. Os artefatos são produzidos nas fases iniciais e são deixados de lado com a evolução do *software*. (KLEPPE, 2003).

2. *FRAMEWORK* MDA

Em função da baixa produtividade no processo de desenvolvimento de *software* tradicional foi criado o *Model Driven Architecture* (MDA) com o objetivo de flexibilizar e agilizar a forma como as aplicações são construídas.

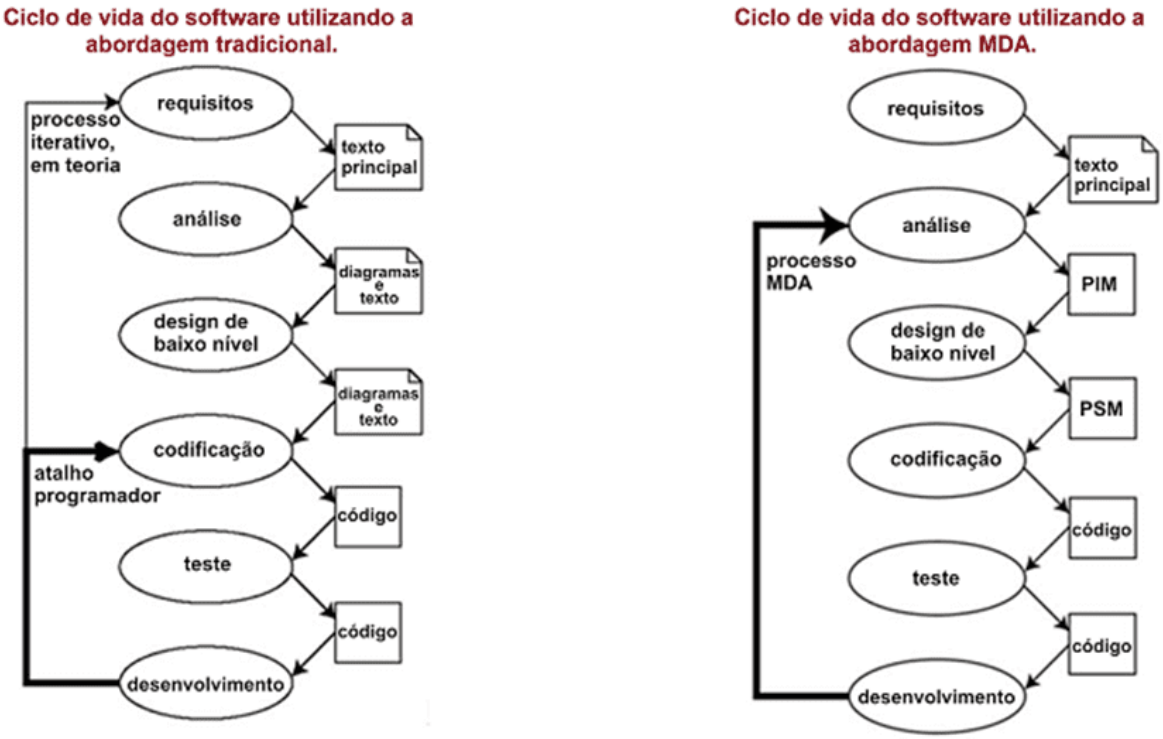
O MDA é um framework para desenvolvimento de *software*, definido pela *Object Management Group* (OMG), o qual é conduzido pela atividade de modelagem. Além disso, ele define o uso de modelos independentes dos detalhes de implementação,

tornando o sistema mais flexível. Grande parte do processo é automatizado, garantindo o aumento da produtividade e qualidade do sistema. Ele promove o uso de uma abordagem que separa a especificação das operações do sistema e os detalhes das funcionalidades em uma plataforma específica (MELLOR, 2004).

O ciclo de desenvolvimento do MDA não parece ser muito diferente do ciclo de vida tradicional, em que as mesmas fases são identificadas nos dois ciclos. Uma das grandes diferenças reside na natureza dos artefatos que são criados durante o processo de desenvolvimento. Os artefatos são modelos formais que podem ser entendidos por computadores.

A Figura 1 representa o ciclo de vida do desenvolvimento de *software* tradicional e o ciclo de vida do desenvolvimento de *software* MDA. No primeiro, percebe-se a existência de bastante documentação em forma de diagramas e texto, e que após a implantação do *software* as manutenções são realizadas diretamente no código sem haver a atualização dos artefatos das fases anteriores. Conduto, na representação usando o processo MDA, toda e qualquer manutenção forçará a atualização dos artefatos desenvolvidos nas fases anteriores.

Figura 1: Ciclo de vida de desenvolvimento tradicional e MDA



Fonte: (KLEPPE, 2003).

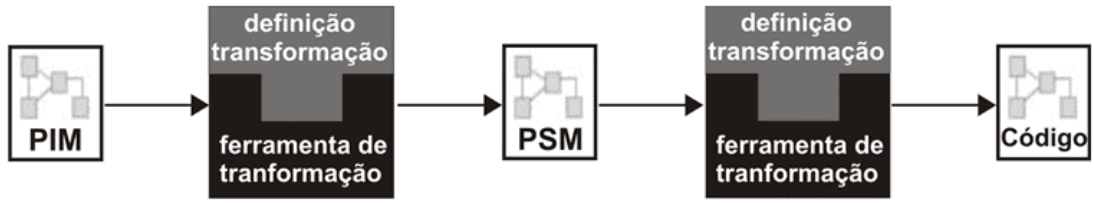
O ciclo de vida MDA compreende três etapas. A primeira etapa consiste em gerar um Modelo Independente de Plataforma (PIM), o qual define um alto nível de abstração e é independente de qualquer tecnologia de implementação. Na segunda etapa, o PIM é transformado em um ou mais Modelos Específicos de Plataforma (PSMs) que é adaptado para especificar o sistema em termos de construções de implementação que estão disponíveis em uma tecnologia específica de codificação. A etapa final do desenvolvimento é a transformação de cada PSM para código.

O PIM, o PSM e o código são mostrados como artefatos de diferentes etapas do ciclo de vida do desenvolvimento. Eles representam diferentes níveis de abstração na especificação do sistema. A capacidade de transformar um PIM em um nível

elevado do PSM aumenta o nível de abstração em que um desenvolvedor pode trabalhar, o que possibilita lidar com sistemas mais complexos e com menos esforço.

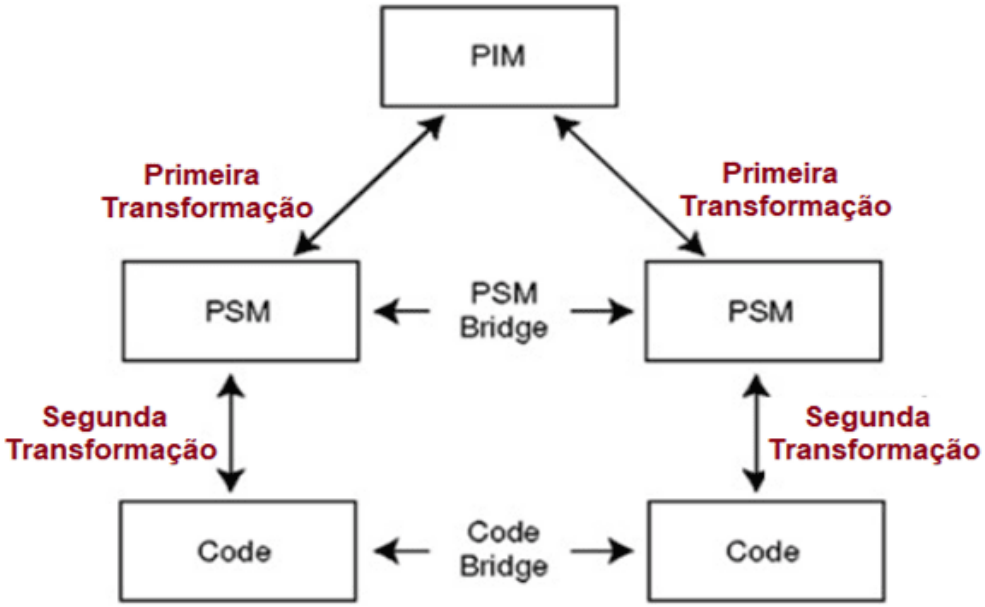
Para realizar as alterações são utilizadas algumas ferramentas que fazem a tradução do PIM para PSM e do PSM para código. Essas transformações são baseadas em mapeamento que consistem em uma coleção de regras que são utilizadas nos modelos (PIM) para gerar diferentes tipos de PSMs. A Figura 2 mostra de maneira resumida como são utilizadas as ferramentas de transformação e a figura 3 representa, de uma forma mais clara, como os artefatos são gerados.

Figura 2: Etapas do desenvolvimento MDA



Fonte: (KLEPPE, 2003).

Figura 3: Geração dos artefatos



Fonte: (BRAGA, 2011).

2.1 BENEFÍCIOS DO MDA

Produtividade: o foco é transformar o modelo conceitual (PIM) em um modelo específico de plataforma (PSM) que será utilizado para a geração do código. Dessa forma, após as regras de transformação serem definidas, apenas o PIM deverá ser atualizado, caso seja necessário realizar alguma alteração no sistema, gerando automaticamente todos os demais modelos garantindo a atualização em todos os artefatos, ganhando produtividade e eficiência. (KLEPPE, 2003).

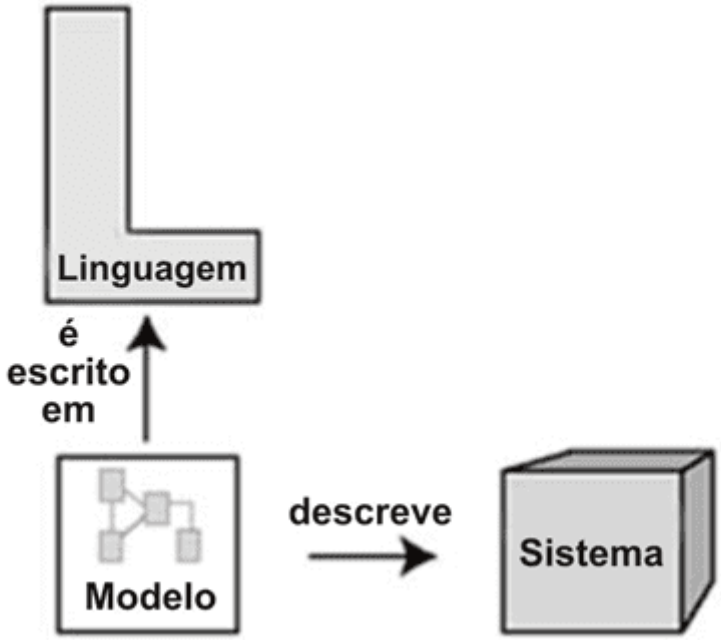
Portabilidade: a portabilidade é alcançada focando no desenvolvimento de PIMs, que são por definição independente de plataforma, o mesmo PIM pode ser transformado automaticamente para vários PSMs específicos para as diferentes plataformas.

Manutenção e Documentação: toda a manutenção é realizada no nível mais alto de abstração forçando o desenvolvedor realizar a alteração diretamente no modelo conceitual (PIM) mantendo todos os artefatos atualizados e consistentes. Dessa forma, é possível garantir que todas as fases (PIM, PSM, código) foram realizadas de forma correta passando mais confiança e credibilidade. (KLEPPE, 2003)

2.2 MODELOS

Modelos consistem em conjuntos de elementos que descrevem uma realidade física, abstrata ou hipotética. Uma chave para a modelagem eficaz no contexto de desenvolvimento de sistemas é o bom uso da abstração e da classificação. A abstração envolve ignorar a informação que não é de interesse em um contexto particular já a classificação envolve agrupar informações importantes com base em propriedades comuns. Especificamente, para o contexto do desenvolvimento de *software*, um modelo é a descrição de um sistema escrito em uma linguagem bem definida com sintaxe e semântica passível de interpretação automática por um computador (KLEPPE, 2003). A Figura 4 representa os relacionamentos entre linguagens, modelos e sistema.

Figura 4: Modelos e linguagens

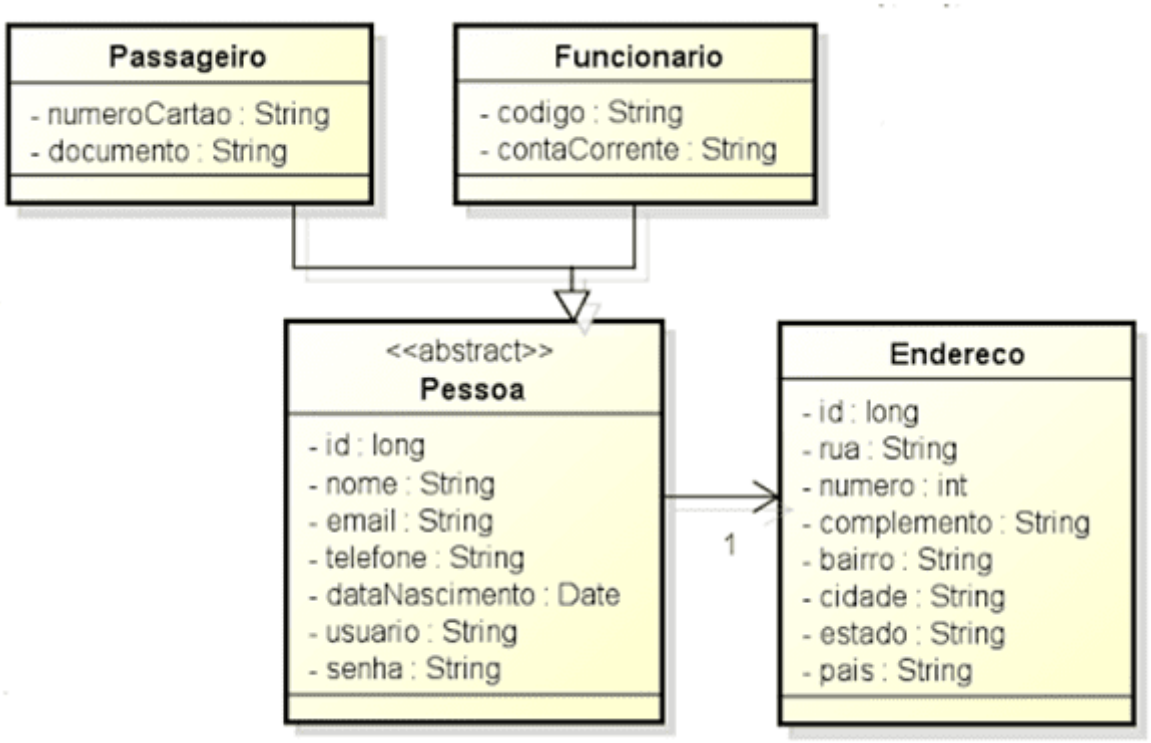


Fonte: (KLEPPE, 2003).

O centro do MDA consiste na criação de modelos em diferentes níveis de abstração que serão ligados para formar uma aplicação. Cada modelo será expresso por meio de uma combinação de texto e vários esquemas complementares e inter-relacionados (MELLOR, 2004).

Na abordagem MDA, o modelo PIM é representado por meio da linguagem de notação *Unified Modeling Language* (UML) que é utilizada para gerar os modelos de classes. A Figura 5 representa um exemplo de um diagrama de classes que faz parte da UML e que pode ser utilizando para gerar os PSMs.

Figura 5: Diagrama de classes



Fonte: autor.

2.3 FUNCIONAMENTO DO MDA

O AndroMDA é um *framework* utilizado no MDA para a gerar códigos a partir de um modelo UML no formato XMI, combinado com cartuchos e bibliotecas que possibilitam criar componentes para qualquer linguagem de programação (Java, PHP, HTML etc.).

Ainda, pode-se mencionar algumas ferramentas disponibilizadas pelo AndroMDA, como o *plugin* do Maven e o Schema2XMI, que faz a leitura do banco de dados e cria um XMI para ser usado em um modelo UML.

Com a codificação das *interfaces* do projeto, o desenvolvedor se preocupará unicamente com o código relacionado ao negócio garantindo agilidade no

desenvolvimento. O AndroMDA tem como relevância o processo de transformação do diagrama para o código que é controlado por meio de cartuchos eliminando as tarefas repetitivas no processo de desenvolvimento.

O AndroMDA atua com a combinação de algumas ferramentas, como, por exemplo, o MagicDraw que é utilizado para modelar os diagramas e gerar o arquivo.xmi que será utilizado para gerar a aplicação. Além da ferramenta CASE, o Mavem é utilizado para gerenciar todo o processo de construção e implantação, o qual automatiza as tarefas repetitivas. Cabe ressaltar também, que o AndroMDA utiliza cartuchos para gerar códigos específicos para determinadas tecnologias.

No AndroMDA, o desenvolvimento de aplicações ocorre a partir de um arquivo, com extensão xmi, gerado preferencialmente pelo MagicDraw, que contém o modelo de mais alto nível (PIM). O Maven, que é um gerenciador de todo o processo, chama as classes do AndroMDA e este, por sua vez, invoca os *templates* definidos no cartucho para gerar as classes da aplicação. O Maven solicita a compilação das classes e empacota os arquivos gerados (arquivo. ear ou .war), tornando a aplicação passível de implantação.

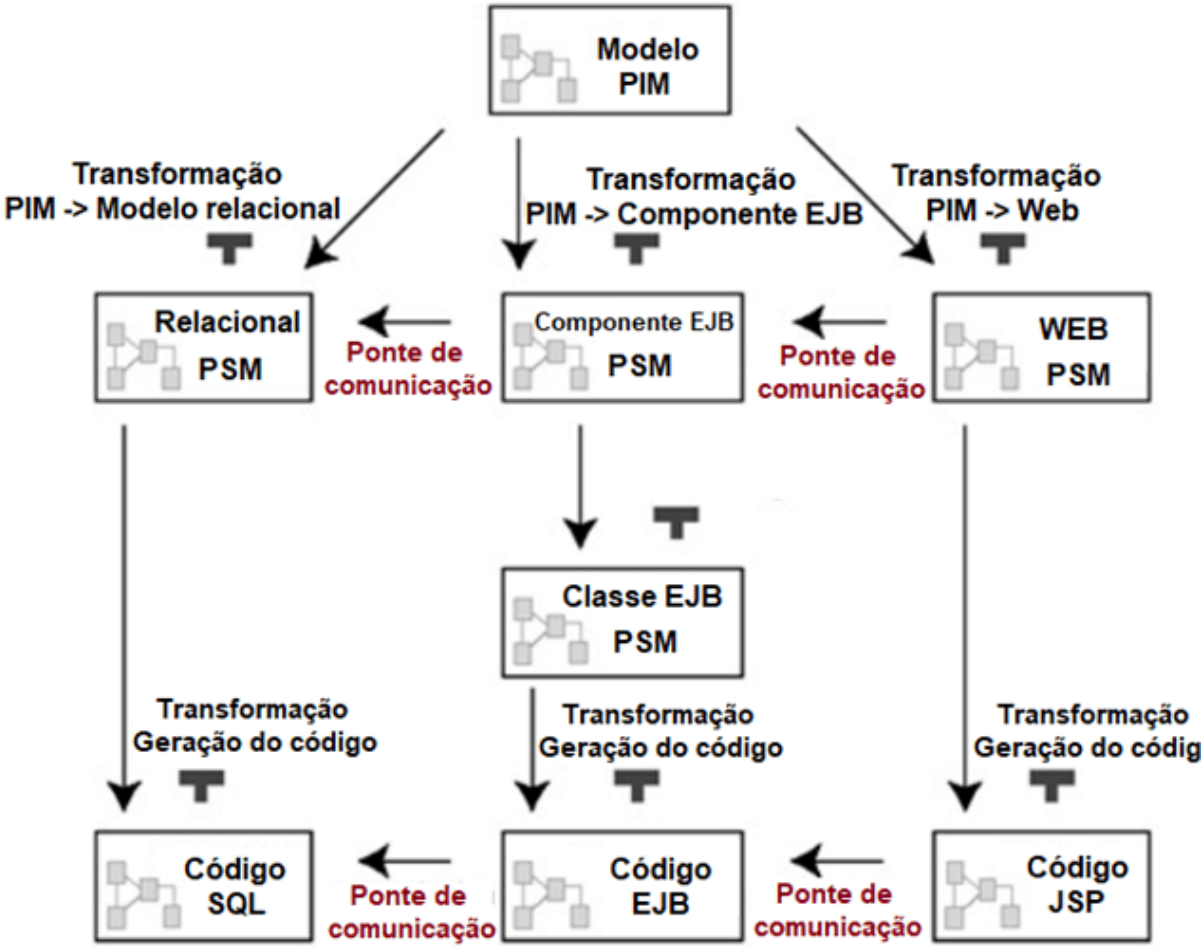
O AndroMDA possui as seguintes vantagens:

- Elimina a necessidade de escrever códigos redundantes;
- Os modelos do projeto refletem o código;
- Facilita a utilização de várias linguagens de programação.

O processo de transformação é controlado usando *plugins* chamados cartuchos (*cartridges*). A geração do código é feita por estereótipos declarados no modelo UML, servindo como guia para o mapeamento realizado pelo cartucho. Estes estereótipos indicam as classes que serão mapeadas, além de aspectos relacionados a outras tecnologias que podem ser usadas no processo de transformação.

Cartuchos consistem em um ou mais arquivos e suportam customizações, o que garante que qualquer código específico pode ser gerado a partir do modelo. A Figura 6 representa o funcionamento do AndroMDA, em que a partir de um modelo independente de plataforma (PIM) pode-se gerar diversos modelos específicos de plataforma (PSM) e consequentemente o código fonte.

Figura 6: Funcionamento do AndroMDA



Fonte: (BRAGA, 2011).

3. CONCLUSÃO

Este trabalho apresentou uma abordagem dirigida por modelos para a geração automática de código, mostrando regras de transformação entre modelos de classes UML, de forma a tornar a arquitetura do projeto mais flexível e adaptável para implementar qualquer linguagem de programação.

No que diz respeito à abordagem proposta, procurou-se demonstrar, através de um exemplo de uso, a ferramenta, os recursos disponíveis, os problemas encontrados e as manipulações necessárias para que as implementações pudessem ser feitas.

REFERÊNCIAS

BROERING, Evandro. **MDA - Model Driven Architeture**. Disponível em: https://www.devmedia.com.br/mda-model-driven-architeture/4547 Acesso em 13/07/2021.

BRAGA, Christiano. **Engenharia de Software**. Disponível em: http://www.ic.uff.br/~cbraga/es/es.pdf Acesso em 13/07/2021.

KLEPPE, Anneke; WARMER, Jos; BAST, Wim. **MDA Explained**: The Model Driven Architecture: Practice and Promise. Addison-Wesley Professional, v. 1, f. 85, 2003. 192 p.

MELLOR, Stephen J; SCOTT, Kendall; UHL, Axel; WEISE, Dirk. **MDA Distilled**: Principles of Model-driven Architecture. Addison-Wesley Professional, v. 1, f. 75, 2004 150 p.

Enviado: Maio, 2021.

Aprovado: Julho, 2021.

RC: 92739
Disponível em: https://www.nucleodoconhecimento.com.br/ciencia-da-computacao/aplicacao-no-processo