

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Rafael Bruno Pessoa de Araújo

**Utilizando Modelos em Tempo de Execução para a Manutenção de
Conteúdo Educacional na Web**

Fortaleza – CE

28 de março de 2019

Rafael Bruno Pessoa de Araújo

Utilizando Modelos em Tempo de Execução para a Manutenção de Conteúdo Educacional na Web

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Linha de Pesquisa: Engenharia de Software

Orientador: Cidcley Teixeira de Souza

Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação (PRPI)

Programa de Pós-Graduação em Ciência da Computação (PPGCC)

Fortaleza – CE

28 de março de 2019

Dados Internacionais de Catalogação na Publicação

Instituto Federal do Ceará - IFCE

Sistema de Bibliotecas - SIBI

Ficha catalográfica elaborada pelo SIBI/IFCE, com os dados fornecidos pelo(a) autor(a)

A658u Araujo, Rafael Bruno Pessoa de.

Utilizando Modelos em Tempo de Execução para a Manutenção de Conteúdo Educacional na Web / Rafael Bruno Pessoa de Araujo. - 2019.

56 f. : il. color.

Dissertação (Mestrado) - Instituto Federal do Ceará, Mestrado em Ciência da Computação, Campus Fortaleza, 2019.

Orientação: Prof. Dr. Cidcley Teixeira de Souza.

1. Engenharia de Software. 2. Modelos em tempo de execução. 3. Linguagens Específicas de Domínio. 4. Arquitetura de software. 5. Sistemas auto adaptativos. I. Título.

CDD 004

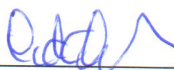
RAFAEL BRUNO PESSOA DE ARAÚJO

Utilizando Modelos em Tempo de Execução para a Manutenção de Conteúdo Educacional na Web

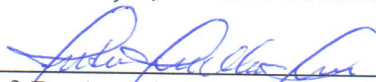
Dissertação submetida à Coordenação do Curso de Pós-graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação, área de concentração Ciência da Computação.

Aprovada em 28 / 03 / 2019

BANCA EXAMINADORA



Prof. Dr. Cidley Teixeira de Souza (Orientador)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - PPGCC/IFCE



Prof. Dr. Antônio Wendell de Oliveira Rodrigues
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - PPGCC/IFCE



Prof. Dr. Tales Paiva Nogueira
Universidade Federal do Ceará - UFC

*Este trabalho é dedicado a todas as pessoas que acharam que não conseguiriam,
mas que foram até o fim, lutaram, e conseguiram.*

Aos guerreiros, a glória.

Agradecimentos

Agradeço ao meu orientador, Prof. Dr. Cidcley, que acreditou neste trabalho mais que eu mesmo acreditei. Espero poder um dia influenciar outras pessoas a acreditarem mais nelas mesmas como ele fez comigo.

Agradeço aos meus outros professores e coordenadores do Programa de Pós Graduação em Ciência da Computação do Instituto Federal do Ceará, assim como aos colegas que ganhei durante o caminho, que também contribuíram para a minha formação.

Agradeço ao meu colega de graduação, Rodrigo Parente, por ter se dedicado várias horas comigo a estudar sobre o ANTLR e me incentivar a terminar este trabalho.

Agradeço à minha amada Vanessa Nobre, que pacientemente me aguentou durante esses anos, em especiais durante este mestrado, que tanto me consumiu e me desgastou mentalmente. Ela que foi o pivô e o incentivo para eu entrar na jornada do mestrado, e que também sempre acreditou em mim e que me deu forças para começar e terminar esta etapa da minha vida.

E por fim, agradeço aos meus pais, por sempre, sempre acreditarem em mim e me mostrarem que a educação é sempre o melhor caminho para a vida e que sempre serviram a mim como espelho.

*“Morrer ignorante sabendo que tinha capacidade de ter sido sábio,
isto sim, é uma tragédia humana.”
(William Shakespeare)*

Resumo

Apesar do grande número de ferramentas e serviços disponíveis para o suporte à autoria de conteúdos web, ainda existem diversas dificuldades na execução dessa tarefa. Esse cenário piora quando os usuários possuem pouco conhecimento em informática. Particularmente no caso de professores, ferramentas como os AVAs (Ambientes Virtuais de Aprendizagem) facilitam a interação do professor com as páginas nas quais os conteúdos devem ser apresentados. Contudo, é necessário que os mesmos ainda precisem de treinamento para usar essas ferramentas e, em grande parte dos casos, esses usuários precisam apenas realizar tarefas simples, como disponibilizar arquivo com slides ou um artigo para leitura da turma, sendo que a complexidade da interação com o AVA acaba sendo desproporcional à simplicidade da tarefa a ser realizada. O propósito desta pesquisa é apresentar uma nova arquitetura para dar suporte a aplicações de autoria web remota através de uma abordagem baseada em modelos de tempo de execução. A proposta é possibilitar que ações realizadas em pastas nos computadores dos usuários, tais como a criação de uma pasta, a inclusão de um arquivo com os slides de uma aula, dentre outras tarefas corriqueiras para quem usa computador, possam ser mapeadas dinamicamente em ações de manutenção em páginas web remotas. Para viabilizar a proposta, foi criado um middleware que utiliza modelos em tempo de execução e uma linguagem específica de domínio. O middleware foi capaz de oferecer um ambiente de autoria web em tempo de execução, permitindo um novo modo de fazer autoria web. Essa abordagem oferece um novo cenário para a manutenção e interação dos usuários entre domínios, máquina local e servidor, diferentes.

Palavras-chave: autoria web. arquitetura de software. ambientes virtuais de aprendizagem. modelos em tempo de execução.

Abstract

Despite the large number of tools and services available to support web content authoring, there are still several difficulties in performing this task. This scenario gets worse when users have limited computer knowledge. Particularly in the case of teachers, tools such as VLEs (Virtual Learning Environments) facilitate the interaction of the teacher with the pages in which the contents should be presented. However, they still need training to use these tools, and in most cases, these users need only to perform simple tasks, such as providing a slide file or an article for class reading, and the complexity of the interaction with the VLE ends up being disproportionate to the simplicity of the task being performed. The purpose of this research is to present a new architecture to support remote web authoring application through a models at runtime approach. The proposal is to enable actions taken in filesystem in the users computers, such as the creation of a folder, the inclusion of a file with the slides of a class, among other ordinary tasks for those who use a computer, to be dynamically mapped in actions of remote web pages. In order to make the proposal viable, a middleware was created that uses runtime models with a domain-specific language. The middleware was able to provide a web authoring environment at runtime, allowing a new way of doing web authoring. This approach provides a new scenario for the maintenance and interaction of users between different domains, local machine and server.

Keywords: web authoring. software architecture. virtual learning environments. runtime models.

Lista de ilustrações

Figura 1 – Árvore de leitura da gramática Hello	32
Figura 2 – Fluxo do ANTLR	33
Figura 3 – Arquitetura proposta. À esquerda há uma representação da estrutura de arquivos do professor. À direita há uma representação do sistema de arquivos no servidor web.	37
Figura 4 – (a) Pasta obedecendo à DSL no sistema de arquivos do professor; (b) Arquivo criado no lado do servidor, resultante da nomeação da pasta no sistema de arquivos do professor.	42
Figura 5 – Ilustração do funcionamento da DSL	44
Figura 6 – Exemplo do sistema de arquivos do professor de História	44
Figura 7 – Resultado no servidor web	45
Figura 8 – Página index. Lista no seu menu superior o nome das páginas, nomeadas de acordo com o nome da pasta que o professor nomeou em seu sistema de arquivos	46
Figura 9 – Página Historia-do-Brasil.jsp. Neste caso, como só existe apenas um arquivo de texto contendo a história do Brasil, este é gerado como um texto dentro da página Historia-do-Brasil.jsp	46
Figura 10 – Página Historia-do-Ceara.jsp. Neste caso, como existe um arquivo pptx e uma imagem, são gerados, respectivamente, um link para download do arquivo e a imagem é renderizada dentro de um objeto img na página Historia-do-Ceara.jsp. O nome da imagem vira automaticamente a legenda da imagem.	47

Lista de tabelas

Tabela 1 – Tipo do token e seu valor correspondente	33
Tabela 2 – Comandos da DSL	39

Lista de abreviaturas e siglas

AVA	Ambientes Virtuais de Aprendizagem
GUI	<i>Graphical User Interface</i>
PC	Computador Pessoal
HTML	<i>HyperText Markup Language</i>
CSS	<i>Cascading Style Sheets</i>
CMS	Sistema de Gerenciamento de Conteúdo
IaaS	Infraestrutura como um Serviço
MDE	Model-Driven Engineering

Sumário

1	INTRODUÇÃO	21
1.1	Motivação para a Dissertação	22
1.2	Objetivos Geral e Específicos	23
1.3	Produção científica	23
1.4	Estrutura da Dissertação	23
2	FUNDAMENTAÇÃO TEÓRICA	25
2.1	Autoria web	25
2.2	Sistemas auto adaptativos	26
2.2.1	Sistemas self-healing	26
2.3	Models@runtime	28
2.3.1	O que são models@runtime	28
2.3.2	Porque modelos em tempo de execução	29
2.3.3	O que constitui um modelo em tempo de execução	30
2.4	ANTLR	31
2.4.1	Parser	32
3	TRABALHOS RELACIONADOS	35
3.1	Características das ferramentas	35
3.1.1	Ferramentas de construção de páginas web	36
3.1.2	AVAs	36
3.1.3	Models at runtime	36
4	AUTORIA WEB EM TEMPO DE EXECUÇÃO	37
4.1	Componentes da arquitetura	38
4.1.1	Metamodelo	38
4.1.2	Modelo	38
4.1.3	Monitor de ambiente	38
4.1.4	Atualizador	38
4.1.5	Base de regras	38
4.1.6	Comunicador	39
4.1.7	Middleware	39
4.2	Linguagem específica de domínio	39
5	ESTUDO DE CASO	41
5.1	Exemplo: cenário do professor de História	41

5.1.1	Sistema de arquivos do professor	44
5.1.2	Sistema de arquivos no servidor web	44
5.1.3	Resultado do cenário	45
6	CONCLUSÃO	49
	REFERÊNCIAS	51

1 Introdução

A interface gráfica, ou GUI (*Graphical User Interface*), foi uma ideia desenvolvida pela Xerox Corporation, mas somente transformada em produto pela Apple. O seu lançamento nos anos 80 foi um sucesso. O motivo de tanta euforia na época é bem claro: a interface gráfica é muito mais amigável, simples e natural do que a interação por linha de comando, que exigia um treinamento bem mais avançado.

Com o advento da internet e da popularização do computador pessoal (PC) nos anos 90, tornou-se comum e até necessário o compartilhamento dos mais diversos tipos de dados e informações na rede mundial de computadores. Muitas ferramentas foram desenvolvidas desde então para tal propósito. No entanto, cada ferramenta tem sua particularidade, regras e funcionamento diferente, o que faz com que o usuário, em especial o leigo em informática, tenha grandes dificuldades de instalação, manuseio e operação. Mesmo tarefas simples, como criar uma página na web e disponibilizar documentos utilizando serviços, produtos ou plataformas que prometem essa funcionalidade de modo fácil pode ser algo complexo para o usuário leigo. Para piorar, muitas vezes além do conhecimento necessário para a autoria web, também é comum em instituições a comunicação ficar centrada em um setor ou pessoa responsável, sendo necessário obedecer uma fila de espera para a inserção de novos conteúdos, criando um gargalo no objetivo final do usuário.

A web 2.0 foi o segundo passo da evolução da internet. Este marco atraiu e atrai até hoje a atenção de profissionais de informática, empresários e usuários. Esta tecnologia permitiu uma internet bem mais interativa e colaborativa, enfatizando principalmente a inteligência coletiva (MURUGESAN, 2007).

Uma técnica bastante popular utilizada no desenvolvimento de sistemas auto-adaptativos é a construção arquitetônica ou comportamental de modelos em tempo de projeto para serem atualizados durante a execução do sistema. Tais modelos são conhecidos como *models@runtime* (BLAIR; BENCOMO; FRANCE, 2009). Segundo (BENCOMO et al., 2012), *models@runtime* também pode ser entendido como uma abstração de um sistema que pode ser manipulado em tempo de execução para um propósito específico.

Dentre os vários objetivos dos modelos em tempo de execução (SZVETITS; ZDUN, 2016), dois podem ser destacados: a adaptação (1) e o monitoramento, simulação e predição (2). A adaptação permite que o sistema seja modificado de acordo com as mudanças ocorridas no ambiente. O monitoramento, simulação e a predição sugerem que o sistema sejam monitorados por meio de modelos de forma a possibilitar o rastreamento do seu comportamento.

Em universidades e escolas, existem várias ferramentas, serviços e plataformas sob

demanda que oferecem ao professor uma solução para criar e manter páginas na web (AVISON D., 2003). Apesar disso, há de se compreender que mesmo com a existência de ferramentas, serviços e produtos, ainda existem muitos profissionais que não tem o conhecimento mínimo em informática para utilizá-las, ou, por questões burocráticas, não tem acesso a elas, pois sua utilização depende da aceitação em sua organização. Além disso, para publicar o seu próprio conteúdo, muitas vezes nestas organizações ou instituições existe uma central de comunicação ou pessoa responsável, sendo necessário obedecer uma fila de espera.

Pensando nestes problemas, esta dissertação propõe uma nova abordagem para que o professor, mesmo o leigo em informática, possa facilmente publicar e alimentar conteúdos na web. Para isso, foi criado uma arquitetura genérica, independente do domínio da aplicação, utilizando DSL (linguagens específicas de domínio) e modelos em tempo de execução.

Explicando de maneira simplificada, este artigo apresenta um estudo de caso (capítulo 5) onde um usuário pode subir um website com textos e mídias (imagens, slides, áudios e outros arquivos) utilizando regras de nomeação de pastas e arquivos através da DSL e da arquitetura proposta por este trabalho.

1.1 Motivação para a Dissertação

Inspirado pela vida acadêmica e observando o cotidiano dos professores, é de se admirar que estes profissionais, mesmo com tantas tarefas no seu dia-a-dia, tenham que se preocupar ainda com tarefas operacionais, que deveriam ser mais simples.

A disponibilização de materiais dos professores para seus alunos, assim como o contrário, é algo bastante comum na rotina de quem vivencia o mundo acadêmico. Apesar de existirem excelentes ferramentas hoje em dia, vemos que o comportamento operacional delas não funciona de maneira favorável: as mídias estão, geralmente, no computador pessoal do professor, no entanto este precisa pegar tais arquivos e fazer um *upload* para um servidor (seja via correio eletrônico, sites ou qualquer outro serviço). Vale também ressaltar que nem todos os professores possuem facilidade com informática. O ideal é que uma linguagem fácil de nomeação de pastas e arquivos fosse o suficiente para que qualquer professor que tivesse o mínimo de conhecimento em informática pudesse facilitar a publicação de material na internet, ganhando mais tempo para os professores e também diminuindo as barreiras. Apesar do estudo de caso deste projeto focar no ambiente acadêmico, a arquitetura proposta por este estudo pode ser generalizada para qualquer área.

Motivado pela situação descrita acima, e com os benefícios apresentados em estudos sobre modelos em tempo de execução (ver seção 2.3 no capítulo 2), este trabalho surgiu

como o primeiro estudo, até o momento, a apresentar uma arquitetura inter-adaptativa, onde uma ação em um modelo causa uma ação em outro modelo.

1.2 Objetivos Geral e Específicos

O objetivo geral deste pesquisa é apresentar uma arquitetura baseada em modelos em tempo de execução para dar suporte à manutenção de páginas web remotas através de ações de manipulação no sistema de arquivos dos usuários.

Em seu sentido mais particular, os seguintes objetivos específicos são:

- Desenvolver o conceito de arquitetura inter-adaptativa no qual ações realizadas em um domínio de aplicação possam ser mapeadas em ações em outro domínio,
- Apresentar um estudo de caso do cenário cotidiano;

1.3 Produção científica

Durante a realização desta pesquisa o seguintes trabalho científico foi aceito para publicação:

- **Rafael Bruno Pessoa de Araújo, Cidcley Teixeira de Souza, 2018. Using Runtime Models in Maintaining Educational Content on the Web.** ICALT 2019: International Conference on Advanced Learning Technologies, Jul 18-19, 2019 in Paris, France;

1.4 Estrutura da Dissertação

Esta dissertação está organizada na seguinte maneira: O capítulo 2 apresenta conceitos básicos sobre autoria web, conceitos fundamentais sobre modelos em tempo de execução e também o funcionamento do ANTLR. O capítulo 3 contém trabalhos relacionados a esta pesquisa. No capítulo 4 será apresentado a abordagem proposta. No capítulo 5 será apresentado o estudo de caso e para finalizar, no capítulo 6, há uma conclusão sobre esta dissertação e sugestões sobre melhorias e ideias futuras.

2 Fundamentação Teórica

2.1 Autoria web

HTML é uma linguagem de marcação e é a base das páginas da web. É o HTML que é renderizado pelos navegadores e pode ser utilizado em conjunto com uma ou mais linguagens de programação. Uma página da web pode conter texto, imagens, animações, vídeos, scripts e muito mais.

A criação na web permite que um indivíduo ou organização se manifeste na Internet. Para fazer isso, você precisa criar uma página da web e alimentá-la com conteúdo e mídia (LEWIS, 2006).

Com uma ampla gama de opções, o tipo, preço, instalação, operação e usabilidade de ferramentas, serviços ou plataformas variam muito. Existem ferramentas de Sistema de Gerenciamento de Conteúdo (CMS), como WordPress¹, Refinery², Joomla³, Drupal⁴ e muitas outras, além de blogs que permitem a publicação de conteúdos, como o Medium⁵.

Há também ferramentas de desenvolvimento e tecnologias para pessoas com habilidades técnicas para desenvolver seu próprio ambiente usando linguagens de programação de uso geral (Java, Python, Ruby, entre outras), linguagens de marcação (HTML / CSS) e uso de sua própria infra-estrutura (ABDULLAH M. S., 2002).

No entanto, todo esse processo de criação da web ainda requer algum conhecimento, mesmo se o usuário estiver usando um serviço de terceiros que automatize boa parte do processo. Desta forma, os usuários devem manipular ferramentas que permitam a geração de páginas, um fato que muitas vezes está longe da realidade de muitos. Assim, usuários sem conhecimento técnico podem utilizar os CMS's para ajudar, mas mesmo assim, é necessário entender o funcionamento dessas ferramentas.

Este trabalho argumenta que, em vários casos, a necessidade de utilizar páginas da web limita-se a permitir a comunicação em larga escala de informações de forma ágil. Assim, professores que querem disponibilizar materiais de aula, profissionais liberais que desejam fornecer informações de contato e portfólios de serviços, estudantes que desejam submeter seus trabalhos, entre outros, acabam encontrando nas ferramentas atuais um conjunto de opções que acabam desestimulando o seu uso.

¹ <https://wordpress.com>

² <http://refinerycms.com>

³ <https://www.joomla.org>

⁴ <https://www.drupal.org>

⁵ <https://medium.com>

Na seção 2.2 será apresentada uma das técnicas modernas de geração de código em tempo real.

2.2 Sistemas auto adaptativos

2.2.1 Sistemas self-healing

Inspirada pela biologia, o *self-healing* (ou autocura) evoluiu como uma propriedade dos sistemas de tecnologia que permite a esses sistemas diagnosticarem-se e reagirem a falhas, de modo que um modo satisfatório de operação seja restaurado após a ocorrência de uma falha ou falhas. A autocura não se limita a uma determinada classe de sistemas de TI, mas abrange o amplo leque de computação, de aplicativos sobre software de sistema a redes, de pequenos dispositivos incorporados em computadores pessoais a GRIDs. Atualmente, a autocura é uma palavra de ordem. Desde a proposta de Koopman de uma “taxonomia para descrever o espaço do problema para sistemas de autocura” (KOOPMAN, 2003), um impulso de projetos de pesquisa e desenvolvimento foi iniciado. Apesar disso, o termo “autocura” ainda é vago, e ainda falta uma definição mais precisa (RODOSEK et al.,).

Mecanismos efetivos de autocura são um dos meios para aumentar a confiabilidade dos sistemas de TI, reagindo a falhas e mantendo o sistema operacional. É claro que existe uma estreita relação com o termo “tolerância a falhas” aqui. Então, qual é esse relacionamento?

Avizienis definiu a tolerância a falhas como “mecanismos e técnicas que permitem entregar seu serviço especificado apesar da presença de falhas” (AVIZIENIS et al., 2004). Um mecanismo popular de tolerância a falhas é a redundância de componentes e a votação dos resultados, como usado, por exemplo, em projetos de *Triple-Modular-Redundancy* (TMR). Claramente, um dispositivo de TMR não seria necessariamente chamado de autocura, porque falta reatividade que vise a cura (ou seja, a reparação) do sistema. Portanto, nem todos os sistemas tolerantes a falhas são sistemas de autocura. Além disso, a tolerância a falhas visa manter o sistema funcionando a 100% de sua funcionalidade projetada, enquanto a autorrecuperação pode significar que, após a cura, o sistema funcione com menos de 100%. Se um sistema é capaz de auto-curar completamente a ocorrência de falhas, de modo que a operação do sistema não seja prejudicada de forma alguma, ele poderia ser chamado de tolerante a falhas. Assim, estamos inclinados a dizer que a resposta à questão levantada por Koopman "Foi toda a auto-cura também tolerância a falhas?" Depende do grau de degradação após uma atividade de auto-cura: O mecanismo de cura em um sistema é tal que sempre restaura o sistema para 100% de funcionalidade, então o sistema é tolerante a falhas.

Ainda segundo (RODOSEK et al.,), os principais desafios enfrentados pelo self-

healing são:

- Monitoramento: Monitorar um sistema refere-se à detecção de eventos relevantes por meio de pesquisa do estado do sistema ou por relatórios do sistema (por exemplo, traps). Isso aborda a capacidade de:
 - receber qualquer tipo de evento dentro e fora de um sistema de autocura e registrá-lo
 - correlacionar eventos “dentro” e “fora” de acordo com restrições temporais, espaciais e funcionais; isso é especialmente necessário quando a causa de uma falha é o resultado de um ataque malicioso externo intencional
 - aplicações de instrumentos e desenvolver novos tipos de sistemas de monitoramento
- Especificação do comportamento esperado (estado crítico, contexto): É possível que um sistema de auto-cura reaja de maneira diferente em outro contexto (isto é, exibe consciência da situação)?
- Análise de falhas e tomada de decisão: quando é necessário tomar quais ações de recuperação?
- Ação de recuperação: Que tipo de ações são necessárias / disponíveis para recuperar um determinado tipo de sistema? Que grau de recuperação é suficiente para “curar” o sistema? O que significa recuperação em computação orientada a serviços com base em contratos de nível de serviço?
- Aprendizagem: A autoaprendizagem envolve a definição das “configurações iniciais”, bem como o processo de como aprender “corretamente” a partir de falhas detectadas, diagnosticadas e reparadas. Um sistema de autocura não pode ser independente do exterior. As capacidades de aprendizado são influenciadas de maneira semelhante nas etapas de análise e decisão.
- Pró-atividade: Na verdade, temos duas perspectivas sobre um sistema de autocura: (i) a visão global com as interdependências e a (ii) visão local com as intra-dependências. Um sistema deve reagir apenas a indicadores (sensores que relatam uma falha) ou deve algo como um “estado de quase falha” ser reconhecido e reagir antes que ocorra uma falha?
- Acompanhamento de ações: Quais ações de acompanhamento são apropriadas para melhorar a estabilidade de um sistema?

Tais desafios só vieram a ser melhores trabalhados com as técnicas de *models@runtime*, que será apresentado na seção 2.3.

2.3 Models@runtime

Na engenharia de software tradicional, a construção de softwares complexos é realizada em etapas. Cada etapa é responsável por tratar aspectos específicos do produto final, sendo que a etapa de execução é deixada fora desse contexto. Dessa forma, a modificação de um software em execução deve ocorrer com a modificação dos seus códigos e com a posterior reexecução do mesmo (SZVETITS; ZDUN, 2016).

A manutenção de um software, geralmente, é responsável pelo maior custo do projeto (KEPHART; CHESS, 2003), o que foi a motivação para estudos sobre sistemas auto-adaptativos. Os sistemas auto-adaptativos possuem a habilidade de modificarem seu próprio comportamento ou estrutura através de adaptações em resposta às mudanças no ambiente em tempo de execução. Uma simples reconfiguração ou uma modificação no modelo do sistema resulta em uma alteração na arquitetura do sistema (CHEN et al., 2014). Uma adaptação pode ocorrer para permitir que um recurso chave do software continue executando ou para garantir requisitos não-funcionais, tais como segurança, disponibilidade, confiabilidade, dentre outros.

2.3.1 O que são models@runtime

Uma técnica bastante popular utilizada no desenvolvimento de sistemas auto-adaptativos é a construção arquitetônica ou comportamental de modelos em tempo de projeto para serem atualizados durante a execução do sistema. Tais modelos são conhecidos como *models@runtime* (BLAIR; BENCOMO; FRANCE, 2009). Segundo (BENCOMO et al., 2012), *models@runtime* também pode ser entendido como uma abstração de um sistema que pode ser manipulado em tempo de execução para um propósito específico.

No MDE (Model-Driven Engineering), um modelo é uma abstração ou representação reduzida de um sistema que é construído para propósitos específicos. Por exemplo, modelos de software independentes de tecnologia descrevem aplicativos usando conceitos abstratos sobre as tecnologias de computação subjacentes. A comunidade *models@runtime* compartilha essa visão do que constitui um modelo e busca uma compreensão dos papéis que esses modelos podem desempenhar em tempo de execução.

Antes de fornecer uma definição de *models@runtime*, é útil contrastar este trabalho com os esforços históricos no campo da reflexão. A comunidade de reflexões preocupa-se em definir representações de um sistema subjacente que sejam auto-representações e causalmente conectadas (MAES, 1987). Ou seja, os modelos devem representar o sistema e devem estar ligados de tal maneira que espelhem constantemente o sistema e seu estado atual e comportamento; Se o sistema mudar, as representações do sistema - os modelos - também devem mudar e vice-versa.

Em *models@runtime*, também se busca auto-representações apropriadas. É essencial

que tais representações estejam causalmente conectadas. Esse é um requisito importante para sistemas adaptativos por dois motivos:

- o modelo como interrogado deve fornecer informações atualizadas e informações exatas sobre o sistema para conduzir decisões de adaptação subsequentes; e
- se o modelo está conectado causalmente, então adaptações pode ser feita no nível do modelo e não no nível do sistema.

Uma conclusão que pode ser tirada é que *models@runtime* é sinônimo de reflexão, mas isso seria errado. O trabalho em reflexão busca modelos que estão intrinsecamente relacionados ao modelo de computação e, portanto, tendem a se basear no espaço da solução e muitas vezes em um nível baixo - por exemplo, expondo o ato de passar mensagens e suportando operações metalevel como a interceptação de tais mensagens.

2.3.2 Porque modelos em tempo de execução

Modelos de desenvolvimento - modelos produzidos em um processo MDE - são modelos de software “em níveis de abstração acima do nível de código” (FRANCE; RUMPE, 2007) e geralmente transmitem significado de design de software. Exemplos de tais modelos são casos de uso e modelos de arquitetura e implantação. Esses modelos estão associados às entidades gerenciadas durante o ciclo de desenvolvimento de software. Em contraste, os modelos de tempo de execução fornecem “abstrações de fenômenos de tempo de execução” (FRANCE; RUMPE, 2007) e diferentes partes interessadas podem usá-los de maneiras diferentes.

Como no caso dos modelos de desenvolvimento tradicionais, um modelo de tempo de execução suporta o raciocínio. Os usuários do sistema também podem usar modelos de tempo de execução para suportar o monitoramento de estado dinâmico e o controle de sistemas durante a execução ou para observar dinamicamente o comportamento do tempo de execução para entender um fenômeno comportamental. Um modelo de tempo de execução também pode potencialmente suportar a integração semântica de elementos de software heterogêneos em tempo de execução - por exemplo, metamodelos dinamicamente adaptáveis - no domínio de sistemas de sistemas (CHAUDRON, 2009). Modelos em tempo de execução também podem auxiliar na geração automatizada de artefatos de implementação que serão inseridos no sistema durante a execução e até mesmo pelo próprio sistema (MORIN et al., 2008).

Em uma abordagem mais visionária, pode-se imaginar o uso de modelos de tempo de execução para corrigir erros de design ou para dobrar novas decisões de projeto em um sistema em execução (FRANCE; RUMPE, 2007) para suportar o projeto em andamento controlado. Espera-se que os mecanismos utilizados para realizar essa visão sejam significativamente mais complexos. Por outro lado, esses mecanismos mais complexos

seriam capazes de suportar modificações imprevistas, um desafio-chave para sistemas adaptativos em geral (CHENG et al., 2009).

O suporte potencial de *models@runtime* para a evolução do design de software pode desfocar a linha entre os modelos de desenvolvimento e os modelos de tempo de execução. A partir dessa perspectiva, um modelo de tempo de execução pode ser visto como um modelo de desenvolvimento dinâmico que permite a evolução dinâmica e a realização de projetos de software.

Modelos de tempo de execução podem suportar decisões de adaptação de seres humanos - por exemplo, gerentes de sistema - através de agentes de adaptação incorporados no próprio sistema ou através de combinações de ambos. A tendência é apoiar o aumento da automação na tomada de decisões em relação à adaptação de sistemas, conforme capturado pelo direcionamento para a computação autônoma, em que os agentes podem aprender estratégias apropriadas para o gerenciamento eficaz de sistemas. Vê-se *models@runtime* como uma contribuição importante para o campo da computação autônoma, fornecendo meta-informações para impulsionar a tomada de decisões autônomicas.

Dentre os vários objetivos dos modelos em tempo de execução (SZVETITS; ZDUN, 2016), dois podem ser destacados: a adaptação (1) e o monitoramento, simulação e predição (2). A adaptação permite que o sistema seja modificado de acordo com as mudanças ocorridas no ambiente. O monitoramento, simulação e a predição sugerem que o sistema sejam monitorados por meio de modelos de forma a possibilitar o rastreamento do seu comportamento.

2.3.3 O que constitui um modelo em tempo de execução

Um modelo pode ser qualquer representação útil do sistema, e não deve ser surpresa que várias técnicas tenham sido investigadas para *models@runtime*. Algumas das principais dimensões incluem o seguinte:

- Estrutura versus comportamento. Como na comunidade de reflexão, os modelos tendem a se concentrar na estrutura do sistema subjacente ou nos aspectos comportamentais associados, mais dinâmicos. Os modelos estruturais tendem a enfatizar como o software é atualmente construído - por exemplo, em termos de objetos, relacionamentos de herança e caminhos de invocação; componentes e suas conexões; ou aspectos e seus padrões de tecelagem. Em contraste, os modelos comportamentais enfatizam como o sistema executa em termos de fluxos de eventos ou traços através do sistema, ou a chegada de eventos e seu caminho para a execução - chegada, enfileiramento, seleção, despacho e assim por diante.
- Procedural versus declarativo. Outra opção importante é se o modelo é procedimental, refletindo as estruturas ou comportamentos reais do sistema (as

porcas e parafusos) ou mais declarativo - por exemplo, em termos de metas do sistema. Há uma clara pressão decorrente da necessidade de espelhar o espaço do problema para modelos mais declarativos.

- Funcional versus não-funcional. A maioria dos modelos que foram considerados baseiam-se na funcionalidade do sistema subjacente, mas existe uma necessidade igual de modelos que capturam e representam características não funcionais.
- Formal versus informal. Vários modelos são inspirados na matemática da computação, enquanto outros são derivados da consideração de modelos de programação ou abstrações de domínio. Esta edição especial apresenta exemplos de ambos. A vantagem dos modelos formais é que eles podem suportar o raciocínio automatizado sobre o estado do sistema, mas também podem não capturar ou expressar adequadamente os conceitos de domínio necessários com tanta elegância.

Observe que, na prática, é provável que vários modelos coexistam (BLAIR G., 2009) e que estilos diferentes de modelos possam ser necessários para capturar diferentes preocupações do sistema.

2.4 ANTLR

Para criar uma linguagem, é necessário criar um conjunto de regras que leiam sentenças e que tenham ações apropriadas para frases e símbolos encontrados. Uma linguagem é um conjunto válido de sentenças. Uma sentença é constituída de frases e uma frase é constituída de sub-frases e símbolos de vocabulário. De modo geral, uma aplicação que computa e executa sentenças é chamado de interpretador. (AHO; SETHI; ULLMAN, 1995)

O interpretador precisa reconhecer todas as sentenças, frases e sub-frases de uma linguagem particular para reagir de forma apropriada. Reconhecer uma frase significa identificar os seus componentes e diferenciá-la de outras frases.

O interpretador faz uso de um *parser* (analisador de sintaxe) para reconhecer a linguagem. Sintaxe refere-se a um conjunto de regras que definem uma linguagem e o conjunto dessas regras formam uma gramática (AHO; SETHI; ULLMAN, 1995).

O ANTLR⁶ (ANother Tool for Language Recognition) é uma ferramenta de programação que cria um *parser* a partir da descrição formal de uma linguagem, ou seja, uma gramática.

Um simples exemplo de uma gramática que reconhece frases como 'hello friends' ou 'hello world' pode ser demonstrada no trecho de código a seguir.

Listing 2.1 – Gramática Hello utilizando Antlr4

⁶ <https://wwwantlr.org>

```

1 grammar Hello ; // Define uma gramática chamada Hello
2 r : 'hello' ID ; // procura corresponder a palavra chave 'hello '
   seguida por uma identificação
3 ID : [a-z]+ ; // procura corresponder com uma cadeia de
   caracteres alfabéticas minúsculas
4 WS : [ \t\r\n]+ -> skip ; // ignora espaços, tabs e linhas em
   branco

```

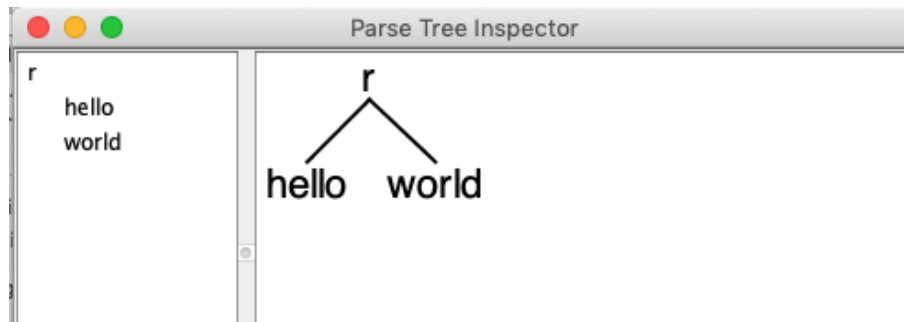


Figura 1 – Árvore de leitura da gramática Hello

A gramática Hello é bem simples. Ela reconhece qualquer cadeia de caracteres que comece com 'hello' seguido de qualquer outra cadeia de caracteres do alfabeto em caixa baixa. Utilizando como entrada 'hello world' na gramática Hello, o ANTLR nos dá como árvore de leitura uma raiz *r* (definida na linha 2) e duas folhas, 'hello' e 'world', como mostra a figura 1.

2.4.1 Parser

O parser pode ser dividido em dois estágios distintos. O primeiro estágio é o processo responsável por analisar a entrada de caracteres e produzir uma sequência de símbolos chamados de 'símbolos léxicos' (lexical tokens), ou somente 'símbolos' (tokens), que podem ser manipulados mais facilmente. Esse estágio é conhecido como análise léxica, tokenizing ou simplesmente lexer.

A tabela 1 mostra quais seriam os símbolos e seus significados depois da etapa de análise léxica, se a entrada fosse a expressão $sp = 100$;

O segundo estágio é o *parser* propriamente dito e recebe como entrada os *tokens* gerados na análise léxica. Por padrão, o ANTLR gera uma estrutura de dados chamada de *parser tree* ou (concrete) *syntax tree*, que registra como o *parser* reconhece a estrutura de entrada, seus componentes e fornece informações úteis de como os símbolos estão agrupados nas frases.

A figura 2 representa o fluxo básico de como o programa funciona.

Tabela 1 – Tipo do token e seu valor correspondente

Tipo do token	texto correspondente
Identificador	<i>sp</i>
Operador de atribuição	=
Inteiro	100
Fim da instrução	;

Fonte: elaborado pelo autor

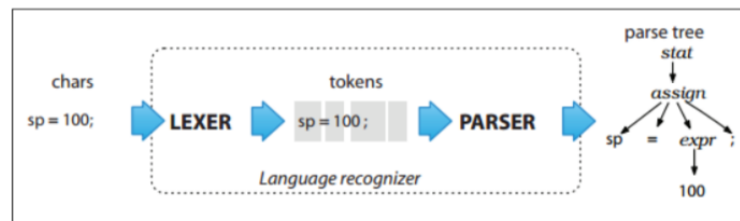


Figura 2 – Fluxo do ANTLR

Os nós do interior da árvore são nomes especificados na gramática e identificam os nós filhos. O nó raiz representa o ponto de partida por onde todas as regras de determinada gramática devem começar a serem analisadas. Os nós folhas são sempre *tokens*.

3 Trabalhos Relacionados

Existem muitas ferramentas que os professores podem utilizar para gerenciar e compartilhar seus conteúdos. Algumas delas são de propósito geral, como o WordPress¹ e o Wix² que permitem um usuário com um pouco de conhecimento em informática criar e publicar uma página na web e, dessa forma, publicar o seu material e conteúdo para seus alunos e interessados. Mas também existem ferramentas como o Moodle³, BlackBoard⁴, Teleduc⁵, que entram na categoria de ferramentas de nicho acadêmico, classificadas como um Ambiente Virtual de Aprendizagem (AVA).

Outra opção que vem sendo utilizada é a construção de DSLs combinadas com modelos em tempo de execução a fim de prover um ambiente para gerenciamento e compartilhamento de conteúdo. Na comunidade de *models@runtime* vários *frameworks* já estabelecem uma relação causal entre um sistema em execução e sua representação de modelo. Kevoree (FOUQUET et al., 2012) oferece a primeira plataforma completa de sistemas distribuídos utilizando *models@runtime* para distribuir aplicações Java, mas não aproveita a infraestrutura como um serviço (IaaS) para operar em sistemas baseados em nuvem. Em contraste, o trabalho de (SHAO et al., 2010) foi a primeira tentativa de construir uma plataforma de modelos em tempo de execução para a nuvem, mas permaneceu restrita ao monitoramento, sem fornecer suporte para promulgação de configuração. Há também o CloudMF de (FERRY et al., 2013), que é um *framework* para modelagem dinâmica de sistemas adaptáveis baseados em nuvem, permitindo a adaptação em tempo de execução. Ele consiste em um DSL de suporte a ferramentas e um ambiente de *models@runtime*. Outro trabalho que se aproxima da contribuição desta pesquisa é o estudo de (LOUKIL; KALLEL; JMAIEL, 2017) que apresenta um middleware responsável por monitorar o sistema e executar a reconfiguração arquitetônica pelo Desenvolvimento de Software Orientado a Aspectos. Concorrência entre reconfigurações também é suportada.

3.1 Características das ferramentas

Nesta seção é apresentado em subtópicos algumas características dos tipos de ferramentas discutidas no início deste capítulo: Ferramentas de construção de páginas web, AVAs e *models@runtime*

¹ <http://wordpress.com>

² <https://pt.wix.com>

³ <https://moodle.org>

⁴ <https://blackboard.com/>

⁵ <http://teleduc.org.br>

3.1.1 Ferramentas de construção de páginas web

As plataformas de criação de site levam uma vantagem maior sobre os outros produtos oferecidos. É fácil lançar um website na internet com WordPress ou Wix, fácil de utilizar, apresentam uma boa curva de aprendizagem. O preço varia um pouco, mas há opções gratuitas e opções pagas. O suporte de ambas plataformas citadas são muito bons, assim como a documentação e nos dois há uma comunidade gigantesca de usuários.

3.1.2 AVAs

Os AVAs por sua vez tiveram uma média 3 (razoável). Dentre os avaliados, Moodle, Blackboard e Teleduc, a facilidade de instalação não pode ser avaliada, pois um deles só era possível obter mediante requisição via formulário de e-mail, o que embarreira o processo de difusão e uso deste. Os outros dois foi uma difícil tentativa, pois a documentação ainda deixa muito a desejar. A comunidade deles variam bastante. O Moodle tem uma expressividade bem maior na internet, enquanto alguns ainda são pequenos, como o Teleduc. O suporte também varia bastante. Alguns tem suporte via e-mail, como o BlackBoard, mas os outros apenas via fórum próprio, como o Moodle e Teleduc. O preço dos AVAs geralmente são gratuitos, porém existem alguns no mercado que são pagos e outros que são gratuitos, mas com suporte pago. A curva de aprendizagem dos AVAs são medianos. Não são difíceis, mas não são fáceis. É necessário algum tempo gasto na plataforma para se ambientar e fazer algumas atividades que deveriam ser mais simples.

3.1.3 Models at runtime

Por fim, os sistemas que utilizam modelos em tempo de execução com DSLs apresentaram boa facilidade de uso e curva de aprendizagem, uma vez que existe uma linguagem específica para auxiliar na execução das tarefas. No entanto, a facilidade de instalação é péssima. Todos os projetos neste tipo de modalidade ainda não são, por assim dizer, comerciais. São gratuitos, de código aberto, o que significa um suporte inexistente ou ineficaz, uma vez que o produtor não tem obrigação de manter o código ou serviço. A comunidade também é muito fraca, visto que apenas uma pequena parcela de pessoas estão testando e utilizando. A documentação técnica é excelente, no entanto, por ser uma documentação técnica, é ainda, inviável, um leigo conseguir instalar sozinho.

No capítulo 4 será apresentado a proposta deste estudo, que é apresentar uma DSL com modelos em tempo de execução para a manutenção de conteúdos na web.

4 Autoria Web em Tempo de Execução

A proposta deste trabalho é considerar os dois objetivos dos modelos em tempo de execução aplicados à realidade do gerenciamento de recursos educacionais pelos professores, que é um dos objetivos dos AVAs.

Nessa arquitetura, apresentada na figura 3, podem ser observados dois ambientes de execução. Um é relacionado ao sistema de arquivos do computador pessoal do professor (lado esquerdo), e o outro é o servidor web no qual os recursos estarão disponíveis para acesso (lado direito). A proposta dessa arquitetura é fornecer um mecanismo de abstração desses dois ambientes de modo a capturar as modificações ocorridas no primeiro e realizar ações de modificação no segundo. Apesar de conterem os quase todos os mesmos elementos nos domínios, esses elementos não são um espelhamento. Cada domínio tem regras, ações e funcionamento diferentes.

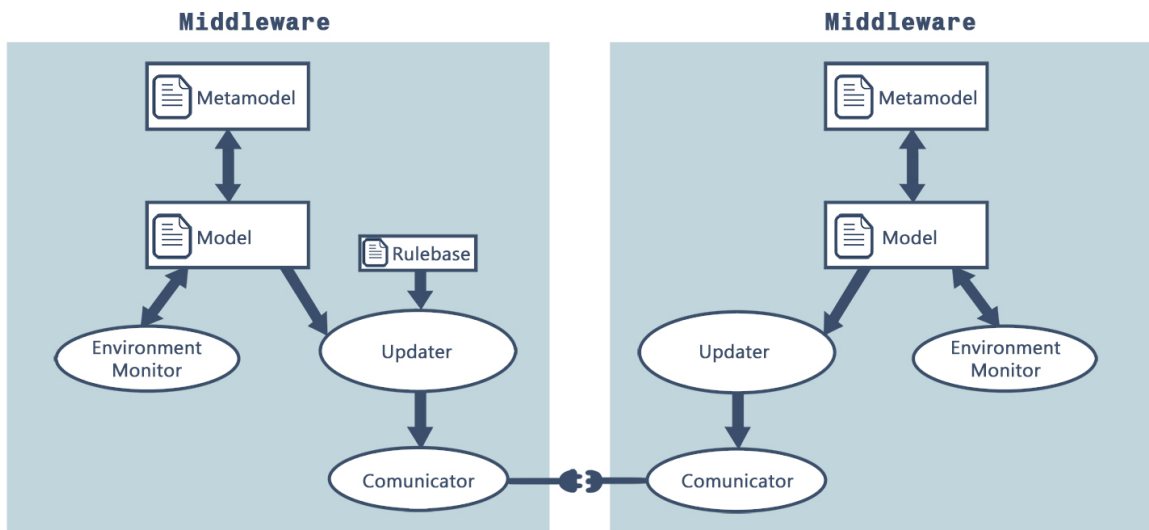


Figura 3 – Arquitetura proposta. À esquerda há uma representação da estrutura de arquivos do professor. À direita há uma representação do sistema de arquivos no servidor web.

Desse modo, os professores poderão realizar ações de criação de pastas, nomeação de arquivos e pastas, inclusão de documentos PDFs, imagens e slides dentro de pastas específicas, e essas ações serão refletidas na criação de páginas web, ou links para arquivos, disponibilizando arquivos PDFs, slides, dentre outros recursos.

Para que essa proposta possa ser viabilizada, algumas regras devem ser aplicadas na criação das pastas pelo professor. Dessa forma, as restrições de criação de arquivos e pastas devem ser monitoradas pelo middleware e, em caso de realização de alguma ação

que não esteja em conformidade com o metamodelo que descreve esse ambiente, essas ações devem ser invalidadas.

O middleware é composto por um metamodelo, um modelo, um monitor de ambiente, um comunicador, um atualizador e, dependendo do ambiente, uma base de regras. Para um melhor entendimento, cada componente será explicado na seção a seguir.

4.1 Componentes da arquitetura

Os componentes da arquitetura proposta serão explicados nas subseções a seguir individualmente, a fim de propor um melhor entendimento dos elementos.

4.1.1 Metamodelo

O metamodelo é um arquivo que define a sintaxe abstrata, como por exemplo as entidades, atributos e relações. O metamodelo é responsável por definir regras, restrições e comportamentos dos modelos.

4.1.2 Modelo

O modelo é um arquivo que implementa as regras do metamodelo. Os modelos em tempo de execução utilizam os modelos para capturar as informações dos softwares em seu ambiente de execução e realizar intervenções nas suas ações.

4.1.3 Monitor de ambiente

O monitor de ambiente é um processo (programa) que é responsável pelo monitoramento do sistema de arquivos. Este programa observa se arquivos ou pastas são criados ou deletados e, em caso afirmativo, o modelo é avisado para tomar as medidas necessárias.

4.1.4 Atualizador

O atualizador é um processo (programa) responsável por receber as mudanças requisitadas pelo modelo e avisar ao comunicador das atualizações do ambiente.

4.1.5 Base de regras

A base de regras é um arquivo texto que descreve as regras do atualizador. Este arquivo contém regras que devem ser refletidas no outro ambiente. Por exemplo, ao criar uma pasta no sistema de arquivos (ambiente do professor), uma página HTML com uma estrutura pré-definida deve ser criada no servidor web.

4.1.6 Comunicador

O comunicador é um processo (software) que é responsável pela comunicação entre os dois ambientes por meio da utilização de sockets. Os comunicadores trocam mensagens sempre que o atualizador tem alguma nova requisição.

4.1.7 Middleware

O middleware engloba todas as partes requeridas de um ambiente. Ele é responsável por empacotar e desempacotar os arquivos necessários para o funcionamento no ambiente instalado.

4.2 Linguagem específica de domínio

Para que as informações disponíveis nas pastas possam, semanticamente, ser suficientes para serem mapeadas por meio das transformações nas informações do metamodelo do servidor web, algumas regras devem ser estabelecidas para a nomeação dos elementos. Dessa forma, uma Linguagem Específica de Domínio (DSL - *Domain Specific Language*) foi desenvolvida. A tabela 2 lista os nomes que os arquivos e pastas devem ter para o funcionamento da arquitetura.

Tabela 2 – Comandos da DSL

Comando	Descrição
Publish ; "nome da página"	Cria uma página no servidor com o nome "nome-da-pagina.html"
Download ; "nome arquivo"	Cria um link para tornar o arquivo "nome arquivo" disponível para download.
Upload ; "Exemplo"	Cria um link com o rótulo "Exemplo" para enviar arquivos para o pasta do usuário.

No capítulo 5 será apresentado um estudo de caso utilizando a arquitetura proposta. O cenário do estudo de caso é um cenário bastante comum na rotina dos professores: estes precisam manipular pastas e arquivos para seus alunos. A proposta é possibilitar que as ações realizadas em pastas nos computadores dos professores, tais como a criação de uma pasta, a inclusão de um arquivo com os slides de uma aula, dentre outras tarefas corriqueiras para quem usa computador, possam ser mapeadas dinamicamente em ações de manutenção em páginas web remotas. Essa abordagem oferece um novo cenário para a manutenção e interação de professores com os AVAs.

5 Estudo de Caso

Um problema recorrente para professores é o compartilhamento de material para várias turmas, tais como slides, artigos, vídeos, entre outras mídias.

Para disponibilizar esses materiais, os professores criam páginas nas quais tudo possa estar disponível, ou utilizam Ambientes Virtuais (AV), tais como o Moodle¹. No primeiro caso, a implementação dessas páginas requer conhecimento técnico, que não é viável para grande parte dos professores. Além disso, é necessário uma constante interação com desenvolvedores sempre que qualquer modificação seja necessária, o que pode tornar a manutenção inviável ao longo do tempo.

Já a utilização de AVs requer, inicialmente, que o mesmo esteja disponível para os professores em sua instituição de ensino. Adicionalmente, cabe ao professor gerenciar a criação de páginas, realizar upload de arquivos e outras ações enfadonhas, que requerem algum grau de conhecimento com o AV utilizado.

Por outro lado, todos os documentos, informações e mídias que os professores pretendem disponibilizar, estão, normalmente, armazenadas no computador pessoal do professor, pois quando um professor vai ministrar um novo curso, no processo de elaboração das suas aulas, normalmente ele utiliza o seu PC para criar pastas e organizar os materiais, notas de aula etc. Ou seja, ele organiza tudo em seu sistema de arquivos do seu computador pessoal.

Assim, pode ser observado dois domínios de execução: o sistema de arquivos do PC do professor, e o servidor web no qual as páginas estão disponibilizadas para acesso externo dos seus alunos. O que é proposto neste estudo é modelar esses ambientes e implantar um middleware para monitorar cada ambiente e realizar ações específicas.

Desse modo, com a utilização desta aplicação proposta nesta pesquisa, a ideia é permitir que as ações que os professores executam em seu ambiente pessoal sejam mapeadas em um outro domínio sem a necessidade de intervenção do professor nem de nenhum outro profissional. Portanto, quando uma pasta for criada no PC do professor para uma nova disciplina, uma nova página seja criada no servidor web disponibilizando o conteúdo que foi depositado na pasta.

5.1 Exemplo: cenário do professor de História

Este é um cenário criado para demonstrar como um professor de História que quer compartilhar com seus alunos um pequeno texto resumido da história do Brasil, um jogo

¹ <https://moodle.org/>

de slides com a história do Ceará para download e exibir uma imagem em uma página web.

Utilizando a arquitetura proposta (capítulo 4), quando o professor cria em seu sistema de arquivos uma pasta chamada 'Publish ; "História do Brasil"' em seu sistema de arquivos, faz com que um arquivo 'Historia-do-Brasil.jsp' seja criada no servidor, conforme ilustra as figuras 4a e 4b.

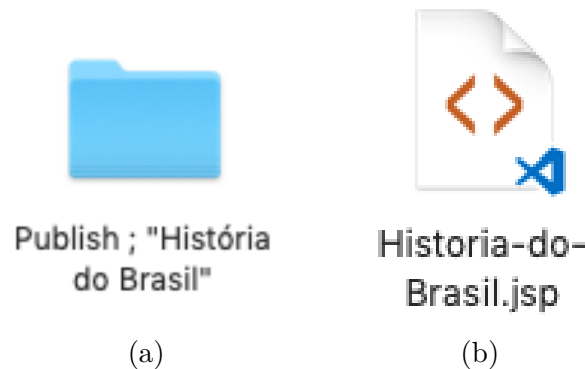


Figura 4 – (a) Pasta obedecendo à DSL no sistema de arquivos do professor; (b) Arquivo criado no lado do servidor, resultante da nomeação da pasta no sistema de arquivos do professor.

O que acontece para fazer essa transformação entre domínios diferentes é que quando a DSL identifica uma pasta que obedece à gramática da linguagem, ela chama o interpretador responsável para executar uma ação relacionada. Neste caso, ao identificar a pasta com nomeada como mostra a figura 4a, o interpretador executa os seguintes códigos:

Listing 5.1 – Método enterInit, chamado automaticamente pelo parser da gramática

```

1 @Override
2 public void enterInit(GrammarDSLParser.InitContext ctx) {
3     super.enterInit(ctx);
4     if (ctx.action() != null && ctx.folder_name() != null) {
5         if (ctx.action().children.get(0).toString().equals("publish")
6             ) {
7             createHtml(ctx.folder_name().children.get(0).toString());
8         }
9     }

```

O pedaço de código descrito em 5.1 checa se é uma ação do tipo 'publish' (as ações podem ser *publish*, *download* ou *upload*), e em caso positivo, ele chama o método createHtml (ver código 5.2, responsável por criar metadados em um arquivo json e enviá-lo ao servidor web).

Listing 5.2 – Método responsável por criar os metadados em um arquivo json

```

1 public void createHtml(String name) {
2     try {
3         String[] fileName = this.myPath.split("/");
4         String tmp = fileName[fileName.length - 1];
5         String[] metadata = tmp.split(";");
6
7         String everything = name;
8         JSONObject jobject = new JSONObject();
9         jobject.put("nome", everything);
10        jobject.put("action", metadata[0].trim());
11        jobject.put("folder_name", metadata[1].trim().replaceAll("
            ^\\|\\|\"$", ""));
12
13        List<String> lines = Arrays.asList(jobject.toJSONString());
14        Path file = Paths.get(this.webAppPath + "/publish_";\json
            \"/" + new Date().getTime() + ".json");
15        Files.write(file, lines, Charset.forName("UTF-8"));
16    } catch (Exception e) {
17        e.printStackTrace();
18    }
19 }

```

O resultado da saída do método `createHtml` é um arquivo json nomeado com um *timestamp* que é enviado para o servidor web em uma pasta só para arquivos no formato json. Um exemplo do conteúdo deste arquivo pode ser visto no código 5.3.

Listing 5.3 – Arquivo json de metadados, contendo informações para a DSL do lado do servidor tomar uma ação

```

1 {"action": "publish", "nome": "\" História do Brasil \", "folder_name
    ": " História do Brasil "}

```

Quando o código do lado do servidor é executado, ele lê todos os arquivos json's encontrados, que são informações (metadados) sobre o que foi encontrado do lado do computador do professor. Ao ler o json no código 5.3, o interpretador do servidor web chama um método apropriado e executa alguma ação de acordo com as informações do metadado, seja criar uma página, inserir texto em um HTML, adicionar um link para download ou qualquer outra funcionalidade que se deseja criar.

A figura 5 ilustra o funcionamento da situação exemplo.

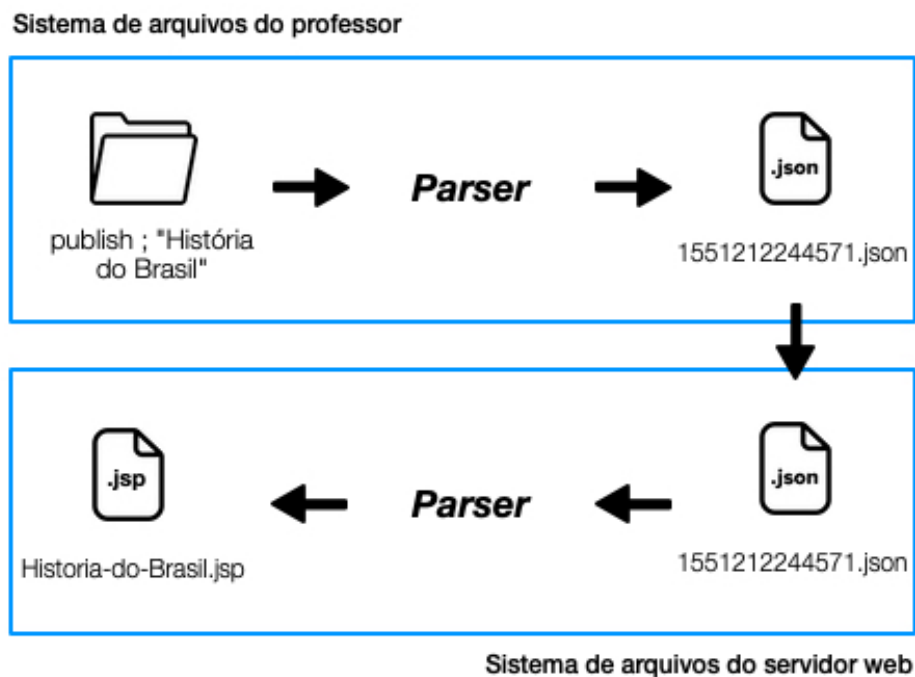


Figura 5 – Ilustração do funcionamento da DSL

5.1.1 Sistema de arquivos do professor

O resultado final do sistema de arquivos do professor é exibido na figura 6.

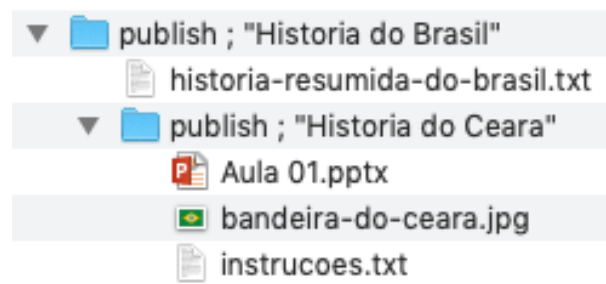


Figura 6 – Exemplo do sistema de arquivos do professor de História

Não é necessário nenhuma configuração ou esforço por parte do professor. Ele apenas precisa conhecer um pequeno conjunto de regras de nomeação de pastas e arquivos.

Na subseção 5.1.2 é exibido como esta estrutura é refletida no servidor web.

5.1.2 Sistema de arquivos no servidor web

O resultado do sistema de arquivos do professor (figura 6) no servidor web é exibido na figura 7.

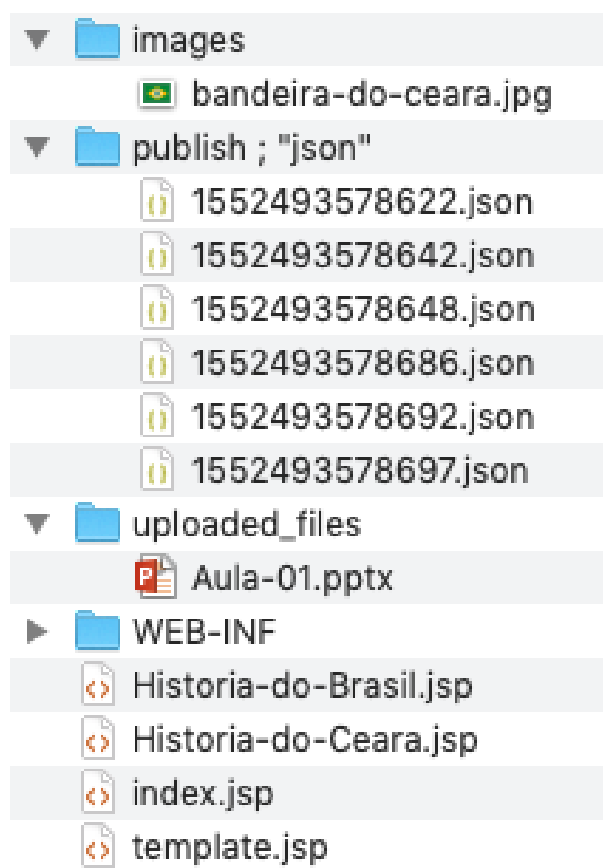


Figura 7 – Resultado no servidor web

Os seis arquivos json são os arquivos que contêm os metadados da página de História do Brasil, do texto da história do Brasil, da página de História do Ceará, do arquivo pptx, da imagem da bandeira e do arquivo de instruções dentro da pasta História do Ceará. O index.jsp é um arquivo padrão que sempre será a página que irá listar as outras páginas e o template.jsp serve como modelo de base para criar as novas páginas geradas pelo interpretador do servidor. A pasta WEB-INF contém apenas informações relacionadas ao servidor.

5.1.3 Resultado do cenário

A saída das páginas geradas pelo professor de História podem ser vistas nas figuras 8, 9 e 10 a seguir.



Figura 8 – Página index. Lista no seu menu superior o nome das páginas, nomeadas de acordo com o nome da pasta que o professor nomeou em seu sistema de arquivos

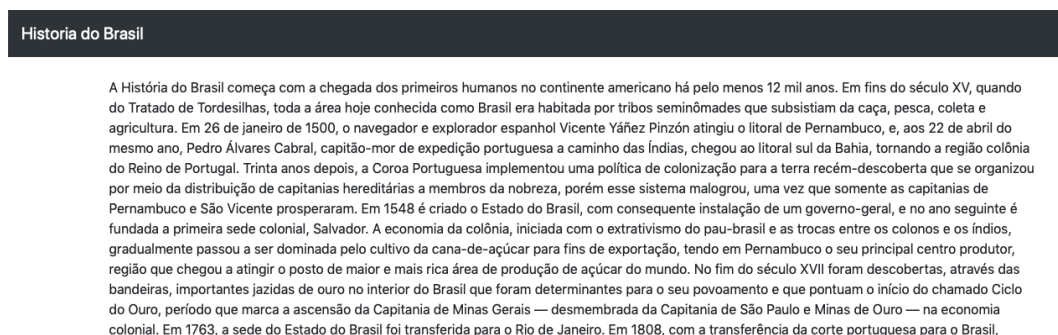


Figura 9 – Página Historia-do-Brasil.jsp. Neste caso, como só existe apenas um arquivo de texto contendo a história do Brasil, este é gerado como um texto dentro da página Historia-do-Brasil.jsp

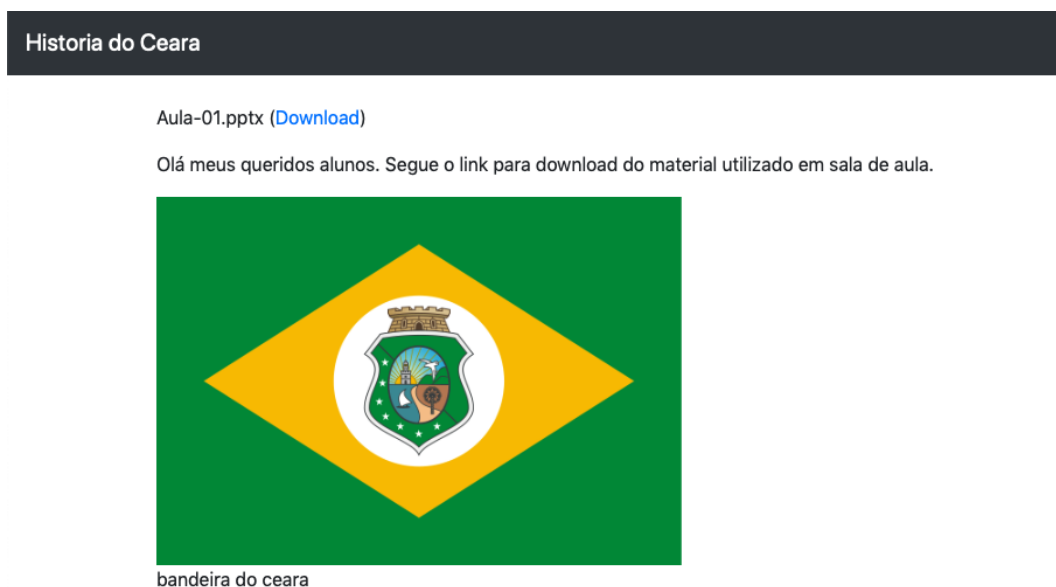


Figura 10 – Página Historia-do-Ceara.jsp. Neste caso, como existe um arquivo pptx e uma imagem, são gerados, respectivamente, um link para download do arquivo e a imagem é renderizada dentro de um objeto img na página Historia-do-Ceara.jsp. O nome da imagem vira automaticamente a legenda da imagem.

Esta arquitetura permite que um professor manipule as informações que ele precisa publicar na web apenas manipulando a estrutura de pastas e arquivos em seu computador utilizando uma simples regra de nomeação da DSL.

No capítulo 6 será apresentado a conclusão deste trabalho.

6 Conclusão

Este trabalho propôs uma arquitetura baseada em modelos de tempo de execução como um mecanismo de interação entre dois domínios diferentes. O exemplo do estudo de caso exibe um cenário onde um professor, mesmo um leigo em informática, poderia utilizar para compartilhar mídias (slides, textos, imagens, dentre outros) com seus alunos apenas se aproveitando de um simples conjunto de regras de nomeação de pastas e arquivos.

A DSL deste trabalho ainda tem diversas limitações e ainda está sendo desenvolvido melhorias, como por exemplo o suporte a multi-usuários. Outro ponto a ser considerado é que esta solução foi apresentada como unidirecional, mas que também poderia ser feito bidirecional, bastando escrever novas regras nos modelos e metamodelos para que estes passassem a ser aptos a trabalharem desta forma.

Como trabalho futuro, espera-se que a continuação deste estudo possa avaliar junto aos professores em um trabalho de campo o benefício e efetividade da contribuição deste estudo. Além disso, esta arquitetura também poderia ser incorporada como uma extensão de AVAs existentes para auxiliar uma interação entre tais softwares a fim de facilitar a rotina dos professores ao utilizar os AVAs.

Um outro trabalho futuro importante é a avaliação mais criteriosa das relações entre serviços de autoria de websites, AVAs e DSLs, de forma a mostrar os reais benefícios da abordagem proposta nesta pesquisa.

Referências

- ABDULLAH M. S., B. I. E. A. K. C. Knowledge modelling techniques for developing knowledge management systems. *Third European Conference on Knowledge Management: Trinity College Dublin*, p. 17, September 2002. Citado na página 25.
- AHO, A. V.; SETHI, R.; ULLMAN, J. D. Compiladores: Principios, técnicas e ferramentas. *LTC, Rio de Janeiro, Brasil*, 1995. Citado na página 31.
- AVISON D., F. G. *Information systems development: methodologies, techniques and tools*. [S.l.]: McGraw Hill, 2003. Citado na página 22.
- AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, IEEE, v. 1, n. 1, p. 11–33, 2004. Citado na página 26.
- BENCOMO, N. et al. Summary of the 7th international workshop on models@run.time. In: *Proceedings of the 7th Workshop on Models@Run.Time*. New York, NY, USA: ACM, 2012. (MRT '12), p. 1–2. ISBN 978-1-4503-1802-0. Disponível em: <<http://doi.acm.org/10.1145/2422518.2422519>>. Citado 2 vezes nas páginas 21 e 28.
- BLAIR, G.; BENCOMO, N.; FRANCE, R. B. Models@ run.time. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 42, n. 10, p. 22–27, out. 2009. ISSN 0018-9162. Disponível em: <<https://doi.org/10.1109/MC.2009.326>>. Citado 2 vezes nas páginas 21 e 28.
- BLAIR G., B. N. F. R. Models@run.time. *Computer* 42, p. 22–27, 2009. Citado na página 31.
- CHAUDRON, M. R. *Models in Software Engineering: Workshops and Symposia at MODELS 2008, Toulouse, France, September 28-October 3, 2008. Reports and Revised Selected Papers*. [S.l.]: Springer, 2009. v. 5421. Citado na página 29.
- CHEN, B. et al. Self-adaptation through incremental generative model transformations at runtime. In: *Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA: ACM, 2014. (ICSE 2014), p. 676–687. ISBN 978-1-4503-2756-5. Disponível em: <<http://doi.acm.org/10.1145/2568225.2568310>>. Citado na página 28.
- CHENG, B. H. et al. Software engineering for self-adaptive systems: A research roadmap. In: *Software engineering for self-adaptive systems*. [S.l.]: Springer, 2009. p. 1–26. Citado na página 30.
- FERRY, N. et al. Managing multi-cloud systems with cloudmf. In: *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies*. New York, NY, USA: ACM, 2013. (NordiCloud '13), p. 38–45. ISBN 978-1-4503-2307-9. Disponível em: <<http://doi.acm.org/10.1145/2513534.2513542>>. Citado na página 35.
- FOUQUET, F. et al. Dissemination of reconfiguration policies on mesh networks. In: *SPRINGER. IFIP International Conference on Distributed Applications and Interoperable Systems*. [S.l.], 2012. p. 16–30. Citado na página 35.

FRANCE, R.; RUMPE, B. Model-driven development of complex software: A research roadmap. In: IEEE COMPUTER SOCIETY. *2007 Future of Software Engineering*. [S.l.], 2007. p. 37–54. Citado na página 29.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 36, n. 1, p. 41–50, jan. 2003. ISSN 0018-9162. Disponível em: <<http://dx.doi.org/10.1109/MC.2003.1160055>>. Citado na página 28.

KOOPMAN, P. Elements of the self-healing system problem space. figshare, 2003. Citado na página 26.

LEWIS, D. What is web 2.0? *Crossroads*, p. 3–3, June 2006. Citado na página 25.

LOUKIL, S.; KALLEL, S.; JMAIEL, M. An approach based on runtime models for developing dynamically adaptive systems. *Future Generation Computer Systems*, v. 68, p. 365 – 375, 2017. ISSN 0167-739X. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0167739X16302369>>. Citado na página 35.

MAES, P. Concepts and experiments in computational reflection. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 22, n. 12, p. 147–155, dez. 1987. ISSN 0362-1340. Disponível em: <<http://doi.acm.org/10.1145/38807.38821>>. Citado na página 28.

MORIN, B. et al. An aspect-oriented and model-driven approach for managing dynamic variability. In: SPRINGER. *international conference on Model Driven Engineering Languages and Systems*. [S.l.], 2008. p. 782–796. Citado na página 29.

MURUGESAN, S. *Understanding Web 2.0. IT professional*. [S.l.]: University of Western Sydney, Australia, 2007. Citado na página 21.

RODOSEK, G. D. et al. Self-healing systems: Foundations and challenges. Citado na página 26.

SHAO, J. et al. A runtime model based monitoring approach for cloud. In: *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, 2010. Disponível em: <<https://doi.org/10.1109/cloud.2010.31>>. Citado na página 35.

SZVETITS, M.; ZDUN, U. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Softw. Syst. Model.*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 15, n. 1, p. 31–69, fev. 2016. ISSN 1619-1366. Disponível em: <<http://dx.doi.org/10.1007/s10270-013-0394-9>>. Citado 3 vezes nas páginas 21, 28 e 30.