UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL - CENTRO DE CIÊNCIAS TECNOLÓGICAS**

SISTEMAS DE INFORMAÇÃO

ROBERTO ELERO JUNIOR

UM FRAMEWORK PARA AUXÍLIO DO DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS INTEGRADAS COM PLATAFORMAS EM NUVEM

Bandeirantes

2015

ROBERTO ELERO JUNIOR

UM FRAMEWORK PARA AUXÍLIO DO DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS INTEGRADAS COM PLATAFORMAS EM NUVEM

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. André Luís Andrade Menolli

Bandeirantes

2015

ROBERTO ELERO JUNIOR

UM FRAMEWORK PARA AUXÍLIO DO DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS INTEGRADAS COM PLATAFORMAS EM NUVEM

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____
Prof. Dr. André Luís Andrade Menolli
UENP – *Campus* Luiz Meneghel

_____
Prof. Me. Glauco Carlos Silva
UENP – *Campus* Luiz Meneghel

_____
Prof. Me. Ricardo Gonçalves Coelho
UENP – *Campus* Luiz Meneghel

Bandeirantes, 19 de Agosto de 2015

Dedico este trabalho à minha avó, a minha motivação eterna.

AGRADECIMENTOS

Agradeço ao meu orientador, Luís, que desde o início aceitou me orientar, mesmo em um caso tão atípico como o meu, ao Prof. Muhammad Ali Babar, que também me aceitou como pesquisador visitante em seu grupo durante meu intercâmbio, no qual desenvolvi a maior parte do presente trabalho, e acima de tudo à minha família, que sempre depositou sua confiança em mim e me deu forças para continuar acreditando nos meus ideais.

Só se pode alcançar um
grande êxito quando
nos mantemos fiéis a
nós mesmos.
(Friedrich Nietzsche)

RESUMO

Avanços em tecnologias móveis têm ocorrido em ritmo acelerado. Não somente dispositivos móveis têm evoluído significantemente, mas também, redes móveis sem fio têm se tornado extremamente mais velozes. No entanto, dispositivos móveis ainda apresentam limitações contrastantes quando comparados à outros dispositivos computacionais. Neste contexto, as vantagens apresentadas pela computação em nuvem, especialmente o *offload*, podem ser combinadas para superar as limitações em dispositivos móveis, proporcionando uma melhor experiência ao usuário. Logo, o presente trabalho tem como objetivo propor um framework que auxilie desenvolvedores a criar aplicações móveis integradas com a nuvem por meio de técnicas de desenvolvimento orientado a modelos, isto é (1) desenvolver um meta-modelo que represente o domínio de aplicações móveis conectadas com a plataforma em nuvem, (2) desenvolver um gerador de código que seja capaz de receber uma instância do meta-modelo como entrada para geração de código fonte para plataformas móveis, (3) superar limitações apresentadas por dispositivos móveis quando comparados com outros dispositivos computacionais; e (4) desenvolver um mecanismo que permita desenvolvedores a determinar quais operações são processadas *on-cloud* e quais operações são processadas *on-device*. Para avaliar a abordagem proposta, criamos uma aplicação móvel integrada com a nuvem utilizando o framework apresentado neste trabalho, efetuando operações distintas e visando ilustrar diferentes cenários nos quais o *offload* de computação pode ser satisfatório ou não satisfatório. Finalmente, analisando os dados obtidos em nosso experimento, chegamos a conclusão que o *offload* pode superar as limitações apresentadas por dispositivos móveis em diferentes cenários.

Palavras-chave: Desenvolvimento Orientado a Modelos, Computação Móvel, Computação em Núvem, Eclipse, EMF, Acceleo, Offloading.

ABSTRACT

Breakthroughs on mobile technologies have taken place fast. Not only mobile devices have significantly evolved, but also, mobile wireless networks have become extremely faster. However, mobile devices still present contrasting limitations when compared to other computational devices. In this context, the advantages presented by cloud computing, especially the *offload*, could be combined in order to overcome limitations of mobile devices, which would provide an enhanced experience to the user. Thus, the present work has as objective propose a framework which assists developers to build mobile application integrated with the cloud through model driven development, namely (1) develop a meta-model that represents the mobile application integrated with the cloud domain, (2) develop a code generator that may receive an instance of the meta-model as input for generation of mobile applications code, (3) overcome limitations on mobile devices when compared to other computational devices, and (4) develop a mechanism that allows developers to determine which operations are performed *on-cloud* and which operations are processed *on-device*. To evaluate the proposed approach, we built an mobile application integrated with the cloud by using our framework, and performing distinct operations, aiming to illustrate different scenarios in which the offload could be satisfactory and could not be satisfactory. Finally, analyzing data obtained from our experiment, we came to the conclusion that offload may overcome limitations presented by mobile devices in different scenarios.

Palavras-chave: Model Driven Development, Mobile Computing, Cloud Computing, Eclipse, EMF, Acceleo, Offloading.

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO

***Background**:* É utilizado para referenciar tarefas ou processos que não necessitam entrada de usuário. Geralmente são processados em segundo plano em uma aplicação.

Clusters: Um cluster de computadores consite de um conjunto de computadores conectados de forma que possam trabalhar juntos em diversos aspectos. De certa forma, o conjunto de computadores pode ser visto como um sistema único.

Cyber-foraging: Cyber-foraging é um outro nome atribuído à *offload.*

Desktop: Representa um computador de mesa.

Graphical User Interface: Interface gráfica de usuário. Geralmente, utiliza-se a sigla GUI para representar este termo.

Namespace: significa espaços de nomes, onde desenvolvedores podem criar suas próprias bibliotecas de elementos.

Offload: Offload de computação é uma abordagem popular empregada para redução de consumo de energia e processamento em dispositivos móveis por meio do envio de operações para servidores remotos.

On-cloud: On-cloud basicamente significa “sob a plataforma em nuvem”. O termo é utilizado quando um processo é executado sobre a plataforma em nuvem, e.g. processamento *on-cloud.*

On-device: On-device basicamente significa “sob o dispositivo local”. O termo é utilizado quando um processo é executado localmente, ou sob o dispositivo local, e.g. processamento *on-device.*

Overhead: Toda computação auxiliar requerida por um algoritmo ou programa.

Platform-Independent Model: Modelo independente de plataforma.

Plugin: Módulo ou software capaz de ser adicionado a um sistema, acrescentando funcionalidades extras a este.

Snippet: pedaço pequeno ou fragmento de código.

Walkthrough: Palavra utilizada para descrever uma tarefa ou sistema passo a passo, geralmente em detalhes.

Workspace: Ambiente de trabalho, ou desenvolvimento, onde os projetos geridos pela plataforma Eclipse são armazenados.

Wizard: assistente utilizado na área de software para criar, alterar, e remover programas ou arquivos.

SUMÁRIO

1. INTRODUÇÃO

A seção introdutória apresenta a contextualização do projeto, formulação e escopo do problema, objetivos do projeto, justificativas para a elaboração do projeto, e finalmente a organização do documento.

1.1 CONTEXTUALIZAÇÃO

Avanços em tecnologias móveis têm ocorrido em ritmo acelerado. De acordo com Bahl, Han e Erran (2012), não somente dispositivos móveis têm evoluído significantemente em termos de velocidade de processamento e armazenamento, mas também, redes móveis sem fio têm se tornado extremamente mais velozes, o que também resulta em uma redução de *overhead* e melhora na resposta de usuário (BAHL; HAN; ERRAN, 2012). Exemplo disso é a série de smartphones *Galaxy*, lançada pela Samsung em 2009. Em 2014, o dispositivo já alcançava mais de duas vezes poder computacional que sua primeira versão, como apresentado na Figura 1.

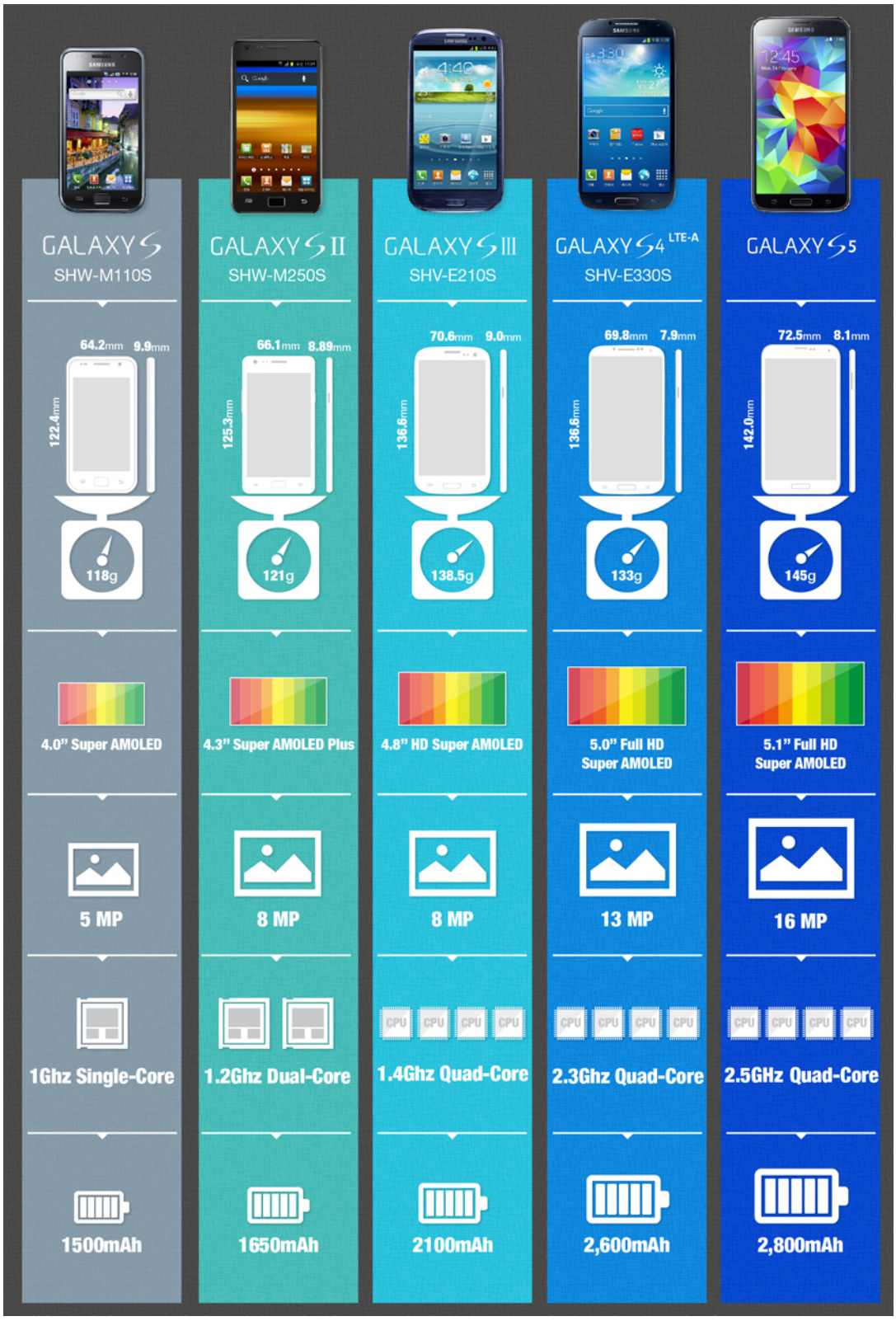


Figura 1 – A evolução da série Galaxy da Samsung. (Samsung, 2015)

Do mesmo modo, computadores pessoais e laptops têm evoluído consideravelmente com o passar dos anos (ALZAHRANI; ALALWAN;

SARRAB, 2014). Segundo Sinha e Kulkarni (2011), embora ambos dispositivos móveis e *desktops* vêm sendo aprimorados em termos de processamento e armazenamento, computadores pessoais e laptops tendem a ser substituídos por dispositivos móveis em função das vantagens com respeito a mobilidade, conectividade e conveniência. Em adição, Sinha e Kulkarni acrescentam que dispositivos móveis ainda não se tornaram o meio computacional primário devido às limitações contrastantes apresentadas por dispositivos móveis quando comparados à outros dispositivos computacionais, e.g. computadores pessoais e laptops. Como exemplo, pode-se observar a Tabela 1, que contrasta diferentes dispositivos computacionais, inclusive móveis, em termos de memória RAM, armazenamento, e CPU nas categorias Smartphone, Tablet, Laptop e PC.

Tabela 1 – Contrastes entre diferentes dispositivos computacionais.

Modelo	**Categoria**	**Memória RAM**	**Armazenamento**	**CPU**
Samsumg Galaxy S5	Smartphone	2GB	Até 128GB	2.5 GHz Quad-Core
iPad Air	Tablet	2GB	Até 32GB	Dual-core 1.3 GHz
Inspiron 13 Série 7000	Laptop	8GB	500GB	Intel Core i5 2.7 GHz
XPS 8700	PC	16GB	2TB	Core i7 4.0 GHz

Fonte: (SAMSUMG, 2015; DELL, 2015; APPLE, 2015).

1.2 FORMULAÇÃO E ESCOPO DO PROBLEMA

Dispositivos móveis ainda são bastante limitados quando comparados com outros dispositivos computacionais, e.g. computadores pessoas e laptops, especialmente em termos de **capacidade computacional** e armazenamento (ALZAHRANI; ALALWAN; SARRAB, 2014). Segundo Balagtas-Fernandez e Hussmann (2008), poder computacional e capacidade de bateria são as principais limitações de dispositivos móveis. Certamente, a limitação de poder computacional em dispositivos móveis restringe significantemente a gama de operações a serem realizadas nestes dispositivos. Além disso, a limitação de capacidade de bateria, também conhecida como limitação de consumo de energia, reduz o tempo para processamento de tais operações (LUO, 2009).

Ao mesmo tempo, a computação móvel em nuvem em certos casos pode não ser uma alternativa apropriada, uma vez que certas operações são melhores executadas sob o dispositivo móvel em si, e não na plataforma em nuvem (KUMAR; YOUNG-HSIANG, 2010). Hassan e Chen (2012) realizaram experimentos distintos com *offload* de dados e computação do dispositivo móvel para a plataforma em nuvem e apontaram casos onde o *offload* resulta no pior desempenho em termos de resposta ao usuário e consumo de energia. Em adição, Hassan e Chen (2012) apontam cenários onde o processamento no dispositivo móvel sem integração com a plataforma em nuvem apresenta a performance menos satisfatória entre todos os demais experimentos.

1.3 OBJETIVO DO PROJETO

Esta seção apresenta os objetivo geral e específico do presente projeto.

1.3.1 OBJETIVO GERAL

O objetivo deste projeto é propor um framework que auxilie desenvolvedores a criar aplicações móveis integradas com a nuvem por meio de técnicas de desenvolvimento orientado a modelos.

1.3.2 OBJETIVOS ESPECÍFICO

O presente projeto tem como objetivos específicos:

a) Desenvolver um meta-modelo que represente o domínio de aplicações móveis conectadas com a plataforma em nuvem;
b) Desenvolver um gerador de código que seja capaz de receber uma instância do meta-modelo, mencionado anteriormente, como entrada para geração de código fonte para plataformas móveis;

c) Superar limitações de capacidade computacional e capacidade de bateria apresentadas por dispositivos móveis quando comparados com outros dispositivos computacionais; e
d) Desenvolver um mecanismo que permita desenvolvedores a determinar quais operações são processadas *on-cloud* e quais operações são processadas *on-device.*

1.4 JUSTIFICATIVA

As vantagens apresentadas pela computação em nuvem e computação móvel, especialmente o *offload*, podem ser combinadas para superar as limitações em dispositivos móveis citadas anteriormente, e portanto, proporcionar uma melhor experiência ao usuário (ALZAHRANI; ALALWAN; SARRAB, 2014).

Em adição, computação móvel em nuvem em certos casos pode não ser uma alternativa apropriada, uma vez que certas operações são melhores executadas sob o dispositivo móvel em si, e não na plataforma em nuvem (KUMAR; YOUNG-HSIANG, 2010).

1.5 ORGANIZAÇÃO

Neste projeto é proposto o MDCM (***Model-driven Development of Cloud-integrated Mobile Applications***), um framework para construção de aplicações móveis integradas com a nuvem que permite desenvolvedores a determinar quando uma operação é executada exclusivamente no dispositivo móvel e quando a operação deve ser enviada para a plataforma em nuvem (i.e. *offload*).

Ao longo do documento é apresentada a Fundamentação Teórica do trabalho, contendo uma introdução ao conceito de Desenvolvimento Orientado a Modelos, Computação Móvel e Computação em Nuvem e uma breve caracterização do conceito de *Offload* de Operações e Dados. Além disso, será apresentada o desenvolvimento de nosso framework, no qual inclui uma visão geral de nosso framework, as especificações de nosso meta-modelo, gerador

de código, e papeis envolvidos no ciclo de vida de nosso framework. Também, apresentamos nosso experimento de avaliação, que é constituído por diferentes ambientes experimentais e finalmente, apresentamos a análise de dados, seguida das considerações finais.

2. FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentadas introduções aos conceitos de Desenvolvimento Orientado a Modelos, Computação Móvel, Computação em Nuvem e *Offload* de Operações e Dados.

2.1 DESENVOLVIMENTO ORIENTADO A MODELOS

Desenvolvimento orientado a modelos, ou MDD, é uma abordagem em engenharia de software que visa elevar o nível de abstração de aplicações e assim, simplificar e formalizar as tarefas e os estágios do ciclo de vida de um software, usando modelos e tecnologias de modelagem (HAILPERN; TARR, 2006).

Segundo Vidyapeetham (2009b), o MDD compreende o uso de modelos no ciclo de vida do desenvolvimento e software e argumenta que MDD automatiza o desenvolvimento de software via processamento de modelos, transformação de modelos, e técnicas de geração de código.

Durante a última década, o MDD tem evoluído significantemente devido à sua flexibilidade e aplicabilidade (VIDYAPEETHAM, 2009a). Convencionalmente, de acordo com Vidyapeetham (2009a), desenvolvedores empregam duas abordagens diferentes em desenvolvimento de software:

a) Projetar a solução de forma visual; e
b) Diretamente codificar solução baseando-se em requisitos funcionais.

A despeito de que ambas abordagens apresentem vantagens e desvantagens, o MDD tende a prover simultaneamente o melhor de cada uma delas. A propósito, por meio do MDD, desenvolvedores são capazes de, ao mesmo tempo, projetar suas soluções e construir artefatos parciais que farão parte do produto final (VIDYAPEETHAM, 2009a).

Do mesmo modo, Selic (2008) afirma que o foco primário do MDD são os modelos ao invés de programas de computador. Além disso, Selic (2008)

aponta a expressão de modelos usando conceitos que não são ligados a linguagens de baixo nível, mas muito embora próximo ao domínio real do problema, como principal vantagem da adoção do MDD.

Abordagens em MDD aplicadas no domínio de aplicações móveis comumente envolvem linguagens de domínio específico (DSLs) ao lugar de representações visuais. Steiner et al. (2013) apresenta uma DSL para desenvolvimento de aplicações nativas multiplataformas e integradas com a nuvem utilizando *Xtext* (http://www.eclipse.org/Xtext/).

Além disso, Heitkötter, Majchrzak e Kuchen (2013) apresentam MD2, uma DSL para desenvolvimento de aplicações móveis multiplataformas, gerando aplicações nativas puras para os sistemas Android e iOS, no entanto, tais aplicações não possuem integração com a nuvem.

Ainda, Ranabahu, Maximilien e Sheth (2011) apresentam uma DSL para aplicações híbridas em escala empresarial, fornecendo abstrações de código suficientes para atender requisitos de aplicações móveis, como também, para integração com a nuvem.

2.2 COMPUTAÇÃO MÓVEL E A COMPUTAÇÃO EM NUVEM

De acordo com Mallikharjuna, Sasidhar e Satyendra (2012), computação em nuvem tem sido um conceito bastante proeminente no momento atual.

O conceito de computação em nuvem corresponde a computação baseada na Internet que provem serviços sob demanda, e.g. aplicações, armazenamento, e servidores por meio de uma conexão Internet (ALZAHRANI; ALALWAN; SARRAB, 2014). Em outras palavras, *clusters* em *background* e servidores efetuam a computação e armazenamento necessários para a execução da tarefa em questão. Como consequência, as vantagens apresentadas pela computação em nuvem e computação móvel podem ser combinadas para superar as limitações em dispositivos móveis citadas anteriormente, e portanto, proporcionar uma melhor experiência ao usuário (ALZAHRANI; ALALWAN; SARRAB, 2014). A propósito, Alzahrani, Alalwan e Sarrab (2014) apresentam uma lista de categorias de aplicações suportadas

pela computação móvel na nuvem incluindo aplicações comerciais, aprendizagem móvel, sistema bancários móveis, sistemas de controle de saúde móveis, e jogos móveis.

2.3 *OFFLOAD* DE COMPUTAÇÃO E DADOS

O *Offload* de computação e dados é uma abordagem popular empregada para redução de consumo de energia e processamento em dispositivos móveis por meio do envio de operações e dados utilizados em tais operações para servidores remotos (NIU; SONG; LIU, 2011).

Também conhecido como *cyber-foraging*, o conceito de *offload* refere-se ao processamento **parcial**, também conhecido como processamento de operações, ou total da aplicação móvel em servidores remotos, e.g. plataformas em nuvem e clusters (SINHA; KULKARNI, 2011). Uma que vez o processamento é concluído nos servidores remotos, o dado é processado é retornado ao dispositivo móvel local. A Figura 2 representa graficamente o as etapas do processamento parcial ou de operações por meio do *offload*. Visando tornar claro o *offload* parcial representado pela Figura 2, as etapas foram numeradas de acordo com sua determinada cronologia no processo.

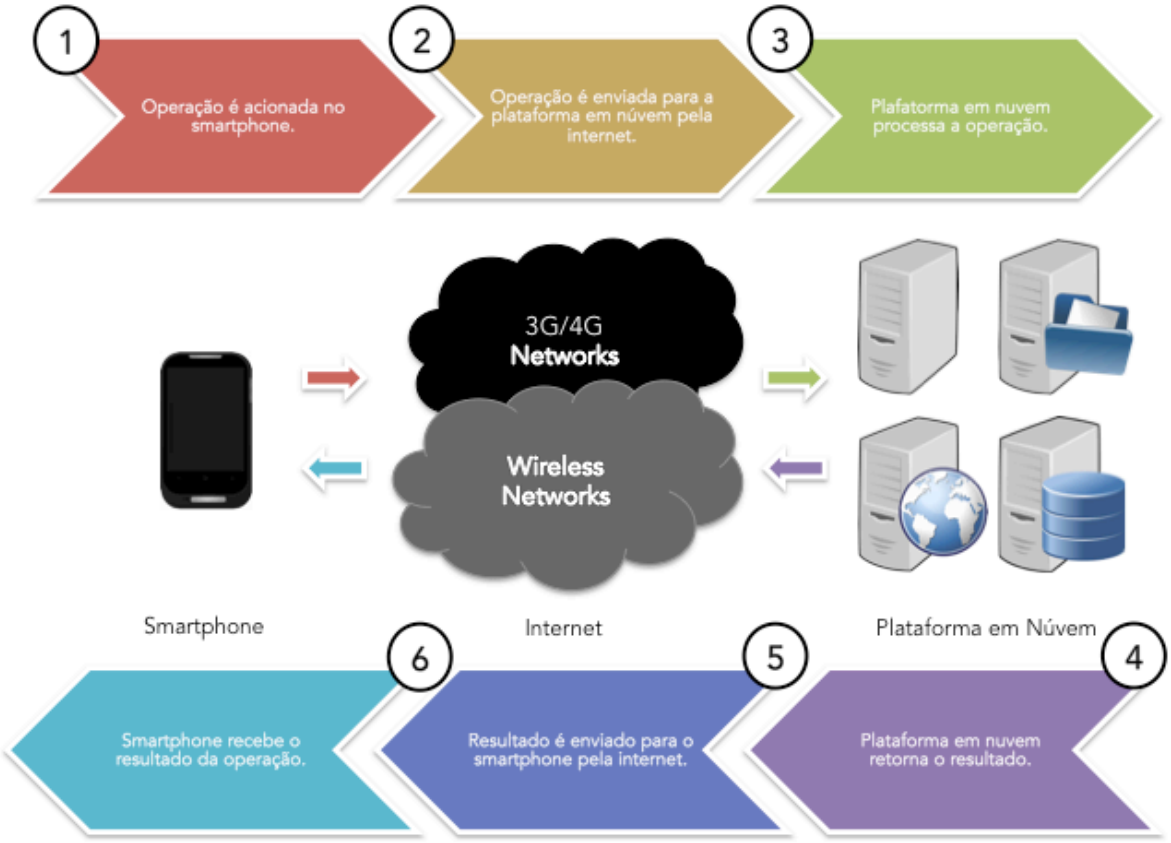


Figura 2 – Processamento parcial ou de operações por meio do *offload*.
(Elaborado pelo Autor)

De acordo com Sinha e Kulkarni (2011), a escolha das operações a serem enviadas para processamento remoto é crucial para o aprimoramento os efeitos causados pelas limitações apresentadas por dispositivos móveis. Dessa forma, o *offload* de operações, em certos casos, pode não ser uma alternativa apropriada, uma vez que certas operações são mais satisfatoriamente executadas sob o dispositivo móvel em si, e não na plataforma em nuvem (KUMAR; YOUNG-HSIANG, 2010). Hassan e Chen (2012) realizam experimentos distintos com *offload* de dados e computação do dispositivo móvel para a plataforma em nuvem *Amazon EC2*, e apontam casos onde o *offload* resulta no pior desempenho em termos de resposta ao usuário e consumo de energia. Em adição, Hassan e Chen (2012) apontam cenários que

o processamento no dispositivo móvel sem integração com a plataforma em nuvem apresenta a pior performance.

À vista disso, o *offload* **parcial** da computação móvel para a plataforma em nuvem representa uma abordagem bastante apropriada. De acordo com Kumar e Yung-Hsiang (2010), o *offload* de computação e dados é benéfico se e somente se grandes quantidades de computação e pequenas quantidades de dados ou comunicação são necessários.

Além do mais, um significante número de pesquisas em como determinar, da melhor forma possível, quando, e onde *offload* de computação e dados em aplicações móveis para dispositivos externos, e.g. computadores próximos, e serviços públicos que fornecem acesso a nuvem têm sido realizadas.

Hassan e Chen (2012) investigam como efetivamente aprimorar o tempo de resposta do usuário em aplicações móveis. Hassan e Chen conduzem experimentos envolvendo a execução de diferentes algoritmos sob o dispositivo móvel, sob computadores próximos, e sob servidores em nuvem. Também, Kumar e Yung-Hsiang (2010) apresentam um estudo em como *offload* em aplicações móveis, em certas circunstâncias pode, e em outras não, melhorar o consumo de energia. De acordo com Kumar e Yung-Hasiang, o *offload* de computação pode ser uma alternativa apropriada quando a quantidade de computação é significantemente maior que a quantidade de comunicação necessária. Miluzzo, Cáceres e Chen (2012) visionam dispositivos móveis como o componente central de arquiteturas em computação móvel na nuvem e apresentam *mCloud*, uma arquitetura para computação móvel em nuvem construída sob dispositivos móveis distintos, no entanto, integrados entre si.

3. ABORDAGEM

Nesta seção será apresentado uma visão geral do framework. Além do mais, partindo da visão geral, apresentaremos os constituintes de nosso framework em detalhes, como nosso meta-modelo, que reflete o domínio de aplicações móveis integradas com a nuvem, nosso gerador de código, e finalmente os diferentes papeis presentes em nosso framework.

3.1 O FRAMEWORK – UMA VISÃO GERAL

O framework emprega uma abordagem orientada a modelos, que é composta por um meta-modelo descrevendo os conceitos essenciais do domínio de aplicações móveis integradas com a nuvem, e um gerador de código para geração de código fonte não somente para a aplicação móvel em si, mas também para a aplicação servidora hospedada na plataforma em nuvem.

O MDCM é composto sobretudo por quatro partes, como apresentado na Figura 3 e descrito abaixo.

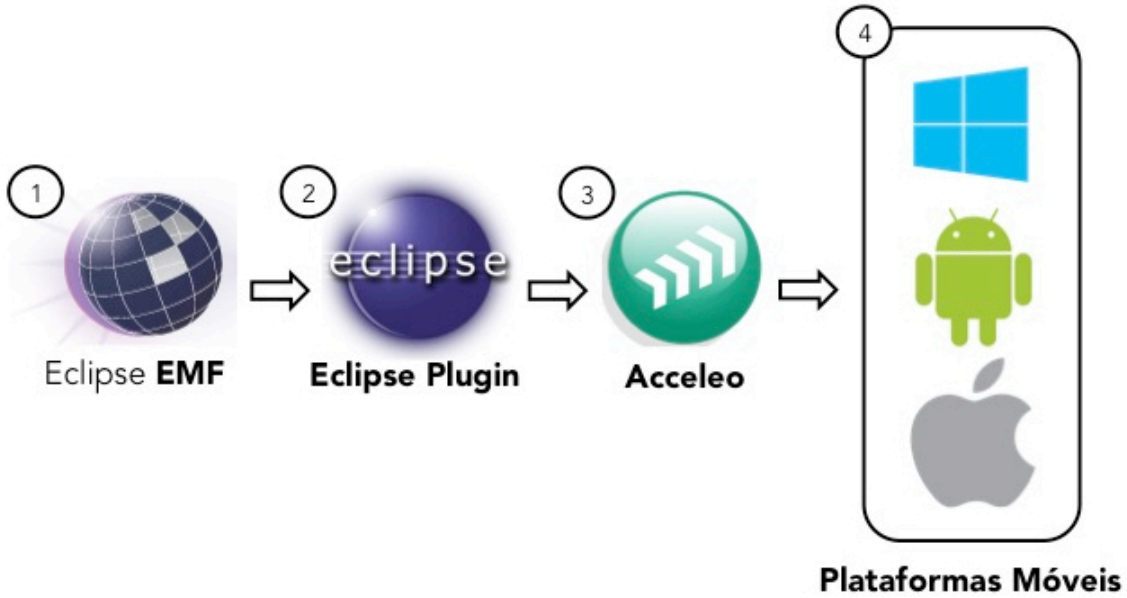


Figura 3 – Composição do MDCM. (Elaborado pelo Autor)

1. **Eclipse EMF:** nesta etapa, o meta-modelo (veja seção 3.2) é criado por meio do Framework de Modelagem do Eclipse

(Eclipse Modelling Framework, **EMF**). O EMF é instalado como um *plugin* da plataforma Eclipse e o processo de modelagem, por completo, é realizado por meio dos mecanismos oferecidos pelo *plugin*, como também pelo IDE. É importante ressaltar que esta etapa é apenas utilizada quando o meta-modelo é criado ou alterado. Desta forma, usualmente, o desenvolvedor começaria pela parte 2, o Eclipse Plugin.

2. **Eclipse Plugin:** uma vez que o meta-modelo é construído, um segundo *plugin* é criado. Este segundo *plugin* consiste de uma segunda instância da plataforma Eclipse que contem todas as entidades, atributos e ferramentas para manipulação do meta-modelo pré-carregados. Desta maneira, através desta segunda instância, desenvolvedores são capazes de modelar (criar um modelo) uma aplicação móvel integrada com a nuvem de acordo com o meta-modelo criado anteriormente.
3. **Acceleo:** nesta etapa, o gerador de código é projetado (veja seção 3.3). Por meio de um projeto do Acceleo, os templates para geração de código são criados. Uma vez que os templates são criados, o gerador está completo. Então, o projeto Acceleo é executado e recebe como entrada o modelo criado na parte 2.
4. **Plataformas Móveis:** uma vez que o projeto Acceleo é executado, código fonte para a plataforma móvel integrada com a nuvem é gerado.

MDCM auxilia desenvolvedores a especificar quando uma dada operação é executada *on-device*, e quando uma dada operação é enviada para a plataforma em nuvem (*on-cloud*) **por meio de um atributo de tipagem**, que é atribuído a operações da aplicação, mapeadas na etapa de modelagem da aplicação. Em consequência, desenvolvedores são capazes de realizar o *offload* de operações, quando o processamento no dispositivo móvel não é apropriado, simplesmente através da atribuição de um valor ao atributo de tipagem da operação específica.

O restante desta seção é organizada da seguinte forma: seção 3.2 apresenta o meta-modelo que representa entidades essenciais do domínio de aplicações móveis integradas com a nuvem, seção 3.3 apresenta as características de nosso gerador de código, e finalmente, seção 3.4 apresenta os diferentes papeis que estão envolvidos no ciclo de vida de nosso framework.

3.2 O META-MODELO

O meta-modelo do MDCM é construído sob o EMF e define elementos essenciais do domínio de aplicações móveis integradas com a nuvem. As entidades e atributos que constituem o meta-modelo proposto neste trabalho têm como base os trabalhos de Steiner et al. (2013) e Heitkötter, Majchrzak e Kuchen (2013). De acordo com Guan et al. (2011), aplicações móveis integradas com a nuvem visam aplicar técnicas em computação na nuvem para processamento e armazenamento de dados em dispositivos móveis. No entanto, o framework MDCM foca em aplicar técnicas de computação em nuvem, especialmente por meio de *offload* de operações, e não de dados. Operações que primariamente seriam executadas no dispositivo móvel em si, para plataformas em nuvem com o objetivo de aperfeiçoar a eficiência e resposta da tarefa a ser executada.

De acordo com Koch (2012), Eclipse é uma plataforma de desenvolvimento baseada em plugins que suporta uma variedade de linguagens de programação, e.g. Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Natural, Perl, PHP, Prolog, Python, R, Ruby, Scala, Clojure, Groovy, Scheme, e Erlang, e outras ferramentas. Conforme citado por Koch (2012), Eclipse apresenta características importantes:

a) Livre integração com outras tecnologias;
b) Vasto suporte;
c) Comunidade atualmente ativa;
d) Disponibilidade de plugins;
e) Suporte para múltiplas linguagens de programação e modelagem; e

f) Ampla adoção e uso por desenvolvedores em serviço.

Além disso, Eclipse e EMF, indiscutivelmente, tem se tornado os padrões na comunidade de engenharia orientada a modelos (*Model Driven Engineering, MDE)*, como também, na maioria das ferramentas MDE no presente mercado (KOLOVOS et al., 2010).

EMF é um framework de modelagem e mecanismo de geração de código para desenvolvimento de aplicações baseadas em um modelo de dados estruturado (NELLAIAPPAN; 2009). Portanto, devido a vasta popularidade e como mencionado, ferramentas disponíveis, Eclipse e EMF são as alternativas mais apropriadas para o desenvolvimento do meta-modelo para o framework proposto neste projeto.

Nosso meta-modelo é construído através do EMF, que por sua vez, utiliza o Ecore para construção dos domínios em questão. O Ecore corresponde a uma extensão do XML que utiliza *namespaces* para definir novas entidades e atributos. Não adicionamos a estrutura do nosso meta-modelo em Ecore neste capítulo por questões de espaço. No entanto, este pode ser encontrado no Apêndice A.

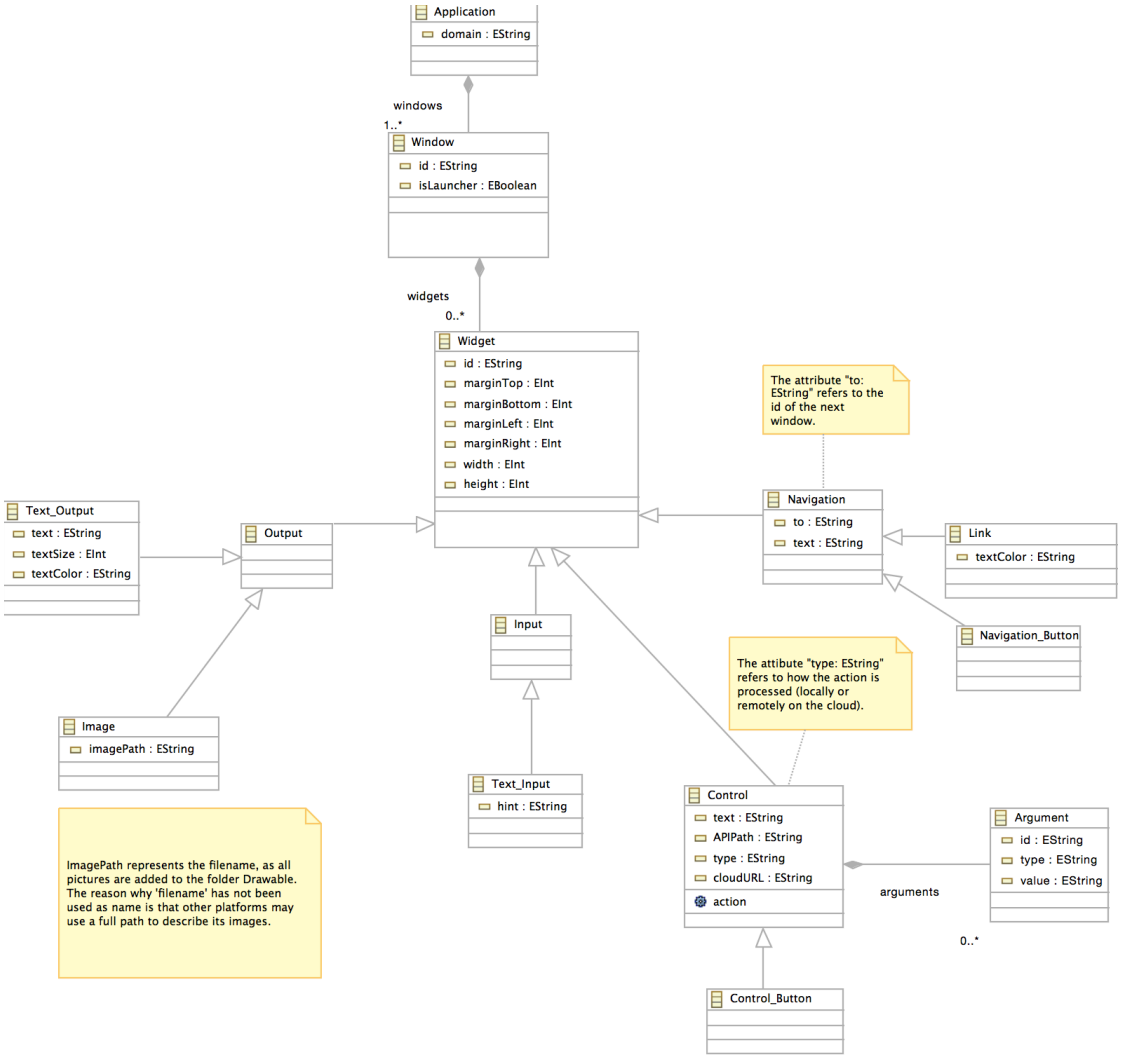


Figura 4 – Meta-modelo do framework MDCM. (Elaborado pelo Autor)

Como apresentado na Figura 4, aplicações móveis consistem de um grupo de interfaces, ou telas (HEITKÖTTER; MAJCHRZAK; KUCHEN, 2013). Além disso, o elemento *Window* contem zero ou mais elementos *Widgets*. *Widgets* são basicamente elementos gráficos de controle que podem ser utilizados e reutilizados em uma interface gráfica de usuário (*Graphical User Interface, GUI*). Elementos *Widgets* são estendidos por quatro elementos: (1) *Input*, elementos de entrada, (2) *Output*, elementos de saída, (3) *Control*, elementos de controle, e (4) *Navigation*, elementos de navegação, como descrito abaixo:

1. *Input*: representa todos os campos que o usuário pode utilizar para inserir dados em uma aplicação. *Input* é estendido por *Text*,

que se refere a componentes para inserção de dados alfanuméricos, e.g. datas, nomes, idades, e *Combo*, que se refere a inserção de dados originados de um conjunto de alternativas pré-definidas, também conhecida como combo-box.

2. *Output:* representa todos os campos que a aplicação pode utilizar para apresentar dados ao usuário. *Output* é herdado não somente de *Text*, que refere-se a elementos para inserção de dados alfanuméricos, como mencionado anteriormente, mas também de *Image*, que se refere a apresentação de dados gráficos, e.g. gráficos, imagens e ícones.
3. *Control*: representa elementos que um usuário pode utilizar para executar uma ação (e.g. submeter um formulário, atualizar uma página, inicializar aplicações e operações em background). *Control* é estendido pelo elemento *Button*, que se refere ao controlador do tipo botão, capaz de acionar operações por meio de cliques. É importante ressaltar que este elemento é responsável, também, por determinar se uma aplicação é executada *on-device* ou *on-cloud* por meio do atributo de tipagem *type*.
4. *Navigation*: embora *Navigation* também represente uma ação, se refere apenas a navegação entre janelas/telas. *Navigation* é estendida por dois elementos: *Button*, que se refere a um botão capaz de navegar de uma tela para outra, e *Link*, que corresponde a um texto/*hiperlink* para navegar entre telas.
5. *Argument*: representa uma argumento sendo enviado para a função executada pela entidade *Control*. Um *Argument* pode possuir dois tipos (*types*) de argumentos: *element* ou *literal.* Argumentos do tipo *element* compreendem uma outra entidade sendo enviada como parâmetro. Desta forma, o atributo *value* recebe o identificador (*id*) da entidade sendo enviada como parâmetro. Já argumentos do tipo *literal* recebem como atributo *value* um valor literal, que é enviado como uma *String* à função sendo executada.

O meta-modelo do framework proposto neste projeto também suporta integração com a nuvem em termos de *offload* de computação. *Widgets* do tipo *Control* estão encarregados de acionar operações, e.g. inicializar aplicações, disparar ações em jogos, entre outros. Embora dispositivos móveis têm evoluído significantemente em termos de performance e processamento, dispositivos móveis ainda podem apresentar limitações, especialmente quando comparados com computadores pessoais, em termos de capacidade de processamento, capacidade de armazenamento, e capacidade de bateria. Portanto, é proposto um meta-modelo que descrimina operações executadas *on-device* e *on-cloud* (*offload* de computação para a plataforma em nuvem). Assim, por exemplo, um usuário poderia efetuar uma operação de busca em um extenso arquivo de texto na nuvem, enviando o padrão sendo pesquisado para a nuvem, e a plataforma em nuvem em si processa a operação e retorna o número de ocorrências do padrão no arquivo de texto. Quando um elemento *Control* é modelado (veja o elemento *Control* na Figura 4), o atributo “type”, de tipo, pode receber os valores “local" e “remote" quando processado localmente, e remotamente, respectivamente.

Apesar de que nosso meta-modelo não suportar a modelagem de regras de negócios, atribuídos dois parâmetros na entidade *Control* visando proporcionar a integração do elemento com uma API contendo a regra de negócio para determinada função sendo invocada: (1) *API-Path* e (2) *cloudURL*, como descrito abaixo:

a) *APIPath:* especifica a API contida na implementação da operação sendo processada (invocada pelo método ligado ao *Control Button*). Caso o processamento seja realizado *on*-cloud, o arquivo da API deve ser hospedado na plataforma em nuvem também, portanto, o caminho atribuído deve representar a localização do arquivo de API não somente no sistema local móvel, mas também no sistema de arquivos na nuvem.
b) *cloudURL*: *cloudURL* é utilizado pela aplicação móvel para mapeamento da aplicação em nuvem visando comunicação com a nuvem.

Também, o meta-modelo do framework visa fornecer um modelo de desenvolvimento simples e flexível para aplicações móveis integradas com a nuvem. Assim, a única restrição dada no meta-modelo é a atribuição de identificadores para os componentes *Window* e *Widget*. Especificamente, os identificadores devem ser únicos (diferentes componentes *Window* e *Widgets* devem possuir identificadores distintos).

3.3 O GERADOR DE CÓDIGO

O gerador de código do framework proposto neste projeto é construído com base no Acceleo e requer uma instância do meta-modelo do MDCM como entrada para geração de código. Na atualidade, diferentes ferramentas para transformação de modelos orientadas a uma variedade de propósitos têm sido explorados na literatura, por exemplo, XSLT (Extensible Stylesheet Language Transformation), QVT (Query-View Transformation Language), ATL (ATLAS transformation language) e Acceleo (KOCH; 2007).

```
[comment encoding = UTF-8 /]
[module generate('http://pretcc.com')]


[template public generateElement(aMyClass : MyClass)]
[comment @main/]
[file (aMyClass.att1.concat('.java'), false, 'UTF-8')]
public class [aMyClass.att1/] {
    public static void main(String['['/]']'/] args) {
        System.out.println("Hello [aMyClass.att2/]!");
    }
}
[/file]
[/template]
```

Figura 5 – Exemplo de um *template* do Acceleo que gera uma classe Java, constituída por um método principal.

Acceleo[1] é uma implementação do padrão modelo para texto (MOF) dirigido pela Object Management Group (OMG). Acceleo fornece mecanismos simples e poderosos para desenvolvimento de geradores de código por meio de *templates*.

Templates são os motores para geração de código no Acceleo. Como apresentado na Figura 5, os *templates* são constituídos por dois tipos de sintaxes: (1) a sintaxe literal, e (2) sintaxe dinâmica. A sintaxe literal (em fonte de cor preta na Figura 5) diz respeito a tudo que será literalmente gerado no arquivo final, enquanto a sintaxe dinâmica (ou programada), escrita entre colchetes, refere-se ao aspecto dinâmico do gerador, isto é, processamento e utilização das entidades e atributos instanciadas no modelo. O *template* apresentado na Figura 5 gera uma classe simples em Java, constituída por o método principal *main*, como demonstrado abaixo:

```
public class MyClass {
        public static void main(String[] args) {
                System.out.println("Hello World!");
        }
}
```

O conteúdo da classe acima é gerada em um arquivo denominado *MyClass.java*, onde o nome do arquivo, como também o nome da classe, é um atributo dinâmico, como também, a palavra “World”, que está contida no atributo dinâmico *att2* em *aMyClass*.

Embora a figura 5 apresente uma visão simplificada do gerador de código para fins de demonstração, a versão completa do mesmo pode ser encontrada no Apêndice A.

Para atender os requisitos do framework proposto, Acceleo aparenta ser a alternativa mais apropriada, especialmente em termos de construção de gerador de código não somente pela simplicidade e facilidade de uso apresentada pela ferramenta, mas também devido a habilidade de automaticamente transformar modelos independentes de plataforma *(Platform-independent Model - PIM)* em código fonte (MTSWENI; 2012).

---

[1] http://www.eclipse.org/acceleo

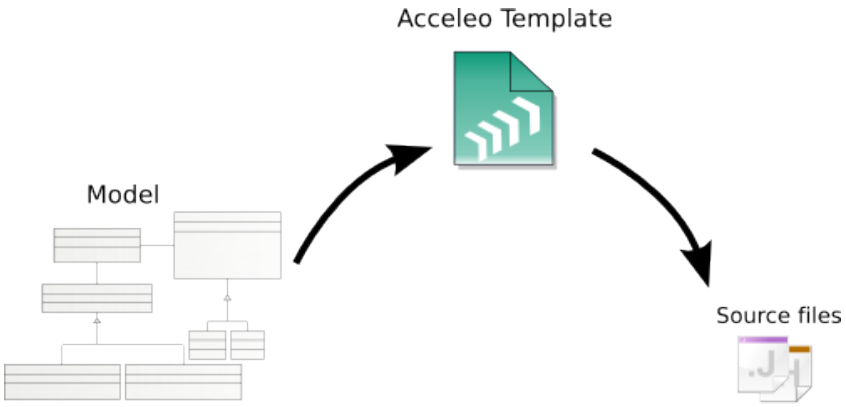


Figura 6 – Princípio genérico para geração de código com Acceleo[2].

Como citado na seção 3.1, para projetar um modelo do framework, um *plugin* do Eclipse é construído por meio de dois outros meta-modelos contidos no EMF: Ecore e Genmodel. Portanto, uma vez que o modelo é construído, o gerador de código gera o código fonte segundo o modelo usado como entrada. A Figura 6 ilustra o princípio geral da ferramenta Acceleo na geração de código.

O esquema adotado para integração dispositivo móvel e nuvem é baseado no esquema cliente-servidor (GUAN ET AL; 2011). Basicamente, o esquema servidor-cliente representa uma estrutura onde um ou mais clientes efetuam solicitações a um servidor, que retorna dados de acordo com o solicitado por clientes. Para os propósitos do framework, o esquema cliente servidor aparenta ser apropriado devido a sua simplicidade e eficácia.

Desta forma, para a integração dispositivo móvel nuvem, o gerador de código gera, também, código fonte para a aplicação em nuvem, que consequentemente deve ser compilado/executado no ambiente em nuvem. A aplicação executada na nuvem, então, é acessada pela aplicação executada no dispositivo móvel, quando invocando operações definidas como remotas.

---

[2] http://www.acceleo.org/pages/first-generator-module/en

3.4 OS PAPEIS

Como apresentado, o framework proposto envolve partes distintas, isto é, (1) criação do meta-modelo, (2) instância de um modelo através do *plugin*, (3) construção do gerador de código, (4) e geração da aplicação móvel em si, que juntas, resultam em nosso produto final.

No entanto, diferentes partes podem envolver diferentes papeis. Em outras palavras, três papeis diferentes estão presentes em nosso framework. A Figura 7 representa uma expansão da Figura 3, e atribui o papel presente em cada elemento de nosso framework. Os papeis presentes em nosso framework são:

a) Desenvolvedor do framework.
b) Desenvolvedor usuário do framework.
c) Usuário das aplicações geradas pelo framework.

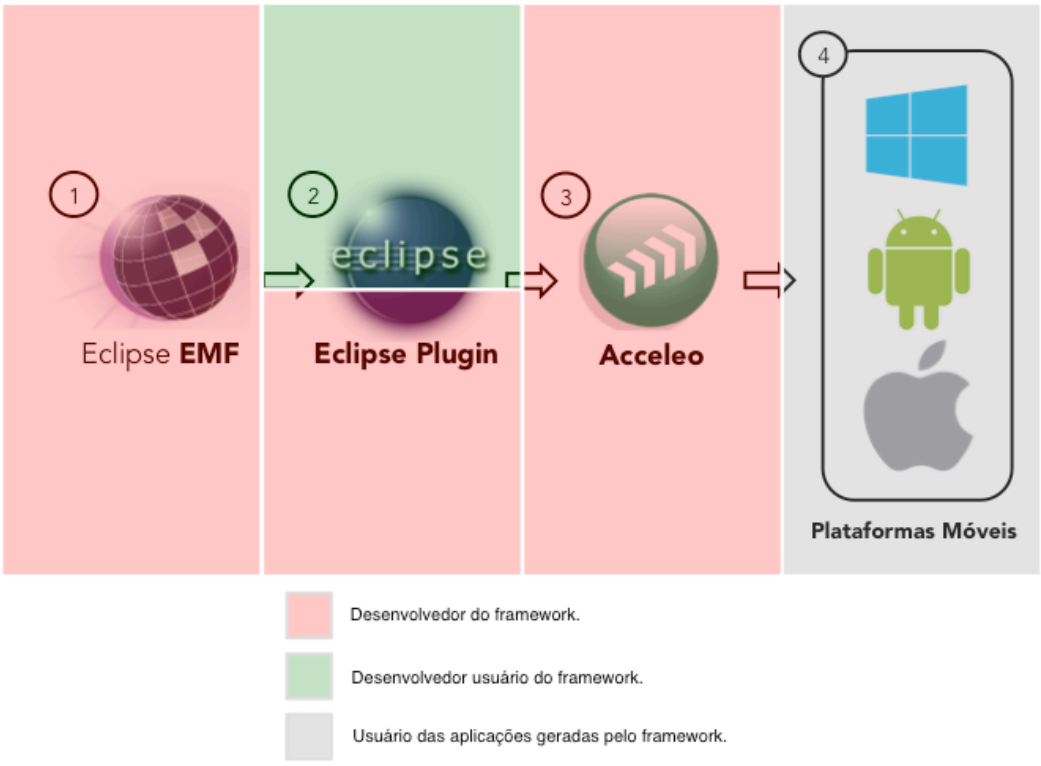


Figura 7 – Diferentes papeis presentes em nosso framework. (Elaborado pelo Autor)

Primeiramente, referente a área vermelha da Figura 7, o desenvolvedor do framework está presente nas etapas de criação do meta-modelo, criação do *plugin*, e construção do gerador de código. Este papel compreende o mais baixo nível na hierarquia em termos de complexidade de suas ações. Este papel também é responsável por futuras extensões de nosso framework. Exemplificando, uma extensão de nosso framework poderia se dar na expansão de nosso meta-modelo, isto é, na criação de novas entidades e atributos visando representar um domínio de aplicações diferente ou mais específico em relação ao proposto no presente framework.

O segundo papel, referente a área verde da Figura 7, compreende o usuário do framework, cujo qual representa nosso público alvo. Este papel representa o desenvolvedor que utiliza nosso framework para desenvolvimento, isto é, geração de aplicações móveis integradas com a nuvem. Por sua vez, está presente somente na etapa de instância do modelo, que ocorre sob o *plugin* previamente construído. Em outras palavras, este usuário basicamente cria um modelo de aplicação, e o executa, o que resulta na geração do código da aplicação. É importante ressaltar que este papel não está envolvido na criação ou expansão do meta-modelo, e não está envolvido no projeto do gerador de códigos, tampouco requer o conhecimento necessário para tais. Também, é importante notar que o papel desenvolvedor do framework também está presente neste passo, no entanto, como o criador do *plugin*, e não como usuário.

Finalmente, referente a área cinza da Figura 7, o usuário das aplicações geradas pelo framework corresponde ao usuário “final”, ou comum, que não possui nenhum conhecimento sob o desenvolvimento de software ou afins, e simplesmente faz uso do produto final gerado por nosso framework.

4. APLICAÇÃO EXPERIMENTAL

Nesta seção será apresentado uma visão geral do experimento. A visão geral é constituída pelos principais pontos abordados no experimento e apresenta tais pontos de forma teórica. Também, neste seção apresentamos um *walkthrough* que contêm cada passo realizado para a construção de nosso experimento.

4.1 O EXPERIMENTO – UMA VISÃO GERAL

Para avaliar a abordagem proposta, criamos uma aplicação móvel integrada com a nuvem utilizando o framework apresentado neste trabalho. Além do mais, adotamos experimentalmente um dispositivo móvel para instalação da aplicação gerada, e um servidor em nuvem, para que assim pudéssemos efetuar operações distintas visando ilustrar diferentes cenários o qual *offload* de computação pode ser satisfatório ou não satisfatório.

Tabela 2 – Especificações do dispositivo móvel e plataforma em nuvem adotadas para o experimento de avaliação.

	Dispositivo Móvel	**Plataforma Cloud**
Sistema Operacional	Android OS, versão 4.3	Ubuntu 11.04
CPU	Quad-core 1.5 GHz Krait	2 vCPUs, 2.5 GHz, Intel Xeon Family
Armazenamento	32 GB	8 GB
RAM	2 GB	4 GB
Wifi	802.11 a/b/g/n, dual-band	-

Fonte: Elaborado pelo Autor.

O dispositivo móvel adotado foi um tablet Nexus 7 e o servidor em nuvem adotado foi o *Amazon EC2*. As especificações do dispositivo móvel e do serviço em nuvem são apresentados na Tabela 2.

Além do mais, como Hassan e Chen (2012), efetuamos não somente experimentos em *offload* de computação, mas também *offload* de dados, uma vez que *offload* de dados, em certos casos, precisa ser realizado.

A operação efetuada no presente experimento é uma simples busca de texto que utiliza um algoritmo baseado no algoritmo de busca de texto apresentado por Hassan e Chen (2012). O algoritmo requer um padrão de texto e um arquivo de texto e procura por ocorrências do padrão no mencionado arquivo de texto fonte através de comparação de caracteres. Neste experimento, usaremos não somente um, mas até três padrões de texto a serem buscados no arquivo de texto fonte (veja Apêndice D para mais informações sobre o algoritmo).

Por sua vez, o arquivo de texto fonte é composto por cadeias de caracteres formadas por meio de combinações dos caracteres abc123. O tamanho das cadeias de caracteres variam entre 1 e 9. Portanto, 12093234 combinações são geradas. Tal arquivo de texto fonte foi gerado com o auxílio da ferramenta *crunch*[3], geralmente empregada para ataques de força bruta.

Os ambientes experimentais foram definidos como demonstrado abaixo:

a) ***Computação "on-device"*:** o arquivo de texto é armazenado no sistema de armazenamento do dispositivo móvel e o processamento também é realizado no dispositivo móvel.
b) ***Offload de computação*:** Realiza-se apenas o *offload* da computação. Neste cenário, o arquivo de texto é armazenado na plataforma em nuvem, então, apenas o padrão sendo procurado é enviado para a instância na nuvem (mas não o arquivo de texto).
c) ***Offload de computação + dados:*** realiza-se o *offload* de computação e dados neste cenário. Este cenário compreende a uma extensão do cenário anterior, no entanto, incluímos o *overhead* do envio do arquivo de texto fonte para nosso servidor em nuvem. É importante ressaltar, como mencionado na seção 3.2, nosso framework foca no *offload* de computação/operação, e não o *offload* de dados. Desta forma, realizamos o envio do arquivo de texto em uma rotina separada.

[3] http://sourceforge.net/projects/crunch-wordlist/

A aplicação gerada é composta por uma única tela. Como demonstrado na Figura 8, a aplicação recebe até três parâmetros através de campos de texto, que por sua vez correspondem aos padrões de texto sendo pesquisados no arquivo de texto fonte. Abaixo, na tela, encontram-se dois rótulos que, respectivamente, compreendem o resultado total das ocorrências do(s) padrão(ões) de texto no arquivo de texto fonte e o tempo total despendido para realização do processamento. Finalmente, no rodapé da tela existe um botão que aciona a operação de busca de texto, usando como parâmetro os padrões de texto fornecidos através dos campos de texto.

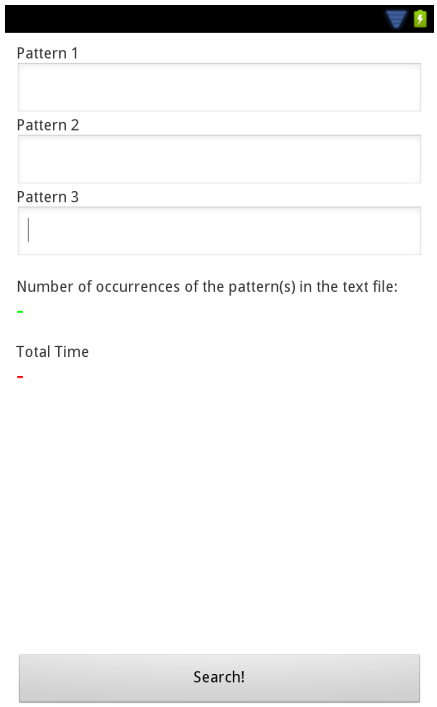


Figura 8 – Captura de tela da aplicação gerada para avaliação experimental do framework.

Uma vez que a operação é acionada, a aplicação pode:

a) Executar a operação de busca localmente (*on*-device).
b) Executar a operação de busca em nuvem (*on-cloud*), no entanto, não enviando o arquivo de texto para a plataforma em nuvem. Neste caso, o arquivo é previamente armazenado na plataforma

em nuvem para que assim a operação possa utilizá-lo sem a necessidade do *offload* de dados.

c) Enviar o arquivo de texto para a nuvem e em seguida, executar a operação de busca na plataforma em nuvem.

Os casos mencionados acima refletem nos ambientes experimentais citados anteriormente. Novamente, todos os três casos foram avaliados e analisados. A seção 5 apresenta a análise completa dos dados, enquanto a próxima seção apresenta um *walkthrough* do experimento, apresentado detalhadamente cada passo do mesmo.

4.2 WALKTHROUGH DO EXPERIMENTO

Esta seção corresponde a descrição passo a passo de nosso experimento, incluindo detalhes práticos de cada passo. As subseções refletem os passos para a geração do produto final: a aplicação móvel conectada com a nuvem. As subseções listadas são: geração do arquivo de texto fonte, construção do modelo da aplicação, interação do modelo com o gerador de código, e *offload* da operação e dados, o que inclui detalhes em como o *offload* de computação/operação e dados foi realizado em nosso experimento.

4.2.1 GERAÇÃO DO ARQUIVO DE TEXTO FONTE

O arquivo de texto fonte utilizado como base para a busca dos padrões foi gerado com o auxílio da ferramenta *crunch*, que é capaz de gerar listas de palavras oriundas de combinações de caracteres de acordo com os requisitos do usuário em termos de tamanho, e conteúdo.

Como mencionado na seção 4.1, o tamanho das palavras variam entre 1 e 9, sendo constituídas pelas combinações dos caracteres abc123. Portanto, 12093234 combinações/palavras são geradas.

A Figura 9 apresenta o uso da ferramenta para geração do arquivo de texto fonte de acordo com os parâmetros anteriormente citados.

```
Robertos-MacBook-Pro:crunch-3.6 robertoelerojunior$ ./crunch 1 9 abc123 -o wordlist
Crunch will now generate the following amount of data: 118513704 bytes
113 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 12093234

crunch: 100% completed generating output
```

Figura 9 – Gerando o arquivo de texto fonte com a ferramenta *crunch*.

Como apresentado na Figura 9, um arquivo nomeado *wordlist*, com o tamanho de 113MB, contendo 12093234 linhas é gerado.

4.2.2 CONSTRUÇÃO DO MODELO DA APLICAÇÃO

Visando construir o modelo da aplicação, devemos considerar que nosso meta-modelo, apresentado na seção 3.2, foi previamente construído e se encontra pronto para utilização. O primeiro passo a ser dado para a construção de nosso modelo, que compreende a modelagem da aplicação a ser gerada, é a criação do *plugin* (veja parte 2 da seção 3.2), que será capaz de construir um modelo que possui como base nosso meta-modelo, previamente construído.

Para criar o *plugin*, devemos executar nosso projeto EMF, que contém nosso meta-modelo, como uma aplicação Eclipse. Tal ação ira resultar em uma nova instância do Eclipse, o *plugin* em sí, que é armazenada em uma pasta separada da instalação original do Eclipse, e pode ser compartilhada para outros desenvolvedores quando necessário.

A Figura 10 apresenta graficamente os passos da criação do *plugin* que irá modelar nossa aplicação móvel conectada com a nuvem. Como demonstrado na imagem, clica-se o botão direito do mouse no projeto EMF que possui nosso meta-modelo, em seguida selecionamos o menu "*Run As*" (Executar Como) para executar nosso projeto como uma aplicação Eclipse (*Eclipse Application*). Uma vez que o menu "*Eclipse Application*" é acionado, uma nova instância do Eclipse é criada, executada, e armazenada em uma pasta dentro do *workspace* do Eclipse. Desta forma, o desenvolvedor usuário de nosso framework poderá receber diretamente o *plugin*, sem se preocupar

com o projeto EMF que contém nosso modelo, tampouco com o passo anterior (geração do plugin).

No entanto, se o desenvolvedor usuário de nosso framework planeja estender nosso meta-modelo, o projeto EMF se faz necessário, como também, a geração do *plugin*, uma vez que o *plugin* deve ser atualizado para refletir as propriedades listadas no meta-modelo.

Com o *plugin* gerado, partiremos para a construção do modelo de nossa aplicação. Primeiro, criamos um projeto Android que corresponde ao alvo de nosso gerador de código (apresentado na próxima seção), isto é, onde os códigos gerados residirão. O modelo é construído dentro da aplicação alvo, e posteriormente é mapeado no gerador de código. A construção do modelo se dá de forma simples: basta usar o *wizard* para criação de arquivos no Eclipse e buscar nosso meta-modelo na pasta "*Example EMF Model Creation Wizards*" (Wizards para criação de modelo de exemplo EMF), como apresentado na Figura 11. Este arquivo de modelagem (*MDCM Model*) só é apresentado no *wizard* porque geramos o *plugin* sob o nosso projeto EMF, que contém nosso meta-modelo, fornecendo os artefatos necessários para criação de uma instância de si mesmo, isto é, o modelo. Em outras palavras, o Eclipse por si só não é capaz de criar modelos baseados em nosso meta-modelo.

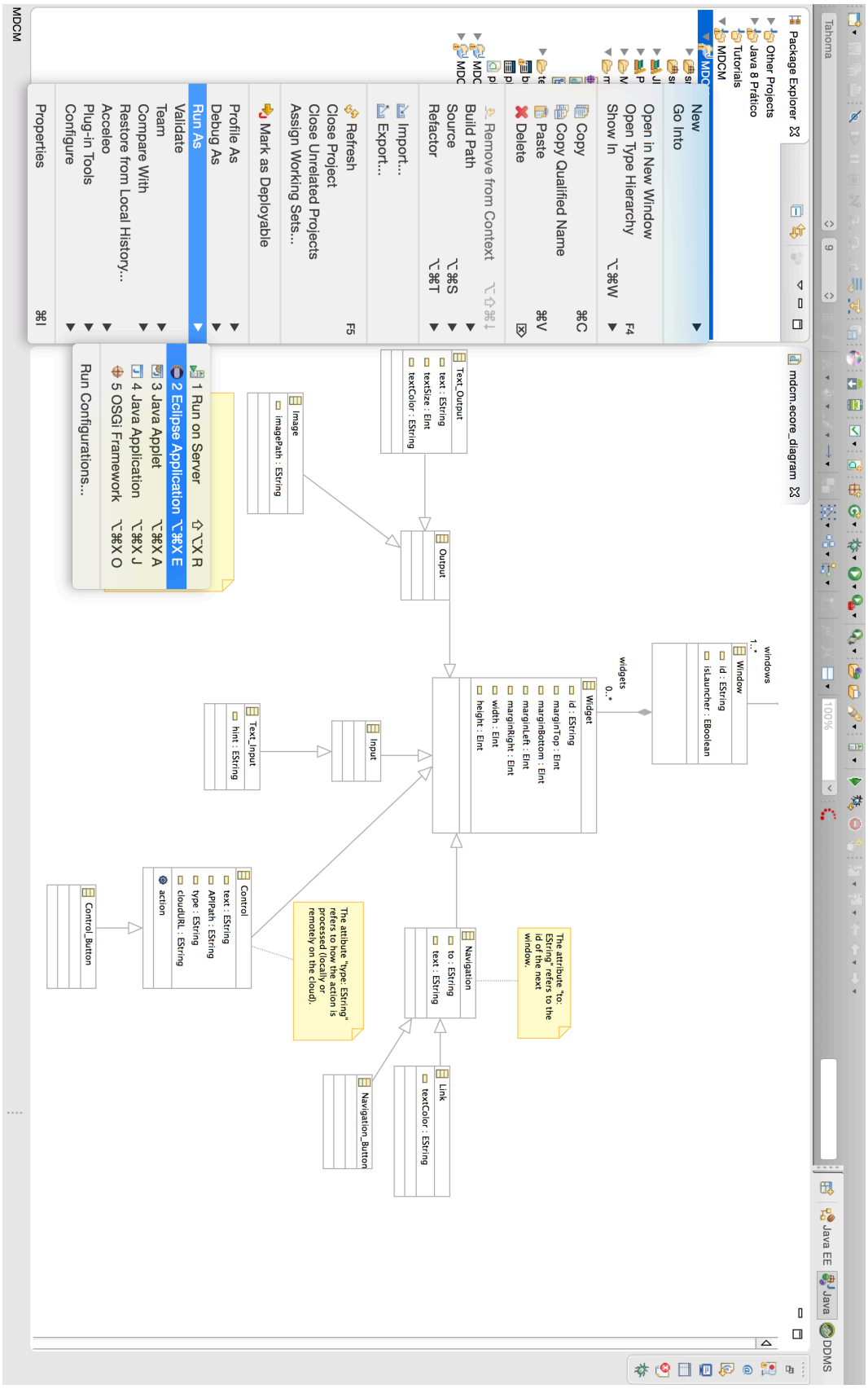


Figura 10 – Gerando um *plugin* para modelagem da aplicação móvel conectada com a nuvem a ser gerada.

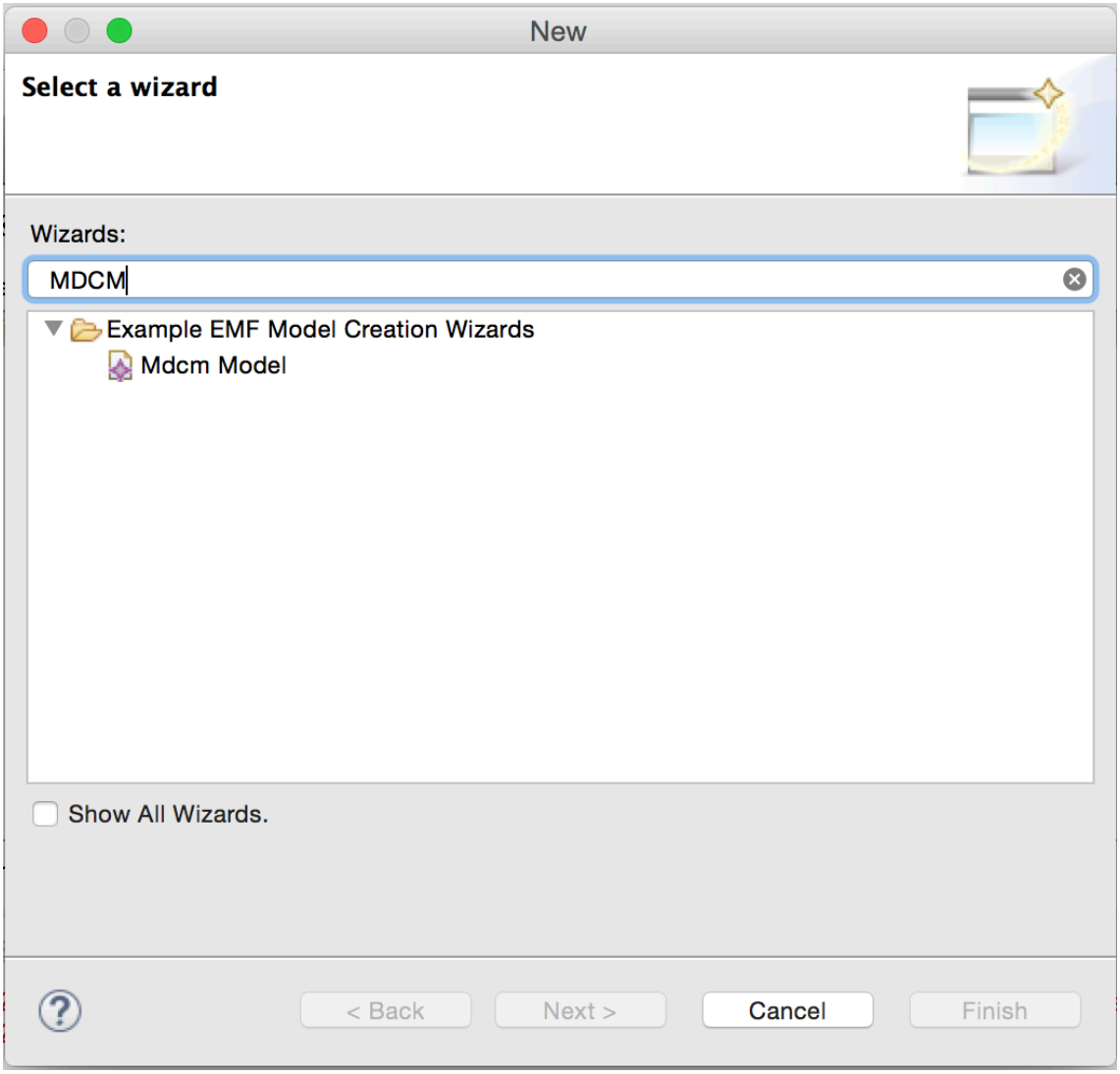


Figura 11 – *Wizard* de criação de arquivos do Eclipse utilizado para criação do arquivo de modelagem que reflete no meta-modelo.

A manipulação do arquivo de modelagem ocorre de forma simples. Tal arquivo é construído utilizando a linguagem XML[4], no entanto, o *plugin* fornece um mecanismo de construção que utiliza uma estrutura em árvore, tornando a criação de elementos e a definição de seus atributos simples e eficaz. A figura 12 demonstra o arquivo XML, e sua respectiva visão em árvore fornecida pelo *plugin*.

[4] http://www.w3.org/XML/

Figura 12 – Arquivo de modelagem XML em contraste com sua representação estruturada em árvore.

A estrutura em árvore mostra-se útil uma vez que a criação do código se da pela interface. Em outras palavras, o usuário cria a estrutura em árvore criando elementos “filhos” em elementos “pais”, e para cada elemento, um conjunto de atributos pode ser definido. Tal hierarquia e conjunto de atributos reflete em nosso meta-modelo, como mencionado anteriormente.

Finalmente, abaixo, apresentamos tabelas com os valores de cada atributo contido nos elementos presentes em nosso modelo (para informações sobre as entidades e atributos, veja seção 3.2).

Tabela 3 – Elemento “com.mdcm.simpletextsearch” de nosso modelo junto com seus respectivos atributos e valores.

Application com.mdcm.simpletextsearch	
Atributo	**Valor**
Domain	*com.mdcm.simpletextsearch*

Fonte: Elaborado pelo Autor.

Tabela 4 – Elemento “home” de nosso modelo junto com seus respectivos atributos e valores.

Window home	
Atributo	**Valor**
Id	*home*
IsLauncher	*true*

Fonte: Elaborado pelo Autor.

Tabela 5 – Elemento “pattern1_label” de nosso modelo junto com seus respectivos atributos e valores.

Text Output pattern1_label	
Atributo	**Valor**
Id	*pattern1_label*
Height	*auto*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*0*
Text	*Pattern 1*
Text Color	*#000*
Text Size	*10*
Width	*auto*

Fonte: Elaborado pelo Autor.

Tabela 6 – Elemento “pattern1_input” de nosso modelo junto com seus respectivos atributos e valores.

Text Input pattern1_input	
Atributo	**Valor**
Id	*pattern1_input*
Height	*30*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*15*
Width	*300*

Fonte: Elaborado pelo Autor.

Tabela 7 – Elemento “pattern2_label” de nosso modelo junto com seus respectivos atributos e valores.

Text Output pattern2_label	
Atributo	**Valor**
Id	*pattern2_label*
Height	*auto*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*50*
Text	*Pattern 2*
Text Color	*#000*
Text Size	*10*
Width	*auto*

Fonte: Elaborado pelo Autor.

Tabela 8 – Elemento “pattern2_input” de nosso modelo junto com seus respectivos atributos e valores.

Text Input pattern2_input	
Atributo	**Valor**
Id	*pattern2_input*
Height	*30*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*65*
Width	*300*

Fonte: Elaborado pelo Autor.

Tabela 9 – Elemento “pattern3_label” de nosso modelo junto com seus respectivos atributos e valores.

Text Output pattern3_label	
Atributo	**Valor**
Id	*pattern3_label*
Height	*auto*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*110*
Text	*Pattern 3*
Text Color	*#000*
Text Size	*10*
Width	*auto*

Fonte: Elaborado pelo Autor.

Tabela 10 – Elemento “pattern3_input” de nosso modelo junto com seus respectivos atributos e valores.

Text Input pattern3_input	
Atributo	**Valor**
Id	*pattern3_input*
Height	*30*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*125*
Width	*300*

Fonte: Elaborado pelo Autor.

Tabela 11 – Elemento “n_occurences” de nosso modelo junto com seus respectivos atributos e valores.

Text Output n_occurences	
Atributo	**Valor**
Id	*n_occurences*
Height	*auto*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*160*
Text	*Number of occurences of the pattern(s) in the text file:*
Text Color	*#000*
Text Size	*10*
Width	*auto*

Fonte: Elaborado pelo Autor.

Tabela 12 – Elemento “occurences” de nosso modelo junto com seus respectivos atributos e valores.

Text Output occurences	
Atributo	**Valor**
Id	*occurences*
Height	*auto*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*170*
Text	*-*
Text Color	*#66CD00*
Text Size	*10*
Width	*auto*

Fonte: Elaborado pelo Autor.

Tabela 13 – Elemento “total_time_label” de nosso modelo junto com seus respectivos atributos e valores.

Text Output total_time_label	
Atributo	**Valor**
Id	*total_time_label*
Height	*Auto*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*190*
Text	*Total time:*
Text Color	*#000*
Text Size	*10*
Width	*auto*

Fonte: Elaborado pelo Autor.

Tabela 14 – Elemento “total_label” de nosso modelo junto com seus respectivos atributos e valores.

Text Output total_label	
Atributo	**Valor**
Id	*total_label*
Height	*auto*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Margin Top	*200*
Text	*-*
Text Color	*#FF0000*
Text Size	*10*
Width	*auto*

Fonte: Elaborado pelo Autor.

Tabela 15 – Elemento “search_button” de nosso modelo junto com seus respectivos atributos e valores.

Control Button search_button	
Atributo	**Valor**
Id	*search_button*
Height	*50*
Width	*300*
Margin Top	*380*
Margin Bottom	*0*
Margin Left	*0*
Margin Right	*0*
Text	*Search!*
Type	*remote/local*
API Path	*/Users/robertoelerojunior/Development/simpletextsearch.jar*
Cloud URL	*ec2-54-213-222-225.us-west-2.compute.amazonaws.com*

Fonte: Elaborado pelo Autor.

Tabela 16 – Elemento “argument_pattern1” de nosso modelo junto com seus respectivos atributos e valores.

Text Output argument_pattern1	
Atributo	**Valor**
Id	*argument_pattern1*
Type	*element*
Value	*pattern1_input*

Fonte: Elaborado pelo Autor.

Tabela 17 – Elemento “argument_pattern2” de nosso modelo junto com seus respectivos atributos e valores.

Text Output argument_pattern2	
Atributo	**Valor**
Id	*argument_pattern2*
Type	*element*
Value	*pattern2_input*

Fonte: Elaborado pelo Autor.

Tabela 18 – Elemento “argument_pattern3” de nosso modelo junto com seus respectivos atributos e valores.

Text Output argument_pattern3	
Atributo	**Valor**
Id	*argument_pattern3*
Type	*element*
Value	*pattern3_input*

Fonte: Elaborado pelo Autor.

Tabela 19 – Elemento “argument_pattern3” de nosso modelo junto com seus respectivos atributos e valores.

Text Output argument_pattern3	
Atributo	**Valor**
Id	*wordlist*
Type	*literal*
Value	*wordlist*

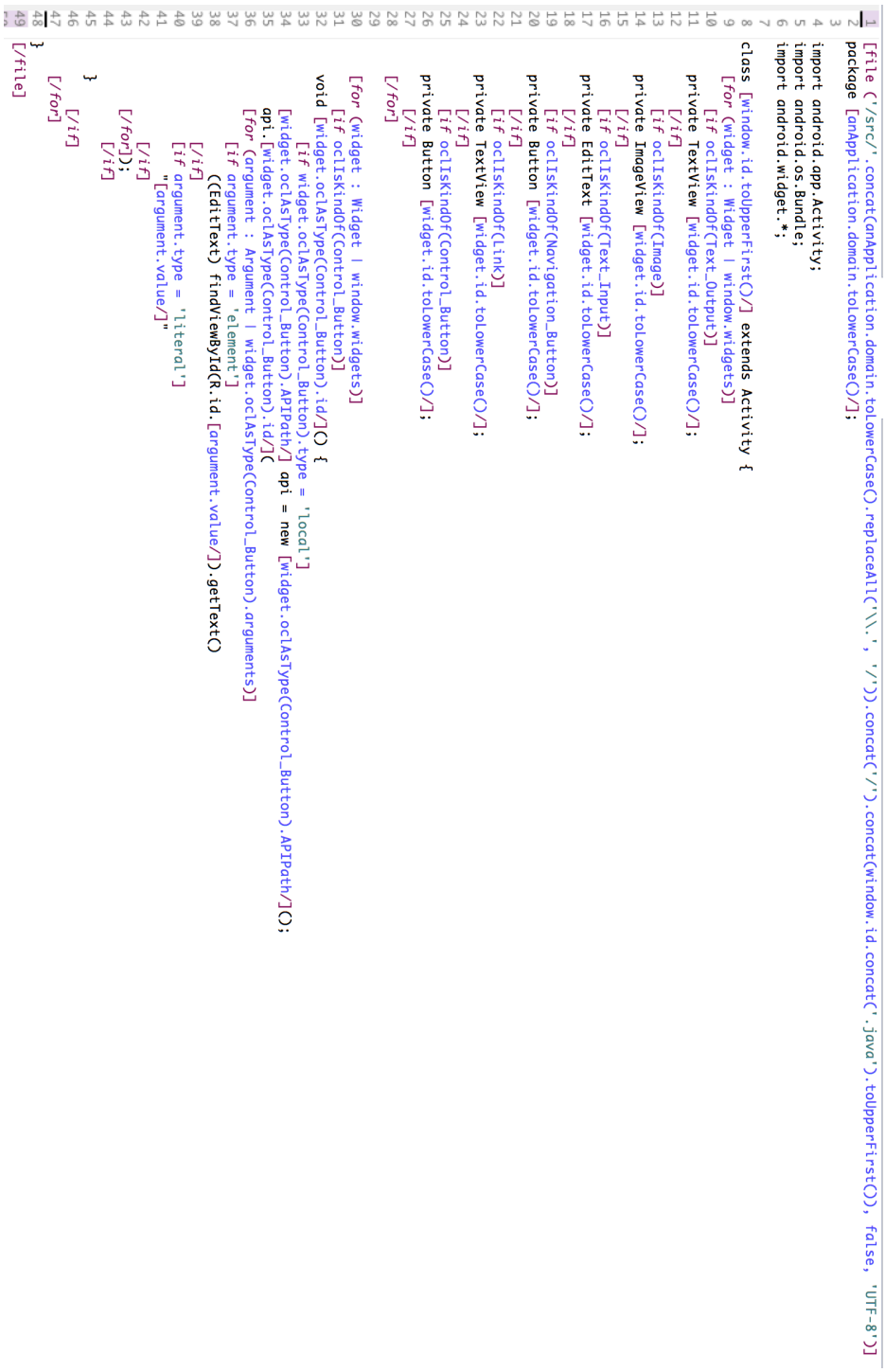
Fonte: Elaborado pelo Autor.

A próxima subseção descreve como o modelo e o gerador de códigos interagem para que a aplicação reflita a modelagem realizada.

4.2.3 INTEGRAÇÃO DO MODELO COM O GERADOR DE CÓDIGO

Na subseção anterior, concluímos nosso modelo e estamos prontos para partir para a geração de código fonte. É importante ressaltar que nosso publico alvo, o desenvolvedor usuário de nosso framework, não precisará mais se importar com modelagem ou a construção do gerador de código, isto é, o mesmo já estará pronto para “executar” o modelo e obter código fonte do produto final, a aplicação móvel conectada com a nuvem.

Como mencionado, nosso gerador de código trabalha com *templates*. Para demonstrar como o modelo previamente construído e o *template* interagem entre si, iremos utilizar um *snippet* da versão completa de nosso *template* (apresentado no Apêndice B) que é utilizado para gerar uma classe parcial que representa uma tela em Android. A Figura 13 representa este *snippet* do *template*.



```
[file ('/src/'.concat(anApplication.domain.toLowerCase().replaceAll('\\.', '/')).concat('/').concat(window.id.concat('.java').toUpperFirst()), false, 'UTF-8']
package [anApplication.domain.toLowerCase()/];

import android.app.Activity;
import android.os.Bundle;
import android.widget.*;

class [window.id.toUpperFirst()/] extends Activity {
    [for (widget : Widget | window.widgets)]
    [if oclIsKindOf(Text_Output)]
    private TextView [widget.id.toLowerCase()/];
    [/if]
    [if oclIsKindOf(Image)]
    private ImageView [widget.id.toLowerCase()/];
    [/if]
    [if oclIsKindOf(Text_Input)]
    private EditText [widget.id.toLowerCase()/];
    [/if]
    [if oclIsKindOf(Navigation_Button)]
    private Button [widget.id.toLowerCase()/];
    [/if]
    [if oclIsKindOf(Link)]
    private TextView [widget.id.toLowerCase()/];
    [/if]
    [if oclIsKindOf(Control_Button)]
    private Button [widget.id.toLowerCase()/];
    [/if]
    [/for]

    [for (widget : Widget | window.widgets)]
    [if oclIsKindOf(Control_Button)]
    void [widget.oclAsType(Control_Button).id/]() {
        [if widget.oclAsType(Control_Button).type = 'local']
        [widget.oclAsType(Control_Button).APIPath/] api = new [widget.oclAsType(Control_Button).APIPath/]();
        api.[widget.oclAsType(Control_Button).id/](
        [for (argument : Argument | widget.oclAsType(Control_Button).arguments)]
            [if argument.type = 'element']
            (EditText) findViewById(R.id.[argument.value/]).getText()
            [/if]
            [if argument.type = 'literal']
            "[argument.value/]"
            [/if]
        [/for]);
        [/if]
    }
    [/if]
    [/for]
}
[/file]
```

Figura 13 - *Snippet* da versão completa de nosso *template* que representa a geração parcial de uma classe em Android.

Na seção 3.3, entendemos que o *template* é constituído por dois tipos de sintaxes, a literal e a dinâmica. A dinâmica sempre inicia-se com colchetes e contém uma instrução programática a ser realizada. Para tornar o entendimento do *snippet* apresentado na Figura 13, iremos abordar cada ponto

do *snippet* que apresente sintaxes dinâmicas, explicando com detalhes como cada uma delas irá retornar o valor esperado em nosso produto final. Cada ponto será referenciado pelo número da linha correspondente no *snippet*. Em algumas linhas, a sintaxe dinâmica não sofre alteração, portanto, tais linhas não serão levadas em consideração.

Linha 1: esta instrução cria um arquivo com o caminho especificado no argumento entre parênteses. Nesse caso, o caminho é baseado no atributo "domain" de "Application", onde os pontos são substituídos por barras (representando um diretório), com adição de "/src/" e ".java" no final do caminho, o que especifica uma classe em Java.

Linha 2: retorna o valor *domain* da *Application* em caixa baixa.

Linha 8: retorna o id capitalizado da *Window*. Este valor corresponde ao nome da classe, que também é utilizado na instrução da linha 1.

Linha 9: estrutura de repetição que itera por todos os *Widgets* contidos em uma *Application*, e usa como identificador de cada um o atributo *widget*.

Linha 10: estrutura condicional que verifica o tipo do *widget* sendo iterado. Neste caso, verifica se o tipo é *Text Output*. Se a condição é satisfeita, o corpo da instrução é realizado (até o fechamento da estrutura).

Linha 11: retorna o *id* do widget em caixa baixa.

Linha 13: estrutura condicional que verifica o tipo do *widget* sendo iterado. Neste caso, verifica se o tipo é *Image*.

Linha 16: estrutura condicional que verifica o tipo do *widget* sendo iterado. Neste caso, verifica se o tipo é *Text Input*.

Linha 19: estrutura condicional que verifica o tipo do *widget* sendo iterado. Neste caso, verifica se o tipo é *Navigation Button*.

Linha 22: estrutura condicional que verifica o tipo do *widget* sendo iterado. Neste caso, verifica se o tipo é *Link*.

Linha 22: estrutura condicional que verifica o tipo do *widget* sendo iterado. Neste caso, verifica se o tipo é *Link*.

Linha 23: estrutura condicional que verifica o tipo do *widget* sendo iterado. Neste caso, verifica se o tipo é *Control Button*.

Linha 32: realiza um *casting* do tipo *Widget* para o tipo *Control Button* e retorna seu atributo *id*.

Linha 33: verifica se o atributo *type* do *Control Button* é *local*.

Linha 34: retorna o valor *APIPath* de *Control Button*.

Linha 36: estrutura de repetição que itera por todos os elementos *Argument* contidos no elemento *Control Button*.

Linha 37: verifica se o elemento *Argument* possui o atributo *type* setado para *element.*

Linha 38: retorna o atributo *value* do elemento *Argument.*

Linha 40: verifica se o elemento *Argument* possui o atributo *type* setado para *literal.*

O resultado do *snippet* é uma classe parcial para Android, que representa uma tela e controla a interação entre a interface e a regra de negócio. A Figura 14 apresenta o código fonte gerado do *snippet* anteriormente apresentado.

```
1   package com.mdcm.simpletextsearch;
2
3   import android.app.Activity;
4   import android.os.Bundle;
5   import android.widget.*;
6
7   class Home extends Activity {
8       private TextView pattern1_label;
9       private EditText pattern1_input;
10      private TextView pattern2_label;
11      private EditText pattern2_input;
12      private TextView pattern3_label;
13      private EditText pattern3_input;
14      private TextView n_occurences_label;
15      private TextView occurences_label;
16      private TextView total_time_label;
17      private TextView total_label;
18      private Button search_button;
19
20      void search_button() {
21          simpletextsearch api = new simpletextsearch();
22          api.search_button(((EditText) findViewById(R.id.pattern1_input).getText(),
23                  ((EditText) findViewById(R.id.pattern2_input).getText(),
24                  ((EditText) findViewById(R.id.pattern3_input).getText(),
25                  "wordlist");
26      }
27  }
```

Figura 14 – Código gerado a partir do *snippet* apresentado na Figura 13.

Embora a classe Android não esteja completa, anexamos a versão completa de nosso gerador no Apêndice B, e a versão completa do código gerado no Apêndice C.

Também, como apresentado na seção 3.2, nosso meta-modelo não suporta a modelagem de regra de negócios, portanto, para referência da tal, se faz o uso do atributo *APIPath* na entidade *Control*. Os arquivos de implementação contidos na API do presente experimento, que por sua vez é referenciada não somente pela aplicação móvel mas também pela aplicação em nuvem, como veremos a seguir, foi adicionada no Apêndice D.

Finalmente, também geramos o código para a aplicação em nuvem na qual a aplicação móvel irá invocar. Tal aplicação se trata de um servidor em PHP que recebe os parâmetros da aplicação via método *GET*, como apresentado na Figura 15.

```php
<?php
    $api = $_GET['api'];
    $param = $_GET['param'];
    $script = "java -jar ".$api." ".$param;

    $out = shell_exec($script);

    echo $out;
?>
```

Figura 15 – Código gerado para a plataforma em nuvem no qual será invocado pela aplicação móvel.

A aplicação servidora recebe o parâmetro *api*, e o parâmetro *param*, e cria um *script* com ambos parâmetros para serem executados na plataforma em nuvem e em seguida imprimi o valor retornado pelo *script*. Neste caso, estamos considerando uma API Java (.jar). O *script* executa o comando *java,* que é responsável por executar aplicações em Java, e em seguida o comando *–jar*, que espera um arquivo .jar a ser executado, o parâmetro *api* corresponderá ao nome da API sendo executada, e o parâmetro corresponde ao parâmetro real enviado para o algoritmo (em nosso caso, os padrões a serem pesquisados).

Embora a implementação genérica da aplicação servidora receba apenas um parâmetro, espera-se que a aplicação servidora seja invocada *n* vezes, onde *n* é o numero de parâmetros enviados, por exemplo, em nossa aplicação, possuímos três parâmetros, que se referem aos padrões de texto a serem enviados, portanto, na API, tal aplicação é invocada três vezes.

4.2.4 *OFFLOADING* DA OPERAÇÃO E DADOS

Em nosso experimento realizamos não somente o *offload* de computação/operação, mas também, o *offload* de dados, como apresentado na seção 4.1.

No *offload* de operações, uma vez que a aplicação servidora gerada por nosso framework está em funcionamento na plataforma em nuvem, em nossa API, utilizamos a biblioteca padrão do Java *java.net* para acessar a plataforma em nuvem e também, retribuir o resultado da operação de busca. A implementação do método é apresentado na Figura 16.

```
private Long findOccurencesRemotely(String pattern, String cloudURL, String apiPath) {
    Long occurences = 0L;

     // String that store the page information
   String externalPage = null;
   try {
       // URL that returns the external HTML Code
       URL url = new URL(cloudURL + "/mdcm.php?api=" + apiPath + "&param=" + pattern);
       // Creating a HTTPUrlConnection with the URL
       HttpURLConnection httpConnection = (HttpURLConnection) url
               .openConnection();
       // Connecting
       httpConnection.connect();
       // Reading the HTML Code from the URL page
       BufferedReader page = new BufferedReader(
               new java.io.InputStreamReader(
               httpConnection.getInputStream()));

       while ((externalPage = page.readLine()) != null && !externalPage.equals("")) {
           occurences = Long.parseLong(externalPage.trim());
       }
       // Closing the BufferReader of the HTML page
       page.close();
   } catch (MalformedURLException ex) {
       ex.printStackTrace();
   } catch (IOException ex) {
       ex.printStackTrace();
   }
    return occurences;
}
```

Figura 16 – Método que invoca a operação na plataforma em nuvem e retorna o resultado da busca.

O método *findOccurencesRemotely* tem como parâmetros o padrão a ser buscado, a URL do servidor em nuvem, e o caminho da API que será utilizada no servidor. Utilizando os parâmetros, o método invoca a aplicação servidora, enviando como parâmetro os valores do padrão, e caminho da API, e retornando o numero de ocorrências do respectivo padrão no arquivo de texto fonte. Tal valor é retornado como uma *String*, portanto, a conversão deste valor para numérico se faz necessária.

Já o *offload* de dados é realizado em uma rotina separada, como mencionado na seção 4.1. Desta forma, utilizamos o comando *scp*[5], que transfere um arquivo utilizando o protocolo de segurança SSH, e o comando *time*[6], que retorna o tempo despendido por dado comando.

O primeiro cenário (**computação "*on-device*"**) foi realizado utilizando o algoritmo padrão, Uma vez que todos os passos do experimento foram executados, obtemos os dados para análise, que por sua vez é realizada no próximo capitulo.

[5] http://sourceforge.net/p/forge/documentation/SCP/

[6] http://pubs.opengroup.org/onlinepubs/9699919799/utilities/time.html

5. ANÁLISE DE DADOS

Nesta seção, apresentaremos os dados obtidos por nosso experimento, e também, uma discussão sob os resultados, o que inclui o que os dados obtidos representam e a viabilidade do framework.

5.1 DADOS

A Tabela 20 representa os resultados obtidos por meio de nosso experimento a partir dos ambientes experimentais apresentados na seção anterior.

Tabela 20 – Resultados obtidos a partir dos ambientes experimentais.

	1 padrão: 'a'	**3 padrões: 'a', 'b', e 'c'.**
Operação e arquivo de texto fonte *on-device*	76 segundos	229 segundos
Operação *on-cloud* (arquivo de texto fonte previamente armazenado na plataforma em nuvem)	1 segundo	3 segundos
Operação *on-cloud* com o envio do arquivo do dispositivo para a plataforma em nuvem	41 segundos	44 segundos

Fonte: Elaborado pelo Autor.

Nosso primeiro ambiente experimental, que se refere a **computação** "**on-device**", evidencia as limitações apresentadas por dispositivos móveis por si só. Apesar de que o arquivo de texto fonte havia sido previamente armazenado no dispositivo, a operação, que por sua vez foi realizada localmente, levou mais de um minuto para processar apenas um padrão 'a', e mais de três para processar três padrões 'a', 'b', e 'c'.

Já nosso segundo ambiente experimental, que se refere ao ***offload* de computação**, o que não inclui o *offload* de dados, isto é, o envio do arquivo de texto fonte como parte da operação, levou apenas 1 segundo para processar a busca por um padrão 'a', e apenas 3 segundos para processar três padrões 'a', 'b', e 'c'. Evidentemente, as limitações apresentadas por dispositivos móveis podem ser superadas por meio da integração em nuvem. No entanto, ainda deve-se considerar o *offload* de dados, que em certos casos se faz necessário.

Mesmo assim, a integração com a nuvem ainda apresenta melhor performance, como apresentado no nosso terceiro ambiente experimental.

O terceiro ambiente experimental, que se refere ao ***offload* de computação e dados**, levou 41 segundos para processar um padrão ‘a’, e apenas 44 segundos para processar três padrões ‘a’, ‘b’, e ‘c’. Aparentemente, o *offload* de dados tomou 40 segundos do tempo, uma vez que o processamento da operação em si custou a mesma quantidade de tempo em relação ao ambiente experimental anterior, que não contava com o *offload* de dados. A despeito do acréscimo de tempo devido ao *offload* de dados, como descrito anteriormente, o tempo despendido ainda é significantemente menor que o tempo despendido pela operação realizada no dispositivo móvel por si só.

5.2 DISCUSSÃO

Os dados oriundos de nosso experimento indicam que o *offload* é uma abordagem viável quando as limitações dos dispositivos móveis entram em palco. Os dados mostram que o envio da operação para a plataforma em nuvem apresenta uma melhor performance sob não somente o processamento local, mas também no processamento remoto que inclui o *offload* de dados.

No entanto, apesar do nosso framework não ter como foco o *offload* de dados, e sim o de computação/operações, vale ressaltar que uma vez que o dado sendo enviado é significantemente extenso, o *offload* pode não ser a melhor abordagem, levando em conta que o *overhead* da transmissão do dado poderia resultar em um tempo de processamento maior que o tempo despendido para executar a operação localmente.

Também, devemos levar em conta que certas operações simplesmente não são suportadas por dispositivos móveis, então, se faz necessário o *offload* para a plataforma em nuvem. Portanto, nosso experimento tem nos levado a conclusão que o *offload* de dados é vantajoso quando a computação necessária é significantemente maior que o dado necessário para a execução da aplicação.

Finalmente, nosso framework se mostrou viável quando o *offload* de operações é suficiente para a execução de uma operação. No entanto, nosso framework não provem suporte para o *offload* de dados, portanto, nosso framework pode não ser uma alternativa quando o *offload* de dados se faz necessário. De qualquer forma, medidas adicionais podem ser tomadas quando o *offload* de dados entra em palco. Um exemplo de tal medida adicional é o *upload* do dado em uma rotina separada, que por sua vez foi utilizado em nosso experimento.

6. CONSIDERAÇÕES FINAIS

Em conclusão, para superar e reduzir as limitações apresentadas por dispositivos móveis em comparação com outros dispositivos computacionais e para explorar os recursos oferecidos pela plataforma em nuvem, nós apresentamos o framework MDCM. Nosso framework tem como proposta facilitar e auxiliar desenvolvedores a criar aplicações móveis integradas com a nuvem por meio de técnicas de desenvolvimento orientado a modelos.

Tendo como núcleo de nosso framework o *offload* de operações, também apresentamos um meta-modelo que suporta o envio de determinadas operações para processo na plataforma em nuvem. Este envio é realizado em função de um atributo de tipagem no meta-modelo, que determina quando a operação é processada "*on-device*" e quando a operação é processada "*on-cloud*".

Também, visando avaliar nossa abordagem, realizamos um experimento real que faz uso de nosso framework para gerar uma aplicação móvel integrada com a nuvem, e ilustra diferentes cenários envolvendo o *offload* de operações e dados.

Depois disso, apresentamos a análise de dados resultados de nossa aplicação experimental. Analisando os dados, chegamos a conclusão que o *offload* pode superar as limitações apresentadas por dispositivos móveis mesmo quando não somente o *offload* de operações, mas também o *offload* de dados é necessário.

Finalmente, após a análise de dados e do framework em si, identificamos alguns passos futuros para nossa pesquisa. Atualmente, nosso framework não suporta o *offload* dinâmico de operações. Em outras palavras, o *offload* das operações é atribuído em "tempo de compilação", melhor dizendo, na criação do modelo da aplicação móvel integrada com a nuvem. Dessa forma, uma vez que aplicação é gerada, tal elemento não pode ser alterado, mesmo se necessário. Portanto, planejamos estender nosso meta-modelo para que o mesmo seja capaz de fornecer uma alternativa para não somente o *offload* de operações estático, como atualmente apresentado, mas também para o *offload* de operações dinâmica.

REFERÊNCIAS

ALZAHRANI, Ahmed; ALALWAN, Nasser; SARRAB, Mohamed. **Mobile cloud computing: advantage, disadvantage and open challenge.** In: 7th Euro American Conference on Telematics and Information Systems. 2014. New York.

APPLE. Disponível em:<https://www.apple.com/ipad/>. Acesso em: 08 jan. 2015.

BAHL, Paramvir; HAN, Richard Y.; ERRAN, Li; SATYANARAYANAN, Mahadev. **Advancing the state of mobile cloud computing.** In: ACM Workshop on Mobile Cloud Computing and Services. 2012. New York.

BALAGTAS-FERNANDEZ, Florence; HUSSMANN, Heinrich. **Model-driven development of mobile applications.** In: 23rd IEEE/ACM International Conference. 2008. L'Aquila, Italy.

DELL. Disponível em:<http://www.dell.com/us/p/laptops>. Acesso em: 08 jan. 2015.

GUAN, Le; KE, Xu; SONG, Meina; SONG, Junde. **A survey of research on mobile cloud computing.** In: 10th IEEE/ACIS International Conference on Computer and Information Science. 2011. Washington, DC, USA.

HAILPERN, B; TARR, P. Model-driven development: The good, the bad, and the ugly. **IBM Systems Journal.** 2006. Riverton, NJ, USA, v. 45, n. 3, p. 451-461.

HASSAN, Mohammed Anowarul; CHEN, Songqing. **Mobile MapReduce: minimizing response time of computing intensive mobile applications**. In: Mobile computing, applications, and services**:** third international conference, MobiCASE 2011, Los Angeles, CA, USA, October 24-27, 2011. Revised selected papers. Los Angeles, CA, USA, v. 95, p. 41-59.

HEITKÖTTER, Henning, MAJCHRZAK, Tim A.; KUCHEN, Herbert. **Cross-Platform Model- Driven Development of Mobile Applications with MD2.** In: 28th Annual ACM Symposium on Applied Computing. 2013. New York.

KOCH, N. Classification of model transformation techniques used in UML-based Web engineering. **Software, IET.** v. 1, n. 3, p. 98-111.

KOLOVOS, Dimitrios S.; ROSE, Louis M.; ABID, Saad B.; Paige, Richard F.;

Polack, Fiona A. C.; BOTTERWECK, Goetz. 2010, 'Taming EMF and GMF Using Model Transformation', in DC Petriu, N Rouquette, Ø Haugen (eds.), Model Driven Engineering Lan- guages and Systems, 1st edn, Springer Berlin Heidelberg, Oslo, Østlandet, Norway.

KUMAR, Karthik; YUNG-HSIANG, Lu. Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? **IEEE Computer Society**. 2010. v. 43, n. 4, p. 51-56.

LUO, Xun. **From Augmented Reality to Augmented Computing: a Look at Cloud-Mobile Convergence.** In: 2009 International Symposium on Ubiquitous Virtual Reality. 2009. San Diego, SA, United States.

MALLIKHARJUNA, N.; SASIDHAR, C.; SATYENDRA, V. Cloud Computing Through Mobile-Learning. **International Journal of Advanced Computer Science and Applications** (**IJACSA**). 2010.

MILUZZO, Emiliano; CÁCERES, Rámon; CHEN, Yih-Farn. **Vision: MClouds - Computing on Clouds of Mobile Devices.** In: Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services. 2012. New York, NY, USA.

MTSWENI, Jabu. **Exploiting UML and Acceleo for Developing Semantic Web Services.** In: The 7th International Conference for Internet Technology and Secured Transactions (ICITST-2012). 2012. Pretoria, South Africa.

NELLAIAPPAN, Rukmani. **An Eclipse-based tool for modeling service-based systems.** 2009. 49 f. Trabalho de Conclusão de Curso (Graduação em Tecnologia da Informação) - Amrita School of Engineering, Coimbatore, Tamil Nadu, India, 2009.

NIU, Ruifang; SONG, Wenfang; LIU, Yong. An Energy-Efficient Multisite Offloading Algorithm for Mobile Devices. **International Journal of Distributed Sensor Networks.** 2013. v. 2013, p. 1-6.

RANABAHU, Ajith H.; MAXIMILIEN, Eugene M.; SHETH, Amit P. **A Domain Specific Language for Enterprise Grade Cloud-Mobile Hybrid Applications.** In: Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11. 2011. New York, NY, USA, p. 77-84.

SAMSUMG. Disponível em:<http://www.samsung.com/us/mobile/cell-phones/all-products?filter=galaxy-s>. Acesso em: 08 jan. 2015.

SELIC, Bran. The pragmatics of model-driven development. **IBM Rational Software**. 2008. v. 20, n. 5, p. 19-25.

SINHA, Kanad; KULKARNI, Milind. **Techniques for fine-grained, multi-site computation offloading.** In: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing Techniques. 2011. West Lafayette, IN, USA, p. 184-194.

STEINER, Dustin; TURLEA, Catalina; CULEA, Cristian; SELINGER, Stephan. Model-Driven Development of Cloud-Connected Mobile Applications Using DSLs with Xtext. In: **Computer Aided Systems Theory - EUROCAST 2013**: 14th International Conference, Las Palmas de Gran Canaria, Spain, February 10-15, 2013. Revised Selected Papers, Part II. Springer Berlin Heidelberg, 2013, v. 8112, p. 409-416.

VIDYAPEETHAM, Amrita Vishwa. **An eclipse-based tool for modeling service-based systems.** 2009. 49 f. Trabalho de Conclusão de Curso (Graduação em Tecnologia da Informação) – Amrita School of Engineering, Coimbatore, Tamil Nadu, India, 2009a.

VIDYAPEETHAM, Amrita Vishwa. **An MDA approach for transforming SoaML models to OSGi declarative services.** 2009. 59 f. Trabalho de Conclusão de Curso (Graduação em Tecnologia da Informação) – Amrita School of Engineering, Coimbatore, 2009b.

APÊNDICE A – ECORE/XML DE NOSSO META-MODELO

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="mdcm" nsURI="http://mdcm.org/metamodel" nsPrefix="mdcm">
  <eClassifiers xsi:type="ecore:EClass" name="Window">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="widgets" upperBound="-1"
        eType="#//Widget" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="isLauncher" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Widget" abstract="true">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="marginTop" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="marginBottom" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="marginLeft" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="marginRight" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="width" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="height" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Output" abstract="true" eSuperTypes="#//Widget"/>
  <eClassifiers xsi:type="ecore:EClass" name="Text_Output" eSuperTypes="#//Output">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="text" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="textSize" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="textColor" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Image" eSuperTypes="#//Output">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="imagePath" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Input" abstract="true" eSuperTypes="#//Widget"/>
  <eClassifiers xsi:type="ecore:EClass" name="Text_Input" eSuperTypes="#//Input">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="hint" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Navigation" abstract="true" eSuperTypes="#//Widget">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="to" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="text" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Navigation_Button" eSuperTypes="#//Navigation"/>
  <eClassifiers xsi:type="ecore:EClass" name="Link" eSuperTypes="#//Navigation">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="textColor" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Control" abstract="true" eSuperTypes="#//Widget">
    <eOperations name="action" eType="ecore:EClass http://www.eclipse.org/emf/2002/Ecore#//EObject"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="text" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="APIPath" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="type" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="cloudURL" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="arguments" upperBound="-1"
        eType="#//Argument" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Control_Button" eSuperTypes="#//Control"/>
  <eClassifiers xsi:type="ecore:EClass" name="Application">
    <eStructuralFeatures xsi:type="ecore:EReference" name="windows" lowerBound="1"
        upperBound="-1" eType="#//Window" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="domain" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Argument">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="type" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="value" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

APÊNDICE B – *TEMPLATE* ACCELEO PARA GERAÇÃO DE CÓDIGO

generate.mtl

```
1 [comment encoding = UTF-8 /]
2 [module generate('http://mdcm.org/metamodel')]
3
4 [template public generateElement(anApplication : Application)]
5 [comment @main/]
6 [file ('AndroidManifest.xml', false, 'UTF-8')]
7 <?xml version="1.0" encoding="utf-8"?>
8 <manifest xmlns:android="http://schemas.android.com/apk/res/
  android"
9     package="[anApplication.domain.toString()/]"
10    android:versionCode="1"
11    android:versionName="1.0" >
12
13    <uses-sdk
14        android:minSdkVersion="8"
15        android:targetSdkVersion="21" />
16
17    <application
18        android:allowBackup="true"
19        android:icon="@drawable/ic_launcher"
20        android:label="@string/app_name"
21        android:theme="@style/AppTheme" >
22        [for (window : Window | anApplication.windows)]
23        <activity
24            android:name=".[window.id.toUpperFirst()/]"
25            android:label="@string/app_name" >
26            [if window.isLauncher]
27            <intent-filter>
28                <action
  android:name="android.intent.action.MAIN" />
29                <category
  android:name="android.intent.category.LAUNCHER" />
30            </intent-filter>
31            [/if]
32        </activity>
33        [/for]
34    </application>
35 </manifest>
36 [/file]
37 [for (window : Window | anApplication.windows)]
38    [file ('/res/
  layout/'.concat(window.id.concat('_layout.xml').toLowerCase()),
  false, 'UTF-8')]
39 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
```

generate.mtl

```
  android"
40      xmlns:tools="http://schemas.android.com/tools"
41      android:layout_width="match_parent"
42      android:layout_height="match_parent"
43      android:paddingBottom="@dimen/activity_vertical_margin"
44      android:paddingLeft="@dimen/activity_horizontal_margin"
45      android:paddingRight="@dimen/activity_horizontal_margin"
46      android:paddingTop="@dimen/activity_vertical_margin"
47
  tools:context="[anApplication.domain.toLowerCase().concat('.'.conc
  at(window.id.toUpperFirst()))/]">
48      [for (widget : Widget | window.widgets)]
49          [if oclIsKindOf(Text_Output)]
50          <TextView
51              android:id="@+id/[widget.id.toLowerCase()/]"
52              android:layout_width="wrap_content"
53              android:layout_height="wrap_content"
54              android:text="[widget.oclAsType(Text_Output).text/]"
55
  android:textColor="[widget.oclAsType(Text_Output).textColor/]"
56
  android:textSize="[widget.oclAsType(Text_Output).textSize/]sp"
57              android:layout_marginLeft="[widget.marginLeft/]dp"
58              android:layout_marginRight="[widget.marginRight/]dp"
59              android:layout_marginTop="[widget.marginTop/]dp"
60
  android:layout_marginBottom="[widget.marginBottom/]dp"/>
61          [/if]
62          [if oclIsKindOf(Image)]
63          <ImageView
64              android:id="@+id/[widget.id.toLowerCase()/]"
65              android:layout_width="[widget.width/]dp"
66              android:layout_height="[widget.height/]dp"
67              android:layout_marginLeft="[widget.marginLeft/]dp"
68              android:layout_marginRight="[widget.marginRight/]dp"
69              android:layout_marginTop="[widget.marginTop/]dp"
70              android:layout_marginBottom="[widget.marginBottom/]dp"
71              android:src="@drawable/
  [widget.oclAsType(Image).imagePath/]"/>
72          [/if]
73          [if oclIsKindOf(Text_Input)]
74          <EditText
75              android:id="@+id/[widget.id.toLowerCase()/]"
76              android:layout_width="[widget.width/]dp"
```

generate.mtl

```
77              android:layout_height="[widget.height/]dp"
78              android:layout_marginLeft="[widget.marginLeft/]dp"
79              android:layout_marginRight="[widget.marginRight/]dp"
80              android:layout_marginTop="[widget.marginTop/]dp"
81              android:layout_marginBottom="[widget.marginBottom/]dp"
82              android:hint="[widget.oclAsType(Text_Input).hint/]"
83              android:inputType="text" />
84          [/if]
85          [if oclIsKindOf(Control_Button)]
86          <Button
87              android:id="@+id/[widget.id.toLowerCase()/]"
88              android:layout_width="[widget.width/]dp"
89              android:layout_height="[widget.height/]dp"
90              android:layout_marginLeft="[widget.marginLeft/]dp"
91              android:layout_marginRight="[widget.marginRight/]dp"
92              android:layout_marginTop="[widget.marginTop/]dp"
93              android:layout_marginBottom="[widget.marginBottom/]dp"
94
  android:text="[widget.oclAsType(Control_Button).text/]"
95              android:onClick="[widget.id/]" />
96          [/if]
97      [/for]
98  </RelativeLayout>
99      [/file]
100     [file ('/
  src/'.concat(anApplication.domain.toLowerCase().replaceAll('\\.',
  '/')).concat('/').concat(window.id.concat('.java').toUpperFirst())
  , false, 'UTF-8')]
101 package [anApplication.domain.toLowerCase()/];
102
103 import android.app.Activity;
104 import android.os.Bundle;
105 import android.widget.*;
106
107 class [window.id.toUpperFirst()/] extends Activity {
108     [for (widget : Widget | window.widgets)]
109         [if oclIsKindOf(Text_Output)]
110     private TextView [widget.id.toLowerCase()/];
111         [/if]
112         [if oclIsKindOf(Image)]
113     private ImageView [widget.id.toLowerCase()/];
114         [/if]
115         [if oclIsKindOf(Text_Input)]
116     private EditText [widget.id.toLowerCase()/];
```

generate.mtl

```
117         [/if]
118         [if oclIsKindOf(Navigation_Button)]
119     private Button [widget.id.toLowerCase()/];
120         [/if]
121         [if oclIsKindOf(Link)]
122     private TextView [widget.id.toLowerCase()/];
123         [/if]
124         [if oclIsKindOf(Control_Button)]
125     private Button [widget.id.toLowerCase()/];
126         [/if]
127     [/for]
128 
129     @Override
130     protected void onCreate(Bundle savedInstanceState) {
131         super.onCreate(savedInstanceState);
132         setContentView(R.layout.
    [window.id.toLowerCase()/]_layout);
133         [for (widget : Widget | window.widgets)]
134         [widget.id.toLowerCase()/] = [if
    oclIsKindOf(Control_Button)](Button)[/if][if
    oclIsKindOf(Text_Output)](TextView)[/if][if oclIsKindOf(Image)]
    (ImageView)[/if][if oclIsKindOf(Text_Input)](EditText)[/
    if]findViewById(R.id.[widget.id.toLowerCase()/]);
135         [if oclIsKindOf(Control_Button)]
136             [widget.id.toLowerCase()/].setOnClickListener(new
    Button.OnClickListener()) {
137                 public void onClick(View v) {
138                 [if widget.oclAsType(Control_Button).type =
    'local']
139                     [widget.oclAsType(Control_Button).APIPath/]
    api = new [widget.oclAsType(Control_Button).APIPath/]();
140                     api.[widget.oclAsType(Control_Button).id/](
141                     [for (argument : Argument |
    widget.oclAsType(Control_Button).arguments)]
142                         [if argument.type = 'element']
143                             ((EditText) findViewById(R.id.
    [argument.value/]),
144                         [/if]
145                         [if argument.type = 'literal'],
146                             "[argument.value/]"
147                         [/if]
148                     [/for]
149                     false);
150                 [/if]
```

generate.mtl

```
151                 [if widget.oclAsType(Control_Button).type =
    'remote']
152                     [widget.oclAsType(Control_Button).APIPath/]
    api = new [widget.oclAsType(Control_Button).APIPath/]();
153                     api.[widget.oclAsType(Control_Button).id/](
154                     [for (argument : Argument |
    widget.oclAsType(Control_Button).arguments)]
155                         [if argument.type = 'element']
156                             ((EditText) findViewById(R.id.
    [argument.value/]),
157                         [/if]
158                         [if argument.type = 'literal']
159                             "[argument.value/]",
160                         [/if]
161                     [/for]
162                     "[widget.oclAsType(Control_Button).cloudURL/]"
    , true);
163                 [/if]
164                 }
165             });
166         [/if]
167     [/for]
168     }
169 }
170     [/file]
171 [/for]
172 [file ('/src/cloud/'.concat('mdcm.php'), false, 'UTF-8')]
173     <?php
174         $api = $_GET['['/]'api'[']'/];
175         $param = $_GET['['/]'param'[']'/];
176         $script = "java -jar ".$api." ".$param;
177
178         $out = shell_exec($script);
179
180         echo $out;
181     ?>
182 [/file]
183 [/template]
184
```

APÊNDICE C – ARTEFATOS GERADOS POR NOSSO FRAMEWORK EM NOSSO EXPERIMENTO

```
package com.mdcm.simpletextsearch;

import android.app.Activity;
import android.os.Bundle;
import android.widget.*;

class Home extends Activity {
    private TextView pattern1_label;
    private EditText pattern1_input;
    private TextView pattern2_label;
    private EditText pattern2_input;
    private TextView pattern3_label;
    private EditText pattern3_input;
    private TextView n_occurences_label;
    private TextView occurences_label;
    private TextView total_time_label;
    private TextView total_label;
    private Button search_button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.home_layout);
        pattern1_label = (TextView)findViewById(R.id.pattern1_label);
        pattern1_input = (EditText)findViewById(R.id.pattern1_input);
        pattern2_label = (TextView)findViewById(R.id.pattern2_label);
        pattern2_input = (EditText)findViewById(R.id.pattern2_input);
        pattern3_label = (TextView)findViewById(R.id.pattern3_label);
        pattern3_input = (EditText)findViewById(R.id.pattern3_input);
        n_occurences_label = (TextView)findViewById(R.id.n_occurences_label);
        occurences_label = (TextView)findViewById(R.id.occurences_label);
        total_time_label = (TextView)findViewById(R.id.total_time_label);
        total_label = (TextView)findViewById(R.id.total_label);
        search_button = (Button)findViewById(R.id.search_button);
            search_button.setOnClickListener(new Button.OnClickListener()) {
                public void onClick(View v) {
                    simpletextsearch api = new simpletextsearch();
                    api.search_button(((EditText) findViewById(R.id.pattern1_input),
                            ((EditText) findViewById(R.id.pattern2_input),
                            ((EditText) findViewById(R.id.pattern3_input),
                            "wordlist"
                            false);
                    simpletextsearch api = new simpletextsearch();
                    api.search_button(
                            ((EditText) findViewById(R.id.pattern1_input),
                            ((EditText) findViewById(R.id.pattern2_input),
                            ((EditText) findViewById(R.id.pattern3_input),
                            "wordlist",
                            "ec2-54-213-222-225.us-west-2.compute.amazonaws.com",
                            true);
                }
            });
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mdcm.simpletextsearch"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="21" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".Home"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```
<?php
    $api = $_GET['api'];
    $param = $_GET['param'];
    $script = "java -jar ".$api." ".$param;

    $out = shell_exec($script);

    echo $out;
?>
```

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.mdcm.simpletextsearch.Home">
        <TextView
            android:id="@+id/pattern1_label"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Pattern 1"
            android:textColor="#000"
            android:textSize="10sp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="0dp"
            android:layout_marginBottom="0dp"/>
        <EditText
            android:id="@+id/pattern1_input"
            android:layout_width="300dp"
            android:layout_height="30dp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="15dp"
            android:layout_marginBottom="0dp"
            android:hint=""
            android:inputType="text" />
        <TextView
            android:id="@+id/pattern2_label"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Pattern 2"
            android:textColor="#000"
            android:textSize="10sp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="50dp"
            android:layout_marginBottom="0dp"/>
        <EditText
            android:id="@+id/pattern2_input"
            android:layout_width="300dp"
            android:layout_height="30dp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="65dp"
            android:layout_marginBottom="0dp"
            android:hint=""
            android:inputType="text" />
        <TextView
            android:id="@+id/pattern3_label"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Pattern 3"
            android:textColor="#000"
            android:textSize="10sp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="110dp"
            android:layout_marginBottom="0dp"/>
        <EditText
            android:id="@+id/pattern3_input"
            android:layout_width="300dp"
            android:layout_height="30dp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="125dp"
            android:layout_marginBottom="0dp"
            android:hint=""
            android:inputType="text" />
        <TextView
            android:id="@+id/n_occurences_label"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Number of occurences of the pattern(s) in the text file:"
            android:textColor="#000"
            android:textSize="10sp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="160dp"
            android:layout_marginBottom="0dp"/>
        <TextView
            android:id="@+id/occurences_label"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="-"
            android:textColor="#66CD00"
            android:textSize="10sp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="170dp"
            android:layout_marginBottom="0dp"/>
        <TextView
            android:id="@+id/total_time_label"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Total time:"
            android:textColor="#000"
            android:textSize="10sp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="190dp"
            android:layout_marginBottom="0dp"/>
        <TextView
            android:id="@+id/total_label"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="-"
            android:textColor="#FF0000"
            android:textSize="10sp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="200dp"
            android:layout_marginBottom="0dp"/>
        <Button
            android:id="@+id/search_button"
            android:layout_width="300dp"
            android:layout_height="50dp"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginTop="380dp"
            android:layout_marginBottom="0dp"
            android:text="Search!"
            android:onClick="search_button" />
</RelativeLayout>
```

APÊNDICE D – API REFERENCIADA PELO PARÂMETRO *APIPATH* DE NOSSO MODELO

simpletextsearch.java

```
 1 package com.mdcm.simpletextsearch;
 2
 3 import java.io.BufferedReader;
 4 import java.io.FileNotFoundException;
 5 import java.io.FileReader;
 6 import java.io.IOException;
 7 import java.net.HttpURLConnection;
 8 import java.net.MalformedURLException;
 9 import java.net.URL;
10 import java.util.Date;
11
12 import android.widget.*;
13 import android.R;
14
15 public class simpletextsearch {
16
17     private BufferedReader fileReader;
18
19     /* Testing patterns
20      */
21     public static void main(String[] args) {
22
23         simpletextsearch sts = new simpletextsearch();
24
25         if (args.length == 0) {
26             System.out.println("-1");
27         } else if (args.length == 1){
28             System.out.println(sts.findOccurrences(args[0]));
29         } else if (args.length > 1){
30             System.out.println(sts.findOccurrences(args[0],
   args[1]));
31         }
32     }
33
34     public long findOccurrences() {
35         String pattern = "a";
36         String fileName = "wordlist";
37
38         long occurences = 0;
39         String line = null;
40
41         try {
42             fileReader = new BufferedReader(new
   FileReader(fileName));
```

simpletextsearch.java

```
43              int count = 0;
44              while ((line = fileReader.readLine()) != null) {
45                  ++count;
46                  int n = line.length();
47                  int m = pattern.length();
48
49                  for (int i=0; i<=(n-m);i++) {
50                      boolean matches = true;
51                      for (int j=0; j<m; j++) {
52                          if (line.charAt(i+j) != pattern.charAt(j))
  {
53                              matches = false;
54                              break;
55                          }
56                      }
57                      if (matches) { occurences++; }
58                  }
59              }
60              return occurences;
61          } catch (FileNotFoundException ex) {
62              ex.printStackTrace();
63          return -1;
64          } catch (IOException ex) {
65              ex.printStackTrace();
66          }
67          return occurences;
68      }
69
70      public long findOccurrences(String pattern, String fileName) {
71          long occurences = 0;
72          String line = null;
73          try {
74              fileReader = new BufferedReader(new
  FileReader(fileName));
75              int count = 0;
76              while ((line = fileReader.readLine()) != null) {
77                  ++count;
78                  int n = line.length();
79                  int m = pattern.length();
80
81                  for (int i=0; i<=(n-m);i++) {
82                      boolean matches = true;
83                      for (int j=0; j<m; j++) {
84                          if (line.charAt(i+j) != pattern.charAt(j))
```

simpletextsearch.java

```
    {
 85                             matches = false;
 86                             break;
 87                         }
 88                     }
 89                     if (matches) { occurences++; }
 90                 }
 91             }
 92             return occurences;
 93         } catch (FileNotFoundException ex) {
 94             ex.printStackTrace();
 95         return -1;
 96         } catch (IOException ex) {
 97             ex.printStackTrace();
 98         }
 99         return occurences;
100     }
101 
102     public long findOccurrences(String pattern) {
103         long occurences = 0;
104         String line = null;
105         try {
106             fileReader = new BufferedReader(new
    FileReader("wordlist"));
107             int count = 0;
108             while ((line = fileReader.readLine()) != null) {
109                 ++count;
110                 int n = line.length();
111                 int m = pattern.length();
112 
113                 for (int i=0; i<=(n-m);i++) {
114                     boolean matches = true;
115                     for (int j=0; j<m; j++) {
116                         if (line.charAt(i+j) != pattern.charAt(j))
    {
117                             matches = false;
118                             break;
119                         }
120                     }
121                     if (matches) { occurences++; }
122                 }
123             }
124             return occurences;
125         } catch (FileNotFoundException ex) {
```

simpletextsearch.java

```
126             ex.printStackTrace();
127         return -1;
128         } catch (IOException ex) {
129             ex.printStackTrace();
130         }
131         return occurences;
132     }
133 
134     private Long findOccurencesRemotely(String pattern, String
    cloudURL, String apiPath) {
135         Long occurences = 0L;
136 
137          // String that store the page information
138        String externalPage = null;
139        try {
140            // URL that returns the external HTML Code
141            URL url = new URL(cloudURL + "/mdcm.php?api=" + apiPath
    + "&param=" + pattern);
142            // Creating a HTTPUrlConnection with the URL
143            HttpURLConnection httpConnection = (HttpURLConnection)
    url
144                    .openConnection();
145            // Connecting
146            httpConnection.connect();
147            // Reading the HTML Code from the URL page
148            BufferedReader page = new BufferedReader(
149                    new java.io.InputStreamReader(
150                    httpConnection.getInputStream()));
151 
152            while ((externalPage = page.readLine()) != null && !
    externalPage.equals("")) {
153                occurences = Long.parseLong(externalPage.trim());
154            }
155            // Closing the BufferReader of the HTML page
156            page.close();
157        } catch (MalformedURLException ex) {
158            ex.printStackTrace();
159        } catch (IOException ex) {
160            ex.printStackTrace();
161        }
162         return occurences;
163     }
164 
165     public void search_button(EditText pattern1, EditText
```

simpletextsearch.java

```
    pattern2, EditText pattern3, String cloudURL, String apiPath,
    boolean isRemote) {
166         Long occurences = 0L;
167         Long time = 0L;
168         Date beggining = new Date();
169 
170         if (isRemote) {
171             occurences +=
    findOccurencesRemotely(pattern1.getText().toString(), cloudURL,
    apiPath);
172             occurences +=
    findOccurencesRemotely(pattern2.getText().toString(), cloudURL,
    apiPath);
173             occurences +=
    findOccurencesRemotely(pattern2.getText().toString(), cloudURL,
    apiPath);
174         } else {
175             occurences +=
    findOccurrences(pattern1.getText().toString());
176             occurences +=
    findOccurrences(pattern2.getText().toString());
177             occurences +=
    findOccurrences(pattern3.getText().toString());
178         }
179 
180         Date end = new Date();
181         time = end.getTime() - beggining.getTime();
182 
183         ((TextView)
    findViewById(R.id.total_label)).setText(String.valueOf(time));
184         ((TextView)
    findViewById(R.id.occurences_label)).setText(String.valueOf(occure
    nces));
185     }
186 }
187 
188 
```