Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
Campus São João da Boa Vista

Gustavo Duran Ferreira

Uma proposta de implementação MDD para geração de códigos-fonte de layout para aplicações web

São João da Boa Vista
2018

Gustavo Duran Ferreira

Uma proposta de implementação MDD para geração de códigos-fonte de layout para aplicações web

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de São Paulo, como parte dos requisitos para a obtenção do grau de Tecnólogo em Sistemas para Internet.

Área de Concentração: Engenharia de software

Orientador: Prof. Dr. Breno Lisi Romano

São João da Boa Vista

2018

Ficha Catalográfica elaborada pela Biblioteca Comunitária
"Wolgran Junqueira Ferreira" do Instituto Federal de São Paulo
Câmpus São João da Boa Vista

Ferreira, Gustavo Duran
F383p Uma proposta de implementação MDD para geração de códigos-fonte de layout para aplicações web / Gustavo Duran Ferreira; orientador Breno Lisi Romano. -- São João da Boa Vista, 2018.
79 p.

Trabalho de Conclusão de Curso (Tecnologia em Sistemas para Internet) -- Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, São João da Boa Vista, 2018.

1. Engenharia de software. 2. Software produtividade. 3. Aplicações web. I. Romano, Breno Lisi, orient. II. Título.

CDD 005.1

Gustavo Duran Ferreira

Uma proposta de implementação MDD para geração de códigos-fonte de layout para aplicações web

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de São Paulo, como parte dos requisitos para a obtenção do grau de Tecnólogo em Sistemas para Internet.

Área de Concentração: Engenharia de software

Aprovado em DIA(número) de MÊS(por extenso) de ANO(Número).

Prof. Dr. Breno Lisi Romano
Orientador
Titulação
Instituição

Professor Convidado 1
Titulação
Instituição

Professor Convidado 2
Titulação
Instituição

São João da Boa Vista
2018

AGRADECIMENTOS

Ao meu orientador, pelo apoio e confiança. Aos meus pais, pelo incentivo e amor incondicional. Aos meus amigos, pelos auxílios e companheirismo. E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Ferreira, Gustavo Duran. **Uma proposta de implementação MDD para geração de códigos-fonte de layout para aplicações web**. 2018. Monografia (Graduação em Tecnólogo em Sistemas para Internet) – Instituto Federal de São Paulo, Câmpus São João da Boa Vista. 2018.

RESUMO

Atualmente, existe uma grande demanda gerada na web para entregar soluções de software rapidamente, com custos de produção reduzidos. A implementação do Desenvolvimento Dirigido a Modelo diminui a necessidade de capacitação técnica e aumenta a produtividade da equipe por gerar código de forma automática. O objetivo desta pesquisa é apresentar o conceito de Desenvolvimento Dirigido a Modelo para geração de códigos-fonte de layout para aplicações web. Para isso, foi realizado uma investigação de trabalhos relacionados, desenvolveu-se um metamodelo para representar elementos de layout e proposto algoritmos de transformação para aplicar em um experimento prático, possibilitando uma análise quali-quantitativa dos resultados obtidos para destacar os ganhos de usabilidade e produtividade. O resultado obtido no experimento prático foi como planejado, já que todas as características de *layout* puderam ser implementadas, utilizando a abordagem MDD por completo, buscando otimização em todos os passos realizados. Com esta pesquisa, busca-se destacar os ganhos de produtividade e usabilidade no desenvolvimento de aplicações web.

Palavras-chave: Modelagem. Engenharia de software. Desenvolvimento. Produtividade. CSS.

Ferreira, Gustavo Duran. **An MDD implementation proposal for generating layout source codes for web applications**. 2018. Monograph (Undergraduate in Technologist in Internet Systems) - Federal Institute of São Paulo, Campus São João da Boa Vista. 2018.

ABSTRACT

Nowadays, there is a great demand generated on the web to deliver software solutions quickly, with reduced production costs. The adoption of Model Driven Development reduces technical training needs and increases team productivity by automatically generating code. The purpose of this research is to present the concept of Model Driven Development to produce layout codes for web applications. For this, an investigation of related works was carried out, a metamodel was developed to represent layout elements and proposed transformation algorithms to apply in a practical experiment, enabling a qualitative and quantitative analysis of the results obtained to highlight usability and productivity gains. The result obtained in the practical experiment was as planned, since all characteristics of layout could be implemented, using the MDD approach completely, seeking optimization in all the steps performed. With this research, we seek to highlight productivity gains and usability in the development of web applications.

Keywords: Modeling. Software Engineering. Development. Productivity. CSS.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CASE	*Computer-Aided Software Engineering*
CIM	*Computacional Independent Model*
CSS	*Cascading Style Sheets*
HTML	*HiperText Markup Language*
JS	JavaScript
MDA	*Model Driven Architecture*
MDD	*Model Driven Development*
MDE	*Model Driven Engineering*
MVC	*Model-View-Controller*
OMG	*Object Management Group*
PHP	*Hypertext PreProcessor*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
SGBD	Sistema de Gerenciamento de Banco de Dados
SOA	*Service Oriented Architecure*
SQL	*Structured Query Language*
T1	Transformação 1
T2	Transformação 2
T3	Transformação 3
UML	*Unified Modeling Language*
W3C	*World Wide Web Consortium*
WHATWE	*Web Hypertext Application Technology Working Group*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O projeto da internet foi concebido por Tim Berners-Lee e Robert Cailliau, em 1990, com o propósito de melhorar o compartilhamento de informação entre os cientistas da Organização Europeia para a Pesquisa Nuclear (*Conseil Européen pour la Recherche Nucléaire* - CERN). Foi criado um protocolo de comunicação de como os arquivos deveriam ser escritos e transitados para que as informações fossem visualizadas de forma idêntica, fácil e relacionada por todos (BERNERS-LEE, 2004). Allsopp (2000) destaca a vantagem da web por não possuir as limitações da mídia impressa, sendo flexível, e recomenda fazer páginas legíveis independente de plataforma, dispositivo ou tamanho de tela.

Conforme dados liberados por Amit Singhal, vice presidente sênior da Google Inc em 2012 que possuia 78% das buscas na internet, a empresa contabilizou: 30 trilhões de URL's únicas encontradas; 20 bilhões de sites rastreados por dia; e 100 bilhões de pesquisas processadas por mês (SULLIVAN, 2012).

Existe uma demanda por *software* onde o acesso às aplicações ocorre de qualquer lugar, a qualquer hora e a partir de diferentes dispositivos. Isso fez surgir um desafio para a Engenharia de *Software* que é a adaptação dos conteúdos de uma aplicação para os inúmeros dispositivos que podem acessá-la em diferentes contextos (CIRILO et al., 2011). Esse problema destacasse em aplicações web que sofre influência de diversos fatores, como por exemplo: tipo de dispositivo (*desktop*, *smartphones*, etc.); diferentes tamanhos de tela; tipo de usabilidade; velocidade de conexão; aplicação de navegador utilizada para acesso; e acessibilidade que deve ser implementada para suprir diferentes deficiências que os usuários possam ter que dificultem na utilização do software (SILVA, 2014).

É necessário uma capacitação maior da equipe para desenvolver uma aplicação web, pois os *browsers* (aplicações utilizadas para acessar a web e navegar pela internet) interpretam três linguagens para renderizar uma interface (*Front-End*) que são a base da web atual. O HTML (*HiperText Markup Language*) descreve a estrutura da interface utilizando uma marcação, o CSS (*Cascading Style Sheets*) descreve como a estrutura HTML deve ser exibida e o JS (*JavaScript*) é utilizado para implementar interações com o usuário (W3C, 2018).

Além das linguagens *Front-End*, outras tecnologias são necessárias em aplicações mais completas para realizar uma comunicação entre a interface e armazenamento de dados que, tradicionalmente, é um banco de dados relacional com uma linguagem SQL (*Structured Query Language*) para manipulação. O PHP (*Hypertext PreProcessor*, originalmente *Personal Home Page*) está entre essas linguagens denominadas *Back-end*, sendo uma linguagem de fácil aprendizado (MINETTO, 2007) e estimasse que seja utilizada em mais

de 80% dos servidores web existentes (DALL'OGLIO, 2015).

Para entregar soluções de *software* de forma mais rápida, os desenvolvedores adotaram a estratégia de utilizar *frameworks* que são coleções de códigos-fonte que servem de base para desenvolver algo maior e mais específico, evitando o trabalho de fazer código que já existe de forma abstraída. Ao utilizar um *framework*, a equipe de desenvolvimento presa por aplicar a reutilização de código e, consequentemente, facilitar o desenvolvimento de novos *softwares*. Porém, isso trás o problema do *software* ser dependente de mais uma tecnologia terceira para funcionar, dando margem para erros imprevistos que podem levar muito tempo para serem descobertos, deixando o ambiente de desenvolvimento mais complexo e exigindo uma capacitação técnica maior da equipe de desenvolvimento (MINETTO, 2007).

Com isso, pode-se destacar os altos custos para evolução ou manutenção, inconsistência entre documentação e sistema final, pouca ou quase que total falta de portabilidade e baixa confiabilidade como uns dos principais problemas para projetos de *software*. A proposta de Desenvolvimento Dirigido por Modelos (*Model Driven Development* - MDD) surgiu para tentar corrigir essas deficiências nos projetos, pois ela muda o foco das linguagens de programação para a modelagem do *software* e, com isso, a equipe de desenvolvimento concentra seus esforços na construção de modelos mais completos e precisos, viabilizando a total geração de código de forma semi-automatizada ou automática (ALVES; MACHADO; RAMALHO, 2008).

1.1 MOTIVAÇÃO

A motivação desta pesquisa surge ao tentar suprir a necessidade dos desenvolvedores de entregar soluções de software rapidamente, com a enorme demanda gerada na web pelos usuários, de maneira que seja reduzido os custos de produção do projeto com a implementação de MDD que diminui a necessidade de capacitação técnica da equipe de desenvolvimento, começando pela geração automática de arquivos CSS responsáveis pelos *layouts* flexíveis das interfaces de aplicações web.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O objetivo desta pesquisa é apresentar a implementação do conceito de MDD para geração de códigos-fonte de layout para aplicações web, visando destacar os ganhos de produtividade e usabilidade no desenvolvimento de aplicações web.

1.2.2 Objetivos Específicos

Para cumprir o objetivo desta pesquisa, as seguintes etapas serão realizadas:

- Investigação dos principais conceitos e trabalhos relacionados com MDD;
- Elaboração de um metamodelo para representar visualmente *layouts* de aplicações web;
- Proposta de algoritmos de transformação de modelos em códigos-fonte usando a abordagem MDD;
- Aplicação dos metamodelos e algoritmos de transformação em um experimento prático;
- Análise quali-quantitativa dos resultados obtidos para destacar os ganhos de usabilidade e produtividade;

1.3 ORGANIZAÇÃO DESTE TRABALHO

Neste primeiro capítulo apresentou-se a motivação e objetivos da pesquisa. O restante da dissertação está estruturado da seguinte maneira:

- No Capítulo 2 é apresentado conceitos sobre a implementação de MDD, os fundamentos para se utilizar está abordagem no desenvolvimento de aplicações web e discutidos alguns modelos para a geração de códigos-fonte para aplicações web;
- No Capítulo 3 é apresentada uma visão geral da abordagem proposta, aplicando-a em um experimento prático;
- No Capítulo 4 é analisado os resultados obtidos, destacando os ganhos de usabilidade e produtividade no desenvolvimento de um experimento prático;
- No Capítulo 5 é realizado a conclusão do trabalho apresentando uma perspectiva futura de projeto.

2 REVISÃO DA LITERATURA

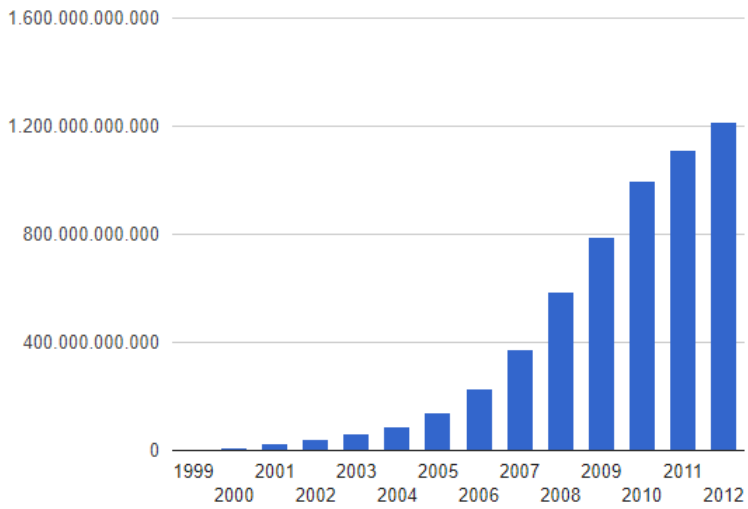
Este capítulo aborda os conceitos e objetivos da web, quais são as dificuldades encontradas ao longo dos anos para o desenvolvimento de aplicações web e as soluções encontradas para resolvê-los. Também apresenta algumas tecnológicas e metodologias utilizadas por desenvolvedores atualmente, para a concepção dessas aplicações.

2.1 APLICAÇÕES WEB

Pode-se observar uma crescente utilização da web ao longo dos anos, analisando o aumento do número de pesquisas computadas pela Google Inc que, em 2006, atendia 10 mil buscas por segundo. Esse valor era atendido por dia quando a empresa foi fundada em 1998 (BATTELLE, 2011).

A Figura 1 apresenta dados liberados em 2012 pelo vice presidente sênior, quando a empresa possuía 78% das buscas na internet mostrando 38 mil pesquisas atendidas, em média, por segundo (SULLIVAN, 2012). Atualmente, estima-se que a empresa atenda 40 mil pesquisas por segundo e exista mais de 1,8 bilhões de domínios para sites (INTERNET. . . , 2018).

Figura 1 – Pesquisas atendidas por ano pela Google Inc



Fonte: (INTERNET. . . , 2018)

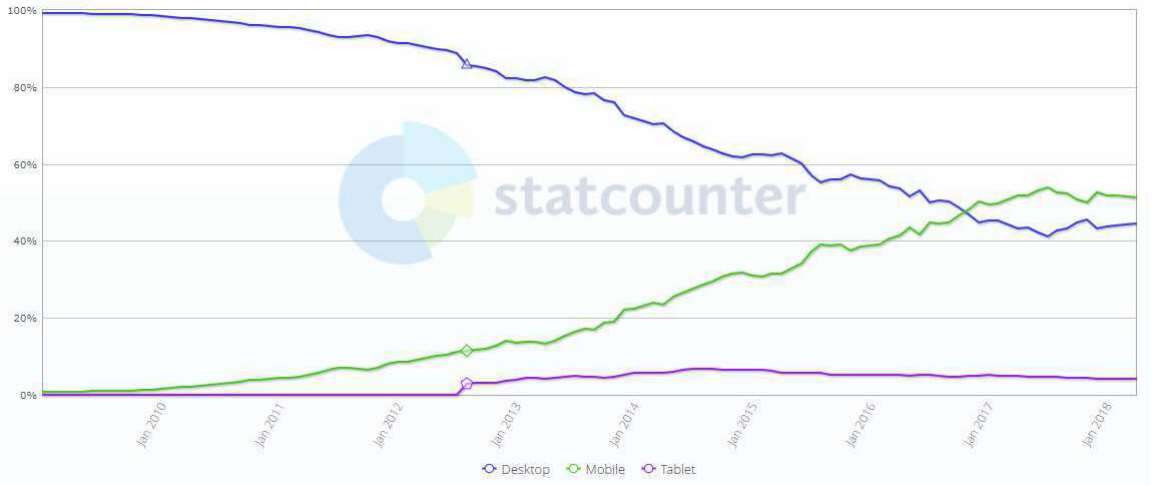
O propósito original nos primórdios da internet era melhorar o compartilhamento de informações entre organização científica. Foi criado a linguagem HTML (*HiperText Markup Language*) para marcar as informações de forma padronizada e a linguagem CSS (*Cascading Style Sheets*) para formatar as informações marcadas. Isso possibilitou uma

comunicação em que todos pudessem enviar e visualizar as informações de forma idêntica (BERNERS-LEE, 2004).

Quando a web parou de ser entendida como a mídia impressa, foi retirado as limitações de tamanho fixo e visualização única, começando a se tornar flexível (ALLSOPP, 2000). Outro problema surgiu com a chegada da era pós-pc, quando fomos apresentados a novas formas de acesso à internet (COELHO, 2016).

Com à acensão do uso de dispositivos móveis trazendo um novo formato de navegação que representam 51.56% dos acessos a web atualmente, como apresenta a figura 2 (STATCOUNTER. . . , 2018), foi necessário criar metodologias para implementar os conceitos de arquitetura responsiva para o desenvolvimento de aplicações adaptativas, como o *Responsive Web Design* (MARCOTTE, 2010).

Figura 2 – Porcentagem de acessos por tipo de dispositivo ao longo dos anos



Fonte: (STATCOUNTER. . . , 2018)

Desde a criação de web servidores, as aplicações são desenvolvidas em uma arquitetura *server-centric* em que o servidor é responsável pelo processamento de dados e atender as requisições dos clientes (qualquer dispositivo que solicite acesso a aplicação), enquanto os clientes somente esperam para renderizar a página quando recebem as respostas. Isso se torna uma vantagem em relação as aplicações nativas por não precisarem de uma instalação para funcionar (FINK et al., 2014).

As aplicações web tentam atender a necessidade de acesso e manipulação das informações de qualquer lugar, a qualquer hora e a partir de diferentes dispositivos (CIRILO et al., 2011), mas o seu desenvolvimento exige preocupações como a usabilidade, velocidade de conexão, acessibilidade e tipos de dispositivo dos usuários que a aplicação deve atender (SILVA, 2014).

As próximas seções têm o objetivo de apresentar as diferentes camadas de desenvolvimento das aplicações web, quais tecnologias são utilizadas para construir cada camada e

quais metodologias atendem as necessidades dos projetos de *software*.

2.2 TECNOLOGIAS PARA DESENVOLVIMENTO

Aplicações web são geralmente parecidas em todas as plataformas e são recomendadas para projetos com a necessidade de várias plataformas, tendo custos e tempo reduzidos a necessidade de várias plataformas. Por ser uma aplicação interpretada pelo navegador do dispositivo, passa a ser visualizada em sistemas operacionais diferentes, sendo assim as aplicações com o menor tempo de projeto (SILVA; PIRES; NETO, 2015).

A compatibilidade dos navegadores entre plataformas tornam as aplicações web mais práticas e dinâmicas, pois possibilita a manutenção e atualização sem distribuição e instalação de software (HALES, 2012). O objetivo desta seção é apresentar as responsabilidades atribuídas a cada camada de desenvolvimento e quais tecnologias são utilizadas para implementá-las.

2.2.1 Front-End

O desenvolvimento *Front-End* foca na produção das interfaces e é fundamental para o sucesso de uma aplicação web, pela importância que a qualidade da implementação das estruturas e interações têm no impacto da experiência do usuário (PAIVA, 2018). É necessário um bom planejamento de projeto para se desenvolver um bom *front-end*, pois geralmente gasta-se mais tempo de desenvolvimento do que o *back-end* (SILVA; FERRARI, 2016). Pode-se exemplificar três linguagens *front-end*, sendo elas: HTML, CSS e JS (EIS, 2015).

O HTML (*HiperText Markup Language*) descreve a estrutura da interface utilizando uma marcação (W3C, 2018). O HTML é o que há de mais importante no desenvolvimento web, estando presente do início ao fim do projeto. Se o HTML for mal feito, o site não será encontrado nos resultados de buscas, a acessibilidade será prejudicada, faltará portabilidade, a manutenção será danificada com o tempo e, consequentemente, haverá problemas gigantes com código legado (EIS, 2015).

O HTML5 surgiu a partir de um consórcio entre a W3C (*World Wide Web Consortium*) e a WHATWE (*Web Hypertext Application Technology Working Group*) trazendo melhorias significativas com novas funcionalidades de semântica, acessibilidade e suporte a multimídia (PILGRIM, 2010).

O CSS (*Cascading Style Sheets*) descreve como a estrutura HTML deve ser exibida (W3C, 2018). Tem o princípio de herança de estilos em cascata para os elementos do HTML, sendo assim, dispensando a necessidade de formatação elemento a elemento (EIS, 2015).

O CSS3 é uma ferramenta para os *designer* web, pois apresenta mudanças para a manipulação de transições, animações e funcionalidades que dão suporte ao *responsive web design* (EIS; FERREIRA, 2012). O CSS não dá significado nenhum as informações como o HTML, sua única responsabilidade é a formatação das informações marcadas, para que sejam bem visualizadas de qualquer dispositivo (EIS, 2015).

O JS (*JavaScript*) é utilizado para implementar interações com o usuário (W3C, 2018). É uma linguagem de programação dinâmica de alto nível adequada ao estilo de programação funcional (OSMANI, 2012).

Segundo Eis (2015), o JS ganhou foco nos últimos anos e foi o responsável pela comunidade de programadores *back-end* aderir um olhar diferente sobre o *front-end*, que até então se resumia a marcar texto com HTML e pintar quadrados com CSS, quando expandiu-se para além do *front-end* com o Node.js que é um interpretador de código JS no servidor, possibilitando uma programação de *back-end* em JS (EIS, 2015).

2.2.2 Back-End

O desenvolvimento *Back-End* foca na implementação no lado do servidor, construindo as regras de negócio presente nas aplicações web (PAIVA, 2018). O *back-end* é responsável pela comunicação entre os dados armazenados no banco de dados e o usuário através das interfaces *front-end*.

Um Banco de Dados pode ser definido como um conjunto de dados relacionados gerenciados por sistemas que surgiram com o intuito de contornar os problemas no formato de armazenamento em arquivos (NETO et al., 2017). São formados por níveis de componentes de hardware e software, com destaque para o SGBD (Sistema de Gerenciamento de Banco de Dados) que possibilita manipulações e persistências dos dados (DATE, 2004), geralmente através de uma linguagem SQL (*Structured Query Language*) como forma de acesso (CALDEIRA, 2015).

Uma arquitetura popular no mercado utilizada para implementar *back-end* em diversas linguagens é o MVC (*Model-View-Controller*) (ANICHE; GEROSA, 2015), criada em 1978, com o objetivo de fazer o modelo mental do usuário e o modelo digital (REENSKAUG, 1979). O MVC divide as responsabilidades em três camadas conhecidas como visão, modelo e controle (FRANCO; PIEDADE; RÊGO, 2014).

As visões (*view*) tem a responsabilidade de exibir uma interface de interação com o usuário para exibição e manipulação dos dados. Os modelos (*model*) tem a responsabilidade de acessar, persistir e gerenciar os dados armazenados no banco de dados que são exibidos nas visões. Os controladores (*controller*) são responsáveis por receber as requisições, decidir qual visão será exibida e realizar a comunicação com os modelos (FRANCO; PIEDADE; RÊGO, 2014).

O PHP (*Hypertext PreProcessor*) é uma linguagem que comumente implementa a arquitetura MVC através de *frameworks* (FRANCO; PIEDADE; RÊGO, 2014). Originalmente *Personal Home Page*, o PHP é uma linguagem de *script* para web amplamente utilizada que ganhou força pelo fácil aprendizado construída como linguagem estruturada, passou por um amadurecimento com suporte ao paradigma de orientação a objetos (MINETTO, 2007) e estima-se que seja utilizada em mais de 80% dos servidores web existentes (DALL'OGLIO, 2015).

As próximas seções tem o objetivo de apresentar algumas metodologias utilizadas para implementar as tecnologias no desenvolvimento de aplicações web, de forma eficaz, visando diminuir os custos de produção de manutenção do software.

2.3 BIBLIOTECAS, PRE-PROCESSADORES E FRAMEWORKS

Utilizar um *framework* exige menos esforço no desenvolvimento e gera produtividade (SILVA; FERRARI, 2016), pois são coleções de códigos-fonte, funções, classes e metodologias genéricas que servem de base para desenvolver algo maior e mais específico (MINETTO, 2007). Existe uma grande variedade de *frameworks* no mercado (FRANCO; PIEDADE; RÊGO, 2014) e a escolha de um deve considerar as tecnologias que são úteis e necessárias para o desenvolvimento (MCARTHUR, 2008).

A principal função do *framework* é facilitar o desenvolvimento de *softwares* evitando contratempos com detalhes de baixo nível, permitindo o foco dos esforços e tempo no desenvolvimento das regras de negócio do sistema. Normalmente possuem componentes de conectividade, implementam bibliotecas de acesso aos bancos de dados e promovem a reutilização de código (ALVES, 2017).

Os *frameworks* oferecem uma arquitetura própria com regras que devem ser seguidas para o desenvolvimento, enquanto a biblioteca disponibiliza uma estrutura de código que se adapta à arquitetura já existente e permite que sejam adicionadas funcionalidades específicas ao projeto (MARTINS, 2014).

Também pode ser entendido como uma plataforma de desenvolvimento e pode se tornar um problema caso seja utilizado de forma incorreta (ALVES, 2017), pois incorpora novas dependências para o projeto, deixa o ambiente de desenvolvimento mais complexo e exige uma capacitação técnica maior da equipe de desenvolvimento (MINETTO, 2007).

O Bootstrap é um dos principais *frameworks* do *front-end* que possui convenções HTML5, CSS3 e uma biblioteca JavaScript. Sua popularidade vem da responsividade de seus componentes e a compatibilidade entre os navegadores. As atuais vantagens do Bootstrap é o suporte os pré-processadores de CSS mais utilizados atualmente (SILVA; FERRARI, 2016).

O JQuery é uma biblioteca JS que simplifica o desenvolvimento de manipulações do HTML, eventos, animações e interações com AJAX. Tornou-se a biblioteca padrão do ASP.NET MVC, em 2008, e do Ruby on Rails, a partir da versão 3 (BALDUÍNO, 2014). O jQuery foi a biblioteca líder durante muito tempo, pois resolvia problemas de comportamento diferente entre os navegadores ao executar código JS (SOUZA, 2016), porém várias de suas funcionalidades foram implementadas nativamente nas linguagens e o uso do JQuery passou a ser considerado uma má pratica por ser uma biblioteca pesada (ANICHE; GEROSA, 2015).

O LESS e o SCSS (nova sintaxe do SASS, *Syntactically Awesome StyleSheets*), influenciada pelo LESS) são linguagens que criam folhas de estilo CSS dinamicamente, denominadas pre-processadores (SOUZA, 2016). O uso dessas ferramentas é muito recomendado (ANICHE; GEROSA, 2015), pois apresenta várias vantagens ao permitir o uso de variáveis, funções, cálculos matemáticos e importação de códigos, aplicando de forma simplificada e organizada os conceitos de código único, limpo e reutilizável (SOUZA, 2016).

Existe uma variedade de *frameworks* PHP disponíveis no mercado, por causa da popularidade da linguagem entre os desenvolvedores de aplicações web. Codeigniter (CODEIGNITER..., 2018), CakePHP (CAKEPHP..., 2018), Zend (ZEND..., 2018), Symphony (SENSIOLABS, 2018) e Laravel (OTWELL, 2018) são os *frameworks* mais utilizados na comunidade, possuem características semelhantes e utilizam a arquitetura MVC (FRANCO; PIEDADE; RÊGO, 2014).

Todos os *frameworks* existentes na atualidade tentam adotar o modelo MVC, contudo podem não seguir este padrão de forma rigorosa. Alguns misturam a responsabilidade do controle na visão e outros adicionam componentes extras dentro da arquitetura, sendo chamados de MV* por possuírem modelo e visão, mas conter um controle distinto ou inexistente (MARTINS, 2014).

A próxima seção tem o objetivo de apresentar conceitos da engenharia de *software* que são utilizados para projetar e desenvolver as aplicações web, selecionando qual a melhor arquitetura para se utilizar e, consequentemente, quais tecnologias serão utilizadas pela equipe na produção do *software*.

2.4 ENGENHARIA DE SOFTWARE

Software é um produto construído para atender as necessidades específicas de um cliente, abrangendo programas executáveis de qualquer porte ou arquitetura. Uma empresa de software se tornou o tipo mais influente do mercado, pois seus produtos foram incorporados em todos os seguimentos de comércio, sendo a mudança radical de antigas tecnologias (mídia), extensão de tecnologias existentes (telecomunicações) ou viabilizador para criação de novas tecnologias (nanotecnologia) (PRESSMAN; MAXIM, 2016).

Abordagens são utilizadas no desenvolvimento de *software* com o objetivo de não repetir os erros de projetos anteriores, sendo comum os problemas de altos custos para evolução ou manutenção, inconsistência entre documentação e sistema final, pouca ou quase que total falta de portabilidade e baixa confiabilidade (ALVES; MACHADO; RAMALHO, 2008).

A engenharia de *software* abrange processos e ferramentas que possibilitam o desenvolvimento de *software* em alta qualidade, com abordagens adaptadas de maneira conveniente, impondo disciplina em um ambiente que tende a se tornar caótico. A comunidade tenta criar tecnologias que tornam mais fácil, rápido e barato o desenvolvimento e manutenção do *software*. Adaptações, correções e melhorias absorvem mais recursos de tempo e esforço profissional do que a criação do mesmo (PRESSMAN; MAXIM, 2016).

Os engenheiros de *software* fazem modelos baseados em requisitos para representar a ideia do software a ser construído de forma gráfica (PRESSMAN; MAXIM, 2016), utilizados para gerenciamento da complexidade, redução de custos no desenvolvimento, comunicação entre as pessoas envolvidas e previsão de comportamento futuro do sistema. Os modelos apresentam coleções de elementos que possuem significados predefinidos, seguindo um padrão lógico denominado de diagrama (BEZERRA, 2017).

2.4.1 Modelagem

A UML (*Unified Modeling Language*) foi definida em 1996 como a melhor forma de representação de diagramas existente em diferentes técnicas de modelagem, sendo uma junção de diversas notações preexistentes com elementos removidos e adicionados com o objetivo de ter mais expressividade (BEZERRA, 2017).

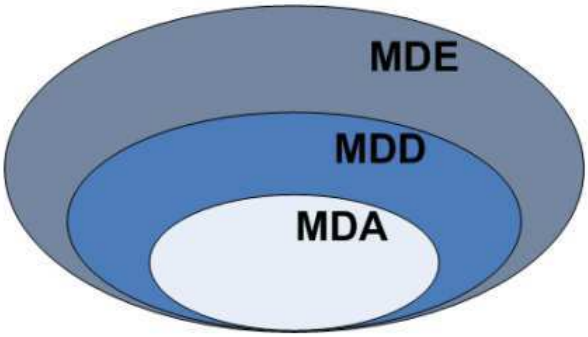
A UML foi construída para modelar sistemas orientados a objetos em que cada elemento visual possui uma sintaxe e semântica, utilizados para representar as operações e estados de cada parte do *software* (BEZERRA, 2017), baseado em visões e cenários possíveis do ponto de vista dos usuários durante seu funcionamento (PRESSMAN; MAXIM, 2016).

Com o aumento da complexidade no desenvolvimento de *software*, a área de Engenharia Dirigida por Modelos (*Model Driven Engineering* - MDE) destaca-se por tentar diminuir a distância entre domínio do problema, sendo os requisitos do usuário, e o domínio da solução, sendo a implementação do *software* (TELES et al., 2017). O modelo conceitual apresenta a classes de domínio e a estrutura de objetos envolvidos na interação entre um usuário e a aplicação (BEZERRA, 2017).

O MDE é um superconjunto que engloba atividades do processo de engenharia de *software* com os conceitos relacionados ao desenvolvimento dirigidas por modelos, como apresenta a figura 3, em que os modelos são os artefatos principais no processo

de desenvolvimento de *software* ao invés do código, pois são abstrações dos cenários que expõem apenas detalhes relevantes (TELES et al., 2017).

Figura 3 – Conjunto da Engenharia Dirigida por Modelos

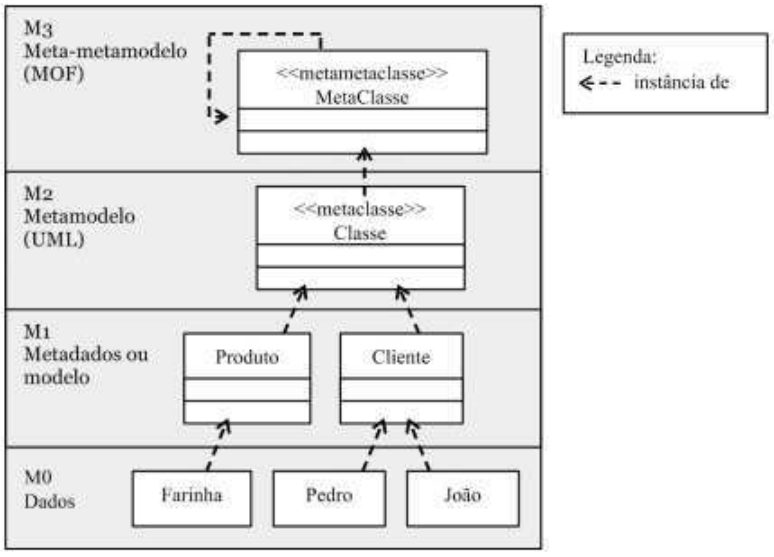


Fonte: (TELES et al., 2017)

O Desenvolvimento Dirigido por Modelos (*Model Driven Development* - MDD) promove o uso intensivo de modelos no desenvolvimento de *software* (REZENDE et al., 2016), pois sua definição de abordagem possibilita a construção um modelo de sistema que sofrerá transformações até tornar-se algo concreto, como código-fonte (TELES et al., 2017).

A figura 4 apresenta o utilização de metamodelos e modelos, conforme recomendações do método MDD em que são produzidos modelos genéricos e modelos mais específicos com base feitos anteriormente, até se ter algo bem próximo dos dados reais (LUCRÉDIO, 2009).

Figura 4 – Conceito do uso de modelos



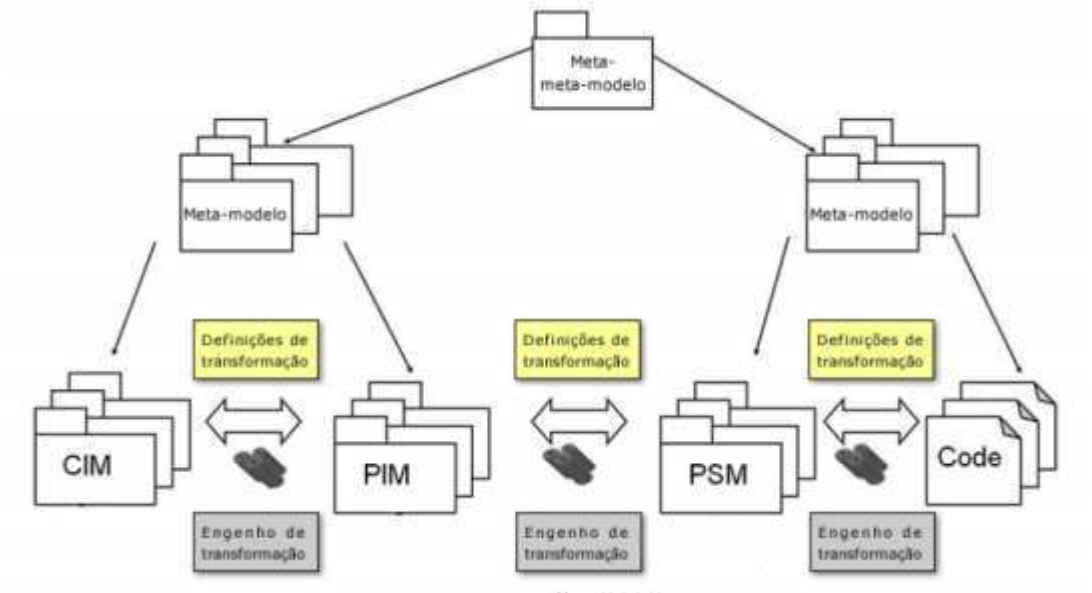
Fonte: (LUCRÉDIO, 2009)

A Arquitetura Dirigida por Modelos (*Model Driven Architecture* - MDA) é uma proposta de *framework* desenvolvida pela OMG, a qual representa uma visão da MDD.

Existem três etapas diferentes de abstrações de modelos, sendo respectivamente Modelo Independente de Computação (*Computacional Independent Model* - CIM), Modelo Independente de Plataforma (*Platform Independent Model* - PIM) e Modelo Específico da Plataforma (*Platform Specific Model* - PSM) (TELES et al., 2017).

O MDA oferece uma abordagem em que os modelos são formais e abstratos ao mesmo tempo, focados em detalhes relevantes para a solução (TELES et al., 2017). A figura 5 mostra os modelos que tem início de uma forma independente de computação, são refinados e transformados em modelos conceituais sem considerar aspectos tecnológicos, que são transformados em outros modelos com base em uma plataforma específica para, por fim, ser gerado o código-fonte (REZENDE et al., 2016).

Figura 5 – Conceito da utilização do MDA



Fonte: (TELES et al., 2017)

As vantagens do uso de MDD incluem velocidade no desenvolvimento alcançada através da automatização em que o código-fonte é gerado a partir das etapas de transformações, melhoria no gerenciamento da complexidade através de abstrações, permitindo alto nível de reusabilidade, produtividade e portabilidade do sistema entre tecnologias, somente modificando as regras de transformações (TELES et al., 2017).

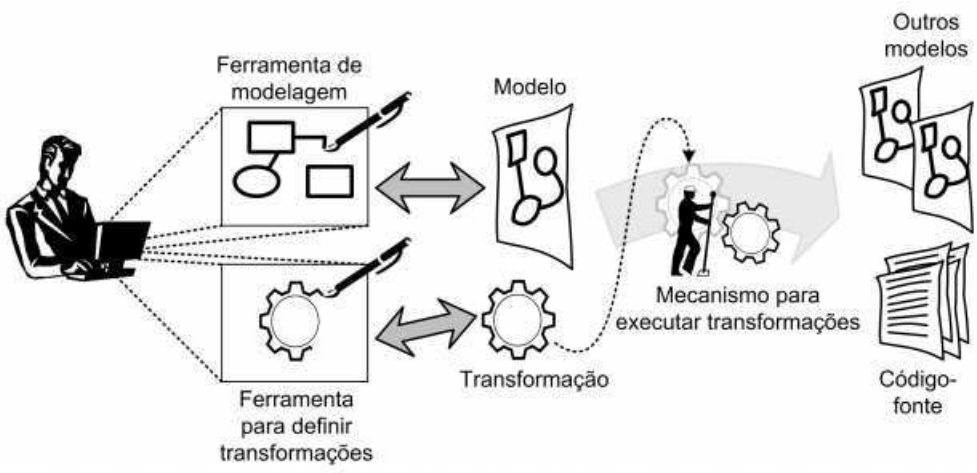
Porém, nenhuma informação é fornecida para indicar que os requisitos permanecem atendidos do início ao fim do processo de transformação entre as etapas. Portanto, é necessário validar as transformações e a verificar se foram realizadas de maneira correta, atendendo os requisitos especificados (REZENDE et al., 2016).

2.4.2 Ferramentas

Segundo Costa e Canedo (2015), existem diversas ferramentas e *frameworks* para auxiliar no desenvolvimento de MDA disponíveis gratuitamente ou pagos no mercado, sendo alguns deles: AndroMDA (BOHLEN, 2018), MagicDraw (INC, 2018), MDArte (SOFTWARE..., 2018) e Papyrus (PAPYRUS..., 2018).

A figura 6 representa o trabalho do desenvolvedor de definir os modelos e as regras de transformação que serão utilizados em um mecanismo de transformação que resultaram em novos modelos e códigos-fonte (LUCRÉDIO, 2009).

Figura 6 – Conceito de utilização das ferramentas

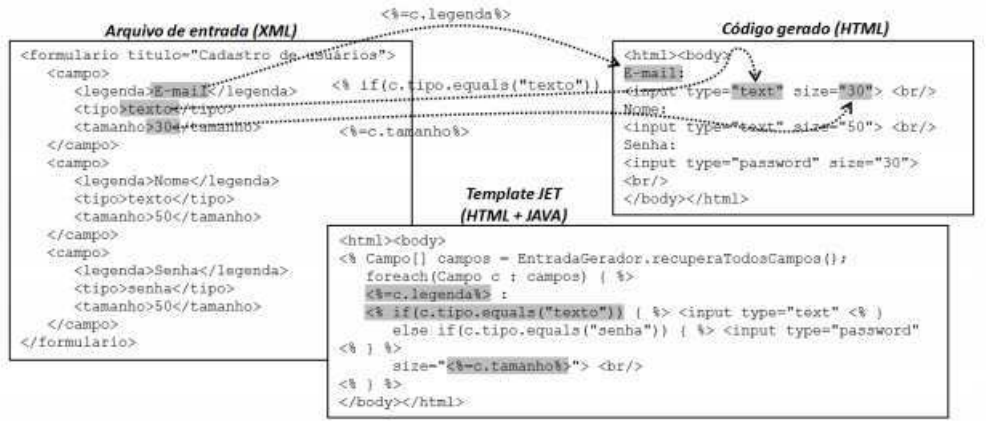


Fonte: (LUCRÉDIO, 2009)

AndroMDA é um *framework* de código-fonte aberto que recebe um modelo UML, combina com *plugins* de tradução e gera componentes customizados para linguagens de programação, como Java e PHP. A ferramenta possibilita que o desenvolvedor foque na lógica de negócio produzindo o PIM, enquanto o PSM é gerado automaticamente finalizando em código-fonte (COSTA; CANEDO, 2015).

A figura 7 apresenta os resultados obtidos após um processo de transformação, em que um arquivo XML é convertido para um arquivo HTML, através de uma regra de transformação em TemplateJET (LUCRÉDIO, 2009).

Figura 7 – Transformações realizadas por ferramentas



Fonte: (LUCRÉDIO, 2009)

Distribuído pela empresa No Magic, MagicDraw é um *software* pago para modelagem de sistemas com suporte ao trabalho em equipe utilizando o Teamwork Server, outro produto da mesma empresa. A ferramenta não está vinculado a nenhuma fase de projeto, pois possibilita modelagem de esquemas de banco de dados, geração de DDL, engenharia reversa e possui um mecanismo de código com suporte as linguagens Java, C#, CL (MSIL) e CORBA IDL (INC, 2018).

MDarte é um *framework* público brasileiro e *open-source* voltado para o desenvolvimento de softwares de governo com uma infra-estrutura MDA. MDarte é uma extensão do *framework* AndroMDA criada para suprir a necessidade do Governo Brasileiro de ter uma ferramenta padrão de desenvolvimento (SOFTWARE. . . , 2018).

Finalizando, a ferramenta Papyrus é um *plugin* do Eclipse de edição gráfica para a UML 2 que visa implementar 100% da especificação do OMG (*Object Management Group*) (PAPYRUS. . . , 2018) e possibilita a manipulação dos *profiles* na modelagem do UML (COSTA; CANEDO, 2015). Porém, existem diversas outras ferramentas disponíveis no mercado que podem possuir uma usabilidade melhor, como Modelio, UML Designer, Astah Professional, Enterprise Architect e IBM Rational Rose (LIMA, 2017).

A próxima seção tem o objetivo de apresentar trabalhos relacionados, destacando a importância do uso da Engenharia Dirigida por Modelos no desenvolvimento de aplicações, utilização de ferramentas facilitadoras disponíveis no mercado tecnológico atual e as vantagens obtidas com essas metodologias.

2.5 TRABALHOS RELACIONADOS

A expansão da web é entendida por Souza (2016), assim como o crescimento da importância e complexidade do desenvolvimento web. A pesquisa estuda os padrões utilizados por desenvolvedores e compara diversas ferramentas em todas as camadas do

front-end como, por exemplo: *frameworks* JS, pré-processadores de CSS e automatizadores de tarefas.

O estudo demonstrou a utilização básica de diversas ferramentas e tecnologias ao implementar uma aplicação web simples. Pode ser utilizado como um guia para a comunidade de desenvolvimento *front-end*, pois apresentada diversos conceitos, tecnologias e ferramentas que possuem extrema importância no desenvolvimento web moderno (SOUZA, 2016).

O trabalho de Silva e Ferrari (2016) apresenta a utilização de *frameworks* no desenvolvimento de interfaces ricas, analisando as vantagens e desvantagens obtidas em relação a produtividade no desenvolvimento, qualidade de código e desempenho da aplicação produzida.

O artigo demonstrou as inúmeras possibilidades na integração entre *frameworks* buscando facilitar e agilizar o desenvolvimento de interfaces ricas, mas destaca que podem ocorrer incompatibilidades entre alguns componentes das ferramentas, que demandaram tempo e conhecimento mais aprofundado para resolver Silva e Ferrari (2016).

O objetivo do trabalho de Lima (2017) foi conceber um protótipo de ferramenta CASE (*Computer-Aided Software Engineering*), pois evidências mostram que muitas ferramentas de modelagem são mais orientadas para funcionalidades do que usabilidade. A pesquisa compara algumas ferramentas que utilizam a linguagem UML e descreve os problemas encontrados conforme a tabela 1.

Tabela 1 – Análise das Ferramentas de Modelagem

Ferramenta	**Tempo de Execução**	**Inf. desnecessária**	**Muitos Ícones**
Papyrus	13 min	X	X
Modelio	11 min	X	X
UML Design	12 min	X	X
IBM Rational Rose	9 min		X
Enterprise Architect	7 min		
Astah Professiona	6 min		

Fonte: (LIMA, 2017)

O resultado desse trabalho serve de orientação para o desenvolvimento de novas ferramentas de modelagem, pois gera um protótipo de interface de usuário para ferramentas CASE de modelagem UML prototipado e avaliado por alunos da área da computação 1.

Costa e Canedo (2015) utiliza a abordagem MDA para demonstrar o aumento da produtividade no desenvolvimento de software. A pesquisa apresentou o potencial da metodologia para lidar com problema de desenvolver sistemas complexos, com funcionalidades ricas e prazos curtos. O trabalho foca no uso das ferramentas para modelagem como o AndroMDA e o Papyrus.

O trabalho de Rezende et al. (2016) apresenta o *framework* R2MDD que visa monitorar e rastrear requisitos durante as transformações de modelos, pois há a necessidade de garantir a integridade dos requisitos do início ao fim do projeto quando código-fonte deixa de ser o foco do desenvolvimento.

O uso do R2MDD no estudo de caso do modelo Qualitas no Hospital Universitário da Universidade Federal de Sergipe apontam o *framework* como de fácil entendimento, enxuto, com poucos níveis e que possibilita a associação de outros *frameworks*, modelos e metodologias, como o próprio modelo Qualitas e o Scrum. Porém, existe uma ausência de ferramentas que suportassem a aplicação do R2MDD (REZENDE et al., 2016).

Finalizando, o trabalho de Teles et al. (2017) apresenta o *framework* BPM4Services dirigido a modelos para automação de processos de negócio baseado em uma Arquitetura Orientada a Serviços (*Service Oriented Architecure* - SOA). A pesquisa demonstra a viabilidade da utilização do *framework*, através de um estudo de caso do processo de Triagem Neonatal no Hospital Universitário da Universidade Federal de Sergipe.

3 METODOLOGIA

O objetivo deste capítulo é apresentar o contexto desta pesquisa e quais serão os passos para a implementação do conceito MDD para geração de código-fonte, visando destacar os ganhos de produtividade e usabilidade no desenvolvimento de aplicações web, através de uma aplicação dos metamodelos e algoritmos de transformação produzidos em um experimento prático.

3.1 CONTEXTO DA PESQUISA

Conforme apresentado no capítulo anterior, existem diversos tipos de soluções de *software* sendo desenvolvidos para resolver problemas do cotidiano, como *softwares desktop*, sistemas embarcados, *apps mobile* e aplicações web. Cada tipo de *software* possui um desenvolvimento com particularidades, vantagens e dificuldades próprias.

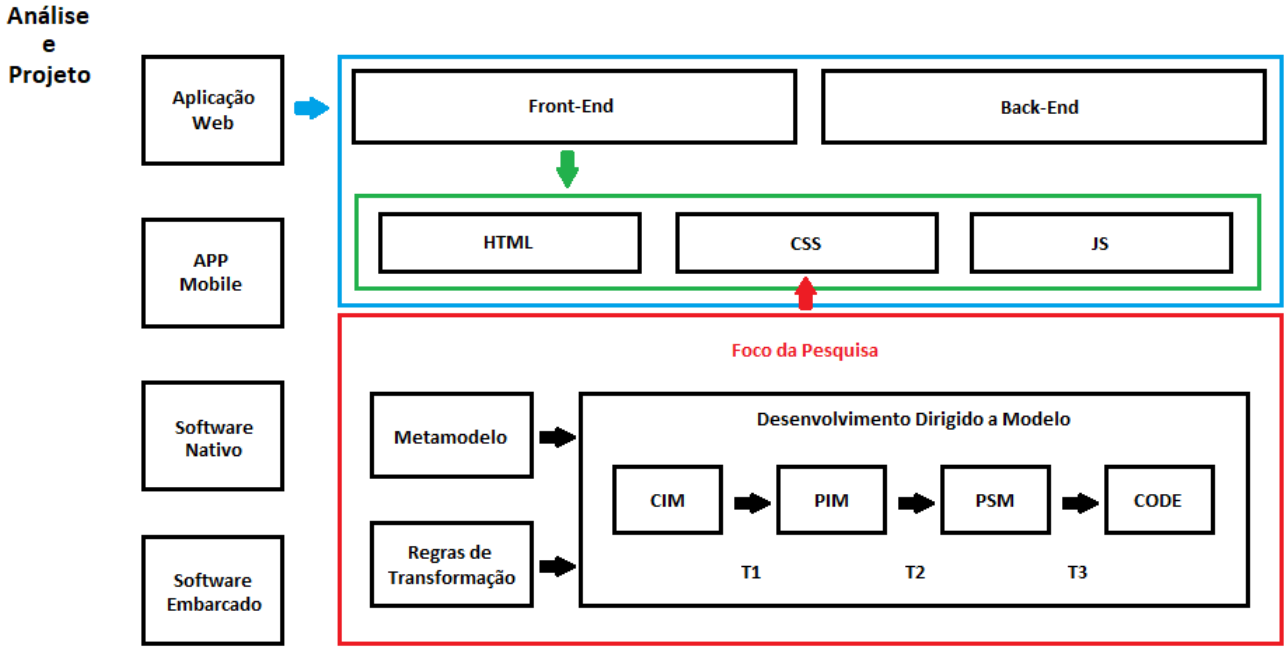
Foi constatado que aplicações web compõem a maior parte do mercado de desenvolvimento de *software*, pois possuem uma alta flexibilidade entre plataformas com poucas alterações em nível de código, mas exigem um melhor planejamento para atender todas as plataformas de maneira similar.

A Figura 8 apresenta que o desenvolvimento web possui duas frentes, sendo o *back-end* e o *front-end*. Segundo pesquisas (SILVA; FERRARI, 2016), é gasto mais tempo de desenvolvimento do *front-end* responsável pelas interfaces entre o usuário e os dados, pois há necessidade de garantir a compatibilidade de experiência entre todas as plataformas. Principal tecnologia que possibilita a adaptabilidade de aplicações web para diversas plataformas é a linguagem CSS.

Esta pesquisa será desenvolvida na área de análise e projeto, focando na modelagem de negócio com a ferramenta CASE Papyrus para produção de diagramas UML e perfis que representam o *layout* de uma aplicação web com o objetivo de ganhar produtividade no desenvolvimento e manutenção de aplicações web, com a implementação do conceito MDD, conforme destacado em vermelho na Figura 8.

A implementação do conceito de MDD utiliza metamodelos UML e algoritmos de transformação para obter código-fonte a partir dos modelos de forma semi-automatizada ou automatizada. Para isso, serão seguidos alguns passos para desenvolvimento da metodologia e comprovação dos resultados que estão detalhados na próxima seção.

Figura 8 – Contexto desta pesquisa



Fonte: Elaborada pelo Autor

3.2 DESENVOLVIMENTO DO CONCEITO MDD PARA GERAÇÃO DE CÓDIGO-FONTE

Esta seção é dedicada a detalhar os passos de desenvolvimento desta metodologia necessários para a implementação do conceito MDD para geração de código-fonte que é o foco desta pesquisa, conforme lista a seguir:

1. Elaboração do metamodelo para representar elementos de *layout* de aplicações web;
2. Desenvolvimento dos algoritmos de transformação de modelos em código-fonte;
3. Definição de um experimento prático para elaboração de modelos aplicando o metamodelo;
4. Aplicação do metamodelo e algoritmos de transformação no experimento prático;
5. Análise quali-quantitativa dos resultados obtidos;

3.2.1 Passo 1: Elaboração do metamodelo para representar elementos de layout de aplicações web

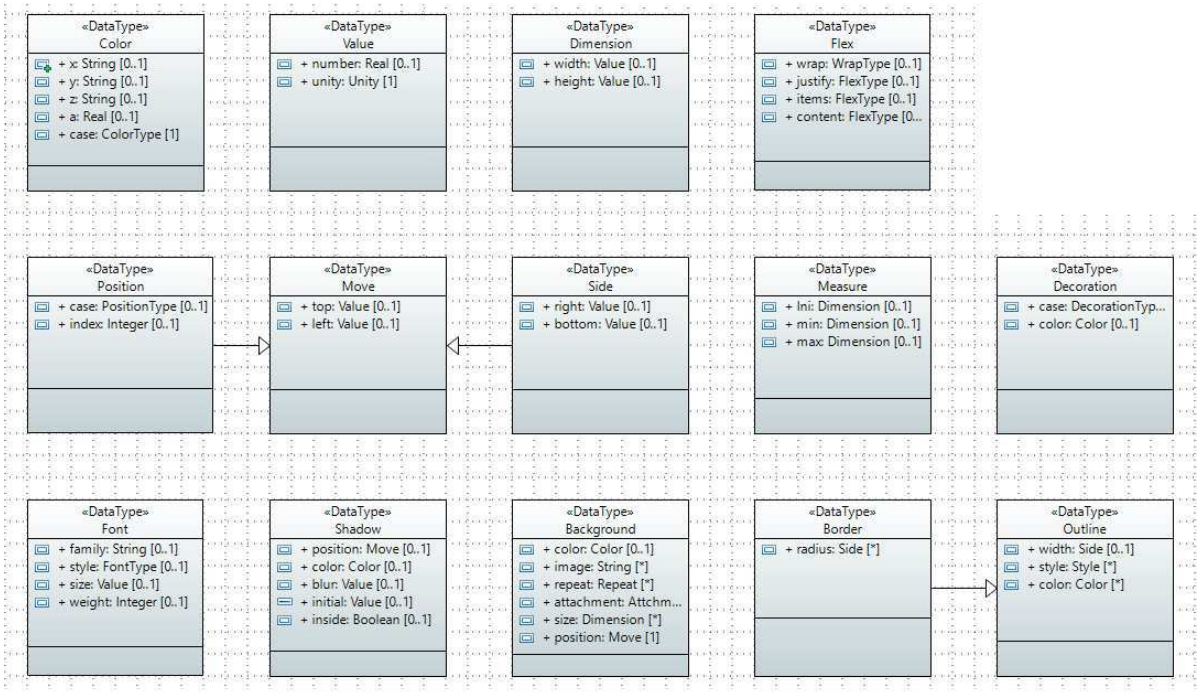
Os modelos são utilizados para representar as interações entre o usuário e a aplicação em um alto nível de abstração, possibilitando que todas sejam entendidas por qualquer

pessoa de forma similar, dispensando grandes conhecimentos técnicos. Os metamodelos são uma base estrutural para a criação de modelos com o mesmo formato, logo que cada modelo representa somente um cenário específico, os metamodelos generalizam todas as possibilidades utilizadas nos modelos através de perfis.

A elaboração de um metamodelo para representar elementos de *layout* é necessária para produzir um padrão de estrutura, sendo estendido nos modelos para representar situações especificas de *layout* das aplicações. Sendo assim, este passo propõe a elaboração de metamodelos para representar elementos de *layout* de aplicações Web, utilizando-se a ferramenta gratuita Papyrus que possibilita o uso de perfis.

Após a instalação da ferramenta, foi realizado um estudo da documentação CSS disponibilizada no site do w3school (**https://www.w3schools.com/cssref/default.asp**) para definição de abordagem ao desenvolver o metamodelo. Esse estudo possibilitou a visualização de alguns padrões entre propriedades que puderam ser abstraídos e simplificados durante a modelagem, resultados em poucos *DataType* que englobam muitas informações quando relacionadas. Alguns *DataTypes* podem ser visualizadas na figura 9.

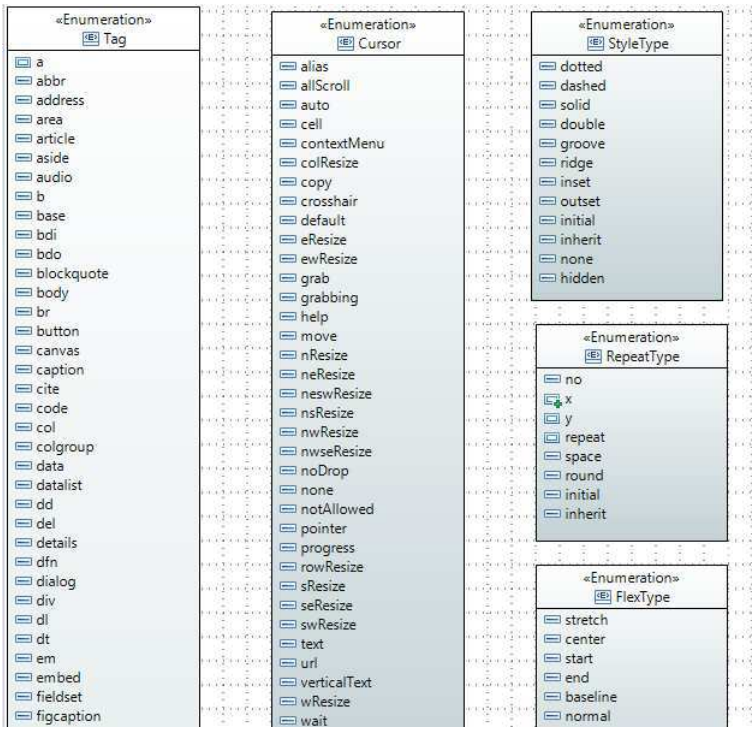
Figura 9 – Representação de alguns DataType's no metamodelo desenvolvido



Fonte: Autor

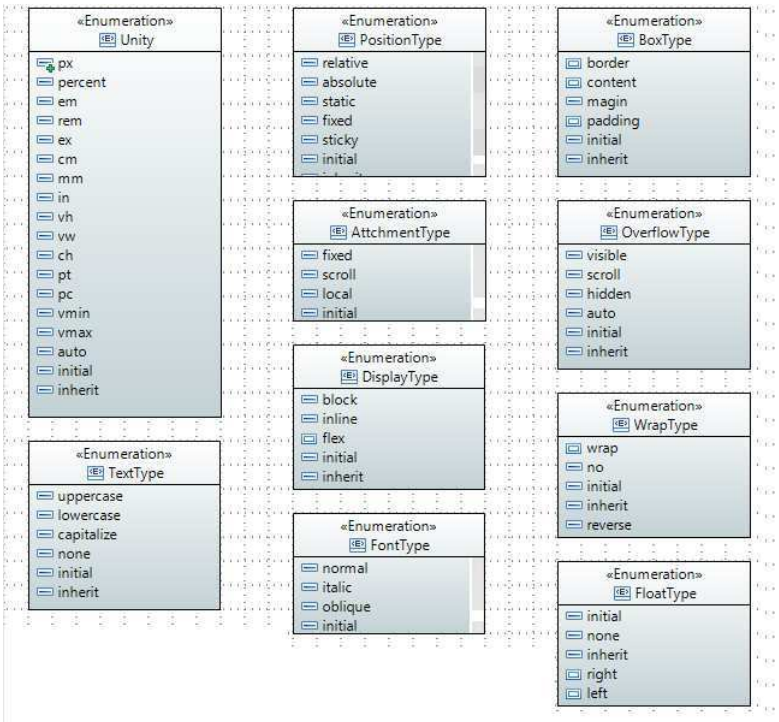
Muitas dessas classes possuem propriedades que devem possuir valores escritos que são escolhidos entre as possibilidades que aquela propriedade permite utilizar. Portanto, foi adotado o uso de *Enumeration's* que são uma lista de valores possíveis a serem utilizados no caso determinado. Este recurso foi utilizado também para informar, no modelo, se a classe CSS será convertida para uma *tag* HTML durante as transformações e parte disso pode ser visualizado nas figuras 10 e 11.

Figura 10 – Representação 1 de alguns Enumeration's no metamodelo desenvolvido



Fonte: Autor

Figura 11 – Representação 2 de alguns Enumeration's no metamodelo desenvolvido



Fonte: Autor

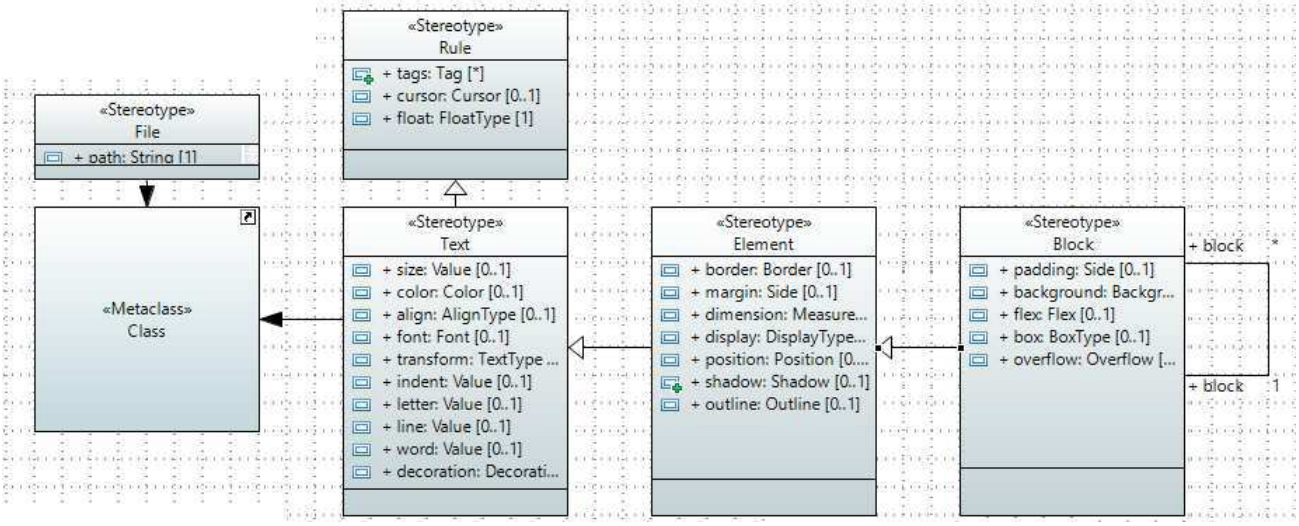
Esses elementos servem de base para a criação dos *Stereotype's* utilizados nos modelos para definir as propriedades dos elementos, classificados pela sua responsabilidade de *layout*. O *Stereotype* **Rule** possui propriedades independentes de responsabilidade,

como qual será o formato do *mouse* ao passar sobre o elemento. O *Stereotype* **Text** é uma especificação de **Rule**, sendo a responsabilidade mais simples de ser utilizada que adiciona as propriedades referentes a texto, como tamanho e cor da fonte.

O *Stereotype* **Element** especifica **Text**, adicionando as propriedades que um elemento de *layout* pode possuir, como borda, margem, largura e altura. O *Stereotype* **Block** especifica **Element**, sendo a responsabilidade mais complexa que pode ser utilizada, pois será um elemento que engloba outros elementos e, portanto, adiciona propriedades como margem interna e plano de fundo.

O último *Stereotype* criado foi **File** que serve para a modelagem da relação entre os arquivos de modelo, possuindo apenas uma propriedade utilizada para localizar o arquivo a que deve ser transformado. Esse modelo será utilizado como entrada para os algoritmos de transformação desenvolvidos no Passo 02. Todos os *Stereotype's* podem ser visualizados na figura 12.

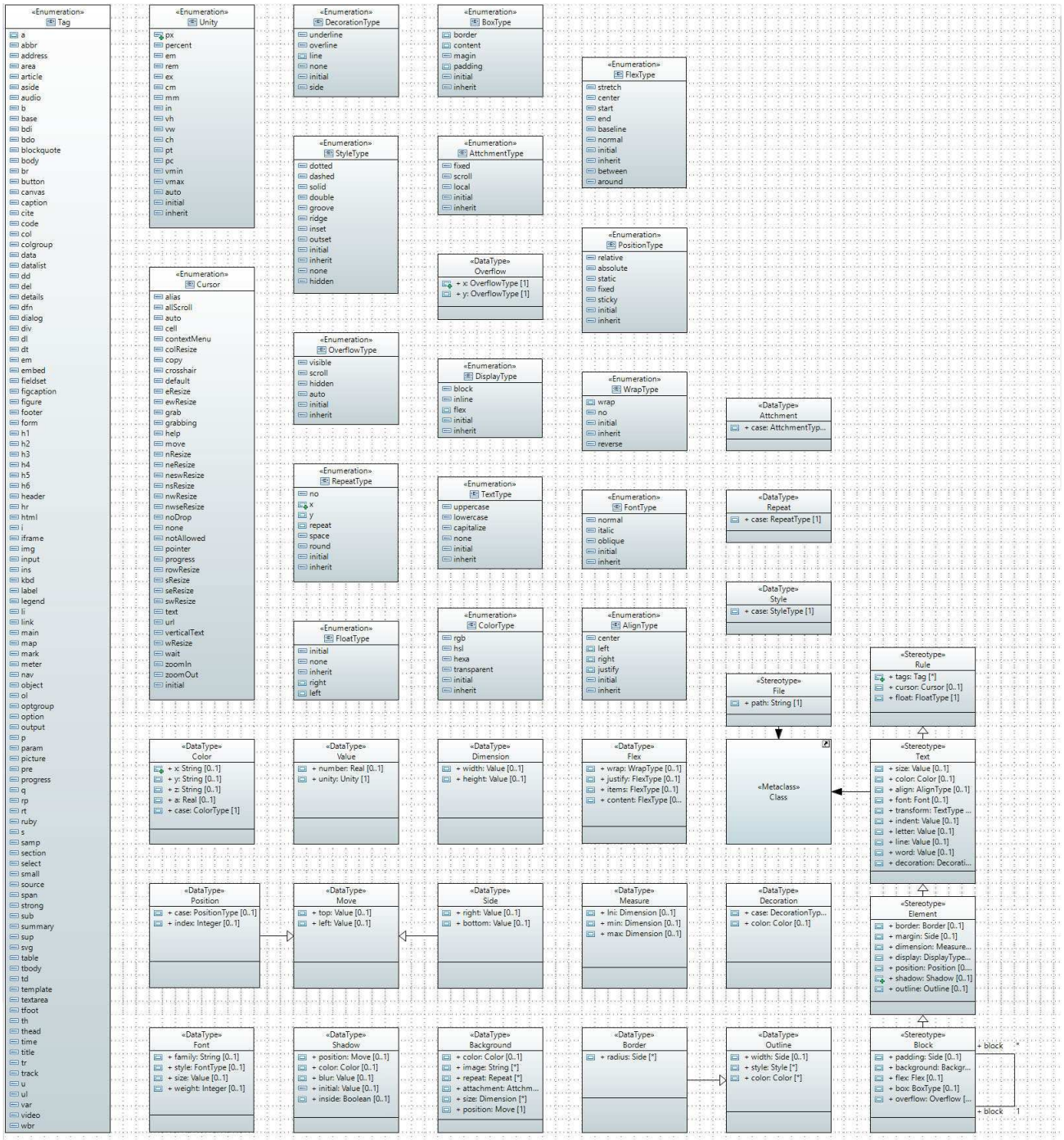
Figura 12 – Representação dos Stereotype's no metamodelo desenvolvido



Fonte: Autor

A representação do metamodelo completo elaborado pode ser visualizado como imagem em **https://github.com/duransbo/TCC/blob/master/modeloCompleto.jpg** e na figura 13.

Figura 13 – Representação do metamodelo completo desenvolvido



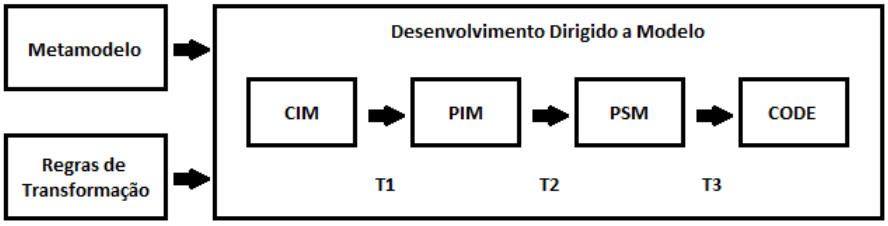
Fonte: Autor

O objetivo deste passo era representar o máximo possível as propriedades disponíveis do CSS, mas algumas funcionalidades não foram possíveis de representar dado a complexidade e necessitam de um estudo mais aprofundado para aplicação, como *media queries*, relação de parentesco entre elementos e seleção por atributo.

3.2.2 Passo 2: Desenvolvimento dos algoritmos de transformação de modelos em código-fonte

Neste passo foram definidos os algoritmos de transformação dos modelos em códigos-fonte, utilizando modelos elaborados com o metamodelo proposto no Passo 01. Assim, na implementação MDD, os algoritmos de transformação de modelos são utilizados para gerar códigos-fonte a partir de uma entrada de dados no formato de diagrama UML, correspondente ao Modelo Independente de Computação (*Computacional Independent Model* - CIM) demonstrado na figura 14.

Figura 14 – Foco da pesquisa



Fonte: Autor

A ferramenta Papyrus possibilita a primeira transformação (T1) do diagrama UML para um arquivo XML, o Modelo Independente de Plataforma (*Platform Independent Model* - PIM). Este será o modelo que deve ser processado pelas regras de transformação, a fim de obter o Modelo Específico da Plataforma (*Platform Specific Model* - PSM), conforme demonstra a figura 14.

A definição dos algoritmos de transformação são necessários para converter o PIM até código-fonte. Este trabalho utiliza o PHP, presente em mais de 80% dos servidores web (DALL'OGLIO, 2015), para interpretar arquivo XML e realizar as transformações do conceito MDD demonstrado na figura 14. Justifica-se a escolha do PHP, em função da familiaridade do autor desta pesquisa com esta linguagem.

3.2.2.1 Segunda Transformação: PIM para PSM

O primeiro algoritmo para transformação do PIM em PSM deve ser responsável por realizar uma limpeza do arquivo PIM, descartando todas as informações irrelevantes ou duplicadas, focando somente em dados vitais para a construção do código-fonte.

Essa aproximação do resultado final, gera um arquivo muito mais simplificado e de fácil utilização para a transformação T3. Portanto, foi adotado a abordagem de transformar o XML para JSON, pois possui um formado mais enxuto, preserva as informações importantes e é facilmente manipulável por várias linguagens, incluindo o PHP.

O arquivo **index.php**, disponíveis em **https://github.com/duransbo/flamel**, realiza a T2 utilizando a função **goldenTouch** da biblioteca **stone.php** para iniciar o processamento, passando como parâmetro o caminho do arquivo XML referente ao diagrama da relação de arquivos elaborado no Passo 03.

```
1 <?php
2     include_once('stone.php');
3     goldenTouch('../modelo/files.uml');
4 ?>
```

O arquivo **stone.php** importa três bibliotecas que possuem funções necessárias para o algoritmo, como manipulação de arquivos e processamento dos modelos. A função **goldenTouch** utiliza o diagrama da relação de arquivos para iniciar um processamento específico sobre esse arquivo que leva a transformação de todos os modelos, com as funções do arquivo **file.php**.

```
1  <?php
2
3      include_once('manipulate.php');
4      include_once('model.php');
5      include_once('file.php');
6
7      function transform ($elements) {
8          foreach ($elements as $x => $file) {
9              if (isset($file['path']) && file_exists('../modelo/'.$file['
                   path'])) {
10                 $json = clearDocument(readXML('../modelo/'.$file['path'
                       ]));
11                 if (isset($file['imports'])) {
12                     foreach ($file['imports'] as $y => $import) {
13                         foreach ($elements as $name => $imported) {
14                             if ($import == $imported['base_Class']) {
15                                 $json['imports'][$y] = $name;
16                             }
17                         }
18                     }
19                 }
20                 writeFile('../json/'.$x.'.json', toJSON($json));
21             }
22         }
23     }
24
```

```
25      function goldenTouch ($diagram) {
26          transform(cleanArray(filesSimple(linkFilePath(scanFiles(readXML(
                $diagram)), readXML($diagram)))));
27      }
28
29  ?>
```

Primeiramente, é realizado uma leitura do arquivo XML, convertendo os dados para um *array* para facilitar a manipulação no PHP. Em seguida, a função **scanFiles** varre o *array*, identificando todos os elementos (*packagedElement*) do diagrama e retorna um novo *array*.

A função **linkFilePath** utiliza o *array* retornado pela função **scanFiles** para varrer novamente o arquivo XML, agora identificando e relacionando os atributos para cada elemento. O resultado dessa função é utilizado na função **filesSimple** para estruturar as informações de forma melhor para o algoritmo.

A função **clearArray** funciona de forma recursiva, removendo todos os atributos vazios e irrelevantes para o processo seguinte. A função **transform** percorre esse *array* limpo para gerar todos os arquivos PSM pela função **clearDocument**, disponível no arquivo **model**, que possui uma lógica similar a função **transform**, porém otimizada para a manipulação dos diagramas UML de CSS.

```
1   <?php
2
3       function scanElements ($document) {
4           $elements = array();
5           $i = 0;
6           foreach ($document['uml:Model']['packagedElement'] as $element)
                {
7               if (isset($element['@attributes'])) {
8                   if ($element['@attributes']['type'] == 'uml:Class') {
9                       $elements[$i]['name'] = $element['@attributes']['
                            name'];
10                      $elements[$i]['id'] = $element['@attributes']['id'];
11                      $i++;
12                  }
13              } else {
14                  if ($element['type'] == 'uml:Class') {
15                      $elements[$i]['name'] = $element['name'];
16                      $elements[$i]['id'] = $element['id'];
17                      $i++;
18                  }
19              }
20          }
21          return $elements;
22      }
```

```
23
24      function linkElementAttributes ($elements, $document) {
25          foreach (array_slice($document,1,-1) as $stereotype) {
26              if (isset($stereotype['@attributes'])) {
27                  $elements[array_search($stereotype['@attributes']['
                        base_Class'], array_column($elements, 'id'))]['
                        attributes'] = $stereotype;
28              } else {
29                  foreach ($stereotype as $class) {
30                      $elements[array_search($class['@attributes']['
                            base_Class'], array_column($elements, 'id'))]['
                            attributes'] = $class;
31                  }
32              }
33          }
34          return $elements;
35      }
36
37      function elementsSimple ($elements) {
38          $clean = array();
39          foreach ($elements as $element) {
40              if (isset($element['attributes'])) {
41                  foreach ($element['attributes'] as $key => $attribute) {
42                      $clean[$element['name']][$key] = $attribute;
43                  }
44              }
45          }
46          return $clean;
47      }
48
49      function cleanArray ($array) {
50          foreach ($array as $key => $value) {
51              if (is_array($value)) {
52                  $array[$key] = cleanArray($value);
53                  if ($key === '@attributes') {
54                      foreach ($array[$key] as $k => $v) {
55                          $array[$k] = $v;
56                      }
57                  }
58                  if (array_key_exists('unity', $value)) {
59                      $array = ($value['unity'] != 'auto' ? floatval(
                            @$value['number']) : '') . ($value['unity'] == '
                            percent' ? '%' : $value['unity']);
60                      break;
61                  }
62              }
63              if ($key === '@attributes' || $key === 'type' || $key === '
```

```
                id' || empty($array[$key])) {
64                  unset($array[$key]);
65              }
66          }
67          return $array;
68      }
69
70      function clearDocument ($document) {
71          return cleanArray(elementsSimple(linkElementAttributes(
                scanElements($document), $document)));
72      }
73
74  ?>
```

Pode-se observar um ganho ao comparar o arquivo XML do menor diagrama (**footer**), com o arquivo JSON gerado após a T2, pois resultou em uma redução de 85% do tamanho em *bytes*.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xmi:XMI xmi:version="20131001" xmlns:xmi="http://www.omg.org/spec/XMI
       /20131001" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:css="http:///schemas/css/_faVFQMucEeipn5oiYGENIQ/27"
       xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" xmlns:uml="http:
       //www.eclipse.org/uml2/5.0.0/UML" xsi:schemaLocation="http:///schemas
       /css/_faVFQMucEeipn5oiYGENIQ/27 /tcc/css.profile.uml#
       _fadoIMucEeipn5oiYGENIQ">
3    <uml:Model xmi:id="_HswfULepEei1WcO5PNJaPw" name="footer">
4      <packageImport xmi:type="uml:PackageImport" xmi:id="
           _H2QZkLepEei1WcO5PNJaPw">
5        <importedPackage xmi:type="uml:Model" href="pathmap://
             UML_LIBRARIES/UMLPrimitiveTypes.library.uml#_0"/>
6      </packageImport>
7      <packagedElement xmi:type="uml:Class" xmi:id="
           _JAuBkLepEei1WcO5PNJaPw" name="BodyFooter"/>
8      <profileApplication xmi:type="uml:ProfileApplication" xmi:id="
           _H1wqULepEei1WcO5PNJaPw">
9        <eAnnotations xmi:type="ecore:EAnnotation" xmi:id="
             _OMYWEMudEeipn5oiYGENIQ" source="PapyrusVersion">
10         <details xmi:type="ecore:EStringToStringMapEntry" xmi:id="
               _OMYWEcudEeipn5oiYGENIQ" key="Version" value="0.0.28"/>
11         <details xmi:type="ecore:EStringToStringMapEntry" xmi:id="
               _OMYWEsudEeipn5oiYGENIQ" key="Comment" value=""/>
12         <details xmi:type="ecore:EStringToStringMapEntry" xmi:id="
               _OMYWE8udEeipn5oiYGENIQ" key="Copyright" value=""/>
13         <details xmi:type="ecore:EStringToStringMapEntry" xmi:id="
               _OMYWFMudEeipn5oiYGENIQ" key="Date" value="2018-10-09"/>
14         <details xmi:type="ecore:EStringToStringMapEntry" xmi:id="
```

```
            _OMYWFcudEeipn5oiYGENIQ" key="Author" value=""/>
15        </eAnnotations>
16        <eAnnotations xmi:type="ecore:EAnnotation" xmi:id="
              _H1wqUbepEei1WcO5PNJaPw" source="http://www.eclipse.org/uml2
              /2.0.0/UML">
17          <references xmi:type="ecore:EPackage" href="/tcc/css.profile.uml
                #_fadoIMucEeipn5oiYGENIQ"/>
18        </eAnnotations>
19        <appliedProfile xmi:type="uml:Profile" href="/tcc/css.profile.uml#
              _hakwkCPpEeibvcGHFL70Mg"/>
20      </profileApplication>
21    </uml:Model>
22    <css:Block xmi:id="_LvJ9wLepEei1WcO5PNJaPw" base_Class="
          _JAuBkLepEei1WcO5PNJaPw">
23      <dimension xmi:type="css:Measure" xmi:id="_M8G5ELepEei1WcO5PNJaPw">
24        <Ini xmi:type="css:Dimension" xmi:id="_NIqHULepEei1WcO5PNJaPw">
25          <height xmi:type="css:Value" xmi:id="_NoSCkLepEei1WcO5PNJaPw"
                number="50.0" unity="px"/>
26        </Ini>
27      </dimension>
28      <background xmi:type="css:Background" xmi:id="_P6sVwMJMEeiU-
            J2K3IPxpQ">
29        <image>../image/logo.png</image>
30        <repeat xmi:type="css:Repeat" xmi:id="_KlkOsMJNEeiU-J2K3IPxpQ"
              case="no"/>
31        <size xmi:type="css:Dimension" xmi:id="_TXohQMJNEeiU-J2K3IPxpQ">
32          <width xmi:type="css:Value" xmi:id="_TXohQcJNEeiU-J2K3IPxpQ"
                unity="auto"/>
33          <height xmi:type="css:Value" xmi:id="_TXohQsJNEeiU-J2K3IPxpQ"
                number="60.0" unity="percent"/>
34        </size>
35        <position xmi:type="css:Move" xmi:id="_MXwlEMJNEeiU-J2K3IPxpQ">
36          <top xmi:type="css:Value" xmi:id="_NcnUUMJNEeiU-J2K3IPxpQ"
                number="50.0" unity="percent"/>
37          <left xmi:type="css:Value" xmi:id="_Nk6UsMJNEeiU-J2K3IPxpQ"
                number="50.0" unity="percent"/>
38        </position>
39      </background>
40    </css:Block>
41  </xmi:XMI>
```

```
1   {
2      "BodyFooter":{
3         "dimension":{
4            "Ini":{
5               "height":"50px"
6            }
```

```
7         },
8         "background":{
9            "image":"..\/image\/logo.png",
10           "repeat":{
11              "case":"no"
12           },
13           "size":{
14              "width":"auto",
15              "height":"60%"
16           },
17           "position":{
18              "top":"50%",
19              "left":"50%"
20           }
21        },
22        "base_Class":"_JAuBkLepEei1WcO5PNJaPw"
23     }
24  }
```

O PSM gerado por esse algoritmo de transformação em PHP é processado em tempo de execução pelo servidor, no momento de acesso a aplicação. Este processamento realiza a terceira transformação (T3) demonstrada na figura 14, retornando um código-fonte CSS possível de ser interpretado pelos navegadores.

Esses processos visam atender as necessidades de portabilidade do *layout* de aplicações web entre as diversas plataformas de acesso disponíveis, utilizando as técnicas de *responsive web design* através de modelos.

3.2.2.2 Terceira Transformação: PSM para código-fonte

Os códigos CSS são interpretados e executados no momento que o navegador carrega todos os códigos-fonte encontrados nos arquivos requisitados. Portanto, a renderização de uma tela só é iniciada após o carregamento dos arquivos acorrentados e processamento realizado nos mesmos.

Os servidores retornam os arquivos requisitados no formato de extensão definida do arquivo para que o navegador realize a interpretação do mesmo, mas arquivos PHP são processados pelo servidor antes de serem enviados para o navegador de maneira que possam ser executados.

O PHP consome o PSM para aproveitar essa funcionalidade de maneira que utiliza o processamento do servidor para implementar as funcionalidades de reaproveitamento de código presentes nos pré-processadores para gerar o código CSS, gerando a terceira transformação em tempo de execução.

```
1  <?php
```

```
2     header('Content-type: text/css');
3     include('metamorfose.php');
4
5     metamorfose('../json/Reset.json');
6     metamorfose('../json/Template.json');
7     metamorfose('../json/Header.json');
8     metamorfose('../json/footer.json');
9 ?>
```

Para o navegador consumir o arquivo CSS retornado após o processamento do PHP, é necessário alterar o **content-type** como na linha 2. Depois de incluir a biblioteca **metamorfose.php** na linha 3, disponível em **https://github.com/duransbo/metamorfose**, o uso da função **metamorfose** utiliza o caminho do arquivo PSM para consumí-lo e gerar o código-fonte final.

A biblioteca metamorfose foi escrita com paradigma procedural, mas uma grande evolução para a escrita do código seria a abordagem Orientada a Objetos, pois o código pode ser escrito baseado no metamodelo elaborado no Passo 01, orientando o desenvolvimento desse algoritmo em paralelo a elaboração dos modelos, obtendo um ganho de produtividade.

Todas as funções estão diretamente ligadas a uma classe do metamodelo elaborado, realizando a logica necessária para a conversão das propriedades modeladas em código-fonte.

```
1  <?php
2
3      // Utilizado para ler arquivos JSON
4      function readJSON ($file) { return json_decode(file_get_contents(
           $file), true);}
5
6      // Escreve uma propriedade basica de chave-valor
7      function write ($a, $b) {
8          if (isset($a)) {
9              echo "\t$b: $a;\n";
10         }
11     }
12
13     // Escreve uma propriedade com um array de valores
14     function writeArray($a, $b) {
15         if (is_array(@$a)) {
16             echo "\t$b:";
17             foreach ($a as $n) {
18                 echo " $n";
19             }
20             echo ";\n";
21         } else {
22             write(@$a, $b);
23         }
```

```
24      }
25
26      // Escreve as propriedades Width e Height
27      function dimension ($a) {
28          if (isset($a)) {
29              write(@$a['width'], 'width');
30              write(@$a['height'], 'height');
31          }
32      }
33
34      // Escreve as propriedades do tipo Size
35      function size ($a, $b) {
36          if (isset($a)) {
37              echo "\t$b: ".(isset($a['width']) ? $a['width'] : '')." ".(
                    isset($a['height']) ? $a['height'] : '').";\n";
38          }
39      }
40
41      // Escreve as propriedades do tipo Position
42      function position ($a, $b) {
43          if (isset($a)) {
44              echo "\t$b: ".(isset($a['top']) ? $a['top'] : '')." ".(isset
                    ($a['left']) ? $a['left'] : '').";\n";
45          }
46      }
47
48      // Escreve as propriedades do tipo Side
49      function side ($a, $b) {
50          if (isset($a)) {
51              echo "\t$b: ".(isset($a['top']) ? $a['top'] : '0px')." ".(
                    isset($a['left']) ? $a['left'] : '0px')." ".(isset($a['
                    bottom']) ? $a['bottom'] : '0px')." ".(isset($a['right'])
                     ? $a['right'] : '0px').";\n";
52          }
53      }
54
55      // Escreve as propriedades do tipo Color
56      function color ($a, $b) {
57          if (isset($a)) {
58              echo "\t$b: ";
59              switch ($a['case']) {
60                  case 'hexa':
61                      echo '#'.$a['x'];
62                      break;
63                  case 'rgb':
64                      echo 'rgba('.$a['x'].','.$a['y'].','.$a['z'].','.(
                            isset($a['a']) ? $a['a'] : 1).')';
```

```
65                      break;
66                  case 'hsl':
67                      echo 'hsla('.$a['x'].','.$a['y'].','.$a['z'].','.(
                            isset($a['a']) ? $a['a'] : 1).')';
68                      break;
69                  default:
70                      echo $a['case'];
71                      break;
72              }
73              echo ";\n";
74          }
75      }
76
77      // Escreve as propriedades do tipo Flex
78      function flex ($a, $b) {
79          if (isset($a)) {
80              echo "\t$b: ";
81              switch ($a) {
82                  case 'start':
83                      echo 'flex-start';
84                      break;
85                  case 'end':
86                      echo 'flex-end';
87                      break;
88                  case 'between':
89                      echo 'space-between';
90                      break;
91                  case 'around':
92                      echo 'space-around';
93                      break;
94                  default:
95                      echo $a;
96                      break;
97              }
98              echo ";\n";
99          }
100     }
101
102     // Escreve as propriedades do tipo Decoration
103     function decoration ($a, $b) {
104         if (isset($a)) {
105             echo "\t$b: ";
106             switch ($a) {
107                 case 'line':
108                     echo 'line-through';
109                     break;
110                 case 'side':
```

```
111                     echo 'overline underline';
112                     break;
113                 default:
114                     echo $a;
115                     break;
116             }
117             echo ";\n";
118         }
119     }
120
121     // Utilizada para processar o arquivo JSON, passando por todas as
            propriedades e escrevendo conforme converso necessaria
122     function metamorfose ($file) {
123         $json = readJSON($file);
124         foreach ($json as $key => $element) {
125             if ($key == 'imports') {
126                 if (is_array($element)) {
127                     foreach ($element as $import) {
128                         echo '@import '. $import.".php\n";
129                     }
130                 } else {
131                     echo '@import '. $element.".php\n";
132                 }
133             } else {
134                 if (isset($element['tags'])) {
135                     if (is_array($element['tags'])) {
136                         foreach ($element['tags'] as $n => $tag) {
137                             echo ($n == 0 ? '' : ",\n").$tag;
138                         }
139                     } else {
140                         echo $element['tags'];
141                     }
142                 } else {
143                     echo ".".$key;
144                 }
145                 echo " {\n";
146                 foreach ($element as $attribute => $value) {
147                     if ($attribute != 'tags' && $attribute != '
                            base_Class' && $attribute != 'imports') {
148                         switch ($attribute) {
149                             case 'size':
150                                 echo "\tfont-size: $value;\n";
151                                 break;
152                             case 'dimension':
153                                 dimension(@$value['Ini']);
154                                 dimension(@$value['max']);
155                                 dimension(@$value['min']);
```

```
156                         break;
157                     case 'outline':
158                         side(@$value['width'], 'outline-width');
159                         writeArray(@$value['style'], 'outline-
                                style');
160                         color(@$value['color'], 'outline-color')
                                ;
161                         break;
162                     case 'border':
163                         side(@$value['radius'], 'border-radius')
                                ;
164                         side(@$value['width'], 'border-width');
165                         writeArray(@$value['style'], 'border-
                                style');
166                         color(@$value['color'], 'border-color');
167                         break;
168                     case 'background':
169                         color(@$value['color'], 'background-
                                color');
170                         writeArray(@$value['repeat'], '
                                background-repeat');
171                         writeArray(@$value['attachment'], '
                                background-attchment');
172                         size(@$value['size'], 'background-size')
                                ;
173                         position(@$value['position'], '
                                background-size');
174                         if (isset($value['image'])) {
175                             echo "\tbackground-image: url('".
                                    $value['image']."');\n";
176                         }
177                         break;
178                     case 'margin':
179                         side(@$value, 'margin');
180                         break;
181                     case 'padding':
182                         side(@$value, 'padding');
183                         break;
184                     case 'align':
185                         echo "\ttext-align: $value;\n";
186                         break;
187                     case 'line':
188                         echo "\tline-height: $value;\n";
189                         break;
190                     case 'display':
191                         echo "\t$attribute: $value;\n";
192                         break;
```

```
193                         case 'box':
194                             echo "\tbox-sizing: $value-box;\n";
195                             break;
196                         case 'transform':
197                             echo "\ttext-transform: $value;\n";
198                             break;
199                         case 'overflow':
200                             write(@$value['x'], 'overflow-x');
201                             write(@$value['y'], 'overflow-y');
202                             break;
203                         case 'cursor':
204                             echo "\t$attribute: $value;\n";
205                             break;
206                         case 'float':
207                             echo "\t$attribute: $value;\n";
208                             break;
209                         case 'color':
210                             color(@$value, 'color');
211                             break;
212                         case 'decoration':
213                             decoration(@$value['case'], 'text-
                                decoration-line');
214                             color(@$value['color'], 'text-decoration
                                -color');
215                             break;
216                         case 'position':
217                             write(@$value['case'], 'position');
218                             write(@$value['index'], 'index');
219                             write(@$value['top'], 'top');
220                             write(@$value['left'], 'left');
221                             break;
222                         case 'flex':
223                             write(@$value['wrap'], 'flex-wrap');
224                             flex(@$value['justify'], 'justify-
                                content');
225                             flex(@$value['items'], 'align-items');
226                             flex(@$value['content'], 'align-content'
                                );
227                             break;
228                         case 'font':
229                             if (isset($value['family'])) {
230                                 echo "\tfont-family: '".$value['
                                    family']."';\n";
231                             }
232                             write(@$value['style'], 'font-style');
233                             write(@$value['size'], 'font-size');
234                             write(@$value['weight'], 'font-weight');
```

```
235                                     break;
236                                 case 'teste':
237                                     echo "\t$attribute: ";
238                                     var_dump($value);
239                                     break;
240                                 default:
241                                     echo "\t// Erro no atributo  $attribute\
                                        n";
242                                     break;
243                             }
244                         }
245                     }
246                     echo "}\n\n";
247                 }
248             }
249         }
250
251 ?>
```

O PSM é transformado para o contexto da tecnologia de forma funcional, conforme observado na T3 do arquivo **footer**. O resultado de um código-fonte funcional sem a necessidade de conhecimento técnico demonstra a grande utilidade da abordagem MDD.

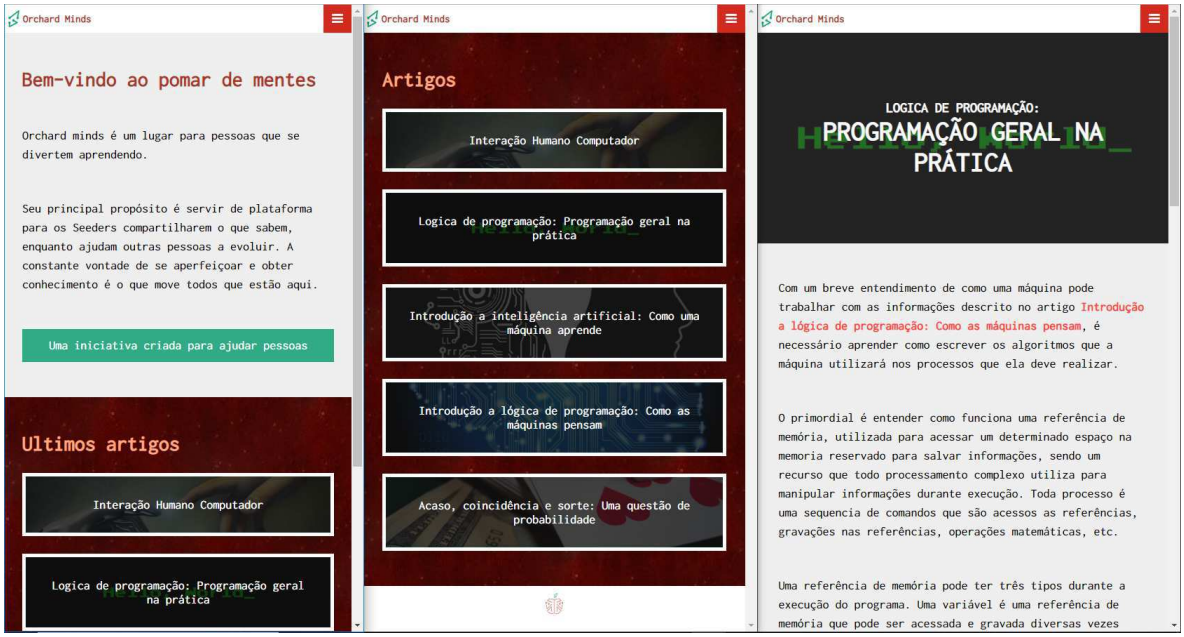
```
1 .BodyFooter {
2         height: 50px;
3         background-repeat: no-repeat;
4         background-size: auto 60%;
5         background-position: 50% 50%;
6         background-image: url('../image/logo.png');
7 }
```

3.2.3 Passo 3: Definição de um experimento prático para elaboração de modelos aplicando o metamodelo

A definição do experimento prático tem o objetivo de focar os esforços para a criação de modelos que representam o *layout* de páginas específicas, aplicando todas as regras desenhadas no metamodelo de CSS. Visando produtividade do trabalho, foi utilizado um projeto preexistente por não exigir preocupações com a criação de um experimento do zero. O projeto **Orchard** que simula um *blog* pode ser visualizado em **https://duransbo.github.com/orchard**.

Os *layout's* que podem ser visualizados na figura 15 são, respectivamente, da página inicial (**home**), a listagem dos artigos (**lista-de-artigos**) e de um artigo aberto (**artigo**). Essas são as três variações de *layout* necessárias para a sistemática de um *blog*.

Figura 15 – Layout's do projeto Orchard que serão representados



Fonte: Autor

O projeto **Orchard** foi desenvolvido utilizando uma plataforma que gera códigos estáticos chamada de **Hexo**. Antes do processo para gerar os códigos estáticos, o projeto possui sete arquivos de estilo para *layout*. O arquivo da pasta **__util** chamado **reset** tem a função de nivelar a base de estilo das *tag's* para padronizar o *layout* inicial em todos os navegadores. Os arquivos da pasta **__partial** são *layout's* de alguns componentes utilizados em diversas páginas, sendo o cabeçalho (**header**) e o rodapé (**footer**).

Os arquivos **home**, **artigo** e **lista-de-artigos** possuem as estruturas específicas de contexto, que possibilitam os *layout's* da figura 15. E o arquivo **template** é responsável por carregar as estruturas utilizadas por todas as páginas, portanto, possui algumas estruturas utilizadas por todo o projeto, independente de componente ou *layout*, e importa os componentes (**header** e **footer**).

Após o desenvolvimento do metamodelo que representa os elementos de *layout* de aplicações web e a definição das regras de transformação para gerar códigos-fonte a partir de modelos, o Passo 4 que realiza o experimento prático pode ser executado com o objetivo de verificar as possibilidades propostas no metamodelo proposto no Passo 01 e validar se as regras de transformações convertem todas as informações dos modelos, comparando se os requisitos representados no CIM continuam fiéis no código-fonte.

4 RESULTADOS

O objetivo deste capítulo é apresentar e analisar os resultados obtidos através da aplicação do metamodelo proposto no Passo 1 e dos algoritmos de transformação desenvolvidos no Passo 2 em um experimento prático definido no Passo 3.

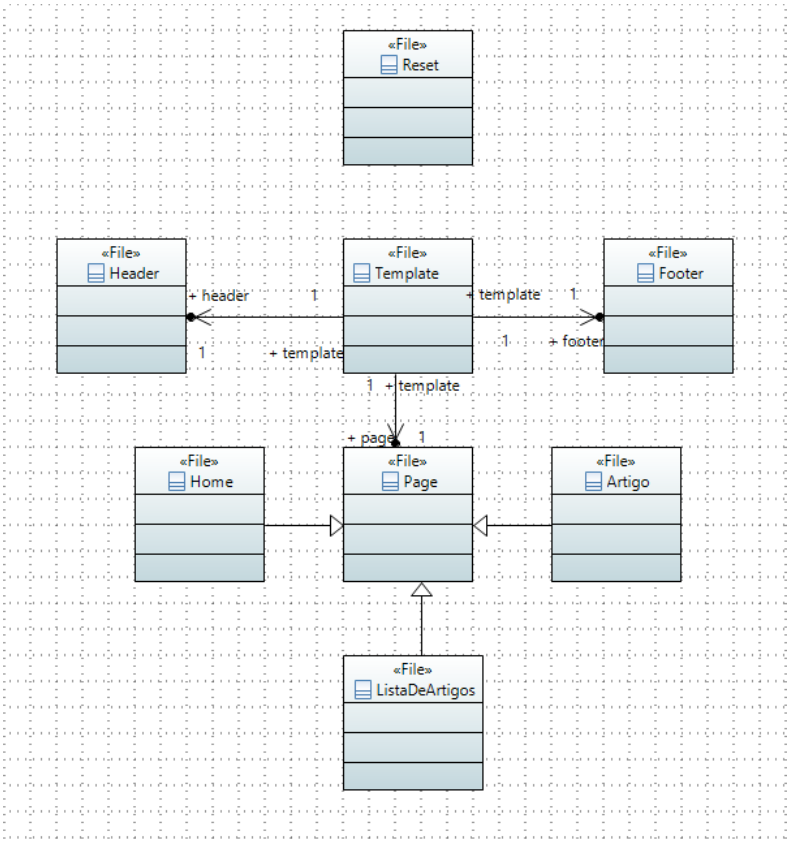
4.1 PASSO 4: APLICAÇÃO DO METAMODELO E ALGORITMOS DE TRANSFORMAÇÃO NO EXPERIMENTO PRÁTICO

O projeto Orchard escolhido como base para ser modelado possui 7 arquivos. Com o processamento da plataforma Hexo, são unidos resultando em dois arquivos finais quando acessado a página **home**. O arquivo **template.css** possui todas as estruturas compartilhadas por todas as páginas (**reset**, **template**, **header** e **footer**), e o arquivo **home.css** possui as estruturas únicas dessa página.

Após o entendimento dessa estrutura, é possível elaborar um diagrama aplicando o metamodelo proposto no Passo 1 que pode ser visualizado na figura 16. Vale destacar que, para este caso, utilizou-se uma engenharia reversa, visando obter o mesmo resultado de CSS do projeto Orchard, mas agora gerando códigos automaticamente, a partir de modelos.

O diagrama possui os sete arquivos de estilo e representa as relações que servem de entrada para os algoritmos do Passo 03. O **reset** é único, enquanto o **template** incorpora os arquivos **header**, **footer** e um arquivo de *layout* específico (**page**). **Page** é uma generalização dos arquivos **home**, **artigo** e **listaDeArtigos**.

Figura 16 – Diagrama de relação dos arquivos de layout



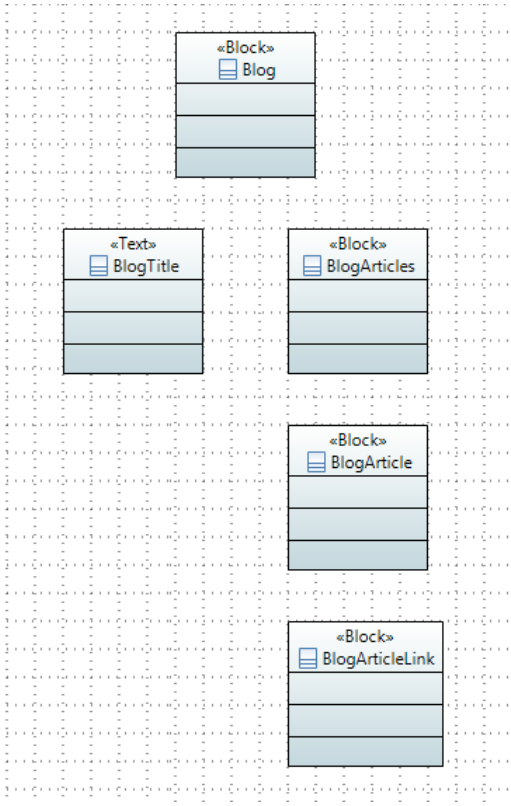
Fonte: Autor

Com as relações desenhadas, foi realizado a elaboração dos modelos que representam as estruturas de cada arquivo. Para demonstração, o arquivo **lista-de-artigos** do projeto **Orchard** foi representado na figura 17, possui o seguinte código-fonte:

```
1  .blog {
2          background-image: url(../image/texture.jpg);
3  }
4  .blog-title {
5          color: rgb(255, 170, 130);
6  }
7  .blog-articles {
8          align-items: stretch;
9          justify-content: space-between;
10 }
11 .blog-article {
12         margin-top: 25px;
13         border: 5px solid #fff;
14         background-position: center;
15         background-size: cover;
16         color: #fff;
17 }
18 .blog-article-link {
```

```
19          width: 100%;
20          height: 100%;
21          box-sizing: border-box;
22          padding: 30px;
23          display: block;
24          background-color: rgba(18, 18, 18, .8);
25          text-align: center;
26  }
27  @media all and (min-width: 600px) {
28          .blog-article {
29                  width: 32%;
30          }
31  }
```

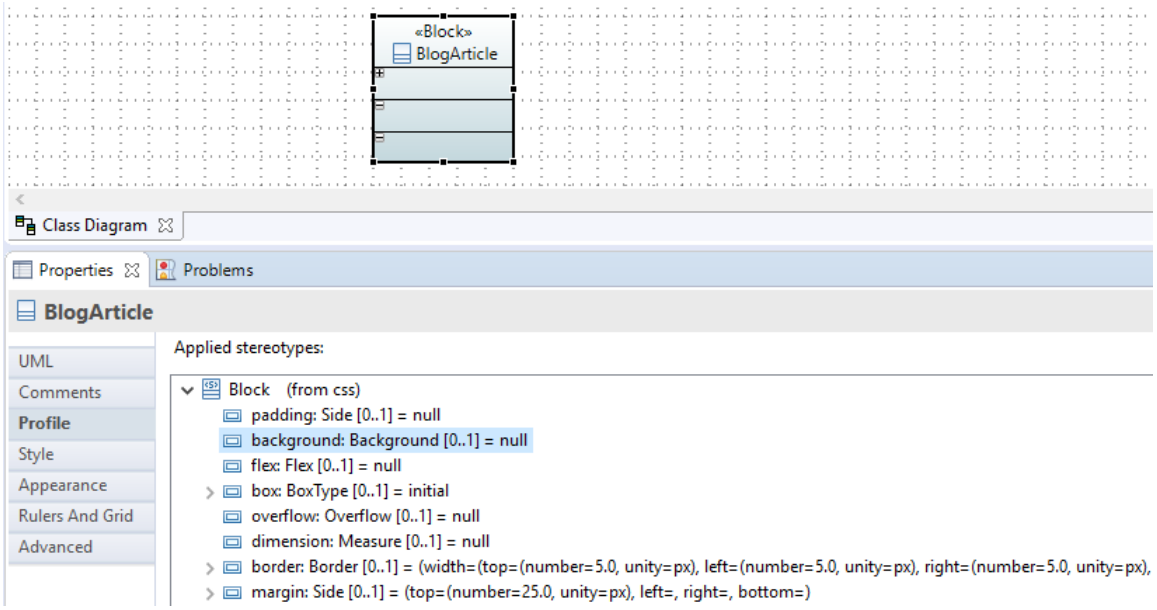
Figura 17 – Modelo do arquivo de layout lista-de-artigos



Fonte: Autor

O modelo da figura 17 aplica o metamodelo de CSS, elaborado no Passo 01, através do uso de perfis que podem ser administrados na aba **profile** da ferramenta Papyrus ao selecionar um elemento, conforme demonstra a figura 18.

Figura 18 – Demonstração do uso de profile



Fonte: Autor

Todos os arquivos de modelagem que utilizam perfis possuem uma complexidade em apresentar as informações dos diagramas, mas podem ser consultados em **https://github.com/duransbo/TCC** e encontram-se no Apêndice A, nas Figuras 21, 22, 23, 24, 25 e 26.

Após todos os arquivos representados em formado de modelo no Papyrus, é possível utilizar os arquivos em formato XML que a ferramenta utiliza para salvar as informações. Essa mudança do estado das informações de UML para XML será a primeira transformação (T1) do processo MDD, conforme mostrado na figura 8. Esses arquivos XML serão utilizados na entrada para o processamento dos algoritmos de transformação descritos no Passo 02.

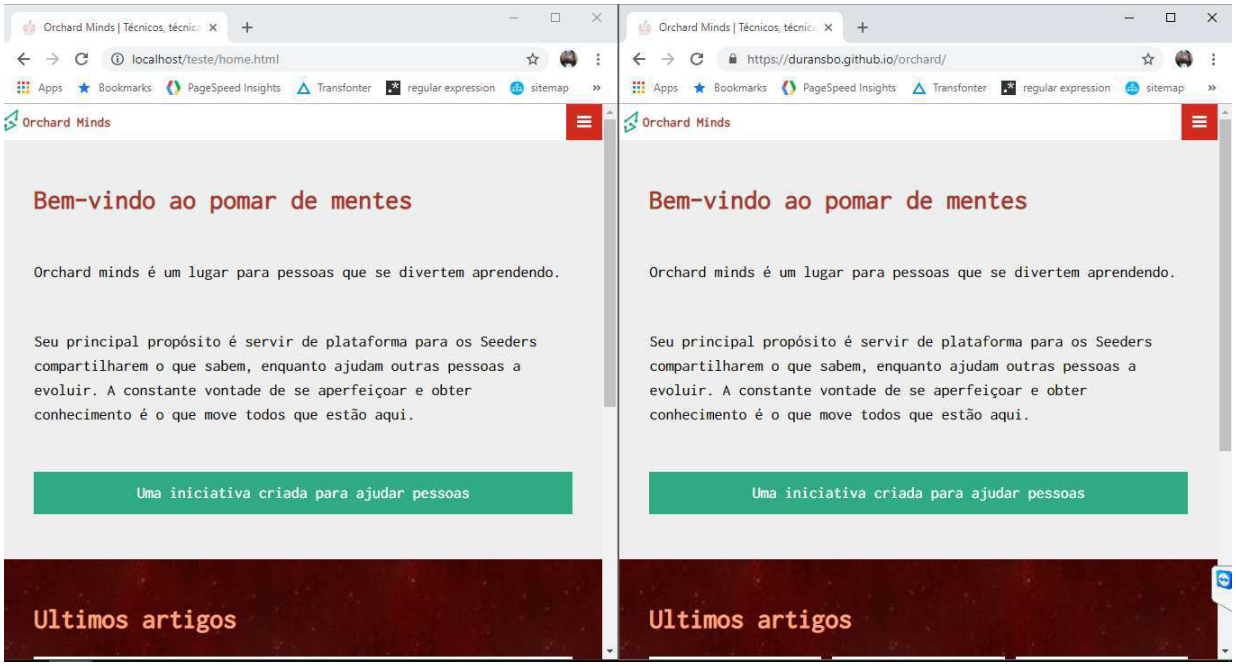
A estrutura HTML do projeto Orchard utilizou uma árvore de *layout* representada na figura 19 a esquerda. A estrutura HTML do experimento prático que utiliza os arquivos gerados pelo processo MDD está representada na figura 19 a direita. Pode-se observar que foi possível implementar uma estrutura muito similar ao projeto original, obtendo o resultado desejado. As classes em vermelho são do arquivo **reset**, as verdes do **template**, azuis do **header**, preta do **footer** e as roxas do arquivo **home**.

Figura 19 – Arvore das classes CSS

Fonte: Autor

Outra prova de sucesso no experimento prático está na comparação lado a lado na figura 20 entre a página **home** do Projeto Orchard e a **home** do experimento prático gerado pelo processo MDD.

Figura 20 – Comparação entre o projeto Orchard e o experimento pratico



Fonte: Autor

4.2 PASSO 5: ANÁLISE QUALI-QUANTITATIVA DOS RESULTADOS OBTIDOS

Com os conceitos aplicados no experimento prático, é possível realizar análises qualitativas e quantitativas dos resultados obtidos na implementação do conceito MDD para geração de código-fonte de *layout* para aplicações web.

Analisando a elaboração do metamodelo e a aplicação nos modelos, há dificuldades ao utilizar a ferramenta Papyrus, mas todas foram possíveis de contornar. Ao conseguir interpretar os atributos nas *tag's* XML, nota-se que existe algumas palavras utilizadas que deve ser evitado o uso para facilitar o desenvolvimento de algoritmos recursivos. Para utilizar em uma única propriedade, múltiplos valores de um *Enumeration* que podem ser repetidos, exige a criação de um *DataType* que possua um atributo do tipo do *Enumeration*, para que esse *DataType* sejá utilizado como tipo da propriedade de múltiplos valores. Também não foi possível encontrar uma forma de duplicar um modelo para reaproveitar classes já criadas.

O metamodelo elaborado possibilitou a modelagem de todo o *layout* proposto para o experimento prático, possuindo grande parte das propriedades CSS. Porém, ainda há recursos importantes que são muito utilizados pela comunidade de desenvolvimento web. Alguns recursos utilizados no próprio projeto Orchard necessitaram de alteração para a implantação do *layout*, por exemplo, a relação entre classes no projeto Orchard se transformou em uma classe no código do experimento prático.

Um estudo focado na documentação CSS pode propor um metamodelo mais robusto do que o proposto nesse trabalho, mas o metamodelo elaborado atendeu os requisitos da proposta de forma satisfatória. Uma escolha de outra ferramenta para modelagem pode facilitar o desenvolvimento, apesar do Papyrus atender sua função, suas limitações deixam a elaboração de modelos mais complexo que o necessário.

Os algoritmos de transformação também podem evoluir, mas funcionaram quando testados no experimento prático. O algoritmo da segunda transformação reduz, em média, 80% do volume de dados em 63ms. Esse tempo é a diferença entre a requisição do inicio do algoritmo e o retorno após finalizado, quando todos os JSON foram gerados.

Após a definição das regras de transformação, não importa a quantidade de modelos elaborados. Essa redução melhora o processo MDD proposto pelo fato dos arquivos consumidos pela terceira transformação serem mínimos, impactando em performance e produtividade.

Depois de desenvolvido o algoritmo da terceira transformação, foi notado a forte ligação com o metamodelo proposto que ajudaria muito no desenvolvimento do algoritmo se descoberto antes. A terceira transformação gera códigos-fonte CSS funcionais a partir

de arquivos pequenos gerados pela segunda transformação.

A comparação entre o acesso ao site estático que possui o *layout* proposto, o projeto que utiliza o preprocessamento da plataforma Hexo e o experimento prático que preprocessa em PHP, demonstra que praticamente não há perda na performance com o uso dessa abordagem. Essa comparação foi realizada utilizando a ferramenta DevTools do Google Chrome versão 70.0.3538.77, desabilitando a função de cache e simulando uma conexão 3G lenta, em um notebook Lenovo com Windows 10 x64, processador Intel Core I5-5200U e 4GB de memoria RAM.

A Tabela 2 exibe o tamanho e tempo necessário para carregamento após requisição de três arquivos estáticos, sendo que o arquivo **index.php** contém a estrutura HTML, o o arquivo **template.css** possui todas as estruturas compartilhadas por várias *layout's* e o arquivo **home.css** possui somente as estruturas únicas da página. O DOM seria o total necessário para todos esses arquivos estarem prontos para uso do navegador, pois existe uma ordem de carregamento em que os arquivos de estilo começam a carregar depois do arquivo HTML.

Tabela 2 – Servidor JS com site estático

Arquivo	**Tamanho**	**Tempo de carregamento**
index.php	3,2KB	2,38s
template.css	3,5KB	2,16s
home.css	1,2KB	2,04s
DOM	7,9KB	4,76s

Fonte: Autor

A Tabela 3 exibe o as mesmas informações, mas as requisições não são realizadas para arquivos estáticos. A plataforma Hexo funciona utilizando NodeJS e faz um pré-processamento com a linguagem JavaScript que transforma os arquivos de estilo escritos em Stylus para CSS. Esse pré-processamento retorna arquivos dinâmicos, da mesma forma que a T3, com impacto quase imperceptível para o usuário.

Tabela 3 – Servidor JS com pré-processamento Hexo

Arquivo	**Tamanho**	**Tempo de carregamento**
index.php	3,1KB	2,39s
template.css	3,4KB	2,07s
home.css	1,1KB	2,31s
DOM	7,6KB	4,80s

Fonte: Autor

A Tabela 4 exibe essas informações das requisições pré-processadas pelo servidor NGINX na versão 1.14 com PHP 7.2.11. Esse pré-processamento realiza a T3 da figura 8, consumindo os arquivos JSON gerados pela T2 e respondendo código-fonte CSS. A

redução do tamanho dos arquivos se deve ao fato do algoritmo descartar informações uteis somente para desenvolvimento, como espaços e indentação. Existe funções de minificação que podem ser aplicadas no pré-processamento do Hexo, mas o tempo de carregamento mostra que é irrelevante para o contexto testado, apesar de ainda ser benéfico para o usuário possuir menos dados trafegados.

Tabela 4 – Servidor PHP com pré-processamento T3

Arquivo	**Tamanho**	**Tempo de carregamento**
index.php	3,2KB	2,39s
template.css	2,4KB	2,10s
home.css	1,0KB	2,04s
DOM	6,6KB	4,80s

Fonte: Autor

Logo, vale destacar que sem perda de performance, a abordagem MDD permite um aumento na produtividade, pois a modelagem pode ser elaborada em paralelo com a regras de transformação, produzindo um resultado viável rapidamente e minimiza capacitação técnica para o desenvolvimento dos modelos, já que todas as regras foram tratadas previamente no metamodelo que precisa somente ser aplicado.

O resultado obtido no experimento prático foi como planejado, já que todas as características de *layout* puderam ser implementadas, utilizando a abordagem MDD por completo, buscando otimização em todos os passos realizados.

5 CONCLUSÕES

Conforme apresentado, o objetivo desta pesquisa era apresentar a implementação do conceito de MDD para geração de códigos-fonte de *layout* para aplicações web.

Após a realização de uma pesquisa aprofundada para atingir o objetivo definido, aplicando conceitos levantados na revisão bibliográfica e estudos de trabalhos relacionados a esta pesquisa, foi possível propor uma metodologia para a realização da pesquisa, possuindo 5 passos, sendo eles:

1. Elaboração do metamodelo para representar elementos de *layout* de aplicações web;
2. Desenvolvimento dos algoritmos de transformação de modelos em código-fonte;
3. Definição de um experimento prático para elaboração de modelos aplicando o metamodelo;
4. Aplicação do metamodelo e algoritmos de transformação no experimento prático;
5. Análise quali-quantitativa dos resultados obtidos;

O uso da ferramenta Papyrus para modelagem trouxe algumas dificuldades com relação a palavras reservadas para o XML, exigindo atenção no desenvolvimento do metamodelo, o uso de múltiplos valores do tipo de um *Enumeration* e reaproveitamento de modelos produzidos. Porém, todas as dificuldades puderam ser contornadas e o objetivo foi atingido.

O metamodelo proposto não atende alguns recursos CSS, como *media queries*, relação de parentesco entre elementos e seleção por atributo. Um estudo focado na documentação CSS pode propor um metamodelo mais robusto, mas o metamodelo elaborado atendeu os requisitos da proposta desse trabalho de forma satisfatória, possibilitando a elaboração de todos os modelos para representar os *layout's* do experimento prático definido.

O desenvolvimento dos algoritmos de transformação utilizando a linguagem PHP que permite uma abordagem multiparadigma proporciona uma flexibilidade ao código. Os algoritmos da transformação T2 e T3 podem ser tratados de forma independente, sendo constado o algoritmo da T3, desenvolvido de forma procedural, pode ter uma organização orientada a objeto, produzido com base no metamodelo proposto.

A abordagem de realizar uma limpeza no arquivo XML, Modelo Independente de Plataforma, transformando para JSON na segunda transformação do processo MDD (T2),

resultou em uma redução de 85% do tamanho em *bytes*. O experimento prático demonstrou que o impacto ao usuário é imperceptível, consumindo os JSON's, Modelo Especifico de Plataforma, em tempo de execução pelo servidor para realizar a terceira transformação (T3) que retorna código-fonte.

O uso da abordagem MDD permitiu um aumento na produtividade, pois a modelagem pode ser elaborada em paralelo com a regras de transformação, produzindo um resultado viável rapidamente e minimiza capacitação técnica para o desenvolvimento dos modelos, já que todas as regras foram tratadas previamente no metamodelo que precisa somente ser aplicado.

Sendo assim, conclui-se que nesta pesquisa foi possível destacar os ganhos de produtividade e usabilidade no desenvolvimento de aplicações web com a implementação do conceito de MDD. Obteve-se sucesso ao propor um metamodelo que representa *layout's*, desenvolveu-se algoritmos que transformaram os modelos elaborados em código-fonte e os aplicou em um experimento prático, atingindo o objetivo esperado de obter arquivos de estilo funcionais.

Como proposta de melhoria ao trabalho apresentado, a realização de um estudo focado na documentação CSS de uma fonte mais presente em implementação, como a documentação Mozilla, para propor um metamodelo mais robusto que atenda os recursos CSS que não foram abordados nessa pesquisa.

Esse trabalho focado no CSS serve de inspiração para aplicar a metodologia para outros focos que podem ser interligados, como o HTML e o JS, gerando códigos-fonte da camada *Front-End* inteira. Sendo possível também, realizar uma analise dos algoritmos desenvolvidos para melhorar sua arquitetura e escrita ou até uma comparação com contras linguagens que tenham a possibilidade de trabalhar com arquivos para determina melhores implementações da proposta MDD.

REFERÊNCIAS

ALLSOPP, J. A dao of web design. *A List Apart*, v. 58, 2000. Citado 2 vezes nas páginas 19 e 24.

ALVES, E. C. Desenvolvimento de um sistema de chamados utilizando asp. net mvc. Niterói, 2017. Citado na página 27.

ALVES, E. L.; MACHADO, P. D.; RAMALHO, F. Uma abordagem integrada para desenvolvimento e teste dirigido por modelos. In: SN. *2nd Brazilian Workshop on Systematic and Automated Software Testing*. [S.l.], 2008. Citado 2 vezes nas páginas 20 e 29.

ANICHE, M. F.; GEROSA, M. A. Boas e más práticas no desenvolvimento web com mvc: Resultados de um questionário com profissionais. 2015. Citado 2 vezes nas páginas 26 e 28.

BALDUÍNO, P. *Dominando JavaScript com jQuery*. [S.l.]: Editora Casa do Código, 2014. Citado na página 28.

BATTELLE, J. *The search: How Google and its rivals rewrote the rules of business and transformed our culture*. [S.l.]: Nicholas Brealey Publishing, 2011. Citado na página 23.

BERNERS-LEE, T. How it all started. *Documento institucional da W3C publicado em*, 2004. Citado 2 vezes nas páginas 19 e 24.

BEZERRA, E. *Princípios de Análise e Projeto de Sistema com UML*. [S.l.]: Elsevier Brasil, 2017. v. 3. Citado na página 29.

BOHLEN, M. *AndroMDA Model Driven Architecture Framework – AndroMDA - Homepage*. 2018. Disponível em: <http://andromda.sourceforge.net/>. Citado na página 32.

CAKEPHP - Build fast, grow solid | PHP Framework | Home. 2018. Disponível em: <https://cakephp.org/>. Citado na página 28.

CALDEIRA, C. *PostgreSQL: Guia Fundamental*. [S.l.]: Edições Sílabo, 2015. Citado na página 26.

CIRILO, C. E. et al. *Model Driven RichUbi-Processo Dirigido a Modelos para a Construção de Interfaces Ricas de Aplicações Ubíquas Sensíveis ao Contexto*. Tese (Doutorado) — Dissertação de Mestrado, Universidade Federal de São Carlos–UFSC, São Carlos–SP, 2011. Citado 2 vezes nas páginas 19 e 24.

CODEIGNITER Rocks. 2018. Disponível em: <https://codeigniter.com/>. Citado na página 28.

COELHO, A. C. A. Web design responsivo: Melhorando interfaces e a experiência do usuário na navegação web. *REVER*, v. 1, n. 1, p. 07–24, 2016. Citado na página 24.

COSTA, P. H. T.; CANEDO, E. D. Using a mdd approach to develop software systems. In: IEEE. *Information Systems and Technologies (CISTI), 2015 10th Iberian Conference on*. [S.l.], 2015. p. 1–6. Citado 3 vezes nas páginas 32, 33 e 34.

DALL'OGLIO, P. *PHP Programando com Orientação a Objetos 3ª Edição*. [S.l.]: Novatec Editora, 2015. Citado 3 vezes nas páginas 20, 27 e 43.

DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier Brasil, 2004. Citado na página 26.

EIS, D. *Guia Front-End: O caminho das pedras para ser um dev Front-End*. [S.l.]: Editora Casa do Código, 2015. Citado 2 vezes nas páginas 25 e 26.

EIS, D.; FERREIRA, E. *HTML5 e CSS3 com farinha e pimenta*. [S.l.]: Lulu. com, 2012. Citado na página 26.

FINK, G. et al. *Pro Single Page Application Development: Using Backbone. Js and ASP. Net*. [S.l.]: Apress, 2014. Citado na página 24.

FRANCO, E.; PIEDADE, M.; RÊGO, R. Protótipo de um framework mvc para aplicações php de pequeno porte. *Anais do Encontro Regional de Computação e Sistemas de Informação*, 2014. Citado 3 vezes nas páginas 26, 27 e 28.

HALES, W. *HTML5 and JavaScript Web Apps: Bridging the Gap Between the Web and the Mobile Web*. [S.l.]: "O'Reilly Media, Inc.", 2012. Citado na página 25.

INC, M. *MagicDraw*. 2018. Disponível em: <https://www.nomagic.com/products/magicdraw>. Citado 2 vezes nas páginas 32 e 33.

INTERNET Live Stats. 2018. Disponível em: <http://www.internetlivestats.com/>. Citado na página 23.

LIMA, M. A. d. O. Um projeto de interface para ferramentas de modelagem uml baseadas em eclipse. Universidade Federal do Pampa, 2017. Citado 2 vezes nas páginas 33 e 34.

LUCRÉDIO, D. Uma abordagem orientada a modelos para reutilização de software. *INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO UNIVERSIDADE DE SÃO PAULO*, p. 37, 2009. Citado 3 vezes nas páginas 30, 32 e 33.

MARCOTTE, E. Responsive web design. *A List Apart*, v. 306, 2010. Citado na página 24.

MARTINS, M. A. T. *Seleção de uma framework MVC client-side para uma aplicação web e mobile*. Dissertação (Mestrado) — Universidade de Aveiro, 2014. Citado 2 vezes nas páginas 27 e 28.

MCARTHUR, K. *Pro PHP: Patterns, Frameworks, Testing and More*. [S.l.]: Apress, 2008. Citado na página 27.

MINETTO, E. L. Frameworks para desenvolvimento em php. *São Paulo: Novatec*, 2007. Citado 3 vezes nas páginas 19, 20 e 27.

NETO, J. D. d. O. et al. Análise comparativa de desempenho de aplicação android com persistência em banco de dados relacional e banco de dados orientado a objetos. *Anais do Encontro de Computação do Oeste Potiguar ECOP/UFERSA (ISSN 2526-7574)*, v. 1, 2017. Citado na página 26.

OSMANI, A. *Learning JavaScript Design Patterns: A JavaScript and jQuery Developer's Guide*. [S.l.]: "O'Reilly Media, Inc.", 2012. Citado na página 26.

OTWELL, T. *Love beautiful code? We do too.* 2018. Disponível em: <https://laravel.com/>. Citado na página 28.

PAIVA, M. S. d. *Técnicas e tecnologias para desenvolvimento de Front-End na empresa PontoPR-Inovação Digital*. Dissertação (Mestrado), 2018. Citado 2 vezes nas páginas 25 e 26.

PAPYRUS Modeling environment. 2018. Disponível em: <https://www.eclipse.org/papyrus/>. Citado 2 vezes nas páginas 32 e 33.

PILGRIM, M. *HTML5: Up and Running: Dive into the Future of Web Development*. [S.l.]: "O'Reilly Media, Inc.", 2010. Citado na página 25.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016. Citado 2 vezes nas páginas 28 e 29.

REENSKAUG, T. Mvc xerox parc 1978-79. *Trygve/MVC*, 1979. Citado na página 26.

REZENDE, I. C. O. et al. R2mdd: um framework para rastreabilidade e monitoramento de requisitos com foco no desenvolvimento dirigido a modelos. Universidade Federal de Sergipe, 2016. Citado 3 vezes nas páginas 30, 31 e 35.

SENSIOLABS. *Symfony, High Performance PHP Framework for Web Development*. 2018. Disponível em: <https://symfony.com/>. Citado na página 28.

SILVA, A. D. A. P. D. Design responsivo: técnicas, frameworks e ferramentas. 2014. Citado 2 vezes nas páginas 19 e 24.

SILVA, L.; PIRES, D. F.; NETO, S. C. Desenvolvimento de aplicações para dispositivos móveis: tipos e exemplo de aplicação na plataforma ios. *Franca/SP*, 2015. Citado na página 25.

SILVA, M. S. da; FERRARI, F. C. Integração de frameworks front-end para desenvolvimento de interfaces ricas com javaserver faces. *Revista TIS*, v. 4, n. 1, 2016. Citado 4 vezes nas páginas 25, 27, 34 e 37.

SOFTWARE Público Brasileiro. 2018. Disponível em: <https://softwarepublico.gov.br/social/mdarte>. Citado 2 vezes nas páginas 32 e 33.

SOUZA, P. M. R. de. *UM ESTUDO SOBRE PADRÕES E TECNOLOGIAS PARA O DESENVOLVIMENTO WEB-FRONT-END*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2016. Citado 3 vezes nas páginas 28, 33 e 34.

STATCOUNTER Global Stats. 2018. Disponível em: <http://gs.statcounter.com/>. Citado na página 24.

SULLIVAN, D. Google: 100 billion searches per month, search to integrate gmail, launching enhanced search app for ios. *Search Engine Land*, 2012. Citado 2 vezes nas páginas 19 e 23.

TELES, Í. P. et al. Bpm4services: framework dirigido a modelos para automação de processos de negócios. Universidade Federal de Sergipe, 2017. Citado 4 vezes nas páginas 29, 30, 31 e 35.

W3C, W. W. W. C. *Acesso em*, 2018. Citado 3 vezes nas páginas 19, 25 e 26.

ZEND the PHP Company. 2018. Disponível em: <http://www.zend.com/>. Citado na página 28.

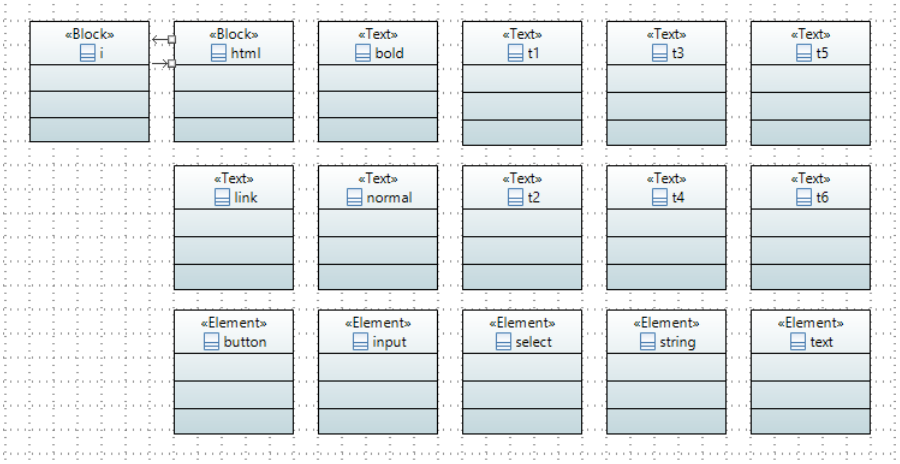
Apêndices

APÊNDICE A – APRESENTAÇÃO DOS MODELOS ELABORADOS APLICANDO O METAMODELO

Este apêndice possui todos os outros modelos elaborados que não foram apresentados no capítulo de resultados. Esses modelos representam arquivos de *layout* do Projeto Orchard que devem ser convertidos aplicando a abordagem MDD com o objetivo de obter código-fonte funcional.

A figura 21 representa o arquivo **reset** que possui a formatação das estruturas básicas do HTML para padronização inicial de *layout* em todos os navegadores.

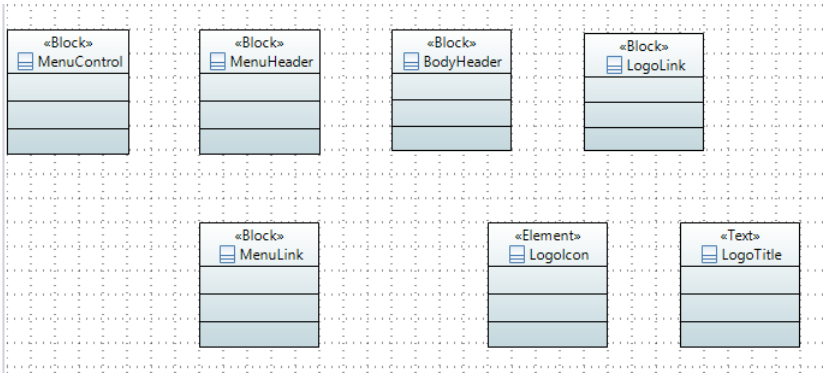
Figura 21 – Modelo do arquivo de layout reset



Fonte: Autor

A figura 22 representa o arquivo **header**, possuindo as estruturas que compõem o cabeçalho da página, como logotipo e menu de navegação.

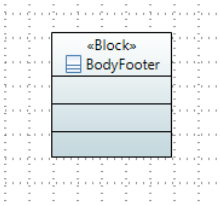
Figura 22 – Modelo do arquivo de layout header



Fonte: Autor

A figura 23 representa o arquivo **footer** que possui a estrutura de rodapé da página.

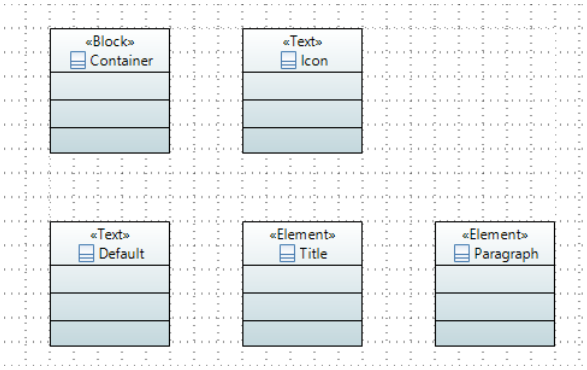
Figura 23 – Modelo do arquivo de layout footer



Fonte: Autor

A figura 24 representa o arquivo **template** que contém formatações utilizadas frequentemente pelas páginas, independente de localização, como ícones.

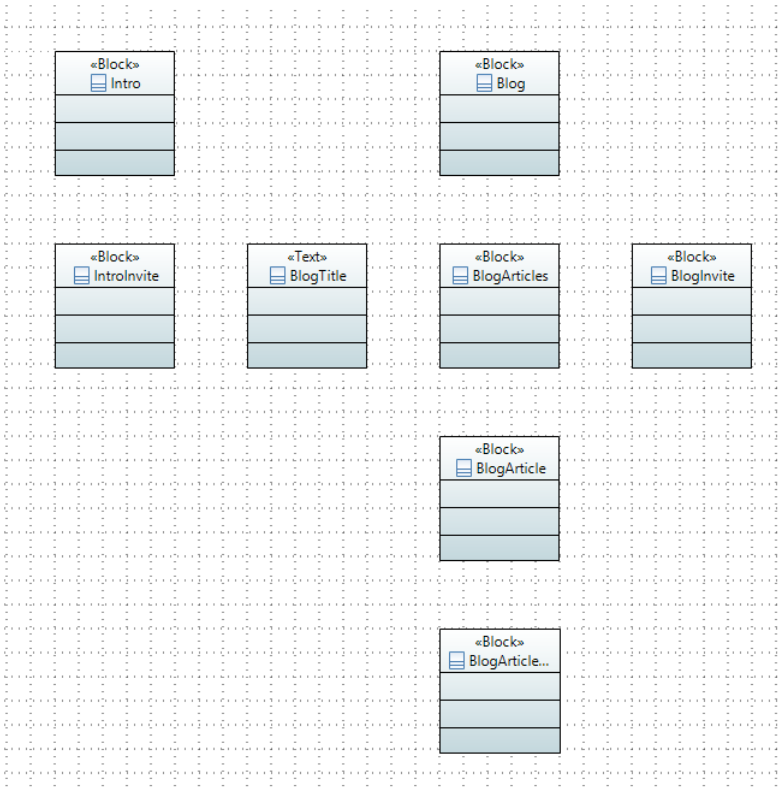
Figura 24 – Modelo do arquivo de layout template



Fonte: Autor

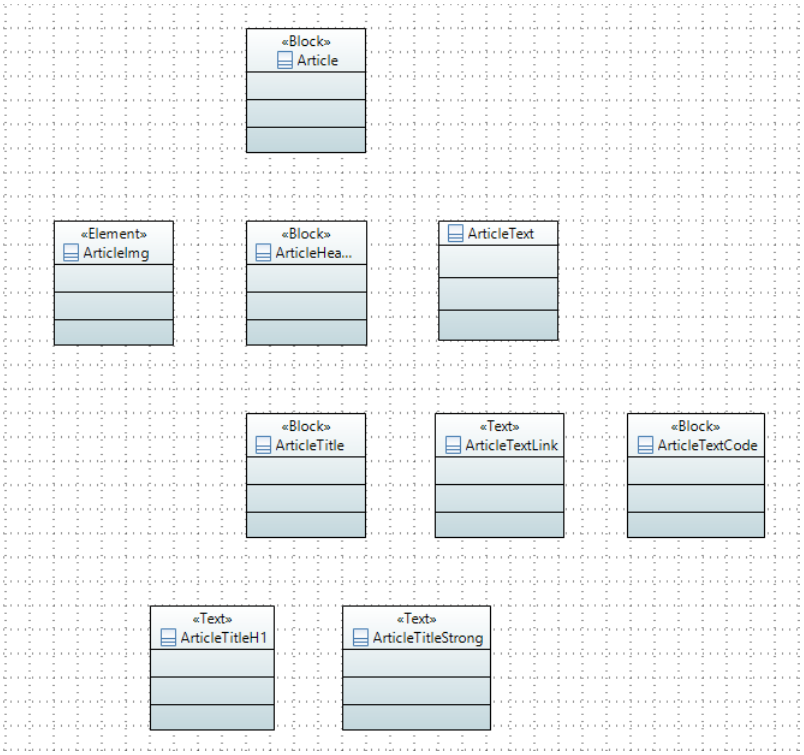
As figuras 25 e 26 representam, respectivamente, os arquivos **home** e **artigo**, contendo estruturas únicas do contexto da página.

Figura 25 – Modelo do arquivos de layout home



Fonte: Autor

Figura 26 – Modelo do arquivos de layout artigo



Fonte: Autor