Universidade Federal de São Carlos – UFSCar

Departamento de Computação

Programa de Pós-Graduação

Juliano Zanuzzio Blanco

Uma abordagem holística para o desenvolvimento de software multiplataforma

Brasil

2020, Outubro

Juliano Zanuzzio Blanco

Uma abordagem holística para o desenvolvimento de software multiplataforma

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação, área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Auri Marcelo Rizzo Vincenzi.

Brasil

2020, Outubro

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Tese de Doutorado do candidato Juliano Zanuzzio Blanco, realizada em 06/10/2020.

Comissão Julgadora:

Prof. Dr. Auri Marcelo Rizzo Vincenzi (UFSCar)

Profa. Dra. Vânia Paula de Almeida Neris (UFSCar)

Prof. Dr. Valter Vieira de Camargo (UFSCar)

Prof. Dr. Windson Viana de Carvalho (UFC)

Prof. Dr. Uira Kulesza (UFRN)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.
O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Agradecimentos

Agradeço ao bom Deus por me abençoar com saúde física e principalmente mental para a execução desse grande desafio.

Agradeço também, em especial, ao professor Dr. Daniel Lucrédio, que me acolheu como aluno de doutorado e acreditou no meu trabalho para o desenvolvimento desta tese. Todo este tempo de convívio, desde meados de 2014, foi um período de muita evolução e aprendizado, que não teria se não fosse sua orientação. Obrigado por todo ensinamento, paciência, direcionamentos e o principal, por ter acredito em mim até o final.

Agradeço em especial aos meus pais, José e Pedrina pelo amor e ajuda em todos os momentos, vocês são meu alicerce e inspiração. Um agradecimento em especial à minha irmã, pelo grande apoio durante o doutorado, assim como no mestrado. Ao meu sogro Joel pelas suas dicas e ajudas e minha sogra Orminda, que sempre preparou tudo em nossas visitas, para que eu pudesse continuar trabalhando. À minha Fabiana, que me apoiou nos momentos difíceis, que não foram poucos, e a todo companheirismo e compreensão. Gostaria de agradecer a Deus novamente por me abençoar com minha maior motivação para continuar encarando todos os desafios da vida, minha filha Sara, que me ensinou o que é o mais puro e verdadeiro sentimento.

À todos os especialistas que contribuíram imensamente com este trabalho, fornecendo informações essências para o progresso desta pesquisa: Mariana, Júlio, Bento, Daniel e Márcio. Agradeço pela condução dos testes de forma imparcial e pela dedicação de horas no aprendizado de uma nova forma de desenvolvimento em pró da ciência.

Agradeço aos colegas do IFSP, que sempre me motivaram. Em especial ao Márcio, Anderson, Luiz, Carlos, Diego, equipe de RH e diretoria. Agradeço também ao próprio IFSP que me deu a oportunidade de realizar este doutorado de forma dedicada.

Agradeço aos avaliadores da minha qualificação de doutorado, por terem incentivado a continuidade da pesquisa e mostrarem novos caminhos. Antecipo meus agradecimentos aos avaliadores desta tese.

Agradeço também a Fundação de Pesquisa de São Paulo (FAPESP) por financiar parte desta pesquisa e me dar a oportunidade de orientar o Matheus como aluno de iniciação cientifica (IC). O Matheus teve uma grande contribuição, continuando a pesquisa sobre realidade aumentada de seu antecessor Afonso (IC), que também contribuiu com esta pesquisa.

Aos professores e funcionários do Departamento de Computação da UFSCar. Enfim, agradeço a todas as pessoas que contribuíram para o desenvolvimento e realização desse trabalho.

“A menos que modifiquemos a nossa maneira de pensar, não seremos capazes de resolver os problemas causados pela forma como nos acostumamos a ver o mundo”. (Albert Einstein)

Resumo

Contexto: Sistemas de software se tornaram fundamentais nas atividades das pessoas e empresas nos últimos anos. A possibilidade de acesso fácil viabilizado pela popularização da plataforma web no primeiro momento, seguido pela plataforma móvel, vêm modificando a necessidade de acesso e processamento de informações das pessoas. Diante desse cenário, o desenvolvimento de software ganha mais um desafio, além da exigência por qualidade e produtividade. Em muitos casos, ele deve ser multiplataforma, e considerar diferenças e limitações dos diversos dispositivos existentes e futuros. Sistemas multiplataforma são normalmente compostos de várias soluções individuais, cada uma para um dispositivo ou plataforma diferente. Essa separação aumenta o custo do desenvolvimento, pois diferentes equipes de desenvolvimento são frequentemente necessárias para lidar com os requisitos específicos de cada dispositivo. Existem soluções multiplataforma disponíveis na indústria e no meio acadêmico, mas normalmente são restritas a plataformas préconfiguradas. Eles também normalmente consideram que as soluções individuais são iguais ou semelhantes, em termos de funcionalidade, entre si. **Objetivo**: Motivado por esse cenário, esta tese de doutorado apresenta uma abordagem em que o desenvolvimento e a manutenção de software multiplataforma ocorrem como uma única entidade de software, adotando uma visão mais holística. A abordagem tira proveito das soluções multiplataforma existentes na academia e indústria e adiciona três novos aspectos em relação à autonomia do desenvolvedor: suporte para incluir novas plataformas, extensão da abordagem e suporte para distribuir funcionalidade entre os dispositivos. **Metodologia**: A abordagem é baseada em uma *General-Purpose Language* (GPL), que permite ao desenvolvedor especificar conceitos de domínio em um nível de abstração mais alto que uma linguagem de programação normal, pois é livre de detalhes da plataforma e inclui alguns elementos de modelagem. As transformações automáticas são usadas para produzir código executável que pode ser adequadamente dividido e implantado separadamente em diferentes plataformas. A abordagem proposta foi avaliada de três maneiras. Na primeira avaliação, um sistema multiplataforma para o domínio de comércio eletrônico foi recriado usando a abordagem, caracterizando a prova de conceito da abordagem. A segunda avaliação foi realizada com 5 especialistas, que testaram a abordagem multiplataforma na prática. A última avaliação foi a solução de uma limitação apontada pelos especialistas através da possibilidade de extensão da abordagem, demonstrando a sua robustez. **Resultados**: Os resultados trouxeram evidências que apoiam as contribuições da abordagem, principalmente: a possibilidade de usar um único ambiente para criar uma representação de alta abstração do software, a capacidade de reutilizar conceitos semelhantes entre plataformas e o potencial de reduzir custos. As avaliações também apontaram algumas limitações, principalmente: a necessidade de um esforço inicial para criar e preparar adequadamente a plataforma usando os IDEs nativos, a necessidade de testes adicionais e alguns detalhes de implementação ausentes.

Palavras-chaves: Sistemas Multiplataforma, Desenvolvimento de Software Dirigido por Modelos, Dispositivos Móveis, Aplicativos web, Aplicativos Híbridos, Aplicativos Nativos, Engenharia Avante, Geração de Código e Modelagem.

Abstract

Background*: Software systems have become fundamental in the activities of individuals and companies in recent years. The possibility of easy access became possible after the popularization of the web platform in the first moment, followed by the mobile platform, have been changing the need for access and information processing. Given this scenario, software development gains a new challenge, beyond the requirement for quality and productivity. In many cases, it must be cross-platform and consider differences and limitations of the various existing and future devices. Cross-platform systems are normally composed of several individual solutions, each for a different device or platform. This separation increases the development cost, as different development teams are often needed for dealing with the specific requirements for each device. There are cross-platform solutions available in the industry and academia, but these are normally restricted to preconfigured platforms. They also normally consider that each individual solution is equal or similar, in terms of functionality, to each other.* ***Objective****: Motivated by this scenario, this doctoral thesis presents an approach where the development and maintenance of cross-platform software occurs as a single software entity, taking a more holistic view. The approach takes advantage of existing cross-platform solutions but adds three new aspects regarding the developer's autonomy: support to include new platforms, extension of the approach and support to distribute functionality across the devices.* ***Method****: The approach is based on a general-purpose language (GPL), which allows the developer to specify domain concepts in an abstraction level that is higher than a normal programming language, as it is free from platform details and includes some modeling elements. Automatic transformations are used to produce executable code that can be properly divided and deployed separately into different platforms. The proposed approach was evaluated in three ways. In the first evaluation, an existing cross-platform system was recreated using the approach. The second evaluation was conducted with 5 experts, who tested the cross-platform approach in practice. The last evaluation was the solution of a pointed out by the specialists through the possibility of extending the approach, demonstrating its robustness.* ***Results****: The results have brought evidence supporting the contributions of the approach, mainly: the possibility to use a single environment to create a high abstraction representation of the software, the ability to reuse similar concepts between platforms and the potential to reduce costs. The evaluations also pointed out some limitations, mainly: the need for an initial effort to properly create and prepare the platform using the native IDEs, the need for additional tests and some missing implementation details.*

Key-words: Cross-Plataform, Model Driven Development, Model Driven Engineering, Mobile Device, Web App, Hybrid App, Native App, Forward Engineering, Code Generation and Modeling.

Lista de ilustrações

Figura 1 – Cenários multiplataforma abordados pela amostragem. 63
Figura 2 – Relação dos benefícios trazidos por cada abordagem pesquisada, onde "S" significa que a abordagem explorou, em algum nível, o benefício. Porém, destaca-se que essa análise simples tem um propósito de sumarização, uma vez que existem diferentes níveis para os benefícios em cada abordagem. 64
Figura 3 – Números de trabalhos utilizando cada um dos benefícios do DSDM. . . 65
Figura 4 – Legenda dos trabalhos apresentados na Figura 5. 66
Figura 5 – Análise técnica das abordagens pesquisadas. 67
Figura 6 – Número de trabalhos que considera cada aspecto do desenvolvimento. . 69
Figura 7 – Número de trabalhos que considera cada estratégia DSDM. 70
Figura 8 – Número de trabalhos que considera cada plataforma. 71
Figura 9 – Número de trabalhos que considera cada linguagem de modelagem. . . 73

Figura 10 – Principais elementos do DSDM (DANIEL LUCRÉDIO, 2009). 76
Figura 11 – Arquitetura clássica de metamodelagem (DANIEL LUCRÉDIO, 2009). 77

Figura 12 – Uma visão geral de uma típica solução multiplataforma baseada em DSDM (esquerda) em comparação com a abordagem proposta nesta pesquisa (direita). 85
Figura 13 – Elementos da abordagem multiplataforma. 87
Figura 14 – Modelos de uso da abordagem proposta e seus principais elementos. . . 97

Figura 15 – Relação entre os objetivos, questões e métricas da abordagem GQM para a avaliação 1. 102
Figura 16 – Relação entre os objetivos, questões e métricas da abordagem GQM para a avaliação 2. 104
Figura 17 – Relação entre os objetivos, questões e métricas da abordagem GQM para a avaliação 3. 105

Figura 18 – Painel administrativo do sistema multiplataforma para plataforma web. 107
Figura 19 – Versão web para usuários finais do sistema multiplataforma. 108
Figura 20 – Versão para dispositivos móveis do sistema multiplataforma. A- Obtenção da localização atual do usuário. B- Visualização do Cardápio. C- Opção de realizar o pedido sem a presença do garçom. 109
Figura 21 – Ambiente de modelagem dos objetos do mundo real da versão de RA. . 109

Figura 22 – Dispositivo de RA e aplicativo de controle (*smartphone*). Quando o usuário seleciona um dos produtos listados no aplicativo (botões vermelhos, no lado B da figura), o mesmo produto no mundo real é destacado através do projetor, executado pelo dispositivo de RA (lado A da figura). 110
Figura 23 – Comunicação entre as versões do sistema multiplataforma. 111
Figura 24 – Modelos criados para a prova de conceito. 115
Figura 25 – Autenticação de usuários nas plataformas: web (A), Android (B) e iOS (C) respectivamente. 123
Figura 26 – Tela de produtos nas plataformas: web (A), Android (B) e iOS (C) respectivamente. 126
Figura 27 – Tela do carrinho de compras nas plataformas: web (A), Android (B) e iOS (C) respectivamente. 129
Figura 28 – Tela do carrinho de compras sendo exibida em quatro dispositivos reais: computador *desktop*, dois *tablets* (Android e iOS) e um *smartphone* (Android). 131
Figura 29 – Tela de pedidos nas plataformas: web (A), Android (B) e iOS (C) respectivamente. 133
Figura 30 – Total de apontamentos Prós (**C+**) e Contras (**C-**) dos especialistas para cada contribuição (**C**) da abordagem desta tese. 166
Figura 31 – Códigos gerados através da abordagem sendo executado nas plataformas iOS, web e Android. 187

Lista de tabelas

Tabela 1 – Resumo das principais abordagens para desenvolvimento multiplataforma. 29
Tabela 2 – Ferramentas apresentadas na Seção 2.2.1. 32
Tabela 3 – Principais ferramentas para desenvolvimento híbrido, interpretado e generativo. 34

Tabela 4 – Anos de publicação organizados por ambiente multiplataforma. 62

Tabela 5 – Comparativo da abordagem proposta com as principais ferramentas acadêmicas e industriais, apontando os pontos fortes e fracos assim como as contribuições **C3, C4 e C5**, originais para esta tese. 82

Tabela 6 – Contribuições elencadas para esta tese. 100

Tabela 7 – Detalhamento dos cálculos das funções comuns e específicas ao domínio do sistema multiplataforma abordado. 112
Tabela 8 – LOC total representando o código comum e específico no domínio de comércio eletrônico. 113
Tabela 9 – Análise das linhas de código em comum nas diferentes classes. 114
Tabela 10 – Exemplos de especificações feitas por meio da GPL e apresentadas neste capítulo. 116
Tabela 11 – Funções globais definidas para a camada de modelo. 117
Tabela 12 – Classes especificadas através da GPL para a camada Modelo. 122
Tabela 13 – Análise do número de LOC dos conceitos comuns especificadas através da GPL. 137
Tabela 14 – Análise da LOC para a prova de conceito. 138

Tabela 15 – Contribuições exploradas por cada tarefa na avaliação. 142
Tabela 16 – Curva de aprendizado da abordagem. 143
Tabela 17 – Avaliação da programação GPL: facilidade (**A**), intuitividade (**A**) e complexidade (**B**). 145
Tabela 18 – Avaliação dos modelos de plataforma. É possível a: inclusão dos detalhes técnicos (**A**), reutilização dos modelos (**B**) e extensão da abordagem(**C**). 147
Tabela 19 – Avaliação dos recursos dos modelos de plataformas. Capacidade de: automatizar a geração de código (**A**) e criar funções customizadas (**B**). 150

Tabela 20 – Avaliação do modelo de *deploy*. Distribuição de funcionalidades (**A**), a troca de plataformas no sistema (**B**) e o gerenciamento da geração de código (**C**). 152
Tabela 21 – Redução de custos e aumento de produtividade com o emprego da abordagem. 155
Tabela 22 – Análise da manutenção de um sistema através da abordagem. 157
Tabela 23 – Benefícios providos pela abordagem. 158
Tabela 24 – Avaliação da abordagem (Notas de 1 a 10) e novas ideias para reduzir custos para o desenvolvimento multiplataforma. 159
Tabela 25 – Notas dos especialistas para o atendimento de cada contribuição elencada para essa tese. 162
Tabela 26 – Tempo gasto por cada especialista em aprender a abordagem e em cada tarefa 164
Tabela 27 – Prós (**C+**) e Contras (**C-**) dos Especialistas (**E**), relatados através das Questões (**Q**) da entrevista, em relação as Contribuições (**C**) da abordagem desta tese. 164

Lista de abreviaturas e siglas

API	*Application Programming Interface*
APE	*Application Presentation Engine*
BPM	*Business Process Modeling*
CaaS	*Comunnication as a Service*
CIM	*Computation Independent Model*
CMS	*Content Management System*
CSS	*Cascading Style Sheets*
CPC	*Cross Platform Code*
DBMS	*Database Management Systems*
DRE	*Data Runtime Engine*
DSDM	Desenvolvimento de Software Dirigido por Modelos
DSL	*Domain Specific Language*
EMF	*Eclipse Modeling Framework*
GPS	*Global Positioning System*
HaaS	*Hardware as a Service*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
JSP	*Java Server Pages*
M2M	*Model to model*
M2T	*Model to Text*
MBE	*Model-Based Engineering*
MDA	*Model-Driven Architecture*
MDD	*Model-Driven Development*

MDE	*Model-Driven Enginering*
MOF	*Meta-Object Facility*
NC	*Native Code*
OMG	*Object Management Group*
OCL	*Object Constraint Language*
ORE	*Organization Runtime Engine*
PaaS	*Plataform as a Service*
PC	*Personal Computer*
PIM	*Plataform Independent Model*
PSM	*Plataform Specific Model*
PWA	*Progressive Web App*
REST	*Representational State Transfer*
SaaS	*Software as a Service*
SDK	*Software development kit*
SLA	*Service Level Agreement*
SO	Sistema Operacional
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
TI	Tecnologia da Informação
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
XaaS	*Everything as a Service*
XMI	*XML Metadata Interchange*
W3C	*World Wide Web Consortium*
WSDL	*Web Service Description Language*

Sumário

1 Introdução e justificativa . . . 16
1.1 Motivação e objetivos . . . 18
1.2 Contribuições . . . 19
1.3 Limitações e escopo . . . 20
1.4 Organização . . . 21

2 Análise do problema . . . 23
2.1 Caracterização do problema . . . 23
2.1.1 Abordagens para desenvolvimento multiplataforma . . . 23
2.1.1.1 Abordagem nativa . . . 23
2.1.1.2 Abordagem híbrida, interpretada e generativa . . . 25
2.1.1.3 Abordagem web . . . 27
2.1.1.4 Resumo . . . 28
2.1.2 Funcionalidade distribuída . . . 30
2.1.3 Resumo dos desafios . . . 30
2.2 Soluções existentes no mercado . . . 31
2.2.1 Soluções para o desenvolvimento nativo . . . 31
2.2.2 Soluções para o desenvolvimento híbrido, interpretado e generativo 33
2.2.3 Soluções para o desenvolvimento web . . . 35
2.2.4 Soluções para adaptação de conteúdo . . . 36
2.2.5 Soluções para a computação em nuvem . . . 37
2.2.6 Resumo . . . 38

3 Trabalhos relacionados . . . 39
3.1 Resultados . . . 45
3.1.1 Plataformas móveis . . . 45
3.1.2 Plataformas móvel, web e interoperabilidade . . . 53
3.1.3 Plataformas móvel, web, interoperabilidade e *desktop* . . . 56
3.1.4 Outras plataformas . . . 59
3.1.5 Análise Geral . . . 61
3.1.5.1 Benefícios explorados do conceito DSDM . . . 62
3.1.5.2 Análise técnica dos trabalhos . . . 66
3.1.5.3 Aspectos do desenvolvimento . . . 68
3.1.5.4 Estratégias DSDM utilizadas . . . 68
3.1.5.5 Plataformas alvo . . . 70
3.1.5.6 Linguagem de modelagem . . . 72

3.1.6 *Frameworks* multiplataforma mais recentes . . . 73

4 Fundamentação teórica: Desenvolvimento de Software Dirigido por Modelos 75
4.1 Abordagens da indústria para o DSDM . . . 78

5 Uma abordagem holística para o desenvolvimento de software multiplataforma . . . 80
5.1 Originalidade da proposta . . . 80
5.2 A Abordagem . . . 85
5.2.1 O modelo do sistema . . . 87
5.2.2 O modelo de plataforma . . . 91
5.2.3 O modelo de *deploy* . . . 95
5.2.4 Geração de código . . . 96

6 Avaliações . . . 100
6.1 Objetivos da avaliação 1 . . . 101
6.2 Objetivos da avaliação 2 . . . 102
6.3 Objetivos da avaliação 3 . . . 103

7 Primeira avaliação: prova de conceito . . . 106
7.1 Desenvolvimento do sistema multiplataforma . . . 106
7.1.1 Análises do sistema multiplataforma . . . 111
7.2 Estudo de caso. Recriando o sistema através da abordagem . . . 114
7.3 Resultados e Discussão . . . 136
7.3.1 Ameaças à validade . . . 139

8 Segunda avaliação: análise por especialistas . . . 140
8.1 Observações dos especialistas . . . 142
8.2 Discussão dos resultados . . . 163
8.2.1 Evidências e limitações apontadas pelos especialistas em relação as contribuições da abordagem desta tese (**M1**) . . . 164
8.2.1.1 Análises da **C1** . . . 167
8.2.1.2 Análises da **C2** . . . 169
8.2.1.3 Análises da **C3** . . . 170
8.2.1.4 Análises da **C4** . . . 171
8.2.1.5 Análises da **C5** . . . 172
8.2.1.6 Análises da **C6** . . . 173
8.2.1.7 Análises da **C7** . . . 174
8.2.2 Limitações . . . 175
8.2.3 Ameaças à validade . . . 176
8.2.4 Considerações finais . . . 178

9 Terceira avaliação: plataforma de testes de unidade 180
9.1 Estudo de caso. Inclusão de testes de unidade na abordagem 180
9.2 Resultados e Discussão 185

10 Considerações Finais 188
10.1 Contribuições da pesquisa 188
10.2 Limitações 190
10.3 Trabalhos futuros 190
10.4 Lições aprendidas 191
10.5 Produção acadêmica desta tese 192

Referências 194

A Apêndice 202

1 Introdução e justificativa

Ao longo dos anos, sistemas de software passaram por diversas evoluções em diferentes aspectos, seja no modo de desenvolvimento, necessidade de uso ou possibilidades de operação. Há alguns anos, os sistemas de software eram utilizados majoritariamente através de computadores de mesa (*desktop*), com baixa mobilidade. Com o surgimento dos computadores portáteis (*laptop*), a mobilidade foi ampliada, porém ainda com certa limitação devido ao tamanho, peso e duração de bateria. Atualmente, temos à disposição diversos tipos de dispositivos móveis e grande disponibilidade de rede para acesso à Internet, o que permitiu a operação de software em praticamente qualquer local. Entre os dispositivos disponíveis, os mais utilizados são: *smartphones*, *tablets*, centrais multimídia e, mais recentemente, os *smartwatches*, todos eles com acesso à Internet e sistemas operacionais que permitem a instalação e uso de diversos tipos de software (UMUHOZA; BRAMBILLA, 2016).

Tal ampliação do uso e diversidade de dispositivos gerou a demanda por software multiplataforma, ou seja, um único sistema de software que executa em vários dispositivos, com sistemas operacionais distintos. Assim, ao invés de se desenvolver para uma plataforma específica, é necessário desenvolver (ou adaptar) uma versão do software para cada plataforma-alvo. Esse problema não é atual, o trabalho de Viana e Andrade (2008) já tratava a geração automática de interfaces para aplicativos móveis e esse problema ainda persiste (CHEN et al., 2019).

Essa situação acaba levando um desafio aos desenvolvedores, os quais precisam ser capazes de lidar com as características específicas de cada dispositivo, tais como: tipos e versões dos sistemas operacionais (plataformas de execução), capacidade para armazenamento de dados, recursos disponíveis (GPS, câmera, status de conexão, etc), tamanho de telas, entre outros (WASSERMAN, 2010). Todas essas características impactam diretamente na produtividade, manutenção e qualidade dos aplicativos. Esse impacto se deve ao fato de que a equipe de desenvolvimento deve tratar essas questões com baixo nível de abstração e muitas vezes gerenciar várias versões do mesmo software, o que acarreta o aumento de custo e perda de foco nos conceitos do domínio.

Como exemplo dessa complexidade, foi divulgada em um site de notícia[1] uma nota curiosa referente às diferenças significativas em funcionalidade e qualidade entre as diferentes versões do aplicativo do Facebook. Segundo noticiado, a versão para Android desse aplicativo era inferior e, por esse motivo, os funcionários do Facebook foram instruídos a utilizar celulares Android para poder melhor vivenciar os problemas relatados e serem ca-

[1] www.businessinsider.com/facebook-Android-app-employees-2012-8

pazes de melhorar sua qualidade. Esse exemplo ilustra a complexidade do desenvolvimento para múltiplos dispositivos.

Além disso, existe um segundo desafio referente a software multiplataforma: pode haver diferenças entre as funcionalidades previstas para cada tipo de dispositivo. Em alguns casos, como no exemplo do aplicativo móvel do Facebook, as funções para Android e iOS devem ser as mesmas, já que o objetivo é maximizar o número de usuários. Mas em outros casos, há a necessidade de se projetar e implementar funções distintas entre as plataformas. Por exemplo, funções de um sistema de software que precisem capturar imagens e sua localização física são melhor executadas em um dispositivo móvel, como um *smartphone* ou câmera inteligente. Outro exemplo são os sistemas da área de saúde e bem estar de monitoramento de corridas e caminhadas. Tais sistemas têm funções que são melhor executadas em um *smartwatch*, como a medição de tempo, velocidade, altitude e distância percorridas durante o exercício, e funções que são melhor executadas em um computador ou *tablet*, como a visualização de um relatório de corrida em um mapa.

Outro exemplo dessas diferenças é o sistema *Google Maps*[2]. Há versões web e móvel desse sistema, e ambas possuem a mesma funcionalidade básica, que é a de mostrar mapas e realizar navegação. Mas a versão web permite a geração de rotas para impressão em papel, mais apropriada para quando o usuário está em casa, enquanto a versão móvel possibilita o acompanhamento curva-a-curva com instruções de voz, mais apropriada para quando o usuário está em movimento, a pé ou de carro.

Existe ainda um terceiro desafio, sempre há o risco do sistema ter que suportar novas plataformas no futuro próximo. A indústria tem investido no uso da computação em vários objetos (HOFFMAN; NOVAK, 2018) e essa realidade deve ser considerada para a engenharia de software. Para manter o software competitivo, em muitos casos, é importante utilizar o maior número possível de plataformas novas, o mais rápido possível.

Ainda com relação a esse ponto, a indústria de computação mostra uma predisposição para o lançamento de novos dispositivos de diferentes formatos e tamanhos para facilitar o acesso aos sistemas de software. Por exemplo, o *Microsoft Hololens*[3] promete ser um dispositivo computacional com um paradigma de interface humano-computador radicalmente diferente dos atuais. Adaptar os sistemas atuais a essa nova realidade irá possivelmente exigir grande esforço dos desenvolvedores no projeto e implementação de aplicações. Essa tendência aumenta ainda mais os desafios do desenvolvimento multiplataforma, uma vez que na prática isso exigirá customizações ou mesmo a completa reengenharia para adaptar sistemas já implantados para esses novos dispositivos.

[2] maps.google.com
[3] www.microsoft.com/microsoft-hololens/en-us

1.1 Motivação e objetivos

Em resumo, podemos elencar três problemas ou desafios principais da engenharia de software referentes ao desenvolvimento de software multiplataforma:

- Desenvolvimento de software compatível para as principais plataformas disponíveis no mercado atualmente e plataformas futuras, satisfeitas a restrição de se utilizar paradigmas conhecidos na computação, como orientado a objetos e estruturado. Para manter o software competitivo, em muitos casos, é importante ser executado no maior número de plataformas possível. Esse fato compromete a produtividade e manutenção devido à quantidade de detalhes técnicos e necessidade de se manter várias versões simultaneamente durante o desenvolvimento;
- Em alguns casos, o software deve ter suas funcionalidades distribuídas entre as versões para plataformas distintas, isto é, nem sempre as funções de uma versão para uma plataforma específica devem ser mantidas em outras versões. Funções distribuídas entre as versões do sistema podem explorar melhor os recursos de cada dispositivo, gerando vantagens competitivas e deixando o sistema mais completo; e
- A abordagem deve permitir que o desenvolvedor consiga incluir ou modificar funções para melhor atender às necessidades de um sistema específico. Isto é, o planejamento da abordagem deve considerar a possibilidade de extensões de sua estrutura pelos usuários finais, evitando que o desenvolvedor fique perigosamente "preso" às liberações dos fabricantes.

Uma caracterização mais detalhada do problema é apresentada no Capítulo 2 desta tese. Atualmente, existem soluções que ajudam a reduzir parte dos desafios citados: desenvolvimento híbrido, desenvolvimento web responsivo, arquitetura orientada a serviços, soluções para armazenamento temporário (*caching*) e sincronização de dados *offline*, computação em nuvem, adaptação de conteúdo, entre outras. A Seção 2.2 apresenta maiores detalhes sobre essas opções existentes. Porém, nenhuma delas resolve a essência do problema, que é o tratamento de todo o desenvolvimento e manutenção do software multiplataforma como um único sistema, considerando os três desafios elencados. Em outras palavras, o desenvolvedor ainda precisa trabalhar com múltiplos sistemas individuais que operam em conjunto.

Na academia também existem trabalhos que buscam abordagens promissoras para resolver esses problemas. Existem pesquisas em diferentes linhas, que propõem abordagens independentes de plataforma para o desenvolvimento de software para múltiplas plataformas. Porém, elas são em geral específicas para algumas plataformas e/ou cenários particulares. Não foi encontrada uma solução que resolve todos os problemas descritos

de forma única. O Capítulo 3 apresenta um estudo do estado da arte relacionado aos problemas aqui descritos.

1.2 Contribuições

Face a estes desafios, esta tese apresenta uma abordagem em que o desenvolvimento e a manutenção consideram o sistema multiplataforma como uma única entidade de software. Portanto, a abordagem é considerada holística, pois abrange várias características e busca solucionar uniformemente os desafios associados ao desenvolvimento multiplataforma. O desenvolvedor cria o software usando um único modelo (ou conjunto de modelos), usando uma *General-Purpose Language* (GPL) desenvolvida para apoiar a abordagem. A GPL é uma linguagem de programação com algumas construções de alto nível que permitem que diferentes conceitos e funções de domínio sejam especificados completamente e independentemente da(s) plataforma(s) em que serão executadas. Por meio de geradores, o código é obtido automaticamente para diferentes plataformas.

A abordagem também é holística no sentido de que não está restrita a um conjunto predefinido de plataformas suportadas. A adição de novas plataformas é incorporada à abordagem por meio da mesma GPL, com a qual os modelos de plataforma podem ser especificados e o código pode ser gerado. Portanto, a abordagem pode abranger uma ampla gama de dispositivos, atuais e futuros. Além disso, a abordagem permite que as plataformas existentes sejam estendidas ou modificadas, para melhor atender às necessidades de um sistema específico.

Finalmente, a abordagem pode ser usada para escolher em qual plataforma cada parte ou função será implantada. Isso facilita o trabalho de distribuir funcionalidades entre dispositivos e plataformas, por exemplo, implantar funções dependentes de GPS e câmera em *smartphones* apenas, e relatórios maiores e planilhas em um aplicativo web. Essa é uma grande diferença em relação às muitas soluções industriais existentes, que normalmente criam versões diferentes do mesmo software para plataformas diferentes.

Para avaliar a abordagem, três estudos foram realizados. Um primeiro estudo foi realizado pelos pesquisadores desta tese e consistiu de uma prova de conceito em que um sistema multiplataforma comercial existente foi recriado através da abordagem. Um segundo estudo envolveu cinco especialistas, que usaram a abordagem para criar partes de um software e em seguida forneceram informações detalhadas sobre o procedimento para uma minuciosa análise dos resultados. Para demonstrar a natureza holística da abordagem no suporte a novas e diferentes plataformas, os estudos envolveram a implantação do software nas plataformas web, Android e iOS, com diferentes tecnologias de armazenamento (*HashMap e SQLite*), diferentes linguagens de programação (Java, C# e Swift) e até incluiu um novo dispositivo criado exclusivamente para esta pesquisa: um dispositivo de

realidade aumentada (RA) baseado em RaspBerry Pi[4], com recursos simples de entrada e saída. Um terceiro estudo teve como objetivo demonstrar a robustez da abordagem, através da solução de um dos principais pontos levantados pelos especialistas no segundo estudo. Trata-se da inclusão de testes de unidade na abordagem desenvolvida, permitindo a especificação de testes genéricos através da GPL.

Os resultados dos estudos mostram que a abordagem pode ser usada com sucesso para tratar o desenvolvimento multiplataforma como uma única entidade de software. Também destacou algumas contribuições. A principal contribuição é a possibilidade de usar um único ambiente para criar uma representação com alto nível de abstração do software. Os resultados também demonstraram a capacidade de reutilizar conceitos semelhantes entre plataformas, possibilidade de distribuição de funcionalidades, extensão da abordagem pelos desenvolvedores e inclusão de suportes a novas plataformas. Os estudos também evidenciaram a robustez da abordagem, que se mostrou capaz de atender a diferentes dispositivos e plataformas, entre elas um dispositivo criado exclusivamente para esta pesquisa, e plataformas de testes de unidade. Por fim, a abordagem tem potencial para reduzir custos.

Para o desenvolvimento da abordagem, foram empregados os conceitos de Desenvolvimento de Software Dirigido por Modelos (DSDM), que consiste na união de técnicas de modelagem com transformação de software para proporcionar independência de plataforma.

1.3 Limitações e escopo

Destaca-se que a tese tem uma motivação prática real, tem características inovadoras com o potencial de contribuições científicas, tem um componente atemporal que tornou a abordagem independente de seu tempo, e uma certa ambição, que são essenciais para uma pesquisa de doutorado. Claro que as contribuições efetivas de uma pesquisa de doutorado certamente não irão atender a todo e qualquer cenário presente e futuro, devido às limitações de tempo e escopo.

Uma primeira limitação diz respeito à abrangência de domínio. A abordagem é holística e poderia, em tese, ser aplicada a diferentes domínios. Mas para esta tese as avaliações envolveram um domínio específico (comércio eletrônico). Ainda que restrito, esse domínio abrange diferentes possibilidades dentro do contexto do desenvolvimento multiplataforma. Porém, para outros domínios (como jogos eletrônicos, por exemplo) podem surgir desafios e detalhes não previstos nesta tese. Também vale destacar que não foram experimentados aspectos da interface com o usuário. A abordagem tem potencial para auxiliar nesta tarefa complexa, mas isso não foi testado devido às restrições de tempo.

[4] https://www.raspberrypi.org/

Uma segunda limitação diz respeito ao suporte aos dispositivos computacionais. A abordagem é genérica e pode gerar código para qualquer dispositivo, presente ou futuro. Porém, para isso é necessário que existam geradores de código para as linguagens de programação suportadas nos dispositivos. Atualmente, foram produzidos geradores para Java, C# e Swift. Outras linguagens podem ser incluídas no futuro, bastando para isso implementar os geradores correspondentes.

Ainda com relação a esse ponto, a abordagem oferece suporte para dispositivos que ainda não existem. Nesse sentido, como a abordagem se baseia na construção de geradores de código, é necessário que esse código utilize paradigmas de programação (e.g. orientação a objetos) e estruturas de código (e.g. laços, iteradores, funções) conhecidos no meio acadêmico e/ou comercial. Nesta tese assume-se que o lançamento de novos dispositivos ainda não existentes irá seguir o paradigma orientado a objetos utilizando as construções das principais linguagens de programação sendo utilizadas atualmente, baseados em estruturas de classes com métodos e funções. Existe a possibilidade do surgimento de novos paradigmas de programação que mudem radicalmente a forma com que o código seja gerado, reduzindo a aplicabilidade dos geradores desenvolvidos.

As avaliações também apontaram outras limitações. A limitação mais importante é a necessidade de um esforço inicial para criar e preparar adequadamente os detalhes da plataforma, antes que a abordagem possa ser útil. Isso é importante, pois a maioria dos desenvolvedores já tem esse suporte disponível nas ferramentas atuais. Usando a abordagem, o desenvolvedor é responsável por criar esse suporte. Além disso, durante a criação das plataformas na abordagem, usar os IDEs nativos das plataformas mostrou-se muito mais vantajoso do que programar diretamente na GPL. Ainda que isso ocorra apenas em um momento inicial, essa dependência vai contra o objetivo de se tratar o software como uma entidade única e em um único ambiente. Uma vez que as plataformas estejam prontas, essa necessidade é bastante reduzida. Os resultados também apontam alguns detalhes de implementação que podem ser aprimorados no futuro. As evidências empíricas coletadas constituem contribuições importantes para a continuidade desta pesquisa no futuro, até que esteja pronta para se tornar uma ferramenta e/ou um ambiente pronto para produção.

Independente dessas limitações, desenvolveu-se um mecanismo essencial, que permitiu superar parte dos desafios do desenvolvimento multiplataforma de forma diferente dos caminhos atualmente trilhados pela indústria.

1.4 Organização

Esta tese está organizada da seguinte maneira. Este primeiro capítulo apresenta uma visão geral da tese, discutindo motivação, objetivos, contribuições e limitações. O

restante do texto aborda de forma detalhada cada assunto. A caracterização do problema é apresentada no Capítulo 2 com mais detalhes, junto com as abordagens comerciais para resolvê-las e algumas ferramentas exemplificando cada abordagem. O Capítulo 4 apresenta informações sobre o conceito DSDM, utilizado para a confecção da abordagem. O capítulo 3 apresenta um estudo dos trabalhos relacionados ao tema desta tese, que constituem nas soluções acadêmicas e industriais que utilizam parte ou totalmente as técnicas mostradas nos Capítulos 2 e 4. O Capítulo 5 descreve a abordagem holística de desenvolvimento multiplataforma em detalhes. O Capítulo 6 contém um breve descrição das três avaliações realizadas. O Capítulo 7 apresenta a primeira avaliação da abordagem, que consistiu de uma prova de conceito. O Capítulo 8 apresenta a segunda avaliação da abordagem, que consistiu de uma avaliação por especialistas. O Capítulo 9 apresenta a terceira avaliação da abordagem, que consistiu na inclusão de uma plataforma de testes de unidade, em resposta a uma das principais limitações encontradas na segunda avaliação. Por fim, o Capítulo 10 apresenta algumas considerações finais e trabalhos futuros.

2 Análise do problema

Este capítulo trata da análise dos desafios relacionados a esta tese. Para isso, é descrito um domínio específico de software que tem a necessidade de ser multiplataforma devido a questões mercadológicas. Para garantir que este software seja multiplataforma sua concepção pode se dar de várias formas, podendo fazer uso de uma gama de tecnologias disponíveis no mercado. A Seção 2.1 apresenta a descrição do software multiplataforma e os principais cenários possíveis para sua concepção, a Seção 2.2 apresenta as principais soluções tecnológicas para o desenvolvimento disponíveis no mercado.

2.1 Caracterização do problema

Para exemplificar os desafios elencados para esta tese, considere um software para pedidos de massas através da Internet, encomendado por uma franquia de comida italiana[1]. Para alcançar todo o público-alvo da empresa é interessante planejar o desenvolvimento de modo a permitir o acesso para o maior número de plataformas possível. Nesse caso foram escolhidas as seguintes plataformas: computadores pessoais, *smartphones* e *tablets*. Cabe ressaltar que, embora os dispositivos móveis venham se popularizando entre os usuários, os computadores pessoais ainda possuem uma quantidade de usuários que deve ser considerada (DANYL, 2019).

A seguir são discutidas algumas alternativas que podem ser utilizadas para o desenvolvimento do software no cenário apresentado.

2.1.1 Abordagens para desenvolvimento multiplataforma

Existem basicamente cinco formas para o desenvolvimento de software multiplataforma atualmente: nativa, híbrida, interpretada, generativa e web. A primeira delas é a forma nativa.

2.1.1.1 Abordagem nativa

A forma nativa corresponde ao desenvolvimento direcionado a cada plataforma, utilizando uma linguagem de programação específica. Esta forma de desenvolvimento se beneficia em alguns fatores, entre eles podemos citar: maior desempenho, usabilidade padronizada pelo sistema operacional e melhor acesso aos recursos do hardware. Porém o desenvolvimento nativo não é multiplataforma, isto é, faz-se necessário desenvolver e

[1] Este é um cenário real vivenciado pelo autor desta tese

manter versões do software de forma individual para todas as plataformas previstas no projeto.

No cenário proposto, a abordagem nativa poderia corresponder à seguinte configuração:

- Existe uma versão web do software de entrega de massas, escrita em HTML5, para compor a interface, e uma linguagem cliente-servidor, assim como: Ruby, Python, PHP, C# ou JSP (SHIOTSU, 2014), e que contém todas as funcionalidades do sistema, incluindo a parte administrativa e a parte de confecção dos pedidos, podendo ser acessada a partir de qualquer computador.

- Existem também duas versões que rodam nos principais tipos de *smartphones* e *Tablets* existentes no mercado atual (LISA, 2013): uma versão para dispositivos que rodam iOS[2] e uma versão para dispositivos que rodam Android[3]. Tratam-se de aplicativos escritos em código nativo dessas plataformas (*Swift*[4] para iOS e Java[5] para Android) que implementa apenas a parte referente à confecção dos pedidos.

Nessa configuração, todo o sistema é acessível em um computador pessoal, via navegador web. Já a parte referente aos pedidos, destinada a ser utilizada pelos usuários, pode ser executada em dispositivos móveis, de forma *offline* e podendo se beneficiar das vantagens do desenvolvimento nativo previamente citadas.

Porém, existe maior esforço no desenvolvimento e manutenção, uma vez que esse cenário exige o desenvolvimento de três versões do mesmo sistema, cada uma desenvolvida em uma linguagem diferente e utilizando uma *Application Programming Interface* (API) diferente. Além disso, para cada versão devem ser planejadas e implementadas, em nível nativo ao sistema operacional, as questões relacionadas à interface, validação de dados, modelagem de dados, segurança, sequência de telas, comunicação e atualização dos dados entre as versões e outros detalhes técnicos que aumentam o custo do projeto.

Essas dificuldades aumentam quando são consideradas as constantes mudanças promovidas pelos fabricantes de dispositivos e evolução de seus sistemas operacionais. Como exemplo dessa situação, vale notar que cada versão do Android introduz mudanças nas APIs, obrigando os desenvolvedores muitas vezes a planejar a adequação de suas aplicações. A recente introdução do *Material Design* na versão *android Lollipop*[6] (2014) forçou algumas mudanças nos principais aplicativos da loja online da Google. De forma si-

[2] www.apple.com/ios/what-is/
[3] www.android.com/
[4] developer.apple.com/swift/blog/
[5] developer.android.com/training/index.html
[6] https://material.io/collections/get-started/

milar, a Apple está promovendo a linguagem *Swift* (lançamento em 2017) em substituição à linguagem *Objective-C*[7], também obrigando os desenvolvedores a promover mudanças.

É possível escolher apenas uma plataforma específica para o desenvolvimento, reduzindo o número de versões a serem desenvolvidas. Porém isso pode reduzir o público-alvo dos aplicativos, visto que existe uma disputa de números de usuários por sistema operacional (IDC, 2020). A falta de versões para os sistemas operacionais com uma grande quantidade de usuários pode representar redução de mercado e desvantagem competitiva em relação aos concorrentes.

2.1.1.2 Abordagem híbrida, interpretada e generativa

A forma nativa tornou o desenvolvimento muito complexo e contribuiu para o surgimento de soluções que resolvem parte do problema, como é o caso dos *frameworks* multiplataforma para desenvolvimento de aplicativos híbridos. O conceito de aplicativos híbridos surgiu como uma forma de centralizar o desenvolvimento para os dispositivos móveis. Utilizando tecnologia web (HTML5)[8] e *frameworks* específicos (HEITKÖTTER; HANSCHKE; MAJCHRZAK, 2013) é possível gerar versões do software para cada plataforma a partir do mesmo código, contribuindo para o aumento da produtividade e manutenção. Aplicativos híbridos utilizam o *browser engine* do dispositivo, renderizando toda programação web em tela cheia. O *browser engine* é uma API baseada nos navegadores móveis e permite iniciar e usar um aplicativo híbrido como se fosse um nativo.

Porém essa técnica não gera versões para computadores pessoais, sendo mais voltados para aplicações móveis. Além disso, não são puramente nativas às tecnologias de cada aparelho, podendo gerar interfaces fora dos padrões de usabilidade dos sistemas operacionais. Outro ponto desfavorável diz respeito ao desempenho: software que exige grandes recursos de processamento e/ou acesso aos recursos do hardware tem desempenho inferior em relação à forma nativa. Isso ocorre pelo fato de utilizar o *browser engine* para interpretar o software, ou seja, não são executados diretamente pelo sistema operacional (CHARLAND; LEROUX, 2011).

A abordagem interpretada é similar a abordagem híbrida, porém utiliza um interpretador de código ao invés do *browser engine*, deixando o desenvolvimento mais próximo da forma nativa. Devido a isso, a abordagem interpretada tem a disposição uma grande quantidade de linguagens, deixando de lado a tecnologia web e podendo utilizar ferramentas como Appcelerator Titanium[9] e LiveCode[10]. O interpretador é instalado junto ao aplicativo e fornece acesso aos recursos nativos do dispositivo em tempo de execução. Assim como na abordagem híbrida, a interpretada tem menor desempenho se comparada

[7] developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC
[8] www.w3.org/TR/html5/
[9] http://www.appcelerator.com/titanium
[10] https://livecode.com/

a forma nativa devido ao tempo de interpretação, mas maior desempenho em relação a forma híbrida. Esse desempenho superior se deve ao fato do interpretador transformar o código diretamente para o processamento interno do dispositivo ao passo que na híbrida o código é executado em *JavaScript* e precisará passar por uma camada de abstração antes de ser executado pelo dispositivo (DALMASSO et al., 2013).

A abordagem generativa é similar a abordagem híbrida e interpretada, no entanto o código é compilado como um aplicativo nativo e não precisa utilizar tecnologia web para o desenvolvimento e sim linguagens específicas, como o Haxe[11]. Outro aspecto que difere da abordagem híbrida e interpretada é a possibilidade de editar o código fonte gerado, permitindo implementar detalhes não suportados pelas ferramentas. Essa abordagem apresenta maior desempenho em relação às outras duas e natureza visual em conformidade com a plataforma (BERNARDES; MIYAKE, 2016).

No cenário proposto, as abordagens apresentadas poderiam corresponder à seguinte configuração:

- Existe uma versão web do software de entrega de massas, escrita em HTML5, para compor a interface, e uma linguagem cliente-servidor, assim como: Ruby, Python, PHP, C# ou JSP (SHIOTSU, 2014), e que contém todas as funcionalidades do sistema, incluindo a parte administrativa e a parte de confecção dos pedidos, podendo ser acessada a partir de qualquer computador. Esse cenário retrata a abordagem híbrida, para as abordagens interpretada e generativa a escrita da versão web poderia ser feita em Haxe ou Titanium por exemplo.

- Existe uma versão híbrida, interpretada ou generativa que roda nos principais tipos de *smartphones* e *tablets* existentes no mercado atual. Trata-se de um aplicativo escrito em código web (HTML5, *JavaScript* e CSS), Haxe ou Titanium que implementa apenas a parte referente à confecção dos pedidos. Diferentemente da abordagem nativa, aqui há menos limitação com relação às plataformas adotadas, pois grande parte dos *frameworks* permite a geração de versões para as plataformas mais utilizadas no mercado (HEITKÖTTER; HANSCHKE; MAJCHRZAK, 2013). Por exemplo, o aplicativo poderia ser executado, além de iOS e Android, também em dispositivos que rodam *windows phone*[12] ou *BlackBerry OS*[13], sem esforço adicional.

Essa configuração funciona de forma similar à nativa, mas com uma versão a menos para ser desenvolvida e mantida, reduzindo os esforços de manutenção de múltiplas versões. Porém, vale destacar que ainda é necessário manter a versão web em paralelo com a versão móvel, pois há diferenças em suas funcionalidades. Devido a essas diferenças, nas

[11] haxe.org/
[12] www.windowsphone.com/en-us/how-to/wp8/basics/whats-new-in-windows-phone
[13] developer.blackberry.com/bbos/java/documentation/

abordagens interpretada e generativa, também se faz necessário manter a versão web separada das versões móveis. Isso porque o desenvolvedor não consegue criar versões móveis e web com funcionalidades distintas através de uma mesma codificação nessas abordagens.

Como ponto negativo, a abordagem híbrida produz aplicativos com alguns problemas, tais como a interface não padronizada e um desempenho inferior. Para o cenário em questão, assim como em muitos outros, isso não é um problema muito crítico. Por esse motivo, a abordagem híbrida é bastante popular atualmente (HU et al., 2019).

Para a abordagem interpretada a questão do desempenho também afeta. E, como já citado, nas abordagens interpretada e generativa é necessário criar e manter a versão web separado da versão móvel, mesmo que utilizando a mesma tecnologia para o cenário proposto.

2.1.1.3 Abordagem web

Uma terceira opção seria trabalhar somente com a versão web do sistema. Trata-se de uma opção razoável, pois atualmente a maioria dos dispositivos possui acesso à Internet e navegadores web bastante similares.

Assim como os híbridos, os sistemas baseados em web também são interpretados e acessados por meio de um navegador, porém esses devem ser acessados através da URL (*Uniform Resource Locator*) e não são instalados nos dispositivos.

Há ainda o problema de leiaute das páginas em telas menores, o que pode ser resolvido com técnicas de interface responsiva. O conceito de responsividade (MARCOTTE, 2010) permite a adaptação da interface do software baseado em web para diferentes tamanhos de tela, tornando-o mais apto para atender aos requisitos de software multiplataforma. Há limitações, é claro, como é o caso dos recentes *smartwatch*, cuja tela é muito menor, exigindo uma adaptação mais radical do conteúdo, mas em geral as páginas responsivas tem boa aceitação.

Em contrapartida, a forma de desenvolvimento web possui algumas desvantagens em relação aos aplicativos nativos, híbridos, interpretados e generativos, como o caso do desempenho e usabilidade inferior (LITAYEM; DHUPIA; RUBAB, 2015). Isso pode reduzir o número de usuários devido à preferência por aplicativos (DANYL, 2019).

No cenário proposto, a abordagem web poderia corresponder à seguinte configuração:

- Existe uma única versão web do software de entrega de massas, escrita em HTML5, para compor a interface, e uma linguagem cliente-servidor, assim como: Ruby, Python, PHP, C# ou JSP (SHIOTSU, 2014), e que contém todas as funcionalida-

des do sistema, incluindo a parte administrativa e a parte de confecção dos pedidos, podendo ser acessada a partir de qualquer computador ou dispositivo móvel que tenha um navegador e uma tela razoável. A interface deve ser projetada de forma responsiva, ou seja, com possibilidade de adaptação conforme o tamanho da tela.

Essa configuração tem a vantagem de possuir o menor esforço de desenvolvimento e manutenção. Além de fornecer recursos que se aproximam muito da experiência de um aplicativo nativo, graças à tecnologia recentemente lançada chamada *Progressive Web App*[14] (PWA). PWA fornece uma série de benefícios ao usuário, entre eles: a utilização do web site no modo *offline*, redução no uso da rede pelo fato de poder transferir apenas o conteúdo e manter o *app shell*, maior velocidade no carregamento mesmo em redes instáveis, utilização de notificação *push* (WARREN et al., 2014) semelhantes aos aplicativos nativos, opção de executar o site em tela cheia, execuções de rotinas em paralelo mesmo com o navegador minimizado, entre outras. O PWA utiliza *Service Workers*[15], que é basicamente um conjunto de *scripts* que permite o processamento do site mesmo em ambientes sem conectividade e em segundo plano.

Porém, mesmo com o uso do PWA, a plataforma web possui limitações no uso de recursos dos dispositivos, assim como: não faz acesso ao *Bluetooth*, NFC, *Fingerprint*, GPS, Câmera, entre outros. Os navegadores compatíveis com o PWA atualmente são: Chrome, Opera e Firefox. Os demais apontam interesse na implementação da tecnologia, porém ainda não oferecem suporte. O espaço de armazenamento disponibilizado por cada navegador[16] também é um fator limitante para os aplicativos baseados em web, apresentando vantagem para os aplicativos nativos nesse aspecto. A interface web não segue os padrões do sistema operacional podendo causar frustração nos usuários. Novamente, nesse cenário, assim como em outros, isso pode não ser um problema crítico.

2.1.1.4 Resumo

A Tabela 1 resume as cinco abordagens de desenvolvimento citadas neste capítulo, suas vantagens e desvantagens.

Com relação à Tabela 1, é importante destacar que as vantagens e desvantagens assinaladas não têm a mesma importância para todos os cenários. Por exemplo, do ponto de vista do usuário, a abordagem nativa é obviamente a melhor, porém a necessidade de múltiplas versões prejudica o desenvolvedor. Por outro lado, a abordagem web é mais vantajosa para o desenvolvedor, mas não para o usuário em muitos casos. Também vale ressaltar que as formas de desenvolvimento híbrido, interpretado, generativo e web não atendem às novas plataformas disponibilizadas no mercado, tais como centrais multimí-

[14] developers.google.com/web/updates/2015/12/getting-started-pwa
[15] developers.google.com/web/fundamentals/primers/service-workers
[16] developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa

Tabela 1 – Resumo das principais abordagens para desenvolvimento multiplataforma.

Abordagem	Vantagens	Desvantagens
Nativa	Maior desempenho. Interface segue o padrão do SO. Funciona *offline*. Aplicativo é instalado. Permite maior aproveitamento do hardware.	É preciso manter uma versão para cada dispositivo. Desenvolvedor deve dominar as linguagens e ambientes para cada SO. É preciso manter uma versão para PCs diferente da móvel. Custo de desenvolvimento e manutenção alto.
Híbrida, Interpretada e Generativa	Uma única versão móvel. Funciona *offline* Aplicativo é instalado . Aproveitamento bom do hardware. Custo de desenvolvimento e manutenção razoável. Generativa: desempenho similar a nativa.	É preciso manter uma versão para PCs diferente da móvel (Apenas no caso da versão híbrida). Interface não segue o padrão do SO (Apenas no caso da versão híbrida). Híbrida: desempenho razoável. Interpretada: desempenho superior a Híbrida e inferior a nativa.
Web	Uma única versão. Funciona *offline* para o PWA Maior parte do processamento e armazenamento é realizado no servidor, aliviando o uso de recursos do dispositivo. Custo de desenvolvimento e manutenção baixo.	É preciso cuidado com a interface (responsiva). Aproveitamento básico do hardware (não é possível armazenar grande volume de dados, por exemplo). Interface não segue o padrão do SO. Desempenho razoável. Limitações para funcionamento *offline*. Aplicativo não é instalado.

dias e *smartwatch*, e muito menos às plataformas futuras. Caso o projeto do cenário em questão precise atender a essas novas plataformas, o desenvolvedor deverá recorrer às soluções nativas, aumentando o número de ambientes para o desenvolvimento. Dessa forma, conclui-se que nenhuma das abordagens é a ideal.

2.1.2 Funcionalidade distribuída

Para exemplificar os conceitos desta seção é apresentado um segundo cenário que trata de um sistema para medição de pragas em produções agrícolas[17]. A diferença desse segundo cenário está na distribuição das funcionalidades entre as versões do software para os dispositivos, que é realizada de modo mais evidente e necessário.

O software deve ser projetado de forma a permitir a realização da checagem de pragas no campo através do processamento de imagens e geolocalização, no intuito de registrar o local exato de cada análise. Uma vez realizada a análise em campo, os dados deverão ser processados e acessados por um especialista. O especialista analisa a quantidade de pesticida sugerida pelo software em cada região processada. Após a análise e aprovação do especialista, os dados são exportados para um veículo responsável por pulverizar a plantação agrícola com pesticidas de forma semiautomática.

Esse cenário apresenta uma situação similar ao do software para pedidos de massas através da Internet. Porém, nesse exemplo, as diferentes versões precisam conter funcionalidades complementares. Portanto a abordagem web é a menos indicada, pois haverá naturalmente versões bastante distintas, fazendo uso especializado dos recursos de hardware. Como resultado, recomenda-se que existam pelo menos duas versões: uma web e uma nativa, híbrida, interpretada ou generativa para dispositivos móveis.

A versão para dispositivos móveis deve funcionar sem conectividade com a Internet (modo *offline*) e também permitir uso dos recursos de GPS (*Global Positioning System*) e fotografia do dispositivo. Os dados gerados das versões do software para os dispositivos móveis devem ser transmitidos, assim que for identificada a presença de Internet, para uso em uma versão web. A versão web irá permitir a utilização de monitores grandes, em computadores pessoais ou *tablets*, para a análise do mapa de forma mais adequada pelo especialista.

Nesse cenário, existe um elemento complicador. Apesar de cada versão possuir funções distintas, certamente haverá partes do software que compartilham conceitos similares. Por exemplo, cálculos matemáticos para geolocalização devem provavelmente existir em ambas as versões. Supondo que haja um defeito em um desses cálculos, ambas as versões precisam ser corrigidas, gerando duplicação de esforço. Esse exemplo evidencia os problemas gerados pelo fato de que, apesar de se tratar de um único sistema conceitualmente, existem múltiplos sistemas individuais operando em conjunto.

2.1.3 Resumo dos desafios

Seja qual for a forma de desenvolvimento, os detalhes técnicos envolvidos no desenvolvimento multiplataforma reduzem a abstração do domínio, além de aumentar o custo

[17] Este é um outro cenário real vivenciado pelo autor desta tese

do desenvolvimento e manutenção. Um único sistema de software não é mais tratado como um sistema único durante o desenvolvimento. Ambos os cenários retratam os desafios no desenvolvimento de software multiplataforma, desafios esses que foram a motivação desta tese:

- Desenvolvimento de software para as principais plataformas disponíveis no mercado e para plataformas futuras;
- Possibilidade de extensões na abordagem pelos desenvolvedores de acordo com a necessidade, evitando que o desenvolvedor fique perigosamente "preso" às liberações dos fabricantes; e
- Funcionalidade distribuída entre as versões para plataformas distintas.

2.2 Soluções existentes no mercado

Devido à natureza dinâmica do desenvolvimento multiplataforma, a indústria está em constante evolução para solucionar os desafios apresentados. Uma série de tecnologias são lançadas e melhoradas no intuito de aliviar a complexidade do desenvolvimento multiplataforma. Nesta seção são apresentadas exemplos de ferramentas para o desenvolvimento de cada abordagem mostrada na seção anterior.

2.2.1 Soluções para o desenvolvimento nativo

Embora existam vários sistemas operacionais móveis disponíveis (HEITKÖTTER; HANSCHKE; MAJCHRZAK, 2013), as plataformas Android e iOS dominam o mercado em respectiva ordem de quantidade de usuários (IDC, 2020). Portanto, essas plataformas devem ser consideradas com maior prioridade para o desenvolvimento multiplataforma. A Tabela 2 apresenta as referências das ferramentas mostradas nesta seção.

Para desenvolver aplicações nativas, devem ser utilizadas as linguagens e ambientes específicos de desenvolvimento para cada plataforma-alvo. No caso do Android, o desenvolvimento pode ser realizado através de um IDE (*Integrated Development Environment*) própria da plataforma *Android Studio* ou outros IDEs que permitam a utilização do Android SDK (*Software Development Kit*) e do ADT (*Android Development Tools*). O SDK contém as ferramentas básicas para o desenvolvimento do aplicativo e o ADT estende a capacidade do SDK, dando suporte, por exemplo, à criação da interface do usuário, criação rápida de novos projetos, exportação do arquivo executável para a distribuição, entre outras funções.

Existem duas linguagens oficiais para desenvolvimento Android, o Java e o Kotlin, porém é possível utilizar outras linguagens como o *Visual Basic* através da ferramenta

Tabela 2 – Ferramentas apresentadas na Seção 2.2.1.

Ferramentas	**Referências**
Android Studio	https://developer.android.com/sdk/index.html
Android SDK	https://developer.android.com/sdk/installing/index.html
Android Development Tools	http://developer.android.com/tools/sdk/eclipse-adt.html
Linguagem Java	https://www.java.com/en/about/
Android Developer	http://developer.android.com/index.html
Basic4android	http://www.b4x.com/
Sistema Operacional OS X	http://www.apple.com/br/osx/what-is/
Apple	https://www.apple.com/br/
iOS SDK	https://goo.gl/B1xYRj
Xcode para iOS	https://developer.apple.com/xcode/
Interface para Builder iOS	https://developer.apple.com/xcode/interface-builder/
Instruments para iOS	https://goo.gl/fyhQvT
Simulator para iOS	https://goo.gl/T73wsC
Visual Studio	https://www.visualstudio.com/pt-br/visual-studio-homepage-vs.aspx
Momentics BlackBerry	http://developer.blackberry.com/native/downloads
Eclipse	http://www.eclipse.org
BlackBerry Java SDK	http://developer.blackberry.com/bbos/java
NetBeans	https://netbeans.org/
BlackBerry Java SDK	http://www.blackberrybrasil.com.br/rim-libera-atulizacoes-java-e-ferramentas-de-desenvolvimento-web-para-blackberry/
BlackBerry Native SDK	https://goo.gl/avX2qJ

Basic4android. O desenvolvimento e a geração do executável do aplicativo para Android pode ser realizado nos principais Sistemas Operacionais (SOs) disponíveis para computadores pessoais, não restringindo o local de criação.

O desenvolvimento de aplicativos nativos para iOS é oficialmente realizado por meio do sistema operacional OS X, pertencente à Apple. Não há uma forma oficial de gerar o executável em outros SOs. O iOS SDK é o pacote de ferramentas necessários para o desenvolvimento de aplicativos para iOS. Esse pacote contém o IDE *Xcode* que corresponde ao ambiente de desenvolvimento, o *Interface Builder* que auxilia na criação da interface, o *Instruments* que analisa o consumo de recursos do aplicativo, e o iOS *Simulator*, responsável por simular a execução do aplicativo.

Aplicativos iOS são desenvolvidos atualmente através da linguagem *Swift*, a qual tomou o lugar da *Objective-C*, devido às melhorias promovidas por essa nova linguagem. Entre as melhorias estão a manutenção facilitada, maior segurança, melhor gerenciamento de memória, maior produtividade, necessidade de menos código, entre outras vantagens (SOLT, 2015).

O IDE oficial para o desenvolvimento de aplicativos nativos para *windows phone* é o *Visual Studio* (VS), da Microsoft. Como padrão, versões do VS 2012 ou superiores já possuem o SDK para *windows phone* integrado. Para as demais versões é necessário realizar a instalação[18]. O VS conta com versões para os principais SOs disponíveis no mercado e permite o desenvolvimento na plataforma de preferência do desenvolvedor. O VS permite a utilização de múltiplas linguagens de programação, portanto aplicativos para *windows phone* podem ser criados através das linguagens: C++, C#, Microsoft *Visual Basic* e *JavaScript*, Embora as linguagens mais utilizadas e com maior quantidade de material disponível são: C# e Microsoft *Visual Basic*[19].

Entre os três SOs apresentados, o *BlackBerry* é a plataforma que possui a menor participação no mercado[20] de *smartphones* e possui probabilidades reais de sair totalmente do mercado, devido à queda nas vendas. Porém, é considerada nesta seção pelo fato de ainda comercializar dispositivos móveis (STATISTA, 2020). Assim como o Android e *windows phone*, o IDE da RIM (*Research In Motion*, fabricante da *BlackBerry*) permite o desenvolvimento de aplicativos em diferentes SOs, por meio da ferramenta *Momentics*. Esse IDE é baseado na plataforma Eclipse e pode ser programada através da linguagem C ou C++. A RIM fornece o *BlackBerry* Java SDK para Eclipse ou NetBeans, permitindo o desenvolvimento de aplicativos através da linguagem Java. Outra possibilidade é a utilização do Visual Studio configurado com o *BlackBerry Native SDK* e o *BlackBerry Native Plug-in* para Microsoft Visual Studio, tendo como linguagem padrão o C e C++.

2.2.2 Soluções para o desenvolvimento híbrido, interpretado e generativo

Essa questão técnica que envolve produtividade e manutenção dos aplicativos, junto com questões comerciais, contribuíram para o surgimento de alternativas à forma nativa de desenvolvimento, de forma a promover maior produtividade e abrangência de plataformas com menor esforço (HEITKÖTTER; HANSCHKE; MAJCHRZAK, 2013). Uma dessas alternativas, como já citadas, são as abordagens híbrida, interpretada e generativa. A Tabela 3 apresenta as referências de algumas ferramentas usadas por cada uma das abordagens apresentadas nesta seção. Mais ferramentas utilizando as abordagens desta seção serão tratadas com mais detalhes no Capítulo 3, destinado aos trabalhos relacionados.

Existe uma série de *frameworks* que apoiam o desenvolvimento híbrido, interpretado e generativo, por meio dos quais é possível desenvolver aplicativos utilizando uma única tecnologia para as várias plataformas, aumentando assim a produtividade. Entre esses *frameworks* multiplataforma alguns estão citados na Tabela 3 como exemplo.

[18] developer.microsoft.com/pt-br/windows/downloads/sdk-archive
[19] msdn.microsoft.com/pt-br/library/windows/apps/dn465799.aspx
[20] www.gartner.com/newsroom/id/3609817

Tabela 3 – Principais ferramentas para desenvolvimento híbrido, interpretado e generativo.

Ferramentas	**Referências para o desenvolvimento híbrido**
Cordova	http://cordova.apache.org
PhoneGap	http://docs.phonegap.com/en/2.3.0
IBM Worklight	shorturl.at/ekmTW
AppMobi	https://appmobi.com
Ferramentas	**Referências para o desenvolvimento interpretado**
Titanium	http://www.appcelerator.com/titanium
Titanium Studio	http://preview.appcelerator.com/studio
LiveCode	https://livecode.com
Rhomobile	http://rms.rhomobile.com
Ferramentas	**Referências para o desenvolvimento generativo**
React Native	facebook.github.io/react-native
Haxe	haxe.org
Kotlin	kotlinlang.org

Cada *framework* possui suas características específicas de desenvolvimento, como IDE, arquitetura de desenvolvimento distinto, permissão para uso de API's, acesso a recursos do dispositivo, entre outras. Todas elas podem ser distribuídos de forma similar ao desenvolvimento nativo, isto é, através das lojas de aplicativos na Internet (*app store*) mantidos por cada sistema operacional.

Um dos *frameworks* multiplataforma mais populares do mercado é o PhoneGap (FISCHER, 2015). Tal *framework Open Source*, distribuído pelo Apache Cordova, permite o desenvolvimento de aplicativos híbridos e podem funcionar de modo *online* e *offline* no dispositivo móvel. O PhoneGap não possui um IDE próprio para o desenvolvimento, mas disponibiliza uma API para ser utilizada no ambiente de preferência do desenvolvedor, que pode ser desde um IDE com muitos recursos até um editor de textos simples. Portanto, é importante que a geração do executável seja realizada em ambiente configurado para cada plataforma. Para o Android, o PhoneGap recomenda o uso do Eclipse com o *Android SDK* e *ADT plugin* configurados. Já a geração do executável para iOS deve ser realizada no IDE *XCode* através da plataforma OS X.

Assim como o PhoneGap, o Titanium também é *Open Source*, porém possui um IDE próprio para o desenvolvimento, o Titanium Studio, que possui versões para os principais SOs disponíveis e é considerada uma abordagem interpretada. Diferente do PhoneGap, o Titanium não utiliza HTML e CSS para construir o aplicativo e sim a linguagem *JavaScript*, que é utilizada para criar a interface, a lógica e o modelo de dados (HEITKÖTTER; HANSCHKE; MAJCHRZAK, 2013). A partir do código base, o Titanium realiza o *deploy* para as plataformas desejadas. No caso do iOS é necessária a plataforma OS X.

Para a abordagem generativa existe o *framework* REACT NATIVE, desenvolvido

pelo Facebook[21]. Este *framework* comercial permite a escrita do código através de tecnologias web (XML, CSS e *JavaScript*) e possui o padrão caixa preta, onde não é possível estender seu metamodelo. Seu diferencial é a geração de códigos nativos legíveis para ambas as plataformas, onde é possível a adição de novas funcionalidades diretamente no código fonte gerado. Embora este *framework* permita o desenvolvimento multiplataforma em apenas uma linguagem, ele não apresenta aumento no nível de abstração, trocando uma linguagem de programação por outra. O REACT NATIVE não permite o acesso as API's nativas de cada plataforma, tal acesso é feito através do *framework*. Esse fato mantém o desenvolvedor dependente de novas atualizações do *framework* para acompanhar a evolução tecnológica das plataformas.

Além dos *frameworks* multiplataforma citados nesta seção, atualmente existe uma gama de outras ferramentas que auxiliam no desenvolvimento de aplicativos, com características próprias de desenvolvimento (REYNOLDS, 2017). Nota-se que essa vasta oferta de soluções evidencia a preocupação da indústria com o desenvolvimento multiplataforma e enfatiza o desafio da produtividade.

2.2.3 Soluções para o desenvolvimento web

O desenvolvimento baseado em web também pode ser considerado uma alternativa para a construção de aplicativos. Assim como os aplicativos híbridos, os sistemas web podem ser construídos utilizando HTML5, CSS e *JavaScript* (AMATYA; KURTI, 2014).

Os sistemas web também são conhecidos como aplicativos HTML5, e se diferem dos aplicativos híbridos em alguns pontos. Uma das principais diferenças é a distribuição que não é feita através das lojas de aplicativos. Os usuários precisam digitar no navegador do dispositivo a URL (*Uniform Resource Locator*) do aplicativo. Outro detalhe se refere ao desenvolvimento que não necessita obrigatoriamente de um *framework* específico, como no caso dos aplicativos híbridos (KONECKI, 2014).

O ajuste do software web para diferentes tamanhos de tela é uma questão crucial para a multiplataforma. Para isso, existe o parâmetro *viewport* do HTML5. Através do qual é possível controlar como a página será exibida no dispositivo móvel, permitindo que o usuário utilize o aplicativo em um computador pessoal ou em um dispositivo móvel.

O conceito usado para a adaptação de aplicativos web para os diferentes tamanhos e orientações de tela é conhecido como responsividade, termo cunhado em 2010 (MARCOTTE, 2010). Esse conceito é responsável pela adaptação da interface do software para diferentes tamanhos e orientação de telas. Tal característica permite que o usuário escolha onde o software é acessado. Em situações onde são realizadas atividades mais complexas, assim como formulários utilizando grandes volumes de dados, manipulação de mapas e

[21] www.facebook.com

imagens, é conveniente a operação em dispositivos com uma tela maior, como computadores pessoais ou *tablets*. Para casos onde a complexidade da tarefa é menor, por exemplo a realização de consultas, o sistema pode ser operado através de um dispositivo móvel.

A possibilidade de acesso do mesmo aplicativo através de diferentes dispositivos contribui para o aumento na produtividade e manutenção do desenvolvimento, uma vez que uma única aplicação web pode ser acessada por qualquer dispositivo que esteja equipado com um navegador (MEHTA; JHA, 2014), fato que não ocorre nos aplicativos híbridos.

Existem ferramentas que auxiliam a criação da responsividade, dentre as quais se destaca o *Bootstrap*[22] como um dos *frameworks* mais utilizados. O *Bootstrap* provê bibliotecas que adaptam a interface de sites para os diferentes tamanhos e orientação de tela. Além disso, ele provê recursos para tornar a interface mais apresentável com menos esforço do desenvolvedor.

Outro ponto que deve ser destacado em relação aos aplicativos web é que, pelo fato de não precisar ser instalado nos dispositivos, o uso de memória é reduzido, já que o armazenamento e processamento da aplicação são realizados quase que totalmente no servidor. Porém, para o funcionamento *offline* é necessário que o aplicativo seja preparado para isso, fazendo o uso de tecnologias como a PWA.

2.2.4 Soluções para adaptação de conteúdo

A adaptação da arquitetura de sistemas de software ou adaptação de conteúdo visa modificar conteúdos direcionados aos computadores pessoais para serem apresentados em dispositivos móveis, como *smartphones* e *tablets*. Software direcionado para computadores pessoais possuem diferenças de recursos que dificultam a criação de modelos para atender outras plataformas, como o caso dos dispositivos móveis. Uma solução para esse problema pode ser utilizar mecanismos que realizem adaptações de conteúdo de acordo com os recursos de cada dispositivo móvel (SANTANA et al., 2007). Segundo Buchholz e Schill (2003), a adaptação de conteúdo pode ser realizada:

a) No servidor de origem do site, obrigando o servidor a possuir diversas versões do mesmo conteúdo. Após reconhecer o dispositivo do usuário de forma explícita ou implícita, o conteúdo é enviado de acordo com as limitações de recursos do dispositivo.

b) No dispositivo do usuário, realizando os processos de adaptação de conteúdo conforme os recursos disponíveis no dispositivo. O ponto negativo desse tipo de adaptação é a sobrecarga de processamento no dispositivo, podendo resultar em uma

[22] getbootstrap.com/

demora no tempo de resposta ao usuário de acordo com os recursos de processamento disponíveis. A adaptação nesse caso pode ser realizada através de comandos do HTML5 que permitem mostrar ou ocultar elementos de acordo com o tamanho e orientação da tela do dispositivo (ROUDAKI; KONG; YU, 2015).

c) Em um servidor de adaptação dedicado, onde um mecanismo de adaptação, rodando em um servidor dedicado, intercepta as requisições de um dispositivo e realiza a adaptação. Esse método de adaptação alivia o servidor de origem e o dispositivo, porém acrescenta uma sobrecarga principalmente na comunicação entre eles. Caso a adaptação de conteúdo ocorra em um servidor dedicado, ele pode ser acessado a partir de dispositivos de borda (*proxy*). Esses servidores podem oferecer uma série de serviços adaptadores com valor agregado, além da conectividade com a web. De acordo com Hofmann, Beck e Condry (2000), dentre esses serviços destacam-se: escaneamento de vírus, classificação e filtragem de conteúdo, tradução de linguagens de marcação, transcodificação de imagens e inserção de publicidade.

2.2.5 Soluções para a computação em nuvem

A computação em nuvem pode prover maior flexibilidade de recursos de hardware em relação a servidores convencionais, melhorando o desempenho dos sistemas de software. Dada a necessidade de se construir software multiplataforma, a computação em nuvem pode facilitar a comunicação entre as versões do software para cada dispositivo, servindo como um único ponto de convergência de processamento e armazenamento.

Entre os exemplos de tecnologia de suporte a essa integração, pode-se citar o *Salesforce Mobile SDK*[23], que é uma API de código aberto que auxilia na sincronização dos dados entre cliente e servidor, e que conta com métodos que monitoram a presença de Internet para enviar ou receber dados do servidor, viabilizando a comunicação com a nuvem de forma segura.

Existem também os serviços de processamento em nuvem, de *Cloud Storage*[24] e *Cloud Computing Hosting* (MORENO-VOZMEDIANO et al., 2017), que possibilitam a integração de aplicativos usando a nuvem como meio. O *Cloud Storage* é um serviço oferecido por várias empresas, entre as quais podemos citar a Google, Amazon[25] e UOL[26]. Esse serviço permite o armazenamento e recuperação de dados gerados por aplicativos para várias finalidades, entre elas armazenamento de fotos, realização de *backup*, arquivamento de relatórios, *disaster recovery*, entre outros. O arquivamento de dados nas nuvens permite a integração dos dados entre as versões dos aplicativos através dos serviços de sincroniza-

[23] developer.salesforce.com/page/MobileSDK
[24] cloud.google.com/storage/docs/overview
[25] aws.amazon.com/pt/s3
[26] www.uolhost.com.br/uol-cloud-computing.html

ção disponíveis para os aplicativos, além de reduzir o espaço utilizado no dispositivo pelo aplicativo.

O *Cloud Computing Hosting* é um serviço de hospedagem de software web com flexibilidade na alocação de recursos, provendo uma solução sob medida para software com diferentes necessidades de recurso. Assim, é muito comum que sistemas multiplataforma rodem parcialmente em hospedagem na nuvem.

2.2.6 Resumo

Analisando-se o cenário atual, nota-se que não há uma solução que consiga resolver os desafios do desenvolvimento de software multiplataforma sem causar algum tipo de prejuízo ao resultado final. O desenvolvedor acaba tendo que escolher entre os problemas causados por múltiplas versões de um mesmo software, e adotar uma solução mais limitada, mas que consiga atender aos requisitos mínimos. Esse não é o cenário ideal.

Além disso, em todos os casos, é necessário lidar com detalhes em um baixo nível de abstração, tratando um único sistema conceitual como múltiplos sistemas que operam em conjunto, levando a dificuldades no desenvolvimento e manutenção dos sistemas atuais. Um conceito que tem mostrado bons resultados para a solução de grande parte da questão apresentada é o DSDM, abordado com mais detalhes no próximo capítulo.

3 Trabalhos relacionados

Foi realizada uma revisão bibliográfica visando obter a maior quantidade possível de trabalhos correlatos e realizar o levantamento do estado da arte de forma aprofundada.

Para atender os requisitos deste tópico, foram pesquisados artigos nos seguintes repositórios de trabalhos científicos: IEEE, Springer, ACM, *Engineering Village* Elsevier, *ScienceDirect*, *Web of Science*, Scopus e Google Acadêmico. Foram utilizadas duas técnicas, sendo a primeira o uso das palavras-chave desta tese de forma aleatória, priorizando os resultados mais recentes. As palavras-chave utilizadas foram: Sistemas Multiplataforma ou *Cross-Plataform Software*, Desenvolvimento de Software Dirigido por Modelos ou *Model Driven Development*, Dispositivos Móveis ou *Mobile Device*, Aplicativos web ou *Web App*, Aplicativos Híbridos ou *Hybrid App*, Aplicativos Nativos ou *Native App*, Engenharia Avante ou *Forward Engineering*, Geração de Código ou *Code Generation* e Modelagem ou *Modeling*. A segunda técnica foi a composição de *strings* de busca, que visam encontrar artigos através das combinações das palavras-chave de uma forma lógica. Como exemplo dessas combinações temos: *((Model-Driven OR Model-Driven Development) AND (Cross-Platform OR Multiple Platforms) AND (Tool OR Approach OR Framework))*.

Essa lógica permitiu a elaboração e uso dos seguintes *strings* de busca:

1. *String* usado na IEEE: ((((Title:"Model-Driven") OR (Title:"Model-Driven Development") OR (Title:"Modeling")) OR ((Title:"Cross-Plataform") OR (Title:"Multiple Platforms")) OR (((Abstract:"Model-Driven Development") OR (Abstract:"Model-Driven")) OR ((Abstract:"Cross-Plataform") OR (Abstract:"Multiple Platforms"))) OR ((Keywords:"Cross-Plataform") OR (Keywords:"application") OR (Keywords: "Web App") OR (Keywords:"Web Application")) OR ((Keywords:"Model-Driven Development") OR (Keywords:"Model-Driven") OR (Keywords:"code generation"))))

2. *String* usado na Scopus: TITLE-ABS-KEY (("cross-platform software"OR "software multiplatforma"OR "cross-platform development"OR "desenvolvimento multiplatforma"OR "cross-platform*"OR "cross platform*") AND ("tool"OR "ferramenta"OR "abordagem"OR "approach"OR “framework”) AND ("model driven"OR "model-driven"OR "desenvolvimento orientado por modelos"OR "model driven engineering"OR "desenvolvimento de software dirigido por modelos"OR "model driven development")) AND (LIMIT-TO (SUBJAREA , "COMP")) AND (LIMIT-TO (SUBJAREA , "COMP"))

Para cada resultado das buscas, foi aplicado o processo de filtragem, que consistiu da leitura do título e leitura do resumo, para verificar a sua relação com o tema desta tese.

Caso houvessem dúvidas, eram lidos mais trechos do artigo até comprovar a relação ou não com esta tese. Os artigos eram incluídos na revisão se correspondessem aos seguintes critérios (com obrigatoriedade para o item 1):

1. Consideravam soluções para o desenvolvimento de sistemas multiplataforma?
2. Aplicavam técnicas de DSDM?
3. Resolviam alguma fase do processo de desenvolvimento?
4. Apresentavam o processo de modelagem?
5. Realizavam a geração de código ou executável?
6. Faziam uma revisão bibliográfica sobre os itens 1 e 2?

Em caso de correspondência negativa, os artigos eram arquivados para o caso de encontrá-los novamente em outra ocasião. Os artigos que apresentassem correspondência positiva, eram incluídos na revisão para a leitura completa em fase posterior.

Além dos critérios de análise supracitados para cada trabalho, a seção de trabalhos correlatos e as referências bibliográficas foram cuidadosamente avaliadas, visando a obtenção de novos artigos. Essa técnica, conhecida como *Snowballing* (WOHLIN, 2014), consiste na incorporação de novos trabalhos ao estudo do estado da arte através do uso das referências bibliográficas. Dessa forma, a cada artigo é possível descobrir novos trabalhos relacionados e a repetição do processo justifica o nome "bola de neve". A seção de trabalhos correlatos permite também um estudo mais abrangente do estado da arte devido às discussões comparativas entre os trabalhos. Essas discussões facilitam o entendimento das abordagens ressaltando os pontos fortes e fracos de cada trabalho.

Além dos trabalhos acadêmicos, as abordagens comerciais que apresentam soluções para o desenvolvimento de sistemas multiplataforma e/ou utilizam DSDM foram consideradas no trabalho. Sua busca foi realizada através das citações feitas pelos trabalhos acadêmicos (*Snowballing*) e através de pesquisas nos principais mecanismos de busca. As abordagens comerciais utilizam técnicas similares às encontradas na academia, porém, a sua análise é fundamental para entender o comportamento do conceito no mercado e sua viabilidade. Uma discussão comparativa foi levantada entre as abordagens comerciais e acadêmicas ressaltando suas potencialidades e fragilidades.

Esta pesquisa selecionou 60 trabalhos relacionados de forma iterativa e incremental durante o desenvolvimento desta tese, entre trabalhos acadêmicos (46) e comerciais (14). Para obter dados analíticos dos trabalhos para estudos comparativos e estatísticos, foram recuperadas as seguintes informações de cada artigo:

- Benefícios do conceito DSDM explorados em cada artigo. O DSDM tem potencial para proporcionar diversos benefícios. Esta revisão buscou avaliar quais desses benefícios foram mais observados em cada artigo que aplicou o DSDM, no contexto do desenvolvimento multiplataforma. A identificação dos benefícios usados por cada trabalho é importante para localizar nichos de pesquisa pouco explorados. Os benefícios elencados estão pontuados na Figura 2, onde é realizado também um estudo comparativo entre as abordagens. Os benefícios DSDM usados para classificar os trabalhos são:

  - Verificação das plataformas atendidas por cada trabalho. O DSDM possui potencial para gerar código em diferentes linguagens, atendendo a várias plataformas, ideal para o cenário multiplataforma (UMUHOZA; BRAMBILLA, 2016). Dessa forma, esse trabalho considerou as seguintes plataformas: web, móvel, *desktop*, comunicação e gerenciamento de dados entre as plataformas (Interoperabilidade) e uso de API (*Application Programming Interface*). Outro aspecto interessante dessa métrica é a análise estatística das plataformas consideradas pelos trabalhos, podendo apresentar lacunas não exploradas na solução da questão multiplataforma;
  - Aumento no nível de abstração dos conceitos do domínio. Com o DSDM é possível melhorar a capacidade de gerenciamento da complexidade de sistemas através do aumento da abstração no desenvolvimento (STAHL; VOLTER, 2006);
  - Tratar todo desenvolvimento em um único ambiente. Um dos grandes desafios para o cenário multiplataforma é a necessidade de vários ambientes de desenvolvimento distintos, um para cada plataforma. Esse desafio impacta em vários aspectos do desenvolvimento: aumento no custo, equipe heterogênea, dificuldade na manutenção, dificuldade no gerenciamento da complexidade do software, dificuldade na participação de *stakeholders*, entre outros. O DSDM oferece condições para o tratamento de todo o software em um único ambiente (UMUHOZA et al., 2015). Portanto, o trabalho analisado recebeu avaliação positiva nesse critério somente se atendeu a todas as plataformas consideradas por esta tese;
  - Reúso e adaptação do código fonte gerado. Novas funcionalidades surgem a todo instante no cenário do desenvolvimento de software, e considerar todas as funcionalidades disponíveis é uma tarefa difícil. Por essa razão, é importante a geração de código legível, isto é, código que permita ao desenvolvedor a inclusão de novas funcionalidades ou detalhes de interface que não foi possível especificar na modelagem (JONES; JIA, 2014);

- Capacidade de Expansão. Esse critério avalia a flexibilidade de cada abordagem para o atendimento de novas tecnologias e/ou plataformas que ainda não existem. As abordagens multiplataforma que utilizam o DSDM podem manter o desenvolvedor limitado no que diz respeito às novas plataformas que venham a surgir. A adoção do conceito de caixa branca (KHAN; KHAN et al., 2012) pelas abordagens é uma medida interessante, uma vez que permite ao desenvolvedor a extensão da abordagem sempre que houver a necessidade;
- Independência da evolução tecnológica. Esse critério trata da flexibilidade no atendimento a novos conceitos e/ou funções disponíveis para o desenvolvimento de software. Como exemplo: interface de comunicação (REST e SOAP), novos recursos de interação humano-computador, uso de novos sensores dos dispositivos móveis, entre outros. Existe a opção do desenvolvedor estender o código gerado pela ferramenta, como descrita no item Reúso e adaptação do código fonte gerado, ou estender a abordagem para que ela gere códigos para novas funções de forma automatizada. Essa última opção se refere ao objetivo desse critério, onde o desenvolvedor pode estender a abordagem, contribuindo para torná-la mais genérica e customizada, atendendo necessidades específicas;
- Independência de plataforma de desenvolvimento. Ferramentas comerciais tendem a manter o usuário preso à ferramenta de desenvolvimento, como é o caso GENEXUS (MARINHO; RESENDE, 2015) e MENDIX (UMUHOZA; BRAMBILLA, 2016). Esse comportamento se deve ao fato que a abordagem segue uma linha caixa preta (KHAN; KHAN et al., 2012), permitindo ao desenvolvedor a utilização apenas das funcionalidades disponibilizadas pela ferramenta, além da impossibilidade da manutenção ser feita em outros ambientes. Algumas ferramentas acadêmicas também possuem esse comportamento, assim como APPIAN (UMUHOZA; BRAMBILLA, 2016), Perchat, Desertot e Lecomte (2013) entre outros;
- Aumento na produtividade. Uma das principais características do conceito DSDM é o aumento de produtividade, isto porque os códigos executáveis podem ser gerados através dos modelos usando uma ou mais etapas de transformação (STAHL; VOLTER, 2006);
- Reengenharia de software. O conceito DSDM inspirou o *Object Management Group* (OMG)[1] a organizar uma área de pesquisa para estabelecer metamodelos padronizados para auxiliar o processo de reengenharia de software. Essa área, chamada de *Architecture-Driven Modernization* (ADM) (SNEED, 2005), visa utilizar esses metamodelos na representação de software e assim realizar sua modernização ou refatoração através da geração de código, utilizando os con-

[1] www.omg.org

ceitos do DSDM. Esse critério foi incluído para avaliar os trabalhos que fazem reengenharia através do DSDM e assim mostrar a flexibilidade do conceito;

  - Interoperabilidade com ferramentas de modernização de código. A *Architecture-Driven Modernization Task Force* (ADMTF)[2] (FORCE, 2012) criou um conjunto de metamodelos para representação de sistemas focado na reengenharia de software. Esses metamodelos têm a capacidade de troca de metadados entre diversas ferramentas de modernização criadas por engenheiros de software para auxiliar a interoperabilidade e poder gerar a modernização em diferentes ambientes. Portanto, este tópico visa avaliar as abordagens que conseguem manter tal interoperabilidade.

- Tipo de solução considerada por cada trabalho. A geração de código, para dispositivos móveis, pode ser realizada de cinco formas: nativa, híbrida, interpretada, generativa ou baseada em web. Cada uma das formas possui suas vantagens e desvantagens. A importância desse item se deve à identificação e classificação estratégica usada por cada trabalho e suas justificativas.

- Detalhes de implementação. Existe uma série de *frameworks* e ferramentas que auxiliam no uso do conceito DSDM. Além de técnicas de modelagem usadas para garantir melhor expressividade ou maior usabilidade para a geração de aplicativos para as diferentes plataformas existentes. Entre as técnicas, podemos citar (UMUHOZA et al., 2015): PIM (*Platform Independent Model*) para NC (*Native Code*), PIM para PSM (*Platform Specific Model*) e para NC, PIM para CPC (*Cross Platform Code*) usando o Cordova[3], PIM para FSM (*Cross Framework Specific Model*) e para CPC. Esses detalhes foram observados para uma análise estatística das técnicas mais usados pelos trabalhos.

- Aspectos considerados para a geração do aplicativo. Existem basicamente quatro aspectos que envolvem a geração de aplicativos: gerenciamento de dados, lógica de negócio, interação com o usuário e a interface do usuário (UMUHOZA; BRAMBILLA, 2016). Alguns trabalhos abordam todos os aspectos, enquanto outros apenas uma parte. Esse item é importante para analisar quais os aspectos mais abordados pelas soluções aqui pesquisadas que tratam dos software multiplataforma.

Para a obtenção de informações mais consistentes e completas dos artigos, em muitos casos, foram realizadas pesquisas em sites oficiais, repositório de códigos, fóruns de discussões técnicas, entre outros.

A busca de novos trabalhos ocorreu durante todo o desenvolvimento da tese, onde novos trabalhos encontrados eram adicionados aos estudos comparativos deste capítulo.

[2] www.omg.org/adm/
[3] cordova.apache.org/

Os trabalhos incluídos na revisão foram categorizados para a geração de estatísticas e informações que visam avaliar e descobrir as técnicas mais usadas pelos artigos e quais os benefícios menos explorados do conceito DSDM.

A categorização baseou-se na verificação das plataformas consideradas por cada abordagem da seguinte forma:

- Plataformas móveis. Existe um grande esforço atualmente na tentativa de solucionar a questão do desenvolvimento de aplicativos para dispositivos móveis (MARINHO; RESENDE, 2015), justamente devido ao número de SO existentes para esse segmento e a exigência de mercado. Esse fato justifica uma categoria destinada exclusivamente para as plataformas móveis, fato que não se repete para outras plataformas. Embora existam vários SO disponíveis para outras plataformas, os usuários não são tão diversificados como no caso da plataforma móvel e o software é normalmente desenvolvido para uma plataforma alvo sob demanda. A mudança de plataforma nesses casos, normalmente, é realizada devido à questão de software legado, não viabilizando grandes estudos nessa área;

- Plataformas Móvel, Web e Interoperabilidade. Abordagens que consideram plataformas móveis e web são menos numerosas devido à complexidade da questão, porém são importantes trabalhos, uma vez que a necessidade do gerenciamento dos dados dos aplicativos móveis em nuvem é muito comum. Portanto, esse tópico inclui desde trabalhos que tratam apenas um aspecto considerado para a geração do aplicativo, até a combinação total entre eles (gerenciamento de dados, lógica de negócio, interação com o usuário e a interface do usuário);

- Plataformas Móvel, Web, Interoperabilidade e *Desktop*. O maior desafio para sistemas multiplataforma é a concentração do seu desenvolvimento em um único ambiente englobando o maior número de plataformas possível. Existem poucas soluções que atendem a essa questão, portanto este tópico apresenta as abordagens que exploram o DSDM no nível mais complexo.

- Outras soluções. Este tópico visa apresentar trabalhos que exploram as vantagens do conceito DSDM no desenvolvimento de software para áreas de intersecção com a multiplataforma. Isto é, áreas que não estão diretamente ligadas à multiplataforma, mas são importantes para apresentar o potencial do DSDM na geração de código em áreas diversas. A apresentação destes trabalhos ajudou a viabilização desta tese, uma vez que o objetivo principal foi propor um ambiente holístico para o desenvolvimento de software.

3.1 Resultados

Devido ao potencial do DSDM para solucionar parte do problema multiplataforma, muitas pesquisas foram conduzidas para avaliar sua viabilidade e explorar seus benefícios. Tais pesquisas é apresentada nesta seção de acordo com a organização apresentada na seção anterior.

3.1.1 Plataformas móveis

A maior parte dos trabalhos da literatura que utilizam o DSDM para tratar o desenvolvimento multiplataforma está voltado para as plataformas móveis. Isto porque a plataforma móvel surgiu com uma grande diversidade de marcas e SO, obrigando os desenvolvedores a lidar com várias versões do mesmo aplicativo para atingir a maior quantidade de usuários possível.

Alguns trabalhos utilizam o DSDM para tratar o desenvolvimento de uma única versão do aplicativo. Para estes casos o apelo é no sentido de elevar o nível de abstração do desenvolvimento e permitir a participação dos *stakeholders* ao projeto. Trabalhos como JUSE4ANDROID (SILVA; ABREU, 2014), MIMIC (ELOUALI et al., 2014) e Cimino e Marcelloni (2012) focam exclusivamente no aspecto da criação de interface (*Grafical User Interface* - GUI) para a plataforma Android. JUSE4ANDROID e Cimino e Marcelloni (2012) utilizam UML para projetar interfaces para aplicativos Android, onde JUSE4ANDROID utiliza uma PSM e Cimino e Marcelloni (2012) utiliza PIM para geração de código nativo para interfaces. MIMIC utiliza o *Mobile MultiModality Modeling Language* (M4L). O M4L é uma linguagem baseado em máquina de estado usada para mapear interfaces de aplicativos Android.

Ainda se tratando no sistema operacional Android, alguns trabalhos consideram outros aspectos relacionados ao desenvolvimento além do GUI, assim como: Lógica do negócio, Interação do usuário e gerenciamento dos dados. Trabalhos como Parada e Brisolara (2012), Ko et al. (2012), Benouda et al. (2016) e ARCTIS (KRAEMER, 2011) propõem uma extensão da linguagem UML para gerar metamodelos mais expressivos e gerar código nativo para a plataforma Android. A diferença entre eles está no método usado para geração de código. No caso do Parada e Brisolara (2012) é usado a ferramenta GenCode desenvolvida como parte do trabalho. Ko et al. (2012) geram códigos nativos seguindo o estilo arquitetural MVC (*Model View Controller*) e utilizando o *framework Acceleo* (ACCELEO, 2010), assim como no trabalho do Benouda et al. (2016). No trabalho de Son, Kim, Kim (2013) o *Meta Object Facility* (MOF) (OMG, 2005) é usado para apresentar uma sequência de diagramas que modela os conceitos de um domínio específico. A partir dessa modelagem, o código é gerado de forma nativa através do *framework Acceleo* para a plataforma Android, onde as regras de transformação são definidas para a

geração de código. O *framework* ARCTIS tem como diferencial a utilização de máquina de estado para gerar códigos para a plataforma Android, além de utilizar APIs oficiais do Android para ficar compatível com futuras atualizações.

Assim como as abordagens que tratam o desenvolvimento exclusivamente para o SO Android, também existem os que tratam para iOS e *windows phone*, porém muito menos numerosos. Behrens (2010) apresenta um ambiente para modelagem textual através de uma DSL específica para o domínio de IPhone onde procura elevar o nível de abstração, porém não trata a questão da geração de códigos. Min et al. (2011) e Benouda et al. (2017) utilizam UML para modelar aplicativos para *windows phone* e gerar códigos nativos. Em Min et al. o código é gerado em C# seguindo o estilo arquitetural MVC através do ambiente *Visual Studio*. Já em Benouda et al. é usado *Acceleo* junto com *Translationist* para a geração da estrutura da aplicação e seu código fonte em C#. Nessa abordagem é gerado desde a camada *Model* (classes e DAO[4]) até a *View* do estilo arquitetural MVC para o usuário. Porém, não é possível a modelagem de regras de negócio muito específicas.

Referente à questão multiplataforma, trabalhos que atendem mais do que um SO para dispositivos móveis representam um impacto mais significativo. A modelagem e geração de código para várias plataformas distintas exploram melhor as vantagens do conceito DSDM, mesmo que direcione o tratamento apenas para o aspecto da GUI, como o caso dos trabalhos RUMO (SCHULER; FRANZ, 2013), Diep, Tran e Tran (2013), Sabraoui, koutbi e Khriss (2012), Chen et al. (2019) e Sabraoui et al. (2019). Nesses trabalhos é realizada a geração de código nativo para a interface do usuário direcionado às duas principais plataformas do mercado: Android, iOS e os três primeiros trabalhos também consideram a plataforma *windows phone*. Os transformadores são providos pelas próprias abordagens. As abordagens se diferenciam pela estratégia usada para prover a modelagem. Em RUMO (*Rule-Based Generation of Mobile User Interfaces*) é proposto um *framework* para o desenvolvimento multiplataforma de interfaces, onde a modelagem visual é realizada através do padrão *Meta-Object Facility* (MOF) e é baseado no modelo independente de plataforma (PIM - *Platform Independent Model*), sendo possível também criar várias versões para a mesma plataforma. No trabalho de Diep, Tran e Tran (2013) é proposta uma abordagem PIM escrito em XML, onde todos os parâmetros da interface são descritos em linguagem com alto nível de detalhamento, mostrando ganhos de produtividade no desenvolvimento com o *framework* quando comparado com a forma tradicional. Sabraoui, koutbi e Khriss (2012) apresentam uma abordagem para geração de interfaces usando UML. Uma vez concebida a modelagem através da UML, a transformação em XMI é realizada através de uma API chamada JDOM e, em seguida, é gerado o *Graphical User Interface* (GUI) para as plataformas móveis.

[4] Considerado um padrão para aplicações que utilizam persistência de dados, possuem separação entre as regras de negócio e a manipulação dos dados persistentes (*Data Access Object*).

Chen et al. (2019) propõem um *framework* que recebe como entrada a codificação da interface (UI), feita nos ambientes de desenvolvimento nativo para Android ou iOS. Em seguida, um algoritmo de classificação usando redes neurais, identifica os componentes e gera os códigos da interface para a plataforma diferente da recebida como entrada. Sabraoui et al. (2019) utiliza DSDM para desenvolver a GUI de forma independente de plataforma. É utilizado uma DSL para modelar a interface, programado através de uma gramática escrita em XText. Essa DSL pode ser visualizado de forma gráfica, como modelos da UML e através de geradores da abordagem, é criado de forma automática os códigos de interface para as plataformas.

A exploração das vantagens do DSDM aumentam quando além do aspecto GUI, as abordagens consideram também os demais aspectos de desenvolvimento e direcionam a geração de código nativo para mais do que uma plataforma móvel, como é o caso dos trabalhos: MD2 (MAJCHRZAK; ERNSTING; KUCHEN, 2015), AXIOM (JONES; JIA, 2014), XIS-MOBILE (RIBEIRO; SILVA, 2014), Perchat, Desertot e Lecomte (2013), MOPPET (USMAN; IQBAL; KHAN, 2017), Taentzer e Vauper (2016), Dageförde et al. (2016), XMOB (GOAER; WALTHAM, 2013), REACT NATIVE[5], NATIVESCRIPT[6], MAG (USMAN; IQBAL; KHAN, 2014), Botturi et al. (2013), Lachgar e Abdali (2017), XAMARIN[7], Vaupel et al. (2018) e Rieger et al. (2020).

Exceto XIS-MOBILE, MAG, Botturi et al. (2013), Lachgar e Abdali (2017) e XAMARIN, todo o restante focam nas plataformas mais populares do mercado: Android e iOS, se diferindo em alguns pontos. O MD2 utiliza os conceitos de DSDM para prover uma modelagem declarativa no intuito de atender aos requisitos típicos dos aplicativos empresariais. A geração de código se dá em duas etapas: após a modelagem é gerado um arquivo XML com a descrição do domínio e o relacionamento entre os modelos. A partir do arquivo em XML os códigos nativos são gerados através de transformadores da própria abordagem no padrão MVC (*Model-View-Controller*) (MAJCHRZAK; ERNSTING; KUCHEN, 2015).

AXIOM utiliza modelos visuais da UML combinado com modelagem abstrata em árvore (*Abstract Model Tree*) para representar as interações e os conceitos genéricos do domínio através de um *Platform Indepent Model* (PIM). Transformações M2M (*Model to Model*) são aplicadas visando gerar um PSM (*Platform Specific Model*), onde especificidades de cada plataforma são incluídas, além de outros detalhes da aplicação. A abordagem usa o conceito de caixa branca e os códigos nativos são gerados para as plataformas através de transformadores da própria abordagem (JONES; JIA, 2014).

Perchat, Desertot e Lecomte (2013) propõem uma linguagem universal em que,

[5] facebook.github.io/react-native
[6] www.nativescript.org
[7] www.xamarin.com

através de um compilador desenvolvido como parte do trabalho, é possível gerar códigos nativos para solucionar parte do desenvolvimento de software para dispositivos móveis. Embora a abordagem mostre ganho na produtividade, a linguagem proposta possui baixo nível de abstração e não se trata de uma linguagem de modelagem, além de não utilizar nenhum *framework* de apoio, fato que pode manter a linguagem com baixo poder de expressividade.

MOPPET é uma ferramenta que utiliza uma abordagem mista, envolvendo os conceitos de Linha de Produto de Software (SPL) e DSDM, definida pelos autores como *product-line model-driven engineering approach*. Com a junção dos dois conceitos, o projeto pretendeu abordar as seguintes variações: na aplicação devido aos SO, de capacidade de hardware entre diferentes dispositivos e de funcionalidades de acordo com cada dispositivo. A ferramenta MOPPET foi desenvolvida para automatizar todo o processo e gerar código nativo. Através dessa ferramenta foi realizado dois casos de uso, onde foi observado redução do esforço e aumento na produtividade com o uso da abordagem (USMAN; IQBAL; KHAN, 2017).

Taentzer e Vauper (2016) propõem a modelagem textual de aplicativos nativos separada em três sub-modelos: modelagem gráfica (GUI), modelo de dados e comportamento do aplicativo (regras de negócio) através do EMF (*Eclipse Model Framework*), usando Xtext e GMF. O diferencial do trabalho foi prover códigos nativos levando em conta na modelagem informações de contexto e funcionamento *off-line*.

Dageförde et al. (2016) provêm uma abordagem que discute formas de agregar o conceito de Linha de Produto de Software ao *framework* MD2. A maior contribuição do trabalho foi a modularização do *framework* MD2, de acordo com os conceitos da SPL, garantindo maior versatilidade e reúso dos artefatos da modelagem.

XMOB é uma linguagem de modelagem utilizada para descrever os conceitos de aplicativos móveis nativos. A abordagem utiliza o conceito de PIM-to-PSM-to-NC e uma DSL declarativa usada para especificar os conceitos do domínio. Porém, não foi apresentado um estudo de caso e nem detalhes da geração de código, de forma que não é possível discutir a viabilidade da abordagem (GOAER; WALTHAM, 2013).

O NATIVESCRIPT foi lançado em 2014 pela Telerik[8] e permite o desenvolvimento de aplicativos nativos utilizando uma única linguagem. No caso, o desenvolvedor poderá optar por *JavaScript*, Angular[9] ou *TypeScript*[10] e customizações de interfaces através do CSS. Essa ferramenta faz acesso às APIs nativas das plataformas que ela oferece suporte: iOS e Android. Dessa forma, o desenvolvedor não fica preso às atualizações das ferramentas, se tornando independente de evoluções tecnológicas das plataformas. O

[8] www.telerik.com
[9] angular.io
[10] www.typescriptlang.org

NATIVESCRIPT utiliza a arquitetura PIM-PSM-NC, onde a maior parte da codificação pode ser reutilizada em ambas as plataformas e algumas especificidades podem ser customizadas para as plataformas de forma individual. A diferença entre o REACT NATIVE e o NATIVESCRIPT é que no primeiro o código é transformado em linguagem nativa, com possibilidade de extensão do fonte pelo desenvolvedor. E, no segundo, o código é entregue aos dispositivos em *JavaScript* e interpretado nativamente através do *Virtual Machine* (VM) de cada plataforma, sendo V8[11] para Android e *JavaScriptCore*[12] para iOS. Assim, como o REACT NATIVE, este *framework* não eleva o nível de abstração do desenvolvimento, forçando o desenvolvedor a lidar com os detalhes técnicos do desenvolvimento.

As abordagens XIS-MOBILE, MAG e Botturi et al. (2013) focam na geração de código para as plataformas Android e *windows phone*. O XIS-MOBILE reusa conceitos da linguagem XIS, que é uma extensão da UML, foca no design interativo dos software seguindo o conceito DSDM. XIS considera três pontos na modelagem: Entidades, Caso de uso e UI (*User Interface*). O XIS-MOBILE trabalha com o conceito de transformação M2M, que compreende PIM para PSM, e M2T, que compreende PSM para NC (*Native Code*). O Gerador de código do XIS-MOBILE é baseado no *Acceleo*, plug-in do Eclipse, o qual pode ser expansível para outras plataformas, bastando para isso o usuário adicionar os *templates*.

MAG (*Mobile Apps Generator*) é uma ferramenta baseada na abordagem DSDM que gera código nativo automático através da utilização de diagramas de caso de uso para modelar as especificações do domínio, diagramas de classe para modelar a estrutura do aplicativo e diagrama baseado em máquina de estados para modelar o comportamento e as regras de negócio do aplicativo. Para avaliar a viabilidade da abordagem, um estudo de caso foi conduzido gerando código multiplataforma para um aplicativo comercial chamado *Scramble*, onde apresenta resultados satisfatórios em relação à produtividade e abstração dos detalhes técnicos de implementação.

Botturi et al. (2013) apresentam uma abordagem para geração de interfaces e regras de negócio independentes para dispositivos móveis, onde o comportamento do aplicativo é definido através de máquina de estado finito. Este trabalho utiliza UML2 para modelar a interface a partir de um modelo genérico PIM que é transformado através de regras M2M para modelagem FSM (*Framework Specific Model*). Nessa etapa, é possível definir especificidades para as plataformas. Um mecanismo de transformação M2T é provido pela abordagem para a geração código legível, onde uma avaliação é conduzida para comparar o código gerado automaticamente com a forma manual. A avaliação tem resultados positivos em relação à elegibilidade do código e ao aumento de performance no desenvolvimento.

[11] developers.google.com/v8
[12] developer.apple.com/documentation/javascriptcore

No trabalho de Vaupel et al. (2018) é apresentada uma abordagem que permite a modelagem de sistemas multiplataforma de forma abstrata e concreta quando isso é necessário. A modelagem utiliza quase que exclusivamente modelos visuais e regras específicas podem ser definidas através de uma linguagem disponibilizada pela abordagem para a geração de códigos nativos (Android e iOS). Porém, a abordagem não considera a sua extensão e a modelagem é gerada de maneira igualitária para ambas as plataformas. Em Rieger et al. (2020), os elementos para a acessibilidade em dispositivos móveis são incluídos no *framework* MD2, visando a redução de custos em sua implementação. O *framework* permite a geração de códigos nativos com elementos visuais que facilitam a usabilidade por pessoas com necessidade especiais. A avaliação conduzida apresenta uma redução considerável no LOC para acessibilidade (3200 para 445) com a utilização do MD2.

Entre as abordagens que geram código nativo e consideram mais de uma plataforma, o trabalho acadêmico de Lachgar e Abdali (2017) e a ferramenta comercial XAMARIN são os mais completos. Isto porque a geração de código é feita para as três plataformas mais populares: Android, iOS e *windows phone*. A abordagem Lachgar e Abdali (2017) usa o conceito de PIM para modelar os conceitos genéricos do aplicativo e, através de regras de transformação M2M escrito em XTend, os conceitos específicos de cada plataforma podem ser expressos na PSM. As regras de transformação M2T, escritas em XTend, realizam a geração de código para as três plataformas. O XAMARIN é uma solução para o desenvolvimento multiplataforma disponibilizada em 2012 e integrada como ferramenta oficial do *Visual Studio* em 2016. Através de uma linguagem única, C#, é possível obter entre 75% e 100% de reutilização de código entre plataformas, dependendo da arquitetura que será utilizada: interface nativa ou *Xamarin.Form*. O *Xamarin.Form* é uma biblioteca que auxilia o desenvolvedor a ter um maior reaproveitamento de código entre as plataformas, apesar da limitação a certos recursos de interface. Os aplicativos gerados pela ferramenta são nativos, graças ao acesso às APIs de cada plataforma, não limitando o desenvolvedor de evoluções tecnológicas de cada dispositivo. Porém, apesar do aumento na produtividade, a ferramenta não aumenta o nível de abstração, mantendo o desenvolvedor envolvido aos detalhes técnicos do projeto.

Além das abordagens multiplataforma para geração de código nativo, existem também as abordagens que direcionam a geração de código utilizando tecnologias web, que é o caso dos aplicativos híbridos. Neste caso, a geração de código web é compatível com alguns *frameworks* híbridos já consolidados no mercado. E, por sua vez, gera códigos para as principais plataformas do mercado. Os trabalhos MOBL (HEMEL; VISSER, 2011), Höpfner et al. (2011), Francese et al. (2015) e WL++ (STROULIA et al., 2015) utilizam o PhoneGape para geração dos códigos executáveis para multiplataforma, neste caso o número de plataformas atendidas é de acordo com a especificação do *framework* utilizado. MOBL usa uma DSL declarativa que gera código em HTML5, CSS e *JavaScript* (JS).

Estes códigos podem ser transferidos para o PhoneGap visando a geração de aplicativos híbridos, executados em diferentes plataformas. Höpfner et al. (2011) propõem uma abordagem baseada em PIS (*Platform Independent Specification*) para a geração de aplicativos híbridos através da geração de linguagem web baseada em uma DSL declarativa. Francese et al. (2015) apresentam uma abordagem que permite a modelagem das questões referentes a GUI e as interações dos usuários, para a geração de aplicativos híbridos. A interação do usuário é modelada através de uma máquina de estados e o código gerado é em JS. O desenvolvedor pode estender os códigos gerados de forma legível no ambiente PhoneGap, visando acrescentar funções não previstas pela abordagem ou recursos nativos do dispositivo. O WL++ é um *framework* que usa os conceitos do DSDM em uma linha diferente dos demais trabalhos. Através de técnicas de reengenharia, o WL++ acessa o sistema web que gerencia os aplicativos (*back-end*) a partir de instâncias do JSON (*RESTful*) e os transforma em modelos que especificam sua estrutura. A partir destes modelos, o *framework* permite que o desenvolvedor estenda a modelagem com detalhes específicos dos conceitos do domínio. Na sequência, é gerado o código fonte na linguagem web, de acordo com as especificações do PhoneGap, onde são gerados os aplicativos híbridos para as principais plataformas.

Outro *framework* híbrido que serve de base pelas abordagens para a geração de códigos executáveis é o Cordova, usado nos trabalhos de Brambilla, Mauri e Umuhoza (2014) e Franzago, Muccini e Malavolta (2014). Assim como o PhoneGap, o Cordova é compatível com as principais plataformas do mercado. No trabalho de Brambilla, Mauri e Umuhoza (2014) é apresentado uma extensão do padrão de modelagem IFML[13] (*Interaction Flow Modeling Language*) adaptado para confecção de aplicativos para dispositivos móveis. Esse padrão é baseado em *Eclipse Modeling Tools* usado para modelar visualmente os elementos da GUI de aplicativos, envolvendo: conteúdo, interação do usuário e controle do comportamento. IFML segue a estratégia PIM e tem o potencial de gerar códigos web através dos conceitos do DSDM. Um estudo de caso foi conduzido visando o desenvolvimento de um aplicativo de comércio eletrônico chamado *RedLaser*, onde foi possível apresentar resultados satisfatórios na comparação com abordagens tradicionais. No trabalho de Franzago, Muccini e Malavolta (2014) é proposto um *framework* colaborativo que utiliza DSDM para o desenvolvimento de aplicativos móveis híbridos baseado em modelos independente de plataforma (PIM) para geração de código web. O mecanismo colaborativo da abordagem permite que a equipe de desenvolvimento possa modelar o sistema em alto nível de abstração e com divisão de tarefas. Modelos de navegação, dados, interface e lógica de negócio pode ser modeladas com a ajuda dos *stakeholders* em todas as fases do desenvolvimento.

As abordagens que geram aplicativos híbridos são normalmente apoiadas por *fra-*

[13] www.ifml.org/

meworks para a geração do código fonte. Isso, faz com que as abordagens não consigam explorar alguns benefícios do DSDM, assim como: independência de plataforma de desenvolvimento, independência da evolução tecnológica e capacidade de expansão. Isso ocorre pelo fato dessas abordagens dependerem de uma ferramenta comercial para a geração de códigos, onde a extensão das mesmas fica impossibilitada.

É comum utilizar o conceito DSDM focando exclusivamente no apoio ao desenvolvimento de aplicativos móveis. De uma forma geral, os trabalhos mostram a viabilidade em se usar o conceito DSDM para solucionar parte dos desafios atribuídos à multiplataforma com inúmeros resultados positivos, comprovando os benefícios que o conceito DSDM pode proporcionar. Os benefícios mais explorados são o aumento da produtividade e o aumento na abstração dos detalhes técnicos, facilitando o envolvimento dos *stakeholders* no projeto. Há grande diversidade de estratégias utilizadas, auxiliando o rumo de novas pesquisas após análise dos resultados. Estratégias como: uso de DSL declarativas com XML, DSL textuais, DSL com apoio visual, DSL baseado nos diagramas da UML, estratégias de modelagem (PIM, PSM, NC, FSM, CPC) e técnicas de transformação M2M e M2T. Essas estratégias foram apresentadas pelos artigos permitindo a análise das vantagens e desvantagens de cada uma e obtendo conhecimentos que ajudam a decidir qual estratégia é mais adequada para cada tipo de projeto. Podemos por exemplo citar o trabalho de Lachgar e Abdali (2017) que direciona o desenvolvimento para as três principais plataformas móveis disponíveis do mercado. Para este caso, utiliza-se PIM-PSM-NC, por se tratar de um número maior de plataformas as especificidades de cada plataforma exigem uma configuração mais completa. Já trabalhos que direcionam o desenvolvimento para menos plataformas como o caso de Ko et al. (2012), utilizam em geral PIM com transformações M2T para NC, onde a DSL utilizada na abordagem já é expressiva o suficiente para a plataforma alvo.

Apesar do fato de os trabalhos mostrarem resultados positivos fazendo uso do conceito DSDM, de modo geral as abordagens não consideraram questões relacionadas à heterogeneidade dos ambientes de execução web e a comunicação de dados entre as plataformas. Essas questões são muito usadas entre os aplicativos para o gerenciamento dos dados e eventuais atualizações. Outras plataformas consideradas em um sistema multiplataforma não foram previstas, assim como sistemas *Desktop* e novas tecnologias que podem surgir a qualquer momento, não provendo uma visão holística do desenvolvimento de software. Mesmo que muitas abordagens apontem flexibilidade para a extensão dos metamodelos, elas se limitam à adição de novas funcionalidades nas plataformas consideradas, não prevendo a inclusão de novas plataformas.

3.1.2 Plataformas móvel, web e interoperabilidade

O gerenciamento dos dados dos aplicativos móveis é uma necessidade muito comum. Para isso, são necessárias abordagens que ofereçam suporte para sistemas web e a comunicação entre dispositivos móveis e servidores web. Sistemas web tem a função de gerenciar e unificar os dados dos aplicativos móveis para deixá-los mais dinâmicos e colaborativos, funções como autenticação e envio de mensagens entre usuários, controle de estoque, gerenciamento de ranking pelos aplicativos e qualquer outra função que exija a troca de dados entre os aplicativos é normalmente gerenciada por sistemas web.

A troca de dados entre as plataformas ou a interoperabilidade garante a comunicação entre os aplicativos, visando manter a integridade e a sincronização dos dados independente do dispositivo utilizado. Para isso é necessário uma estrutura de comunicação que pode ser automatizada através do DSDM. Nessa linha, algumas abordagens tratam exclusivamente a comunicação e geram códigos a partir da análise da arquitetura do sistema. Grace, Pickering, Surridge (2016) apresentam um método, baseado nos conceitos de engenharia dirigida por modelos, que reduz o esforço na geração de serviços que garantem a interoperabilidade de sistemas complexos. Este método utiliza a combinação entre os modelos que especifica a arquitetura e o comportamento do software para realizar a geração correta de interoperabilidade entre aplicações através da composição de *Web Service*. Os autores utilizam uma máquina de estado finito (*Finite State Machines*) e a técnica de modelagem FSM para gerenciar a comunicação entre os aplicativos em cada plataforma, os quais são representados pelos estados e as transições são alterações nos mesmos, requisitando comunicação entre os dispositivos. A abordagem disponibiliza uma ferramenta onde é possível realizar a modelagem da máquina de estado de forma visual. A modelagem por sua vez é convertida em XML e traduzida em serviços de comunicação REST/Json ou REST/XML.

CANAPPI utiliza uma DSL declarativa para geração de interfaces de comunicação para *Web Service* (JONES; JIA, 2014). Essas interfaces auxiliam a comunicação entre aplicativos nas nuvens e dispositivos móveis, porém o CANAPPI foi descontinuado. WS SOFTWARE FACTORY[14] (SILVA, 2015) é uma abordagem comercial baseada em DSDM que provê recursos para auxiliar no desenvolvimento de serviços para interoperabilidade usando padrões de *design* conhecidos da engenharia de software. Esses recursos são disponibilizados através de modelos visuais ou orientações textuais que auxiliam na geração de códigos integrados ao ambiente de desenvolvimento *Visual Studio*. Embora utilize uma linguagem que aumente o nível de abstração, essa DSL é uma ferramenta acoplada a um ambiente de desenvolvimento específico, não permitindo a independência de plataforma. Essas abordagens são importantes pois apresentam aumento na produtividade e manutenção dos serviços de comunicação e tratam a questão da heterogeneidade de

[14] servicefactory.codeplex.com/

ambientes de execução de software. Porém as abordagens não tratam as outras questões multiplataforma, servindo como uma ferramenta de gerenciamento da comunicação.

Abordagens que consideram plataformas móveis e web possuem um nível de complexidade superior aos apresentados na seção anterior, como é o caso do *Model-Driven APPLAUSE*[15] (DAGEFÖRDE et al., 2016). *APPLAUSE* é um *framework* que provê uma DSL declarativa escrita em XText com *templates* para geração de código em XTend, usadas para especificar aplicativos para várias plataformas nativas, entre eles: iOS, Android, *windows phone* 7 e *Google App Engine*. Porém o foco da modelagem envolve detalhes técnicos do domínio ao invés de elementos conceituais, mantendo um baixo nível de abstração e comprometendo a produtividade, além da abordagem não considerar a comunicação de dados entre o servidor web e os dispositivos móveis. O *Model-Driven APPLAUSE* não apresenta continuidade na pesquisa.

O APPIAN[16] é um *framework* comercial que apoia o desenvolvimento de aplicativos móveis e a interoperabilidade de dados entre os aplicativos. Este *framework* permite ao desenvolvedor modelar a interface da aplicação arrastando e soltando elementos visuais, mantendo um desenho único para todas as plataformas. As regras de negócio e comportamento do aplicativo são definidas em alto nível de abstração através de editores customizados pela ferramenta e os códigos nativos são gerados para as seguintes plataformas: Android, iOS e *BlackBerry*. A comunicação de dados entre as plataformas móveis e os serviços web também são consideradas através da geração de código REST. A ferramenta foca na geração de executáveis da aplicação para cada plataforma, de forma que não é possível estender o código fonte gerado, limitando o desenvolvedor com as funções previstas pela ferramenta, além de não considerar a geração dos serviços web, somente a estrutura de comunicação.

Outra ferramenta comercial que considera plataformas móveis e interoperabilidade é o MENDIX[17], se diferenciando do APPIAN por gerar código também para sistemas web. MENDIX é uma plataforma de desenvolvimento comercial baseado em DSDM que usa o PhoneGap como base para geração de aplicativos híbridos para as principais plataformas do mercado. Esse *framework* permite a modelagem visual dos conceitos do domínio em alto nível de abstração e gera código legível baseado em tecnologias web compatível com o PhoneGap, onde é possível adicionar novas funcionalidades. O MENDIX trata não só a comunicação de dados entre as plataformas através de *Web Services*, mas também o sistema web responsável em gerenciar os dados dos aplicativos dos usuários, fornecendo um ambiente mais completo de desenvolvimento do que os apresentados. Embora o MENDIX resolva bem a parte de unificação do desenvolvimento móvel em um único ambiente, ele não considera a geração de aplicativos nativos nem o desenvolvimento de sistemas

[15] github.com/applause/applause
[16] www.appian.com/
[17] www.mendix.com

para *desktop*, além de manter o desenvolvedor preso às funcionalidades previstas pela plataforma, por se tratar de um padrão caixa preta.

No trabalho de Inayatullah et al. (2019), é proposto um *framework* baseado em MDA (*Model-Driven Architecture*), que disponibiliza uma modelagem em UML para expressar os conceitos de um domínio. O *framework* utiliza *Acceleo* para gerar códigos para dispositivos móveis e aplicações web com operações CRUD (*Create, Read, Update and Delete*). A ferramenta gera Angular e ReactJS[18] para web, Ionic[19] e ReactNative[20] para dispositivos móveis e *REStful Service*[21] em Asp.Net para a comunicação entre eles. Dois estudos de caso foram conduzidos para validar o *framework*, gerando o esqueleto de um CRUD para multiplataforma, a partir de modelos em alto nível. Mostrando que o método simplifica a criação de operações CRUD, para as plataformas móveis e web. Porém, além de gerar aplicativos híbridos, questões como extensão e gerenciamento de funções entre as plataformas não são abordadas no trabalho.

WEBRATIO[22] (ACERBIS et al., 2015) é uma plataforma comercial de desenvolvimento orientada a modelo baseadas no IFML e com um alto nível de abstração, usado para o desenvolvimento de aplicativos híbridos para várias plataformas conforme especificação do Cordova. Utilizando técnicas DSDM, o WEBRATIO permite a modelagem da aparência e comportamento dos aplicativos móveis e a criação de *Web Services* e serviços web em um único ambiente. O código fonte gerado pela ferramenta pode ser estendido pelo desenvolvedor através da linguagem *JavaScript* (JS), onde é possível gerar códigos para as principais plataformas através da API Cordova. Originalmente, o WEBRATIO tinha foco apenas no desenvolvimento de aplicações web, passando a atender também dispositivos móveis. O WEBRATIO possui três ambientes integrados que são usados para o desenvolvimento completo do software. O primeiro é o ambiente de modelagem, que fornece suporte às especificações de diagramas IFML para descrição visual e diagramas de classe UML para o projeto da lógica do sistema. Em seguida há o ambiente de desenvolvimento, que fornece apoio a implementação de componentes personalizados, a fim de realizar extensões personalizadas da modelagem baseada em necessidades específicas de plataformas ou funções nativas. Por fim, o ambiente para projetar a interface, onde é possível projetar os desenhos das telas em alto nível, suportados pelo HTML5, JS e estilos baseado em CSS. Com base nas informações fornecidas através dos ambientes, a ferramenta WEBRATIO realiza a validação do modelo, geração do código completo, gerenciamento do ciclo de vida e permite o trabalho em grupo. O WEBRATIO possui o diferencial de suportar funções *offline*, e permite a geração de código para a sincronização de dados entre os dispositivos móveis e o servidor web. Essa ferramenta apresenta grande produtividade no

[18] https://pt-br.reactjs.org/docs/getting-started.html
[19] https://ionicframework.com/
[20] https://reactnative.dev/docs/getting-started
[21] https://restfulapi.net/
[22] www.webratio.com

desenvolvimento, porém tem os ônus relativos ao desenvolvimento híbrido. O ambiente visual de modelagem é uma caixa preta, de modo que torna não aplicável alguns benefícios do DSDM, assim como: capacidade de expansão, independência da evolução tecnológica e independência de plataforma de desenvolvimento.

É notório que os trabalhos, principalmente acadêmicos, que consideram um número maior de plataformas, são cada vez menos volumosos devido à sua complexidade. Nessa linha, pode-se dizer que o trabalho de Miravet et al. (2014) é uma das abordagens acadêmicas mais completas pelo fato de considerar o desenvolvimento nativo para duas plataformas móveis (Android e *windows phone*), sistema web (Servelets Java ME) para o gerenciamento dos dados das aplicações e a comunicação entre eles através do SOAP[23] (*Simple Object Access Protocol*). Miravet et al. (2014) propõem DIMAG (*device-independent mobile application generation*) um *framework* para especificar aplicações móveis, onde o lado do cliente (aplicativos) e servidor (sistema web) são modeladas de forma independente. Isto é, todo o código para sincronização e transições de dados entre aplicações web e móveis, construção de regras de negócio do aplicativo e sistema web, controle de versões do aplicativo, comportamento do software, aparência e definição da interface entre outros detalhes são modelados em XML. A interface é modelada através de uma máquina de estado descrita em XML. A partir da modelagem em XML o *framework* DIMAG gera um *Abstract syntax tree* (AST) que é utilizado pelo gerador DIMAG para gerar código para as principais plataformas. Os elementos XML disponíveis para realizar a modelagem são prioritariamente fixos (desc: controle de versão, flow: ciclo do aplicativo, ui: documentos de interface, entre outros) e o gerador de código é customizado de acordo com esses elementos, o que mantém a modelagem engessada para novas funcionalidades. Isto é, a extensão do *framework* é muito custosa, inviabilizando a adaptação para o atendimento de novos recursos ou novas tecnologias.

3.1.3 Plataformas móvel, web, interoperabilidade e *desktop*

Para atender um sistema multiplataforma na essência do termo é necessário uma abordagem que ofereça uma visão holística de todo o processo, isto é, um ambiente único de modelagem que permita a especificação de todas as plataformas: móvel, web, *desktop* e a comunicação entre elas. Abordagens com essa visão holística são mais difíceis de encontrar.

HAXE[24] é uma linguagem de código aberto que garante alto nível de abstração, usada para compilar código para várias plataformas distintas. HAXE suporta a geração de nove linguagens: *JavaScript*, Neko, PHP, Python, C++, C#, Java, ActionScript 3 e Flash, a partir de um código base. É uma linguagem muito popular para a produção

[23] www.w3.org/TR/soap/
[24] haxe.org/

de jogos, devido à sua diversidade de bibliotecas, curva de aprendizado rápido e o fator multiplataforma. Porém a linguagem gera *bytecode* para algumas plataformas, não permitindo a extensão do fonte, além de comprometer a independência de plataforma de desenvolvimento pelo fato do alto custo em estender a linguagem. Embora HAXE possa gerar código para várias plataformas, ele não gera de forma automática a interoperabilidade entre as aplicações criados por ele, mas oferece uma base para o desenvolvimento manual através de web *request* e um JSON *parser*[25]. A linguagem não foi projetada para desenvolver um sistema multiplataforma e sim gerar código para plataformas de forma individual. Nesse caso, a abordagem completa utiliza o metamodelo KDM (*Knowledge Discovery Metamodel*) para transcrever um sistema legado para a linguagem de modelagem KDM, na sequência é realizado a transformação para a linguagem de modelagem HAXE através de técnicas M2T (MARTINEZ; PEREIRA; FAVRE, 2017). O HAXE faz uso do *framework Acceleo* (ACCELEO, 2010) para geração de código nativo para as plataformas alvo, incluindo: Android, iOS, BlackBerry, além de aplicações *desktop* e web.

KOTLIN[26] é uma linguagem de programação estaticamente tipada, criada para o reaproveitamento de código, permitindo que grande parte do código usado em um produto ou plataforma seja utilizado em outro. KOTLIN foi lançada em 15 de fevereiro de 2016 e é considerada linguagem oficial para a plataforma Android, junto com o Java. Atualmente, ela pode ser usada para o desenvolvimento web através da linguagem *JavaScript* (*client-side* e *server-side*), *desktop* e nativamente para o desenvolvimento Android, além de prover base para a comunicação de dados entre as plataformas através do *RESTful*. KOTLIN está desenvolvendo a tecnologia Kotlin/Native que permitirá a geração de executáveis nativos para rodar em outras plataformas que não utilizam o JVM, assim como iOS. Parte do Kotlin/Native já está disponível e suporta o desenvolvimento para as seguintes plataformas: Windows, Linux, MacOS, iOS e Android. O fato de ser uma linguagem de caráter universal, atribui a ela as mesmas características da linguagem HAXE: compromete a independência de plataforma, faz a geração de *bytecode*, não permitindo a extensão do código fonte, e substitui uma linguagem por outra, não elevando de forma considerável o nível de abstração.

A ferramenta IBM RATIONAL RHAPSODY[27] possibilita o desenvolvimento de sistemas embarcado, sistema de tempo-real e software comerciais através de uma linguagem de modelagem visual baseado na UML ou *Systems Modeling Language* (SysML). Os modelos podem ser simulados para realizar testes e usados para gerar códigos. Também é possível sincronizar alterações nos modelos com a geração de códigos e vice-versa. A ferramenta gera códigos para as linguagens C, C++, Ada, Java e C *Sharpe*, atendendo às plataformas móveis, web e *desktop*. A política usada pela ferramenta é manter todo

[25] api.haxe.org/haxe/Json.html
[26] kotlinlang.org
[27] www-03.ibm.com/software/products/en/ratirhapfami

o desenvolvimento e manutenção do software diretamente no ambiente de modelagem, não recomendando a extensão do código fonte gerado, uma vez que afetará a função de reengenharia e engenharia avante para futuras manutenções. O único ponto não previsto pela ferramenta é o tratamento da comunicação de dados entre os aplicativos gerados. Assim como o HAXE, o IBM RATIONAL RHAPSODY compromete a independência de plataforma de desenvolvimento e limita o desenvolvedor às funcionalidades previstas na ferramenta.

O GENEXUS[28] é uma ferramenta comercial que usa o conceito do DSDM para modelar de forma visual sistemas complexos. GENEXUS trabalha com o conceito de *Model First*, com a definição prévia da base de dados, seguido pela geração automática das operações do tipo CRUD para cada entidade. Em seguida a modelagem do sistema é definida em um ambiente visual de forma modularizada, onde, para cada componente de interface adicionado na modelagem. Pode-se especificar regras de negócio e controlar o comportamento através de uma linguagem generativa disponibilizada pela ferramenta. Com os conceitos do domínio expressos através da modelagem a definição das plataformas para geração de código fonte pode ser selecionado. O GENEXUS oferece suporte para muitas plataformas. Para as plataformas móveis são suportadas a geração de código nativo para Android, *BlackBerry*, iOS e *windows phone*. Para a geração da plataforma web é suportado Ruby, Java e .Net, além das tecnologias de interface responsiva (HTML5, CSS e JS). Para os sistemas *desktop* a ferramenta oferece a geração de código em C# ou Java. Além dessa diversidade de linguagens suportadas, o serviço de comunicação entre as plataformas pode ser especificados através da modelagem e gerados de forma automática nos padrões SOAP ou REST, tornando a ferramenta mais completa investigada por esta pesquisa.

Embora a ferramenta GENEXUS apresente um perfil holístico de desenvolvimento, combinando com a definição dos conceitos de um sistema multiplataforma efetivamente em um único ambiente, ele apresenta alguns pontos a serem considerados. O fato da ferramenta ser comercial e usar o padrão caixa preta, não permite a extensão para suporte a outras funções além das pré-estabelecidas pela ferramenta, limitando o comportamento das funcionalidades que os aplicativos poderão oferecer. Os códigos-fonte gerados pela ferramenta não são legíveis, dificultando a adição de funcionalidades não disponíveis pela ferramenta ao produto final, além de ser uma prática não recomendada pois a extensão do código pode inviabilizar a manutenção do software. Outro ponto negativo relacionado à manutenção é o fato que o desenvolvedor fica dependente do sistema de modelagem interpretado unicamente pelo GENEXUS. Portanto, tanto na expansão tecnológica quanto na independência de evolução tecnológica o usuário fica perigosamente preso a ferramenta.

[28] www.genexus.com/

3.1.4 Outras plataformas

O conceito DSDM tem mostrado flexibilidade na geração de código para diversas áreas em diversos ambientes. Para comprovar o potencial do conceito, este tópico apresenta alguns trabalhos com o foco em áreas pouco divergentes com o projeto. Um dos objetivos desta tese é gerar código para novas tecnologias, permitindo a capacidade de expansão e a proposta de uma abordagem atemporal, de forma que o desenvolvedor possa estendê-la para a inclusão de novas tecnologias.

Alguns autores trabalham na geração de código e modelagem, porém não especificamente na área da multiplataforma. Os trabalhos de Bērziša et al. (2015), Hinkel e GoldSchmidt (2016) e SILABMDD (SILVA et al., 2014) tratam exclusivamente da questão da modelagem. Neste caso, a maior contribuição são modelos com expressividade para tratar em alto nível os dinâmicos ambientes de negócio do mundo corporativo. Bērziša et al. (2015) propõem o *Capability Driven Development* (CDD), trata-se de um metamodelo que considera as mudanças organizacionais de uma empresa e permite com que elas sejam integradas em alto nível aos sistemas de informação gerenciais, de forma que inclua todos os *stakeholders* do projeto. O objetivo principal é elevar o nível de abstração e disponibilizar um ambiente de modelagem que possa expressar e documentar os conceitos do domínio, além de facilitar as alterações do sistema. Hinkel e GoldSchmidt (2016) tratam de transformações de modelos usando a linguagem NTL (NMF (*.NET Modeling Framework*) *Transformation Language*)) que provê suporte ao DSDM na plataforma .NET e demonstra sua implementação. Os autores propõem uma ferramenta para mapear modelos em códigos usando a linguagem NTL. A geração de código é realizada através de compiladores desenvolvido como parte do trabalho e não focam em uma plataforma específica. É apresentada a viabilidade em se usar a linguagem NTL frente a outros *frameworks* DSDM, tratando questões como expressividade da modelagem. SILABMDD é uma abordagem DSDM que utiliza a linguagem SilabReq para a modelagem de requisitos baseado nas especificações dos casos de uso, disponibilizando uma modelagem textual que permite com que usuários possam definir e gerenciar os requisitos do software (SILVA et al., 2014). O objetivo principal é prover aos desenvolvedores um ambiente de modelagem completa referente à expressividade dos conceitos de um domínio. Nenhum dos autores tratam as questões de geração de código, nem focam em algum tipo de plataforma específica, forçando esforços na documentação, modelagem e concepção em alto nível abstração do software.

Para garantir reúso de código, abstração e encapsulamento de funcionalidades, muitos desenvolvedores criam APIs (*Aplications Programming Intefaces*) e disponibilizam para o aumento de produtividade. Nesse caso, o DSDM pode ser uma alternativa para a criação de APIs devido a vantagens como: expressividade, manutenção, legibilidade e o fato de ser conciso, além de poder ser melhor adaptado em diferentes ambientes de

desenvolvimento. Nessa linha, a abordagem DSLIT (COSENTINO; TISI; IZQUIERDO, 2015) provê mecanismos para gerar automaticamente DSL externa a partir de uma API orientada a objetos. Para validar a abordagem, a ferramenta DSLIT gera uma DSL textual em Xtext a partir de uma APIs baseada na linguagem Java. A ferramenta é integrada na plataforma Eclipse como um *plugin*, e utiliza técnicas dirigidas por modelos para representar as APIs em alto nível de abstração através de uma DSL declarativa. Por fim, a transformação da modelagem em código java é feita pela ferramenta *Acceleo*, que permite inclusive a geração para outras linguagens suportadas pelo gerador.

A flexibilidade do DSDM é comprovada pela apresentação de abordagens que divergem do objetivo principal desta tese e apresenta melhoria no processo de desenvolvimento, como é o caso da DSL PYTF (HINKEL et al., 2016). Essa DSL é usada para modelar em XML os conceitos da Neuro-Robótica. Uma vez concebida a DSL, é possível gerar códigos Python (linguagem muito usada pela Neuro-robótica) através do gerador *Jinja2*[29]. Os componentes utilizados no NRP(*Neurorobotics Platform*) foram baseados em C++ e Python e a DSL PYTF é usada para automatizar a geração de parte do código para o domínio de neuro-robótica. O artigo aponta que a neuro-robótica é melhor gerenciado através de modelos, representando a interconexão entre a parte robótica e as redes neurais. Portanto, para aumentar o nível de abstração foi utilizado DSDM, permitindo especificações em alto nível de abstração. A ideia principal seria a divisão do código em 3 partes: Plataforma, código repetitivo e código individual. Os resultados em se usar uma linguagem com pouco suporte ao DSDM, foram relatadas como válidas, uma vez que foi possível aumentar o nível de abstração na criação dos modelos no contexto na neurociência e gerar códigos usando o gerador *Jinja2*. Embora o artigo trate as questões do domínio da Neuro-Robótica, ele apresenta desafios similares a multiplataforma, uma vez que precisa lidar com modelagem e geração de códigos distintas das comumente usadas pelo DSDM.

Martinez, Pereira e Favre (2017) descrevem a migração de uma aplicação feita em C/C++ para diferentes plataformas, através do metamodelo KDM (*Knowledge Discovery Metamodel*). O KDM faz parte do conjunto de soluções para modernização de software proposta pela *Architecture-Driven Modernization* (ADM) e tem o potencial de modelar um software legado com alto nível de abstração. A partir dessa modelagem é possível a geração de código para várias plataformas através de geradores de código. O processo é orientado a modelos, ocorrendo transformações M2M de KDM para a modelagem HAXE, e este por sua vez, utiliza o *Acceleo* para gerar transformações M2T para código nativo. O KDM é focado em especificar e definir refatorações com o objetivo de auxiliar a modernização de software independente de plataforma ou linguagem de programação. A ideia principal do KDM é a representação de artefatos, associações e elementos de um software com o intuito

[29] jinja.pocoo.org/

de realizar transformações em seu código-fonte, visando a modernização ou alteração da linguagem de programação (PÉREZ-CASTILLO; GUZMAN; PIATTINI, 2011).

O KDM contém meta-classes específicas para representar desde código-fonte até a arquitetura de um determinado software e é focado em reengenharia de software. Isto é, seu objetivo principal seria representar um software legado em uma modelagem baseada no *Meta-Object Facility* (MOF) (SON; KIM; KIM, 2013). Essa modelagem por sua vez teria a capacidade de troca de metadados entre diversas ferramentas de modernização criadas por engenheiros de software para auxiliar a interoperabilidade e poder gerar a modernização em diferentes ambientes. Porém, o KDM é utilizado para representação de software a partir de seus metamodelos pré-definidos. Pois ele, como padrão, requisita um artefato de entrada para a definição dos conceitos de algum software. Portanto, o KDM não possui a estrutura adequada para o desenvolvimento multiplataforma utilizando um único ambiente, alguns elementos dessa estrutura podem ser: definição de plataformas, configuração da geração de código para a distribuição de funcionalidades e escrita de regras de negócio. Outro aspecto diz respeito a extensão do KDM, isso poderia comprometer a interoperabilidade com outras ferramentas de modernização, pelo fato de alterar o metamodelo KDM. Dessa forma, ele não é utilizado neste trabalho, uma vez que um dos objetivos é desenvolver um metamodelo para engenharia avante, com expressividades para o domínio de sistemas multiplataforma. Espera-se compor na modelagem instrumentos que permitam expressar nas plataformas que serão destinados, certos artefatos de software previamente modelados.

3.1.5 Análise Geral

Como apresentado nesta pesquisa, existem muitas opções na literatura e no mercado que visam solucionar a questão da multiplataforma com base no DSDM, comprovando o potencial do conceito em tratar esse problema. A análise dos trabalhos apresentados nesta revisão forneceu uma base consistente sobre o estado da arte no tratamento da questão multiplataforma através do uso do DSDM. Muitos grupos de pesquisa acadêmicos e comerciais apresentam grande empenho na solução dos desafios da multiplataforma com o uso do DSDM, soluções cada vez mais complexas e com maior eficiência surgem a todo instante. Os inúmeros resultados positivos apresentados pelos trabalhos mostram que a comunidade científica está no caminho de uma abordagem que ofereça contribuições para a engenharia de software e ofereça vantagens para que a indústria passe a utilizar mais este conceito, atualmente pouco explorado pelo mercado (HINKEL; GOLDSCHMIDT, 2016). Visando obter conclusões mais consistentes sobre os trabalhos, foram selecionados dois critérios de análise. Inicialmente, focando na análise dos benefícios mais explorados do conceito DSDM e, em seguida, uma análise mais técnica para identificação de tendências e estratégias mais assertivas no campo do DSDM.

3.1.5.1 Benefícios explorados do conceito DSDM

É visível que a medida que o software dependa de mais plataformas a complexidade do desenvolvimento aumenta e o número de trabalhos que tratam a questão diminui. Essa redução se dá devido à complexidade da abordagem no atendimento a várias plataformas, sendo que o atendimento efetivo a plataforma móvel já torna a abordagem complexa. Outro fato é que os dispositivos móveis foram o grande motivador para a condução de abordagens que reduzam o custo do desenvolvimento multiplataforma, devido aos vários sistemas operacionais comercializados junto aos dispositivos (UMUHOZA; BRAMBILLA, 2016). Tal fato pode ser observado na Tabela 4, onde é possível verificar os ambientes multiplataforma considerados pelos artigos e os anos de publicação. Os grupos de pesquisas deram preferência em tratar a questão do ambiente móvel, visto que os anos dos trabalhos publicados em geral são muito próximo um do outro. Esse fato dificulta a previsão do rumo das pesquisas para aplicações multiplataforma, já que não foi constatada uma progressão ordenada.

Tabela 4 – Anos de publicação organizados por ambiente multiplataforma.

Ambientes Multiplataforma	Abordados por artigos
Móvel	1 trabalho em 2010 4 trabalhos em 2011 5 trabalhos em 2012 7 trabalhos em 2013 7 trabalhos em 2014 4 trabalhos em 2015 3 trabalhos em 2016 3 trabalhos em 2017 1 trabalho em 2018 2 trabalhos em 2019 1 trabalho em 2020
Móvel e web	1 trabalho em 2014
Móvel e interoperabilidade	1 trabalho em 2012
Móvel, web e *desktop*	2 trabalhos em 2011
Móvel, web e interoperabilidade	1 trabalho em cada ano: 2009, 2011, 2015 e 2019
Móvel, web, interoperabilidade e *desktop*	1 trabalho em 2014 e 1 em 2016
Modelagem de ambiente de negócio	1 trabalho em cada ano: 2015, 2015 e 2016
Interoperabilidade	1 trabalho em cada ano: 2011, 2012 e 2016
Outros	1 trabalho em cada ano: 2015, 2016 e 2017

A relação do número de trabalhos que consideram cada plataforma é melhor visualizada na Figura 1. Sintetizando este gráfico, as abordagens que consideram apenas plataformas móveis lideram com 38 trabalhos, seguidas por trabalhos que tratam a combinação Móvel-Web-Interoperabilidade (4), apenas a questão da interoperabilidade (3), modelagem do ambiente de negócios (3) e outros (como: neurótica e reengenharia). Na

sequência, com 2 trabalhos encontrados, estão as categorias: Móvel-Web-*Desktop* e Móvel-Web-Interoperabilidade-*Desktop*. Com apenas 1 trabalho encontrado, estão as abordagens que consideraram: Móvel-Web e Móvel-Interoperabilidade.

Outros pontos alavancados pela análise gráfica é a identificação das plataformas ou as combinações de plataformas mais consideradas nas pesquisas e, em alguns casos, é possível verificar a ausência de pesquisas. É o caso das combinações de plataformas Web-*Desktop*, assim como Móvel-*Desktop*. Para essas combinações não foram localizados grupos de pesquisas atuantes, talvez pela baixa necessidade percebida pelo mercado, não viabilizando esforços para atender a esses cenários. Já combinações como Móvel-Web-*Desktop* e Móvel-Web-Interoperabilidade, que possuem mais plataformas e maior complexidade, apresentam uma quantidade maior de trabalhos, sinalizando maior interesse pela comunidade científica e pelo mercado em atender esses nichos.

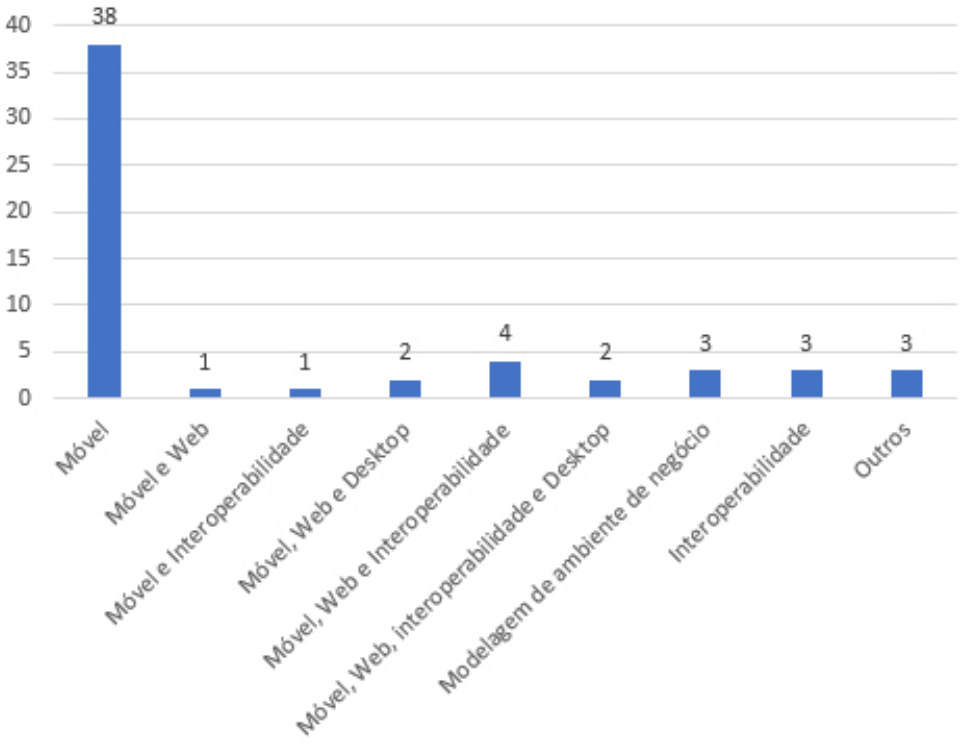

Figura 1 – Cenários multiplataforma abordados pela amostragem.

A análise dos benefícios do conceito DSDM explorados por cada trabalho é visualizada na Figura 2. Essa figura apresenta um resumo geral de todos os trabalhos considerados nessa pesquisa, onde a letra "S" significa que a abordagem explorou o benefício supracitado. Para cada trabalho é possível identificar quais os benefícios do conceito DSDM que foram efetivamente explorados, permitindo a descoberta de caminhos não explorados. A classificação para alguns trabalhos pode ser não precisa ou atender com certa parcialidade. Dessa forma, essa análise foi feita considerando o foco de cada trabalho e os apontamentos no artigo. Caso não houvesse nenhum apontamento para certo bene-

fício, era observado outras fontes, como sites, repositório de códigos e comentários em outros artigos, para assim classificar como não atendida (não-S). Se o artigo apresentasse apontamentos para determinado benefício, também eram observadas outras fontes para confirmar. De modo geral, podemos concluir que, de fato, existem lacunas que podem ser exploradas por futuras pesquisas, visto que existem benefícios pouco explorados pelas abordagens. Uma abordagem de modelagem flexível o suficiente para comportar as prin-

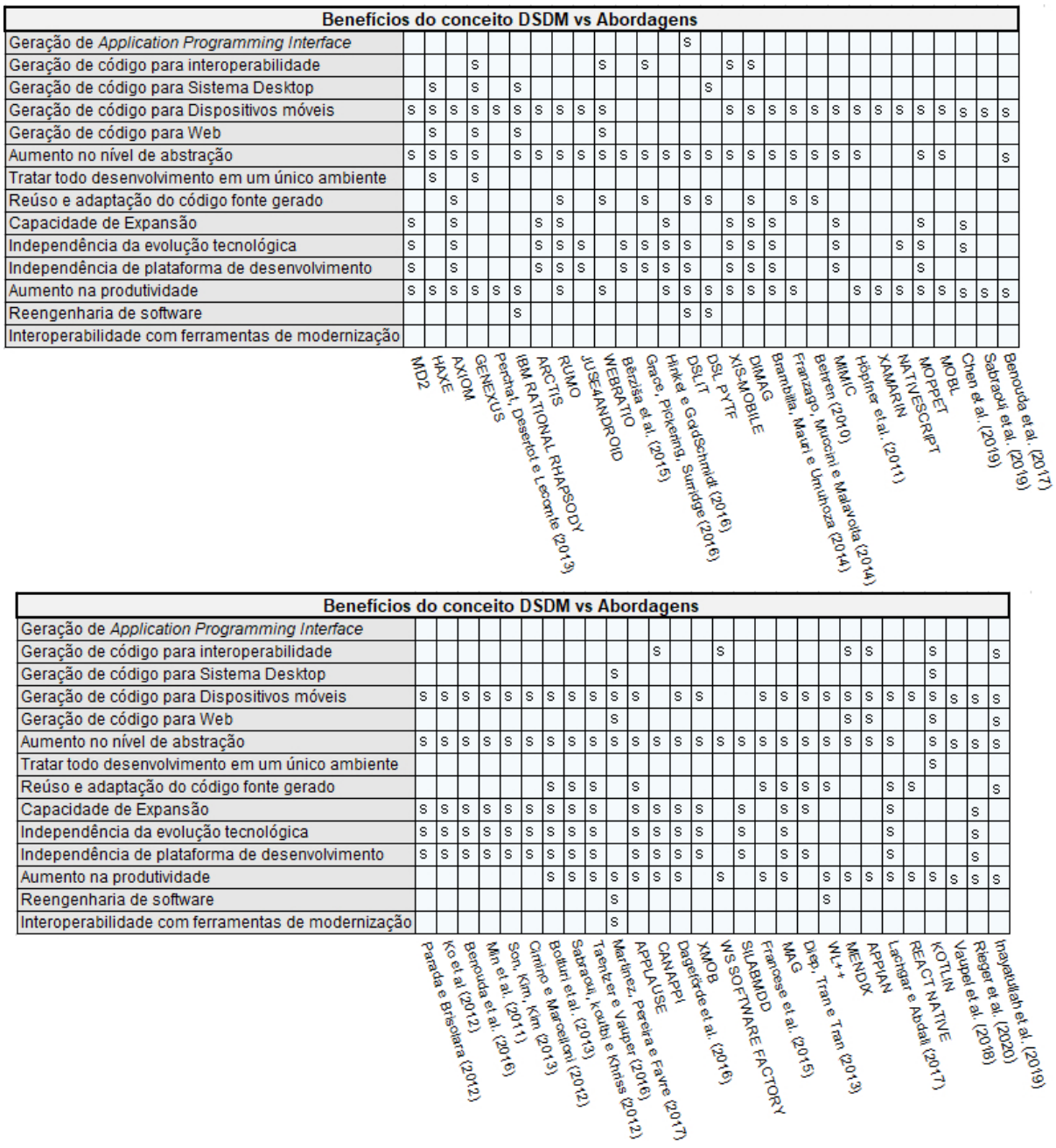

Figura 2 – Relação dos benefícios trazidos por cada abordagem pesquisada, onde "S" significa que a abordagem explorou, em algum nível, o benefício. Porém, destaca-se que essa análise simples tem um propósito de sumarização, uma vez que existem diferentes níveis para os benefícios em cada abordagem.

cipais plataformas do mercado e customizada para garantir a extensão do metamodelo a novas funcionalidades e novas tecnologias é passível de grande contribuição para este segmento.

O número de trabalhos que exploram cada benefício do conceito DSDM também foi contabilizado visando a localização de nichos não explorados pelas pesquisas. Os benefícios mais usados pelos trabalhos são também os mais conhecidos do conceito, assim como: aumento no nível de abstração dos conceitos do domínio, aumento na produtividade, independência da evolução tecnológica e independência de plataforma de desenvolvimento. Esses dados mostram quais os principais desafios do cenário multiplataforma atacados pelos grupos de pesquisas. A Figura 3 apresenta um panorama geral do número de trabalhos que exploram cada benefício. Através do gráfico da Figura 3, é possível identificar os potenciais do DSDM pouco explorados pelas abordagens para o atendimento do cenário multiplataforma. O tratamento de todo desenvolvimento em um único ambiente mostra-se um potencial pouco explorado pelas abordagens e pode ser um aspecto importante para promover o aumento de produtividade. Esse potencial, associado à capacidade de extensão dos modelos e código fonte, pode ser um nicho de pesquisa que traga importantes contribuições para a solução de parte dos desafios da multiplataforma.

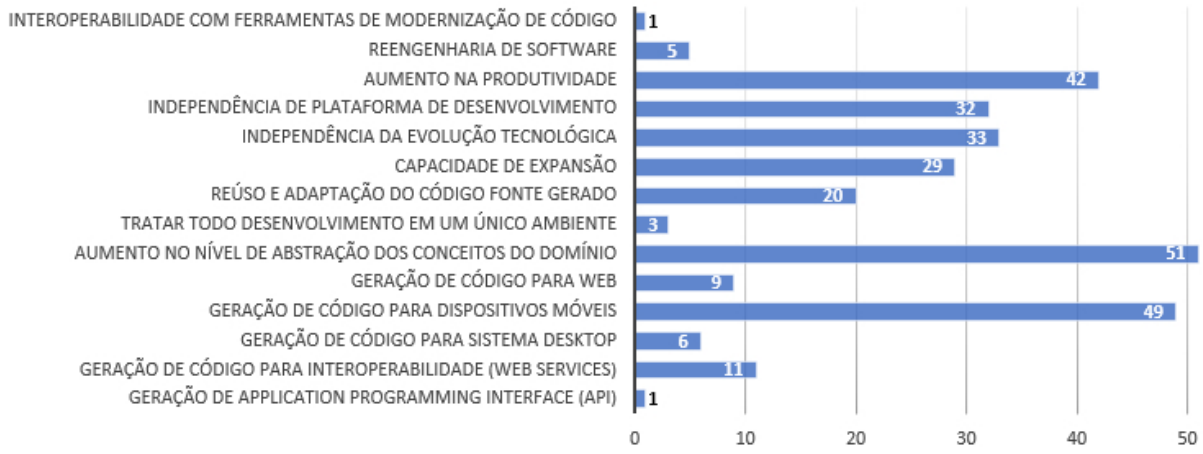

Figura 3 – Números de trabalhos utilizando cada um dos benefícios do DSDM.

Identificou-se que os benefícios do conceito DSDM para Capacidade de Expansão (inclusão de novas plataformas à ferramenta) e Independência da evolução tecnológica (Inclusão de novas funcionalidades) são apontados em uma quantidade razoável de trabalhos, mostrados na Figura 2 e os valores absolutos na Figura 3. Mas isso se resume à liberação do código-fonte para a comunidade. Ainda que o código-fonte esteja bem preparado para extensão, mexer em geradores e gramáticas não é uma tarefa trivial, pois envolve metaprogramação e conhecimentos na construção de linguagens.

3.1.5.2 Análise técnica dos trabalhos

Baseado nos dados coletados, conduziu-se uma segunda análise para verificar os detalhes técnicos de cada trabalho. O intuito dessa análise foi levantar informações baseadas em 4 critérios relacionados ao uso do conceito DSDM e ao desenvolvimento multiplataforma. Tais critérios são: aspectos do desenvolvimento, técnicas DSDM utilizadas, plataformas alvo de cada trabalho e linguagem de modelagem utilizada. Parte dos critérios (aspectos do desenvolvimento e linguagem de modelagem) foram baseados no trabalho de Umuhoza e Brambilla (2016).

- Aspectos do desenvolvimento: refere-se aos aspectos que envolvem a geração de aplicativos e são basicamente gerenciamento de dados, lógica de negócio, interação com o usuário e a interface do usuário (GUI);
- Técnicas DSDM utilizadas: este critério apresenta as estratégias de modelagem utilizadas pelos trabalhos (3);
- Plataformas alvo: apresenta de forma pontual quais as plataformas consideradas por cada trabalho e realiza uma visão genérica dos mais usados;
- Linguagem de modelagem: mostra quais as linguagens de modelagem selecionadas por cada trabalho e apresenta uma discussão sobre o assunto.

A Figura 5 apresenta uma síntese das abordagens pesquisadas de acordo com os critérios supracitados. O carácter "S" representa que a abordagem atendeu ao item apontado. O critério Plataformas Alvo, possui um carácter a mais em sua indicação, o

Legenda dos trabalhos investigados		
1 JUSE4ANDROID	20 Dageförde et al. (2016)	39 DSL PYTF
2 MIMIC	21 XMOB	40 Martinez, Pereira e Favre (2017)
3 Cimino e Marcelloni (2012)	22 MAG	41 Chen et al. (2019)
4 Parada e Brisolara (2012)	23 Botturi et al. (2013)	42 Sabraoui et al. (2019)
5 Ko et al (2012)	24 Lachgar e Abdali (2017)	43 Benouda et al. (2017)
6 Benouda et al. (2016)	25 MOBL	44 Vaupel et al. (2018)
7 ARCTIS	26 Höpfner et al. (2011)	45 Rieger et al. (2020)
8 Son, Kim, Kim (2013)	27 Francese et al. (2015)	46 Inayatullah et al. (2019)
9 Behren (2010)	28 WL++	47 WS SOFTWARE FACTORY
10 Min et al. (2011)	29 Brambilla, Mauri e Umuhoza (2014)	48 REACT NATIVE
11 RUMO	30 Franzago, Muccini e Malavolta (2014)	49 KOTLIN
12 Diep, Tran e Tran (2013)	31 Grace, Pickering, Surridge (2016)	50 XAMARIN
13 Sabraoui, koutbi e Khriss (2012)	32 CANAPPI	51 NATIVESCRIPT
14 MD2	33 APPLAUSE	52 APPIAN
15 AXIOM	34 DIMAG	53 MENDIX
16 XIS-MOBILE	35 Bērziša et al. (2015)	54 WEBRATIO
17 Perchat, Desertot e Lecomte (2013)	36 Hinkel e GoldSchmidt (2016)	55 HAXE
18 MOPPET	37 SILABMDD	56 IBM RATIONAL RHAPSODY
19 Taentzer e Vauper (2016)	38 DSLIT	57 GENEXUS

Figura 4 – Legenda dos trabalhos apresentados na Figura 5.

Abordagens	Aspectos				Estratégia					Plataformas Alvo							Ling. Modelagem							
	GUI	Gerenc. De Dados	Lógica de negócio	Interação do usuário	PIM-to-NC	PIM-to-PSM-to-NC	PSM-to-NC	PIM-to-CPC	PIM-to-FSM-to-CPC	Android	iOS	Windows Phone	Web	Desktop	Interoperabilidade	Outras	Extensão UML	MOF	Máquina de Estado	Linguagem Universal	DSL Textual	XML	DSL Visual	
1	$						$			SN							$							Abordagens Acadêmicas
2	$				$					SN							$							
3	$				$					SN									$					
4	$	$	$	$	$					SN							$							
5	$	$	$	$	$					SN							$							
6	$	$	$	$	$					SN							$							
7	$	$	$	$	$					SN							$		$					
8	$	$	$	$	$					SN							$	$						
9	$	$	$	$	$						SN						$				$			
10	$	$	$	$	$							SN					$							
11	$				$					SN	SN	SN						$						
12	$				$					SN	SN	SN										$		
13	$				$					SN	SN	SN					$							
14	$	$	$	$		$				SN	SN										$	$		
15	$	$	$	$		$				SN	SN						$				$			
16	$	$	$	$		$					SN	SN					$							
17	$	$	$	$	$					SN	SN									$				
18	$	$	$	$		$				SN	SN						$		$			$		
19	$	$	$	$	$					SN	SN										$			
20	$	$	$	$		$				SN	SN										$	$		
21	$	$	$	$		$				SN	SN										$			
22	$	$	$	$	$						SN	SN					$		$					
23	$	$	$	$		$					SN	SN					$		$					
24	$	$	$	$		$				SN	SN	SN									$			
25	$	$	$	$				$		SH	SH	SH									$			
26	$	$	$	$					$	SH	SH	SH									$			
27	$			$				$		SH	SH	SH							$					
28	$	$	$	$				$		SH	SH	SH									$			
29	$	$	$	$				$		SH	SH	SH								$				
30	$	$	$	$				$		SH	SH	SH									$			
31		$							$						$				$			$	$	
32		$			$										$						$			
33	$	$	$	$	$					SN	SN	SN	$								$			
34	$	$	$	$			$			SN		SN	$		$					$		$		
35		$	$		$											$							$	
36		$	$		$											$							$	
37		$	$		$											$	$			$				
38			$		$											$					$			
39			$		$											$						$		
40	$	$	$	$	$					SN	SN	SN	$	$				$						
41	$				$					SN	SN								$					
42	$				$					SN											$			
43	$	$		$	$							SN					$							
44	$	$	$	$		$				SN	SN												$	
45	$					$				SN	SN										$	$		
46	$	$		$	$					SH	SH		$		$		$							
47	$	$	$	$			$			SN	SN				$					$			$	Comerciais
48	$	$	$	$				$		SH	SH	SH	$		$					$			$	
49	$	$	$	$				$		SH	SH	SH	$		$		$						$	
50	$	$	$	$			$			SN	SN	SN	$	$						$				
51	$	$	$	$			$			SN	SN	SN	$	$			$							
52	$	$	$	$			$			SN	SN	SN	$	$	$					$			$	
53		$			$										$						$		$	
54	$	$	$	$	$					SN	SN													
55	$	$	$	$			$			SN	SN	SN	$	$	$					$				
56	$	$	$	$		$				SN	SN	SN					$							
57	$	$	$	$	$					SN	SN									$				
Soma:	49	45	42	40	30	11	7	7	2	43	37	25	10	5	10	5	20	3	8	9	17	8	9	

Figura 5 – Análise técnica das abordagens pesquisadas.

"N" significa Nativo e o "H" significa híbrido ou interpretado ou generativo. Sendo assim, "SN" indica que a abordagem atende ao item através da geração de aplicativos nativos e o "SH" através da geração de aplicativos híbridos ou interpretados ou generativos. Com essa tabela é possível inferir algumas informações a respeito dos critérios analisados em cada trabalho. A Figura 4 apresenta a numeração de cada abordagem mostrado na Figura 5.

3.1.5.3 Aspectos do desenvolvimento

O aspecto de desenvolvimento mais considerado pelas abordagens é o GUI, atendido por 85% dos trabalhos, sendo que 16% destes trabalhos consideram apenas o GUI para geração de código. Isto comprova que um dos pontos mais críticos no desenvolvimento de aplicativos móveis é a interface do usuário, pelo fato de possuir diferenças consideráveis entre as plataformas na forma de desenvolvimento.

Outro aspecto bastante considerado entre as abordagens é o gerenciamento de dados, com 78% entre os trabalhos pesquisados, seguido pela lógica de negócio com 72%. O gerenciamento de dados possui destaque por se tratar de uma área de suma importância principalmente para as plataformas móveis, pois diz respeito ao armazenamento dos dados no cliente e no servidor. A lógica de negócio é crucial para funcionamento do aplicativo, porém é uma área que já foi bastante explorada pela engenharia de software e pode ser modelada através de abordagens tradicionais como UML, BPM (*Business Process Modeling*), entre outros. Portanto, não é considerada um desafio tão grande quanto os outros aspectos para o desenvolvimento multiplataforma. A interação do usuário é o aspecto menos considerado entre as abordagens, considerada em 70% dos trabalhos. Porém ela acompanha de perto a GUI, considerada um complemento desse aspecto. Isto é, sempre que a interação é considerada pelas abordagens obrigatoriamente a GUI também é considerada, sendo que o contrário não é verdade. A interação do usuário é um dos grandes desafios para o desenvolvimento multiplataforma devido à sua complexidade, sendo tratado pela maior parte dos trabalhos através de uma máquina de estados, onde o mapeamento é expressado através da linguagem XML ou através de uma DSL textual. A Figura 6 mostra um panorama geral dos aspectos mais considerados pelas abordagens.

3.1.5.4 Estratégias DSDM utilizadas

Este critério analisa a quantidade de camadas de modelagem usadas para expressar os conceitos do domínio por cada abordagem e pode auxiliar a descoberta das melhores técnicas para determinados nichos. A estratégia PIM-to-NC (*Platform-Independent Model to Native Code*) lidera, presente em 52% (30) das abordagens pesquisadas, onde apenas 5% são abordagens comerciais. Isso mostra que abordagens acadêmicas normalmente optam pela PIM, independente de considerar uma ou mais plataformas alvo, visto que o

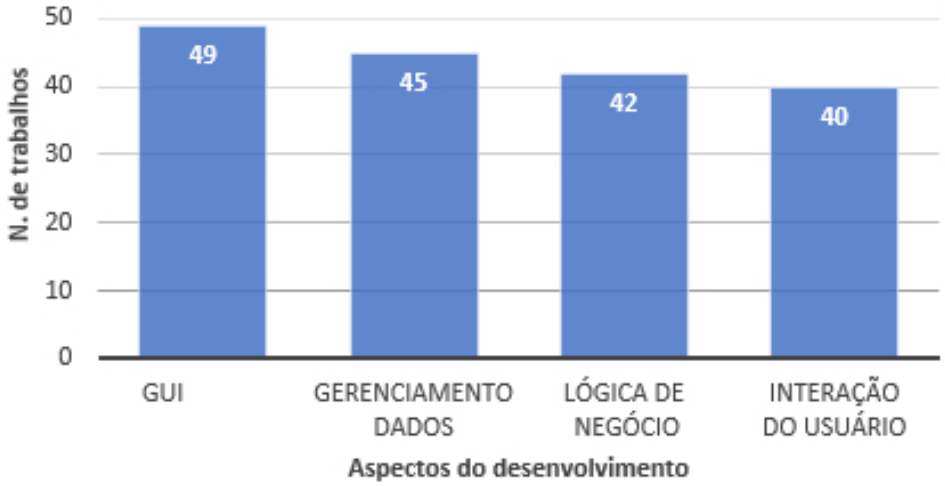

Figura 6 – Número de trabalhos que considera cada aspecto do desenvolvimento.

número de abordagens que tratam apenas uma plataforma (9 abordagens) e abordagens que tratam mais que uma abordagem (8 abordagens), são muito próximas.

Já a estratégia de modelagem PIM-to-PSM-to-NC (*Platform-Independent Model to Platform-Specific Model to Native Code*), usada por 19% (11) dos trabalhos, apresenta um padrão bem definido, onde todas as abordagens que utilizaram essa técnica consideraram mais que uma plataforma de desenvolvimento. Coincidentemente, todas as abordagens que usaram essa estratégia trabalharam apenas com as plataformas móveis, formando um outro padrão de comportamento. A estratégia PIM-to-PSM-to-NC permite que os engenheiros possam modelar algumas especificidades, como utilizar recursos disponíveis a apenas uma plataforma específica ou modelar funções distintas entre os SO, aproveitando melhor o que cada plataforma tem a oferecer. Porém essa prática prejudica a manutenção, uma vez que o engenheiro precisa alterar os modelos específicos de plataforma de forma individual. Talvez esta seja a razão de não ser a estratégia mais usada entre as abordagens.

As abordagens comerciais que focam na geração de código nativo para várias plataformas usam a estratégia PSM-to-NC (*Platform-Specific Model to Native Code*) em sua totalidade, mostrando um outro padrão de comportamento. A estratégia PSM-to-NC, usada em 12% (7) dos trabalhos pesquisados, possibilita a disponibilização de um ambiente de modelagem específico para cada plataforma suportada e uma linguagem universal para definição das regras de negócio. Isso permite a definição específica das funcionalidades de cada plataforma. Porém, essa estratégia é pouco usada entre as abordagens acadêmicas, totalizando 4% entre as abordagens pesquisadas.

A estratégia PIM-to-CPC (*Platform-Independent Model to Cross-Platform Code*) é usada pelas abordagens que geram aplicativos híbridos e se apoiam em algum *framework* para a geração dos executáveis. Portanto, visa gerar códigos web utilizando uma mode-

lagem independente de plataforma. Essa estratégia é usada em 12% (7) das abordagens acadêmicas e 4% entre as comerciais. Outra estratégia que possui poucos adeptos é a PIM-to-PSM-to-CPC (*Platform-Independent Model to Platform-Specific Model to Cross-Platform Code*), que também gera códigos web com o diferencial de permitir especificações através da camada PSM, utilizada em apenas 4% (2) das abordagens acadêmicas e nenhuma na comercial, resultando na menor escolha entre as abordagens. As abordagens que utilizam a geração de aplicativos híbridos aparecem em menor número, um dos motivos pode ser a dependência de um *framework* comercial para a geração dos códigos executáveis e dos aspectos positivos apresentados pelos aplicativos nativos.

A Figura 7 apresenta o balanço geral das técnicas DSDM utilizadas pelas abordagens

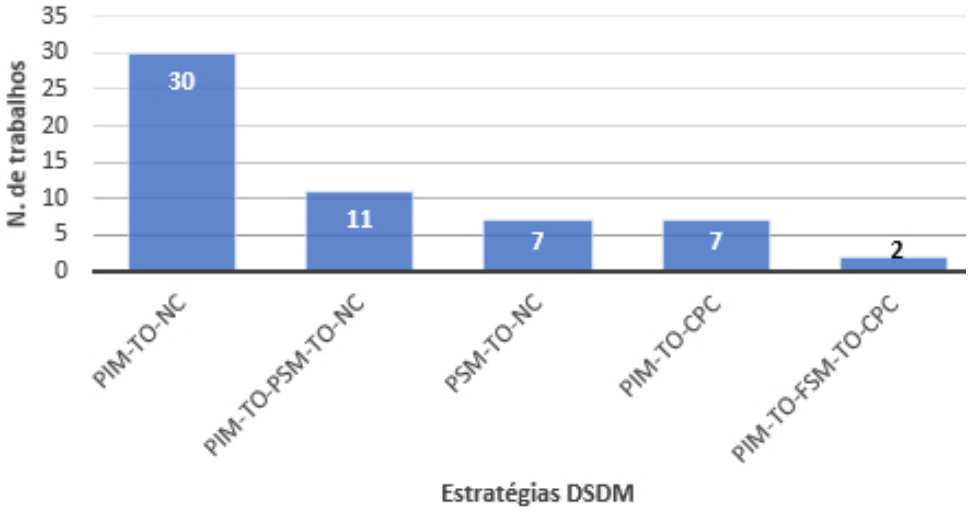

Figura 7 – Número de trabalhos que considera cada estratégia DSDM.

3.1.5.5 Plataformas alvo

Como já mencionado, as plataformas móveis são as mais consideradas para o tratamento da questão multiplataforma, usadas em 86% do total dos trabalhos pesquisados, sendo que 66% dos trabalhos tratam unicamente as plataformas móveis. Entre as abordagens que consideram plataformas móveis, o Android é o mais usado, com 87,5% dos trabalhos, seguido pela plataforma iOS (75%) e o *windows phone* (55%). Analisando ainda os trabalhos que consideram plataformas móveis (49 trabalhos), 40 focam no desenvolvimento de código nativo (81%, demonstrado pelos caracteres "SN"), enquanto o restante (19%, demonstrado pelos caracteres "SH") focam nos aplicativos híbridos. Tal fato indica uma preferência para a geração de códigos nativos, independente da abordagem ser comercial ou acadêmica.

Através desses dados, é possível observar algumas tendências nas conduções dos trabalhos que envolvem a multiplataforma e o conceito DSDM. É sugestivo, por exemplo, considerar as plataformas móveis em futuros projetos, de preferência Android e iOS. Mostra-se também, uma pré-disposição na geração de código nativo legível, visando a sua extensão e a inclusão de funções não previstas pela modelagem. A menor parte dos trabalhos (39%) relata explicitamente a preocupação em gerar códigos legíveis. O restante dos trabalhos não relata tal preocupação e não mencionam a possibilidade de extensão de código fonte. De qualquer forma, o fato de uma quantidade expressiva de trabalhos relatar a geração de códigos legíveis pode apontar um importante aspecto a ser considerado em trabalhos futuros. A Figura 8 apresenta o número de abordagens que consideram cada plataforma.

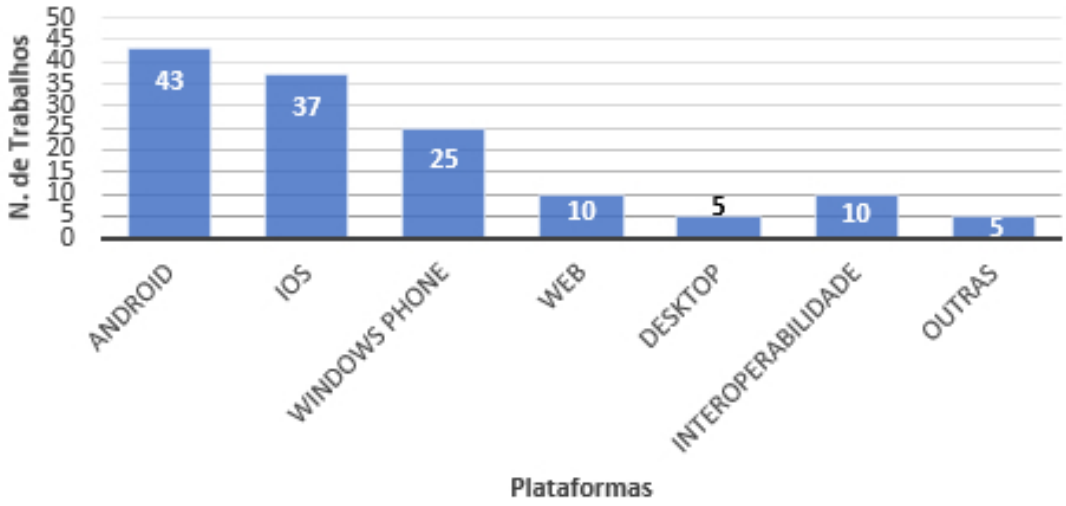

Figura 8 – Número de trabalhos que considera cada plataforma.

As outras plataformas são consideradas em menor número pelos trabalhos e em sua maioria servindo de apoio às plataformas móveis e ao ambiente multiplataforma. Como é o caso da interoperabilidade, que é considerada em 17% dos trabalhos pesquisados, mas é uma das mais utilizadas após as plataformas móveis. Este dado reflete a importância da interoperabilidade nos ambientes multiplataforma, sendo considerada principalmente em abordagens comerciais (63% dos trabalhos), onde o intuito é prover um ambiente de desenvolvimento mais completo e operacional possível. A plataforma web possui os mesmos 17% que os serviços de interoperabilidade. Essa plataforma promove normalmente o processamento dos dados dos dispositivos que compõem o sistema multiplataforma, além de possibilitar o acesso ao sistema através de outros dispositivos. Tanto a interoperabilidade, quanto a plataforma web são mais consideradas em ambientes de desenvolvimento comerciais, mostrando que são essenciais na composição de um cenário multiplataforma. Já a plataforma *desktop* possui o menor índice de uso entre os aplicativos (8%) indicando a baixa necessidade entre as abordagens.

3.1.5.6 Linguagem de modelagem

A análise das linguagens de modelagem usadas pelos artigos, também pode contribuir para a tomada de decisões de futuros trabalhos. A linguagem mais utilizada pelas abordagens é a UML, com 35% entre as abordagens pesquisadas. Essa linguagem é usada preferencialmente por abordagens acadêmicas, 88%, que focam exclusivamente em aplicativos móveis nativos. A UML é uma das linguagens de modelagem mais consolidadas no meio científico e tem o potencial de envolver os *stakeholders* devido ao apelo visual e a capacidade de elevar o nível de abstração. Normalmente, as abordagens que utilizam essa linguagem estendem os seus diagramas no intuito de aumentar a sua expressividade e adaptá-los para definição de funções específicas.

O MOF é uma meta-linguagem e é utilizado para a definição de outras linguagens de modelagem, como a UML. Mas é possível fazer o seu direto e nesse caso é usado por 5% dos trabalhos pesquisados, todos de caráter acadêmico. O uso do MOF de forma direta é bastante usado em abordagens que usam o DSDM para reengenharia de software (MARTINEZ; PEREIRA; FAVRE, 2017), tendo baixa preferência em abordagens que utilizam a engenharia avante.

O conceito de máquina de estados é explorado em sua maior parte para tratar a interação do usuário com o aplicativo e é utilizada somente em abordagens acadêmicas, com 14% de adesão. Máquina de estado pode organizar as interações dos usuários no aplicativo, de modo que cada estado normalmente representa uma função e as transições fornecem dados sobre alterações nas funções (USMAN; IQBAL; KHAN, 2017), auxiliando a geração de código.

A Linguagem Universal é uma alternativa desenvolvida pelas abordagens para unificar o desenvolvimento através de um código base, e possui diretivas para compilar em diferentes plataformas. É usada preferencialmente por abordagens comerciais (10% dos trabalhos pesquisados) para definição de regras de negócio e na maior parte dos casos serve como um auxílio de uma DSL visual. Essa linguagem é utilizada também por algumas abordagens acadêmicas (8% dos trabalhos pesquisados) no intuito de atender a ideia de escrever uma vez e rodar em qualquer plataforma, como o caso da abordagem Perchat, Desertot e Lecomte (2013). Porém, embora consiga unificar o desenvolvimento multiplataforma em um único código base, é necessário programar diretivas de compilação e detalhes técnicos das plataformas.

A DSL Textual é usada predominantemente pelas abordagens acadêmicas e é a segunda linguagem mais utilizada, com 29% do total. A DSL textual é capaz de expressar uma quantidade maior de conceitos com maior detalhamento e elevando o nível de abstração. O XML é usado em 14% dos trabalhos, sendo todos acadêmicos e é usado muitas vezes para representar informações sobre o mapeamento das interações do usuários.

A DSL Visual apresenta 11% de adesão entre as abordagens comerciais e 7% entre as acadêmicas, aproveitando da perspectiva visual que reduz a curva de aprendizado e facilita a integração dos *stakeholders* ao projeto. Normalmente, a DSL Visual, utiliza uma linguagem universal para expressar regras de negócios específicas.

A Figura 9 apresenta as linguagens de modelagem e os trabalhos que as consideram.

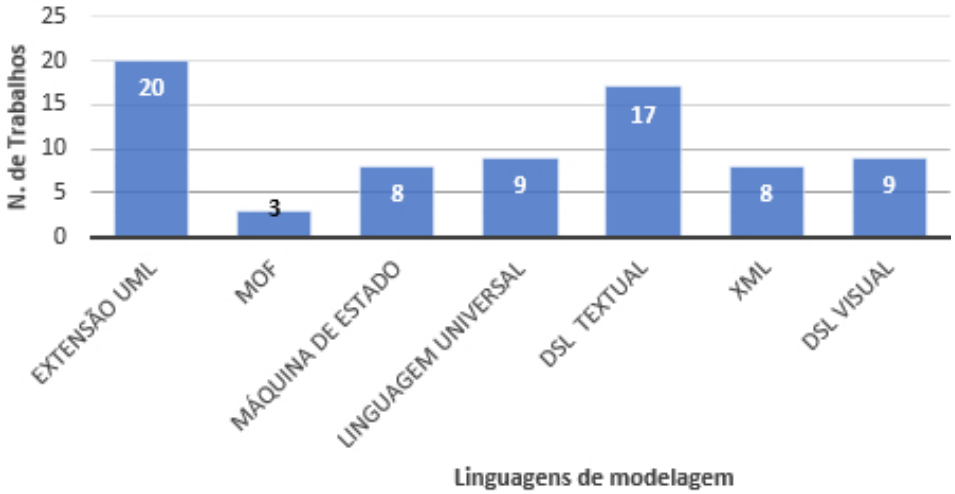

Figura 9 – Número de trabalhos que considera cada linguagem de modelagem.

Analisando-se o cenário atual, tanto na indústria como na academia, constata-se que não existe uma solução multiplataforma capaz de reunir todas características necessárias para compor um ambiente holístico de desenvolvimento explorando as principais vantagens do DSDM. O único trabalho que consegue reunir todas as plataformas em um único ambiente é a ferramenta comercial GENEXUS, porém impossibilita a extensão da ferramenta pelo desenvolvedor, o que compromete a liberdade de desenvolvimento. Para gerar uma importante contribuição para a engenharia de software e motivar a adoção do conceito DSDM em larga escala pela indústria, deve-se desenvolver uma abordagem que consiga explorar os seguintes benefícios: Aumento na produtividade, Independência de plataforma de desenvolvimento, Independência da evolução tecnológica, Capacidade de Expansão, Reúso e adaptação do código fonte gerado. Tratar todo desenvolvimento em um único ambiente e aumento no nível de abstração dos conceitos do domínio. Esta tese visa estudar formas de conceber um ambiente que explore esses benefícios.

3.1.6 *Frameworks* multiplataforma mais recentes

Nas fases finais do desenvolvimento da abordagem proposta, novos *fameworks* multiplataforma se popularizaram e portanto foram incluídos em uma seção separada neste capítulo, já que parte deles foram citadas durante as avaliações conduzidas. É o

caso do *framework* Flutter[30], Ionic[31] e Qt[32].

Embora o Flutter tenha sido apresentando pela primeira vez em 2015 (em um evento sobre a linguagem Dart[33]), foi em poucos anos atrás que ganhou notoriedade entre os desenvolvedores. Flutter é um *framework* que utiliza a linguagem Dart para modelar e gerar códigos executáveis para web, dispositivos móveis e *desktop* e é considerado uma abordagem interpretada. Flutter utiliza um compilador Dart para converter o código em linguagem de maquina nativo e assim rodar no dispositivo, junto com um mecanismo de renderização de interface. Dessa forma, o Flutter não utiliza elementos nativos para renderização de interface, prejudicando um pouco o seu desempenho.

O Ionic foi criado em 2013, mas assim como o Flutter, se popularizou recentemente. Esse *framework* era baseado no Cordova e em seguida foi aprimorado para rodar em cima do React Native. Essa mudança foi importante para aumentar o desempenho durante a execução do aplicativo, já que o React Native utiliza elementos nativos para executação. O Ionic utiliza tecnologias web conhecidas (HTML, CSS, JavaScript) para o desenvolvimento e permite a integração com outros *framework*, entre eles ReactJS, Angular ou VueJS. Existe uma versão (Ionic Pro) que permite o desenvolvimento para *Cloud* e oferece suporte aos usuários. Por se tratar de um *framework* híbrido, ele é executado no *browser engine*, o que prejudica o desempenho em relação aos aplicativos nativos.

Diferente dos demais, o Qt é um *framework* amplamente usado no ambiente *desktop* Linux, macOS e Windows. Mas também oferece suporte para outras plataformas, assim como Android, iOS, UWP para dispositivos móveis, web, plataformas de televisões *smart*, automóveis e também inclui várias bibliotecas para desenvolvimento de jogos. O grande diferencial desse *framework* é o desempenho que é considerado bom na maior parte das plataformas. Qt usa o C++ padrão para o desenvolvimento multiplataforma de interfaces, com geração de código através do seu *Meta Object Compiler*. Porém, o Qt não oferece suporte a outros aspectos do desenvolvimento, além de possuir leves discrepâncias na emulação do visual nativo para algumas plataformas.

[30] https://flutter.dev/
[31] https://ionicframework.com/
[32] https://wiki.qt.io/Portal:Mobile
[33] https://dart.dev/guides

4 Fundamentação teórica: Desenvolvimento de Software Dirigido por Modelos

Conforme apresentado no Capítulo 3, que contextualiza o desenvolvimento multiplataforma na academia e indústria, o conceito Desenvolvimento de Software Dirigido por Modelos (DSDM) é utilizado pela ampla maioria dos trabalhos. Tal fato mostra a eficiência do conceito no tratamento do problema, apresentando em sua maior parte resultados consistentes e satisfatórios, motivo pelo qual foi escolhido também para ser usado nesta tese. O DSDM é a combinação de programação generativa com linguagens específicas de domínio (DSL – *Domain-Specific Language*) e técnicas de transformação de software, aspectos esses que já vêm sendo explorados desde a década de 1980. O seu objetivo principal é proporcionar ao desenvolvedor de software a possibilidade de atuação em um nível mais elevado de abstração do que o proporcionado pelas linguagens de programação e plataformas de desenvolvimento existentes. Isso se dá pela redução da distância semântica entre o problema a ser resolvido e a implementação/solução através de modelos de alto nível que protegem e procuram isolar o desenvolvedor das complexidades inerentes às plataformas usuais de implementação (FRANCE et al., 2007).

O DSDM funciona com base em modelagem e transformação de software. A modelagem normalmente é feita com o apoio de uma linguagem, que pode ser a UML (FRANCE et al., 2006), ou uma DSL, que é uma linguagem projetada especificamente para um determinado propósito. Essa última é mais indicada para uso no DSDM, pois possui maior poder expressivo e foco no domínio de interesse (HOYOS; GARCÍA-MOLINA; BOTÍA, 2013). A definição de uma sintaxe abstrata (meta modelo) é a primeira etapa para se especificar uma DSL, seguida da definição de uma sintaxe concreta, permitindo ao engenheiro de software instanciar os modelos de acordo com a DSL (KOLOVOS et al., 2009).

Os modelos servem de entrada para uma cadeia de transformação de software, responsável por produzir código executável. Essa cadeia é composta por transformadores modelo-para-modelo (M2M - *Model-to-Model*) e modelo-para-texto (M2T *Model-to-Text*). As transformações M2M geralmente introduzem refinamentos e modificações nos modelos, por exemplo, adicionando detalhes da plataforma, enquanto as transformações M2T produzem código. Recomenda-se que os modelos intermediários produzidos ao longo dessa cadeia não sejam modificados manualmente, para possibilitar a automação do processo (VOELTER, 2009).

Entre as vantagens do DSDM estão a portabilidade e o aumento do nível de abstração para o desenvolvimento do software. A portabilidade é uma importante característica do DSDM devido ao uso de modelos independentes de plataformas que podem ser reu-

tilizados em diferentes plataformas por meio de geradores de códigos especializados. O aumento no nível de abstração também é importante, pois permite que o desenvolvedor trabalhe com as tarefas conceituais mais importantes do domínio do problema, ao passo que tarefas repetitivas são realizadas de forma automática.

Além disso, os modelos são utilizados como documentação, retirando do programador o complexo trabalho de criar e manter esses importantes artefatos para a evolução do software (ANNEKE KLEPPE, JOS WARMER, 2004). A manutenção do software é realizada diretamente nos modelos, passando a fazer parte do software e não utilizados apenas em um primeiro momento, como o caso da modelagem via UML.

Dessa forma, a modelagem permite o melhor envolvimento dos *stakeholder* no projeto através de um vocabulário unificado. A geração de código através da concepção dos modelos podem ser ferramentas úteis no auxílio do desenvolvimento multiplataforma, uma vez que é possível a geração e manutenção da maior parte do código para software destinados a computadores pessoais e dispositivos móveis. A Figura 10 apresenta a combinação dos principais elementos necessários para o uso do DSDM. Nessa Figura é possível visualizar o desenvolvedor interagindo com as linguagens de modelagem (DSL), visando expressar os conceitos do domínio em um nível mais alto de abstração. Uma vez realizado a modelagem, os geradores criam o código-fonte e/ou outros modelos necessários para conceber o sistema alvo.

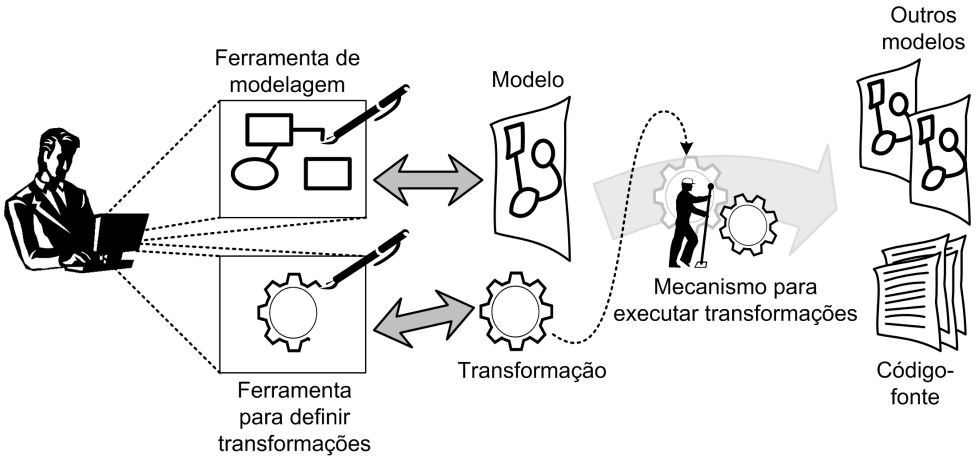

Figura 10 – Principais elementos do DSDM (DANIEL LUCRÉDIO, 2009).

Ferramenta de modelagem: usada para auxiliar engenheiros e desenvolvedores, através de editores gráficos ou textuais, a modelagem de requisitos, arquiteturas, estruturas de dados, comportamentos e outras características do sistema (DEURSEN; VISSER; WARMER, 2007). Os modelos criados por essa ferramenta precisam seguir regras de semântica, uma vez que são interpretados pelo computador e normalmente é utilizada uma DSL.

Ferramenta para definir transformações: usadas para transformações M2M e M2T. A transformação M2M corresponde a geração de outros a partir de modelos recebidos como entrada. Através delas são construídas as regras de mapeamento. Transformações M2T são usadas para transformar os modelos em código fonte, automatizando o processo de desenvolvimento.

Modelos: servem de entrada para transformações que irão gerar outros modelos ou o código-fonte.

Mecanismos para executar transformação: contém regras de mapeamento definidas pelo engenheiro de software, responsáveis em aplicar as transformações, visando manter informações de rastreabilidade, assim possibilitando saber a origem de cada elemento gerado, seja no modelo ou código-fonte.

Outros modelos e código-fonte: resultam do processo de transformação.

Com intuito de abstrair da melhor maneira possível linguagens e modelos, e possibilitar a definição e execução de transformações genéricas, as principais abordagens existentes na indústria para DSDM se baseiam no conceito de metamodelagem (OMG, 2003), apresentado na Figura 11.

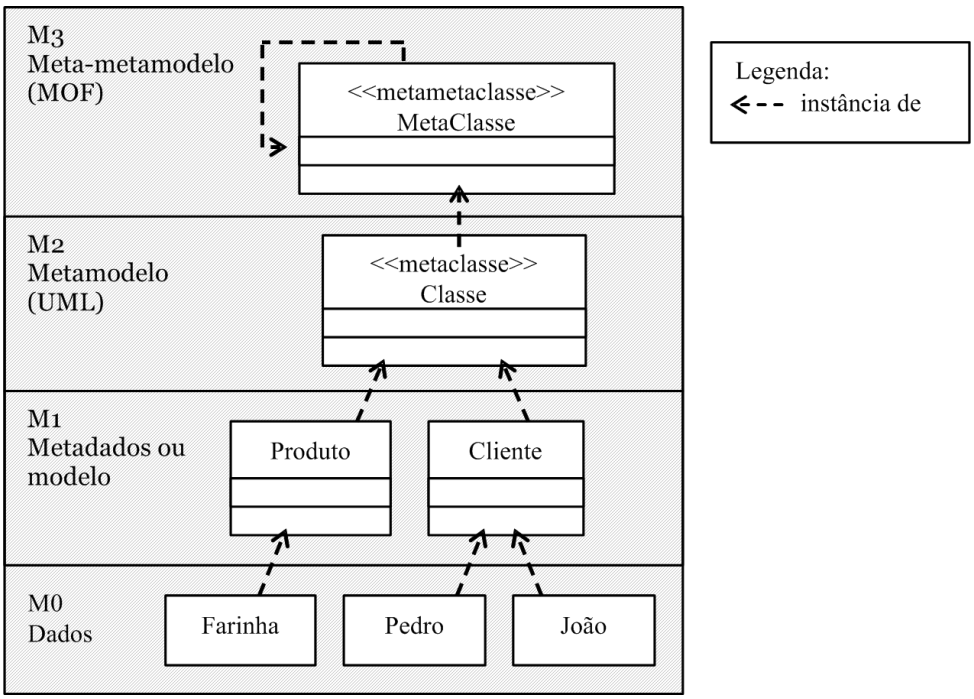

Figura 11 – Arquitetura clássica de metamodelagem (DANIEL LUCRÉDIO, 2009).

O primeiro nível (M0) corresponde aos dados. A seguir, o segundo nível (M1) eleva o nível de abstração, sendo responsável pelos metadados e modelos. São dados

que descrevem os dados. Os modelos definidos através da UML[1] são um exemplo desse nível. Um nível acima (M2) temos o que chamamos de metamodelo, que é utilizado para especificação de modelos. A especificação UML é um exemplo de metamodelo. Por fim, o último nível (M3) é a camada de meta-metamodelo. Essa camada é utilizada para definir linguagens de modelagem (como a UML). Normalmente, um meta-metamodelo é uma instância de si próprio.

4.1 Abordagens da indústria para o DSDM

Uma das principais linguagens que dão suporte ao DSDM é o MOF (*Meta Object Facility*), um padrão mantido pelo OMG (*Object Managment Group*) (OMG, 2003). O MOF é um metametamodelo baseado em uma simplificação de recursos da linguagem da modelagem UML (*Unified Modeling Language*) (OMG, 2005). O MOF permite a criação de metamodelos com interoperabilidade e portabilidade para atender à necessidade de se projetar uma DSL orientada a objetos. A representação dos modelos baseados no MOF pode ser feita via XMI (*XML Metadata Interchange*) ou JMI (*Java Metadata Interface*), mapeando os modelos para XML e Java, respectivamente (DAVID FRANKEL, 2006).

Existem alguns *frameworks* que utilizam os conceitos DSDM para a criação de software. O EMF (*Eclipse Modelling Framework*) (KOLOVOS et al., 2009) é um dos mais populares *frameworks*, tendo sido influenciado pelo metamodelo MOF (GERBER; RAYMOND, 2003). Através do EMF é possível a criação e manipulação, em memória, de modelos baseados nos conceitos de diagrama de classes, permitindo a inclusão de atributos e métodos para o domínio de interesse, definindo as entidades criadas e suas relações.

O EMF serve apenas para manipulação do metamodelo. Para a criação das ferramentas que possibilitam a geração visual de diagramas e seus elementos, existe o GMF (*Graphical Modeling Framework*). Baseado no EMF, é um ambiente de modelagem visual disponível para o IDE Eclipse. A partir do metamodelo é possível instanciar uma ferramenta de modelagem, com elementos visuais e diagramas personalizados para qualquer domínio (KOLOVOS et al., 2009). O Graphiti (GRAPHITI, 2008) também serve o mesmo propósito, permitindo a criação de ferramentas de modelagem visual no Eclipse. Tanto o GMF como o Graphiti estão disponíveis como parte da plataforma Eclipse[2].

Diagramas visuais são adequados para prover uma visão hierárquica ou organizacional de algum aspecto da modelagem. Aspectos algorítmicos, especificações de invariantes ou algum tipo de comportamento sequencial, são mais adequadamente expressos em linguagens textuais. Para isso, existe o Xtext (XTEXT, 2006), que é um ambiente de modelagem textual. Também baseado em EMF, está disponível como parte da plataforma

[1] UML - Unified Modeling Language - www.uml.org/
[2] www.eclipse.org/modeling/gmp

Eclipse[3].

Seja em forma visual ou textual, a criação de DSLs torna possível aumentar o nível de abstração e criar modelos que podem ser acoplados a geradores de códigos. Nesse sentido, pode-se citar o Xpand (XPAND, 2008), *Acceleo* (ACCELEO, 2010) e Xtend[4] como ferramentas disponíveis, que são geradores baseados em *templates*, servindo ao propósito de completar o processo de geração de código sem a necessidade de criar essa infraestrutura a partir do início.

Fora do ambiente Eclipse, outra ferramenta para utilização dos conceitos DSDM é o Visual Studio da Microsoft, que possui o DSL Toolkit. Essa ferramenta pode ser utilizada para construir modelos visuais para o problema do domínio (COOK et al., 2007). O DSL Toolkit também pode gerar códigos a partir dos modelos criados e também gerenciar as atualizações. Normalmente é utilizada a geração de códigos em C#, mas também é possível a geração para outras linguagens como *Visual Basic*.

Existem outras ferramentas que oferecem suporte ao DSDM em diferentes plataformas. Em todas elas, o ponto em comum é a base no conceito de metamodelagem para aumentar o nível de abstração dos domínios de interesse (LATIF et al., 2016).

[3] eclipse.org/Xtext
[4] eclipse.org/xtend/

5 Uma abordagem holística para o desenvolvimento de software multiplataforma

A tese aqui defendida é a de que é possível adotar uma solução multiplataforma que aborda de maneira única os diferentes desafios do desenvolvimento multiplataforma, ao invés de abordar separadamente os problemas específicos, como a geração de diferentes versões para um mesmo software, ou a interface com o usuário. Defende-se, portanto, uma visão mais holística do problema, na qual seja possível propor uma solução que seja capaz de efetivamente tratar o desenvolvimento multiplataforma como um esforço único e centralizado.

Como detalhado no capítulo anterior, existe uma série de ferramentas comerciais disponíveis no mercado para o desenvolvimento multiplataforma, conforme apresentado na Seção 2.2. Porém nenhuma delas resolve a essência do problema, que é permitir todo o desenvolvimento multiplataforma em um único ambiente sem manter o desenvolvedor dependente das liberações dos fabricantes e permitindo a distribuição de funcionalidades entre os dispositivos. Foi também demonstrado no capítulo anterior que vários trabalhos acadêmicos têm mostrado resultados satisfatórios com a utilização do conceito DSDM, mas ainda existem lacunas não cobertas pela comunidade científica. Essas lacunas são exploradas neste trabalho de uma forma diferente das encontradas na literatura, conforme discutido a seguir.

5.1 Originalidade da proposta

Frente às soluções investigadas, percebe-se que o volume de trabalhos que consideram todas as plataformas atuais é reduzido, ainda mais no meio acadêmico. Retomando a caracterização do problema na Seção 2.1, apenas as soluções comerciais poderiam de fato suprir parte das questões levantadas, podendo citar: IBM *Rational Rhapsody*, a GENEXUS e as linguagens HAXE e KOTLIN. Porém, essas abordagens esbarram em questões comuns à maioria das ferramentas comerciais, assim como:

- Não é possível estender o metamodelo para a inclusão de novas funcionalidades por se tratar de caixa preta, limitando o comportamento do software e a criatividade da equipe de desenvolvimento;
- Não é possível estender a ferramenta visando considerar outras plataformas atuais e futuras ou até mesmo evoluções da tecnologia atual;

- O desenvolvedor fica dependente da plataforma de desenvolvimento. Isto é, todo o desenvolvimento e manutenção devem ser realizados na ferramenta, correndo o risco de ela ser descontinuada, prejudicando todo o projeto;
- Não há recurso para a distribuição de funcionalidades entre as plataformas e a troca de plataformas usadas no sistema de forma facilitada; e
- Em muitos casos não existe a preocupação com a legibilidade do código fonte gerado, dificultando a sua extensão e aproveitamento em caso de descontinuidade da ferramenta.

Na academia também existem soluções que abordam o problema. À abordagem acadêmica mais próxima de solucionar as questões retratadas na Seção 2.1, é o trabalho de Miravet et al. (2014), uma vez que os autores consideram o desenvolvimento nativo para plataformas móveis, o desenvolvimento web e a comunicação entre as plataformas. Porém, a abordagem mantém um conjunto de elementos previamente criados para a modelagem em XML. Esses elementos são compilados por transformadores próprios, o que dificulta a extensão da abordagem, além de não prever a inclusão de outras plataformas atuais e futuras.

A originalidade desta tese, que portanto a diferencia das soluções comerciais e acadêmicas existentes, está em abordar esses problemas ao mesmo tempo, por meio de uma solução capaz de resolvê-los de maneira unificada. Para isso, a abordagem busca unir os aspectos (chamados também de contribuições - **C**) menos explorados no desenvolvimento multiplataforma (Capítulo 3), sem no entanto desprezar os avanços e resultados positivos já obtidos, ainda que por poucos trabalhos. A abordagem portanto aborda, de maneira holística:

- **C1** - Modelagem dos conceitos de domínio em alto nível de abstração e em um único ambiente. Esse último aspecto é considerado por uma pequena parte dos trabalhos pesquisados, diferente do aumento do nível de abstração que é bastante usado;
- **C2** - Aproveitamento dos conceitos similares entre as plataformas na modelagem. Esse é um aspecto bastante presente nos trabalhos relacionados e ferramentas comerciais;
- **C3** - Redução do custo na configuração de um sistema multiplataforma. Isso inclui distribuir as funcionalidades entre plataformas e alternar as plataformas que estão sendo usadas no sistema. Essa característica não existe nas ferramentas existentes;
- **C4** - Inclusão de novas plataformas na abordagem com menor impacto no código. As soluções existentes normalmente são restritas a um conjunto predeterminado de plataformas;

- **C5** - Extensão da abordagem pelos seus usuários finais (desenvolvedores) através de recursos projetados para esse fim. As soluções existentes não são facilmente extensíveis pelo usuário final;
- **C6** - Reutilização de códigos entre domínios. Normalmente as ferramentas existentes possuem bom suporte para vários domínios; e
- **C7** - Redução do custo de desenvolvimento e manutenção. Em geral, isso é de fato alcançado por meio das soluções existentes, e nesta abordagem isso não pode ser diferente.

A Tabela 5 apresenta uma comparação da abordagem proposta com as principais ferramentas multiplataforma disponíveis na academia e indústria. As primeiras 6 ferramentas são industriais e as três últimas são acadêmicas, incluindo a abordagem proposta. Nessa tabela é possível verificar de forma facilitada as principais originalidades proposta nesta tese.

Tabela 5 – Comparativo da abordagem proposta com as principais ferramentas acadêmicas e industriais, apontando os pontos fortes e fracos assim como as contribuições **C3, C4 e C5**, originais para esta tese.

Abordagens	**Pontos Positivos**	**Pontos Negativos**	**C3**	**C4**	**C5**
Flutter	-Linguagem amigável; -Interface rica; -Muitos recursos; -Versão web Beta; -Documentação completa; -Desempenho; -Único ambiente; -Redução do custo de desenvolvimento e manutenção.	-Não gera código legível; -Dependência do fabricante; -Não aumenta a abstração;	Não	Não	Sim
Xamarin	-Linguagem amigável; -Muitos recursos; -Documentação boa; -Único ambiente; -Redução do custo de desenvolvimento e manutenção.	-Não gera código legível; -Dependência do fabricante; -Desempenho; -Não aumenta a abstração.	Não	Não	Não

Ionic	-Linguagem web; -Gera códigos nativos; -Único ambiente; -Redução do custo de desenvolvimento e manutenção.	-Não gera código legível; -Dependência do fabricante; -Desempenho; -Não aumenta a abstração;	Não	Não	Não
React Native	-Linguagem web; -Gera código legível; -Muitos recursos; -Documentação boa; -Único ambiente; -Redução do custo de desenvolvimento e manutenção.	-Dependência do fabricante; -Desempenho; -Não aumenta a abstração.	Não	Não	Não
Kotlin	-Linguagem amigável; -Desempenho; -Único ambiente; -Redução do custo de desenvolvimento e manutenção.	-Não gera código legível; -Dependência do fabricante; -Interface não considerada; -Não aumenta a abstração.	Não	Não	Não
Genexus	-Geração para muitas plataformas; -Linguagem visual e textual; -Único ambiente; -Redução do custo de desenvolvimento e manutenção; -Aumenta a abstração.	-Não gera código legível; -Dependência do fabricante; -Interface padronizada; -Desempenho; -Recursos limitados; -Não aproveitamento de código entre as plataformas .	Não	Não	Não
MD2	-Gera código legível; -Desempenho; -Único ambiente; -Redução do custo de desenvolvimento e manutenção; -Aumenta a abstração.	-Nova linguagem -Dependência do fabricante; -Poucos recursos; -Interface padronizada; -Pouca documentação;	Não	Não	Não

Miravet et al. (2014)	-Utiliza XML; -Gera código legível; -Geração para muitas plataformas; -Desempenho; -Único ambiente; -Redução do custo de desenvolvimento e manutenção; -Aumenta a abstração.	-Dependência do fabricante; -Poucos recursos; -Interface padronizada; -Pouca documentação.	Não	Não	Não
Abordagem Proposta	-Geração para muitas plataformas; -Desempenho; -Gera código legível; -Independência do fabricante; -Único ambiente; -Redução do custo de desenvolvimento e manutenção; -Aumenta a abstração.	-Nova linguagem; -Interface não suportada; -Pouca documentação.	Sim	Sim	Sim

Existe uma ressalva na Tabela 5, onde é possível verificar a ferramenta Flutter apresentando suporte para a **C5**, referente a extensão. De fato, essa ferramenta, oferece a possibilidade de utilizar IDEs nativas para realizar a inclusão de funções não previstas pelo fabricante. Porém, existe uma limitação para a inclusão de novas funções, para tal é necessário possuir APIs disponíveis nas versões Android e iOS, (encontrou dificuldades) não suportando a inclusão de novas funções para a versão web (Conforme documentação oficial[1]). Além disso, algumas funções disponíveis para Android e iOS não são suportadas, conforme relato de um especialista na avaliação por especialistas, apresentada no Capítulo 8.

Para comprovar esta tese, foi realizado o planejamento, concepção e testes de uma abordagem para o desenvolvimento multiplataforma que apoia o desenvolvimento de sistemas que precisam ser executados em diferentes dispositivos e que permite o aproveitamento e a distribuição dos conceitos entre as diferentes plataformas através de uma linguagem única de modelagem. Os desafios principais da tese são a redução de custos e o aumento da produtividade e manutenção no cenário multiplataforma (conforme descritos

[1] https://flutter.dev/docs/development/platform-integration/platform-channels?tab=ios-channel-swift-tab

no Capítulo 2) através de uma abordagem que permita a sua extensão pelos desenvolvedores.

5.2 A Abordagem

A abordagem empresta as principais ideias do DSDM (FRANCE et al., 2007), que consiste em combinar programação generativa, técnicas de transformação de software e linguagens específicas de domínio (DSL). Mas apenas o uso do DSDM para o desenvolvimento multiplataforma já foi proposto e testado com sucesso por outros, conforme discutido no Capítulo 3. A principal diferença na forma de DSDM adotada nesta pesquisa é como os detalhes específicos de plataforma são levados em consideração. Uma comparação entre uma típica solução para o desenvolvimento multiplataforma e a abordagem proposta nesta pesquisa pode ser vista na Figura 12.

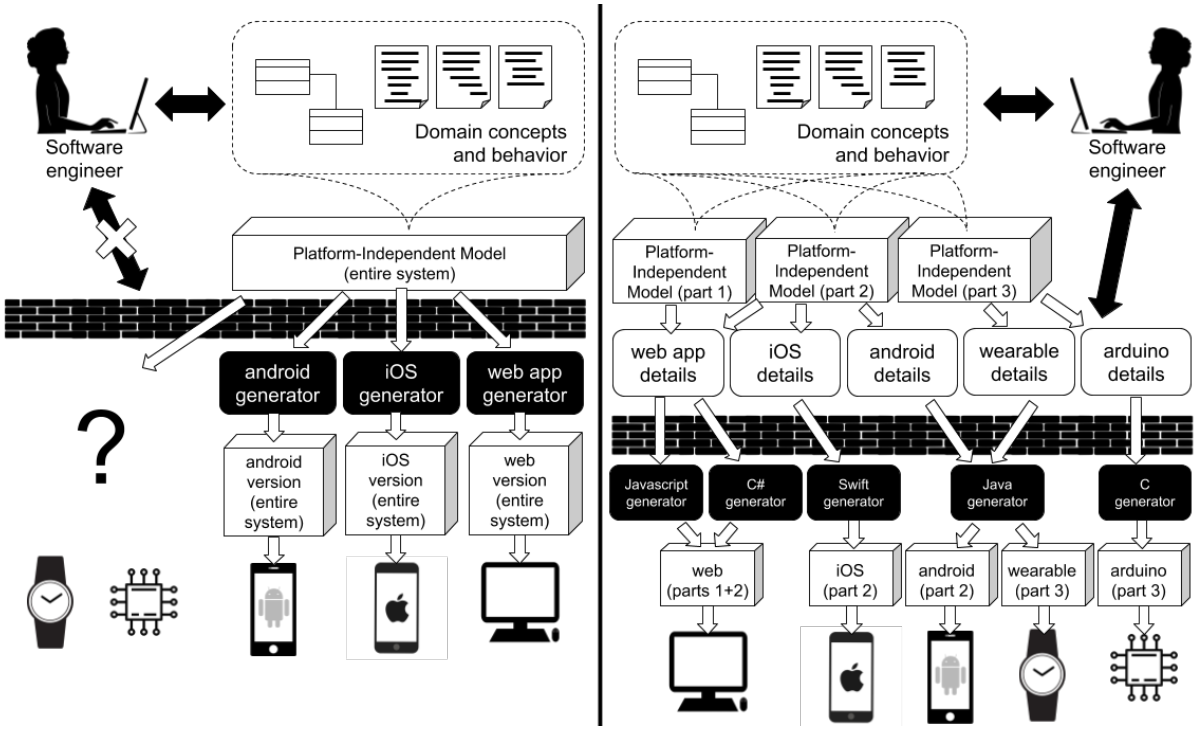

Figura 12 – Uma visão geral de uma típica solução multiplataforma baseada em DSDM (esquerda) em comparação com a abordagem proposta nesta pesquisa (direita).

O lado esquerdo da Figura 12 mostra uma típica solução multiplataforma baseada em DSDM. O engenheiro de software trabalha em um nível de abstração mais elevado, para produzir um modelo independente de plataforma. Ela é protegida dos detalhes das plataformas por meio de geradores e transformadores específicos de plataforma. Esses artefatos são fornecidos ao desenvolvedor como um componente caixa preta, para serem usados como estão ou com algumas parametrizações. Como resultado, o desenvolvedor pode se concentrar nos conceitos do domínio, conforme pretendido pelo DSDM. Mas não

há flexibilidade para modificar os detalhes das plataformas, como incluir novas funcionalidades e/ou recursos ou incluir novas plataformas. A única solução seria modificar os geradores ou transformadores de código, mas como parte deles foram projetados para serem usados como componentes caixa preta, essa é uma tarefa muito difícil. Para as abordagens caixa branca, embora possível, a sua modificação não é trivial. Além disso, na maioria das soluções existentes, os geradores de código produzem versões equivalentes do mesmo software. Isso significa que todas as funções para todo o sistema são geradas para todas as plataformas igualmente.

Em contraste, a abordagem proposta adota um ponto de vista mais holístico de todo o desenvolvimento. Como mostra o lado direito da Figura 12, tanto os conceitos do domínio como os detalhes das plataformas estão sob o controle do desenvolvedor. Os únicos componentes caixa preta são os geradores de códigos para as linguagens específicas, que precisarão mudar apenas se a linguagem alvo sofrer alterações, o que provavelmente não ocorrerá com frequência. Isso permite que as plataformas sejam modificadas ou adicionadas à solução pelo desenvolvedor. A abordagem também oferece a capacidade de escolher onde cada parte do software será implantada. Por exemplo, uma área administrativa de um sistema pode ser direcionada apenas para a versão web. O *front-end* principal pode ser direcionado para as plataformas web e móvel, e a parte do sistema que lida com sensores pode ser direcionada para plataformas *wearable* e arduino. Todas essas especificações são independentes de plataforma e toda a solução é tratada como uma única entidade de software.

Isso é possível devido ao uso de uma *General-Purpose Language* (GPL) projetada especificamente para esse objetivo. Essa GPL é uma linguagem de programação textual com algumas construções de alto nível que permitem a especificação de conceitos do domínio do problema e da solução, comportamento detalhado, detalhes de plataformas para geração de código e distribuição de funcionalidades. Portanto, se for necessária uma maneira diferente de gerar código para uma plataforma específica, ou se uma nova plataforma precisar ser suportada, o desenvolvedor poderá usar a GPL, em vez de lidar com geradores de código ou modificar o código gerado.

A Figura 13 mostra o que precisa ser especificado pelo engenheiro de software ao criar um sistema multiplataforma usando a abordagem proposta. Existem três modelos básicos: o modelo do sistema (*System model* - à esquerda), o modelo de plataforma (*Platform Model* - à direita) e o modelo de *deploy* (*deployment model* - ao meio). Eles são especificados usando a linguagem GPL definida para esta abordagem. Cada modelo da abordagem é responsável pela especificação de uma parte do software e pode ser reutilizado em outros sistemas de diferentes domínios.

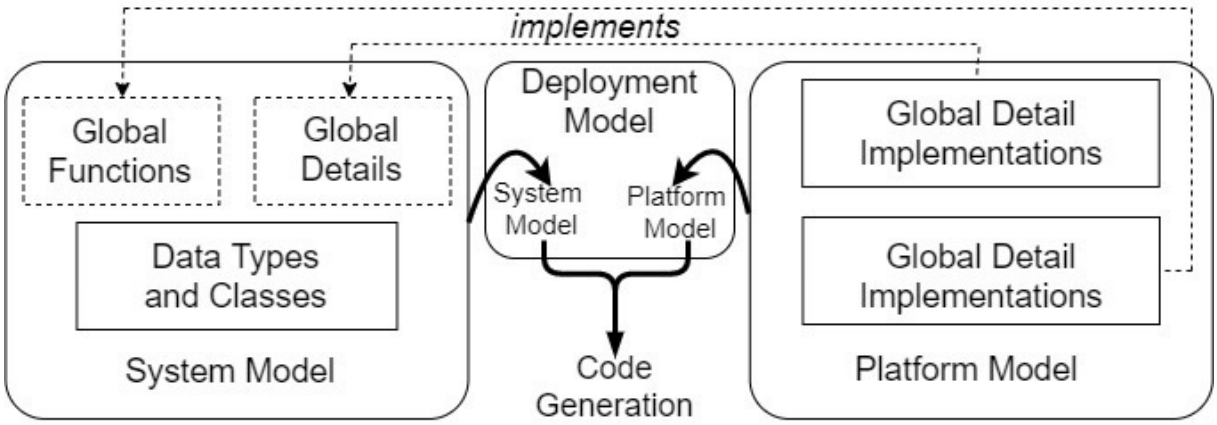

Figura 13 – Elementos da abordagem multiplataforma.

5.2.1 O modelo do sistema

O modelo do sistema é composto por três partes principais: tipos e classes de dados (*Data Types e Classes*), funções globais (*Global Functions*) e detalhes globais (*Global Details*) .

O primeiro passo para o seu uso é a especificação dos tipos de dados. Esta especificação pode ser feita em um arquivo dedicado e importado em todos os novos projetos. A Listagem 5.1 apresenta a declaração de alguns tipos utilizando a GPL proposta.

```
1  datatype int
2  datatype string
3  datatype float
4  datatype double
5  datatype bool
6  datatype dateTime
7  datatype void
8  datatype list<E>
9  datatype linkedHashMap<E, T>
10 datatype img
11 datatype Task
```

Listagem 5.1 – Declaração de tipos de dados utilizando a linguagem da abordagem.

O segundo passo se dá com a especificação das classes e a definição de funções e regras de negócio na GPL. Cada classe tem atributos e operações (funções), especificadas na GPL, independente de plataforma. O trecho da GPL apresentado na Listagem 5.2 apresenta a especificação das classes `DBConf` (linhas 1 a 8) e `ShoppingCartDAO` (linhas 9 a 23). A classe `DBConf` serve para o desenvolvedor configurar os dados de conexão com o banco de dados de forma genérica. A classe `ShoppingCartDAO` representa o acesso aos dados de um carrinho de compras, em um sistema de *e-commerce*, onde é definida uma

função `addProduct` (linhas 14-22), usada para inserir um produto em um carrinho de compras. Essas são construções regulares orientadas a objetos para especificar a estrutura e o comportamento de um sistema.

```
1  class DBConf {
2      ecommerce: string         //database name
3      db_rest_user: string      //user id
4      restCommerce2019: string //password
5      MYSQL5011: string         //Server
6      Smarterasp: string        //Server
7      net: string               //Server
8  }
9  class ShoppingCartDAO {
10     usesGlobalDetails <DBConf> DBConnection
11     usesGlobal SelectCustomer(id: string): Customer
12     usesGlobal SelectProduct(id: string): Product
13     usesGlobal InsertProductIntoCart(customer: Customer, product:
       Product, quantity: int): string
14     operation addProduct(custId: string, prodId: string, quantity
       : int): string {
15         objCust: Customer
16         objProd: Product
17         newProd: string
18         objCust := global SelectCustomer(custId)
19         objProd := global SelectProduct(prodId)
20         newProd := global InsertProductIntoCart(objCust, objProd,
       quantity)
21         return newProd
22     }
23 }
```

Listagem 5.2 – Exemplo mostrando a declaração de classes, operações e o uso de funções globais e detalhes globais através da GPL.

Como pode ser notado, essa listagem não apresenta detalhes dependentes de plataforma. No entanto, em determinados momentos (linhas 18, 19 e 20) é feita uma consulta em um banco de dados persistente para encontrar os clientes e produtos cadastrados, e adicionar o produto no carrinho. Na abordagem proposta, esse tipo de funcionalidade precisa ser implementada em uma plataforma específica. Para evitar que essa dependência de plataforma prejudique a reutilização das classes, a abordagem prevê declarações de funções globais. Funções globais (palavra-chave *global*) podem ser definidas como algoritmos que contém códigos específicos de plataformas e podem ser adaptadas em tempo de geração de código para qualquer classe e plataforma do sistema. Esses são os elementos mais im-

portantes da abordagem, pois estabelecem o relacionamento entre modelos independentes de plataforma e os modelos de plataforma. A Listagem 5.3 apresenta a declaração de quatro funções globais, sendo que três foram utilizadas na Listagem 5.2: `SelectCustomer`, `SelectProduct` e `InsertProductIntoCart` (linhas 1, 2 e 3) e a `getGPSPosition` será abordada mais à frente. A declaração consiste em um protótipo de função, que contém todos os parâmetros e tipo de retorno necessários para utilização.

```
1 global SelectCustomer(id: string): Customer
2 global SelectProduct(id: string): Product
3 global InsertProductIntoCart(customer: Customer, product: Product
    , quantity: int): string
4 global getGPSPosition(): Coordinates
```

Listagem 5.3 – Exemplo de declaração de funções globais.

As funções globais podem ser utilizadas em qualquer classe sem que esta necessite conhecer sua implementação, como por exemplo acontece na Listagem 5.2. Para isso, basta que seja definido que aquela classe utiliza uma função global, por meio da palavra-chave `usesGlobal` (linhas 11-13 da Listagem 5.2). Considere, por exemplo, uma função para obter a localização do dispositivo. Cada plataforma (Android, iOS, web etc.) terá uma implementação diferente e apresentará suas próprias particularidades e, possivelmente, uma assinatura de função diferente (parâmetros, tipos de retorno, entre outros detalhes). Mas, em essência, eles podem ser representados como uma função, chamada "getLocation()" que retorna um par de coordenadas (latitude e longitude). Na abordagem, é exatamente isso que é uma função global: uma abstração para uma função necessária ao sistema, tem a mesma assinatura (ou muito semelhante), mas requer implementações específicas da plataforma.

Ainda na Listagem 5.3, é possível ver que a abordagem suporta funções globais entre domínios, como `getGPSPosition` (linha 4), para obter a posição de um dispositivo, por exemplo. Esse tipo de função pode ser usado em diferentes aplicativos, de diferentes domínios. A abordagem também oferece suporte a funções globais específicas do domínio, como `InsertProductIntoCart`, `SelectCustomer` e `SelectProduct` (linhas 1-3). Eles são específicos para um domínio de comércio eletrônico.

Uma função global deve ser suportada por diferentes plataformas. Isso é possível desde que as plataformas possam implementar o protótipo da função declarada. Por exemplo, a maioria dos *smartphones* são capazes de obter suas posições via satélites através do GPS, portanto, provavelmente será capaz de fornecer uma implementação compatível para a função `getGPSPosition`. Mas um computador de mesa ou portátil não possui um receptor GPS, portanto, não poderá implementar esta função[2]. Como discutido anterior-

[2] Estes computadores possuem sistemas de localização, mas normalmente dependem de antenas fixas. Portanto, podem não funcionar em uma fazenda ou o oceano, por exemplo.

mente, a abordagem não pressupõe que todas as plataformas executem versões idênticas de todo o sistema. Dessa forma, nem todas as funções globais serão implementadas por todas as plataformas. Em vez disso, a abordagem pressupõe que o sistema exige que essas funções sejam implementadas por pelo menos uma plataforma. Caso contrário, deve-se questionar a existência dessa função global.

A terceira parte do modelo de sistema são os detalhes globais. Essas são declarações genéricas que representam códigos específicos das plataformas (campos, métodos, anotações, classes aninhadas, enumerações, entre outros trechos de código) que só fazem sentido em uma plataforma específica e, portanto, não podem ser generalizadas como uma função global. Por exemplo, embora a função para obter uma posição GPS possa ser generalizada como global, ela requer um código de inicialização exclusivo para cada plataforma e que pode nem existir em determinada plataforma. Este código, portanto, não pode ser generalizado como funções globais. Outros exemplos incluem inicializar uma conexão com o banco de dados ou declarar o fluxo de controle em um estilo arquitetural MVC.

A declaração dos detalhes globais é muito simples, como mostra a Listagem 5.4. Na listagem, um tipo genérico `<E>` é especificado como parte desta declaração específica, para permitir uma personalização adicional, conforme descrito posteriormente. Um exemplo de sua implementação será explicada mais adiante.

```
1 globalDetails <E> DBConnection
```

Listagem 5.4 – Exemplo de declaração de um detalhe global.

Funções globais e detalhes globais podem ser usados por qualquer classe. Um exemplo é mostrado na Listagem 5.2. O uso de detalhes globais em uma classe é declarado com a palavra-chave `usesGlobalDetails` (linha 10). Nesse exemplo, esses detalhes representam o código de inicialização da conexão com o banco de dados, com os parâmetros especificados na classe `DBConf` (linhas 1-8). O uso de uma função global em uma classe é declarado com a palavra-chave `usesGlobal`(linhas 11-13). Para melhorar a legibilidade, a declaração completa da função global deve ser repetida na classe. Em seguida, essas funções podem ser chamadas normalmente (linhas 19-21). A única diferença de uma chamada de método normal é a palavra-chave `global`, que indica que essa é uma chamada de função global.

A ideia é que cada função global tenha uma implementação diferente em cada plataforma. No exemplo da Listagem 5.2, as chamadas globais nas linhas 19 a 21 podem se referir a um banco de dados SQL se a classe for implantada em uma plataforma web, ou em um banco de dados SQLite se a classe for implantada em uma plataforma para dispositivos móveis. Mas aqui no modelo do sistema, isso é intencionalmente deixado indefinido. Não importa qual plataforma executará esse código, desde que seja capaz de fornecer uma implementação compatível para essas funções.

5.2.2 O modelo de plataforma

O modelo de plataforma consiste em implementações para as funções globais e detalhes globais. Os protótipos das funções são declarados usando a sintaxe da GPL, mas a implementação real dessas funções pode ser escrita usando a GPL ou a linguagem de programação nativa da plataforma. É preferível usar a linguagem nativa, pois permite acesso mais direto a todos os recursos da plataforma.

Cada função global terá uma implementação diferente em cada plataforma. Cada plataforma é declarada por meio de um nome e uma linguagem. Como discutido anteriormente, a abordagem holística não pressupõe que todas as plataformas executem versões idênticas de todo o sistema. No exemplo da Listagem 5.5 são apresentadas quatro plataformas: web/C#, Android/Java, iOS/Swift e *Augmented Reality* (RA), uma plataforma inteiramente nova desenvolvida para esta tese, em C#. Essas plataformas contém três implementações diferentes para a função `InsertProductIntoCart`: baseada em SQL, para a plataforma web/C# (linhas 2-5), baseada em *Hashmap*, para a plataforma Android/-Java (linhas 8-11) e baseada em *SQLite*, para a plataforma iOS/Swift (linhas 14-17). É importante observar que a abordagem oferece ao engenheiro de software a flexibilidade de fornecer implementações personalizadas para cada plataforma.

A função `InsertProductIntoCart` não é suportada pela plataforma *Augmented Reality*, portanto não apresenta sua implementação, e sim de outra função global, `GetDiskImagesAsync` (linhas 20 a 22), que é uma função para exibir imagens. Nota-se também, que essas implementações podem ser feitas diretamente na linguagem da plataforma, possibilitando assim controle total sobre a implementação.

```
1  platform Web: CSharp {
2    implementsGlobal InsertProductIntoCart(customer: Customer,
     product: Product, quantity: int): string {
3      //C# code that implements the function on the web platform,
4      //using SQL for persistence
5    }
6  }
7  platform Android: Java {
8    implementsGlobal InsertProductIntoCart(customer: Customer,
     product: Product, quantity: int): string {
9      // Java code that implements the function on the Android
     platform,
10     // using HashMap for memory persistence
11   }
12 }
13 platform iOS: Swift{
14   implementsGlobal InsertProductIntoCart(customer: Customer,
```

```
    product: Product, quantity: int): string {
15    // Swift code that implements the function on the iOS
    platform,
16    // using SQLite for device persistence
17  }
18 }
19 platform AugmentedReality: CSharp {
20    implementsGlobal GetDiskImagesAsync(picturesFolder:
    StorageFolder): list<img> {
21      //C# code that implements the function on
22      //the platform for Augmented Reality
23    }
24 }
```

Listagem 5.5 – Implementação de funções globais em diferentes plataformas.

Também para aumentar a flexibilidade, a abordagem oferece suporte para modelos do Apache Velocity[3]. Isso permite que as implementações globais usem também os recursos de metaprogramação e geração de código, tornando-os mais flexíveis e aumentando sua reutilização. Como exemplo desse recurso adicional, a Listagem 5.6 mostra uma maneira independente de domínio para declarar, implementar e usar uma função global `InsertObject`, a qual pode inserir qualquer objeto em um banco de dados SQL. Na parte da declaração (linha 2), não há nada diferente de uma declaração global normal, exceto o tipo genérico `<E>`, que será usado na implementação e uso.

Na implementação (linhas 4-22), apóstrofos triplos (’’’ nas linhas 5 e 21) são usados para delimitar o conteúdo do modelo. As tags do *Apache Velocity* (começando com # e $) são usadas para consultar o tipo genérico `<E>` em busca de seus recursos. É possível, por exemplo, percorrer os atributos de `E` (`$E.attributes` nas linhas 7 e 12) especificar os parâmetros de consulta (linha 9) e valores SQL específicos (linha 12) ou consultar o nome da entidade `E` (`$E.name` na linha 12) para gerar uma instrução SQL.

Ao usar esta função global (`InsertObject`), é necessário especificar uma classe concreta a ser usada para geração de código. Nesse exemplo (Listagem 5.6), a classe `Order` (linha 24) será o objeto que está sendo inserido. A abordagem cuida de vincular essas definições e gerar o código correto para a plataforma. O resultado é que essa função global pode ser reutilizada em diferentes domínios.

É importante destacar que o suporte à metaprogramação por meio do Velocity é um complemento à abordagem, porém não é sua principal contribuição. Seria possível implementar as mesmas funções, em múltiplas plataformas, sem esse suporte, apenas exigiria mais trabalho de codificação. No exemplo da Listagem 5.6, seria necessário criar

[3] velocity.apache.org

um método `Insert` para cada entidade do sistema.

```
1 //Declaration
2 global <E> InsertObject(e: <E>): string
3 //Implementation
4 implementsGlobal <E> InsertObject(e: <E>): string {
5 '''
6     try {
7         #foreach($f in $E.attributes)
8             #if($f.name != 'id')
9                 cmd.Parameters.AddWithValue("@${f.name}", ${f.
    name});
10             #end
11         #end
12         SQL = "insert into $E.name (#foreach($f in $E.attributes)
    #if($f.name != 'id')${f.name}#if($foreach.hasNext) , #end#end#
    end) values (#foreach($f in $E.attributes)#if($f.name != 'id')
    @${f.name}#if($foreach.hasNext) , #end#end#end)";
13         cmd.Connection = Conn;
14         cmd.CommandText = SQL;
15         cmd.ExecuteNonQuery();
16         result="Data inserted into database successfully!";
17         objCon.CloseConnection();
18     } catch (MySql.Data.MySqlClient.MySqlException ex) {
19         result = "Error " + ex.Number + " has occurred: " + ex.
    Message;
20     }
21 '''
22 }
23 // Use
24 usesGlobal <Order> InsertObject(e: Order): string
```

Listagem 5.6 – Declarando, implementando e usando uma função global independente de domínio com modelos do Apache Velocity.

Além das funções globais, outro recurso usado para a geração de código multiplataforma são os detalhes globais, semelhantes às funções globais. Como discutido anteriormente, os detalhes globais estão em um nível acima das funções globais e podem representar códigos específicos das plataformas (anotações, trechos de código, campos estáticos, etc.). Elas são especificadas nos modelos de plataformas e representam código que não pode ser generalizado como funções globais. Esses detalhes são recomendados quando uma determinada operação requer detalhes específicos de plataformas e são complexas e/ou possuem muitas diferenças entre as plataformas para se programar de forma unifi-

cada. Assim como a função global, o detalhe global também é dividido entre declaração, implementação e uso.

A Listagem 5.7 apresenta um exemplo do detalhe global `DBConnection`. Na parte da implementação (linhas 4-18), nota-se que dentro da função existem outros métodos (linhas 7-13) com detalhes específicos de conexão. Esses detalhes e métodos se diferem muito entre as plataformas, tornando o detalhe global ideal para esses casos. Essa implementação gerará três elementos: um campo para a conexão (linha 6), um método para abrir a conexão (linhas 7-13) e um método para fechar a conexão (linhas 14-16). Nesse exemplo, as tags Apache Velocity são usadas para consultar os parâmetros de conexão, como nome de usuário, senha e servidor, de uma classe do sistema (classe `DBConf` especificado na Listagem 5.2, linhas 23-30), mas outras opções de configuração seriam possíveis, como usar um arquivo externo. Na linha 21 da Listagem 5.7 é apresentada a chamada da função dentro da classe `Connection`. Nessa chamada é possível ver a classe `DBConf` sendo passada como parâmetro genérico. O exemplo da Listagem 5.2 é utilizado para as plataformas web, Android e iOS.

```
1  // Declaracao:
2  globalDetails <E> DBConnection
3  // Implementacao:
4  implementsGlobalDetails <E> DBConnection{
5    '''
6      private MySqlConnection Conn;
7      public MySqlConnection OpenConnection(){
8        #set($server=$E.attributes.get(3).name + "." + $E.
   attributes.get(4).name + "." + $E.attributes.get(5).name)
9        Conn = new MySqlConnection("server=$server;database=$E.
   attributes.get(0).name;"
10           + "uid=$E.attributes.get(1).name;pwd=$E.attributes.
   get(2).name");
11       Conn.Open();
12       return Conn;
13     }
14     public void CloseConnection(){
15       Conn.Close();
16     }
17   '''
18 }
19 // Uso:
20 class Connection{
21   usesGlobalDetails <DBConf> DBConnection
22 }
```

```
23 class DBConf {
24   ecommerce: string         //name of database
25   db_rest_user: string      //user
26   restCommerce2019: string //password
27   MYSQL5011: string         //Server
28   Smarterasp: string        //Server
29   net: string               //Server
30 }
```

Listagem 5.7 – Declaração, implementação e uso de um detalhe global.

Se uma plataforma não precisar de detalhes específicos, ela pode deixar em branco a implementação do detalhe global. Como resultado, nenhum código adicional será gerado para essa plataforma.

5.2.3 O modelo de *deploy*

O modelo de *deploy* simplesmente define quais classes serão executadas em quais plataformas permitindo a distribuição de funcionalidades e servindo como um guia para a geração de código. A Listagem 5.8 apresenta um exemplo de implantação de diferentes classes em diferentes plataformas. A única restrição é que a plataforma escolhida para a implantação de uma classe específica deve ter uma implementação para todas as funções globais e detalhes globais usados por essa classe. Essa flexibilidade de escolher qual classe será implantada em cada plataforma faz parte da abordagem holística. Nesse exemplo, a classe `ShoppingCartDAO` está sendo implantada nas plataformas Android (linha 11) e iOS (linha 12), mas não na plataforma da web.

```
1 deploy Ecommerce{
2     User: Android
3     User: iOS
4     User: Web
5     UserDAO: Android
6     UserDAO: iOS
7     UserDAO: Web
8     Connection: Android
9     Connection: iOS
10    Connection: Web
11    ShoppingCartDAO: Android
12    ShoppingCartDAO: iOS
13  }
```

Listagem 5.8 – Configuração da implantação (modelo de *deploy*).

A estrutura da abordagem permite a modelagem dos conceitos de um sistema de forma genérica e facilita a troca da configuração de um sistema multiplataforma. A troca de configuração pode ser desde a escolha das plataformas que irão rodar certas classes até a troca da forma como determinadas classes operam. Para esse último, por exemplo, poderia ser criada uma versão da plataforma Android, onde a implementação da persistência do carrinho de compras seja feita em SQLite, ao invés de armazenamento em memória (*HashMap*). Assim, seria possível trocar a forma como a classe `ShoppingCartDAO` funciona apenas realizando sua implantação nesta nova versão da plataforma.

5.2.4 Geração de código

O resultado final do processo é a geração de código que pode ser executado em cada plataforma com autonomia de escolha pelos desenvolvedores. A abordagem permite a criação de geradores para diferentes linguagens. Atualmente existem geradores para as linguagens Java, C# e Swift, portanto é possível definir plataformas baseadas nessas linguagens apenas declarando-as. Essa prática garante que plataformas distintas possam utilizar a mesma linguagem de programação, assim como o exemplo das plataformas RA e web que utilizam a linguagem CSharp.

Novos geradores podem ser adicionados à abordagem, sendo necessário para isso implementar um mapeamento entre as construções da linguagem genérica e as construções da linguagem alvo. Apesar de não ser uma tarefa complexa, não se trata de uma atividade projetada para ser realizada pelos usuários finais da abordagem, sendo considerada caixa preta e mais apropriada para os desenvolvedores da abordagem ou pessoas interessadas em contribuir com o projeto. Esse não é o caso das extensões nos modelos de plataformas, que, conforme discutido anteriormente, foram projetadas para serem editadas pelos desenvolvedores. A GPL e os geradores de código foram implementados usando o Eclipse.

Para a produção de novos geradores é recomendável tomar como base algum gerador em produção e modificá-lo para que passe a gerar código para a nova linguagem alvo. O gerador em produção já possui todas as regras de mapeamento para a GPL, portanto o trabalho é facilitado desa maneira. Para esta tese, os geradores tem em média 1100 linhas de códigos e sua redefinição pode ser feita em poucos dias (por volta de 2 ou 3 dias), caso se conheça as regras de sintaxe da nova linguagem.

A Figura 14 apresenta a interação das pessoas envolvidas no sistema na abordagem proposta e o uso dos seus principais elementos. Nessa figura é possível visualizar três personagens, o especialista em DSDM, o engenheiro de software e o especialista de domínio. O especialista em DSDM é responsável em criar e manter toda a abordagem. Para isso foi necessário a escrita de uma gramática em XText (BETTINI, 2016) responsável em compor os principais elementos da GPL. Essa gramática é genérica o suficiente para considerar os geradores disponíveis na abordagem, a inclusão de novos geradores

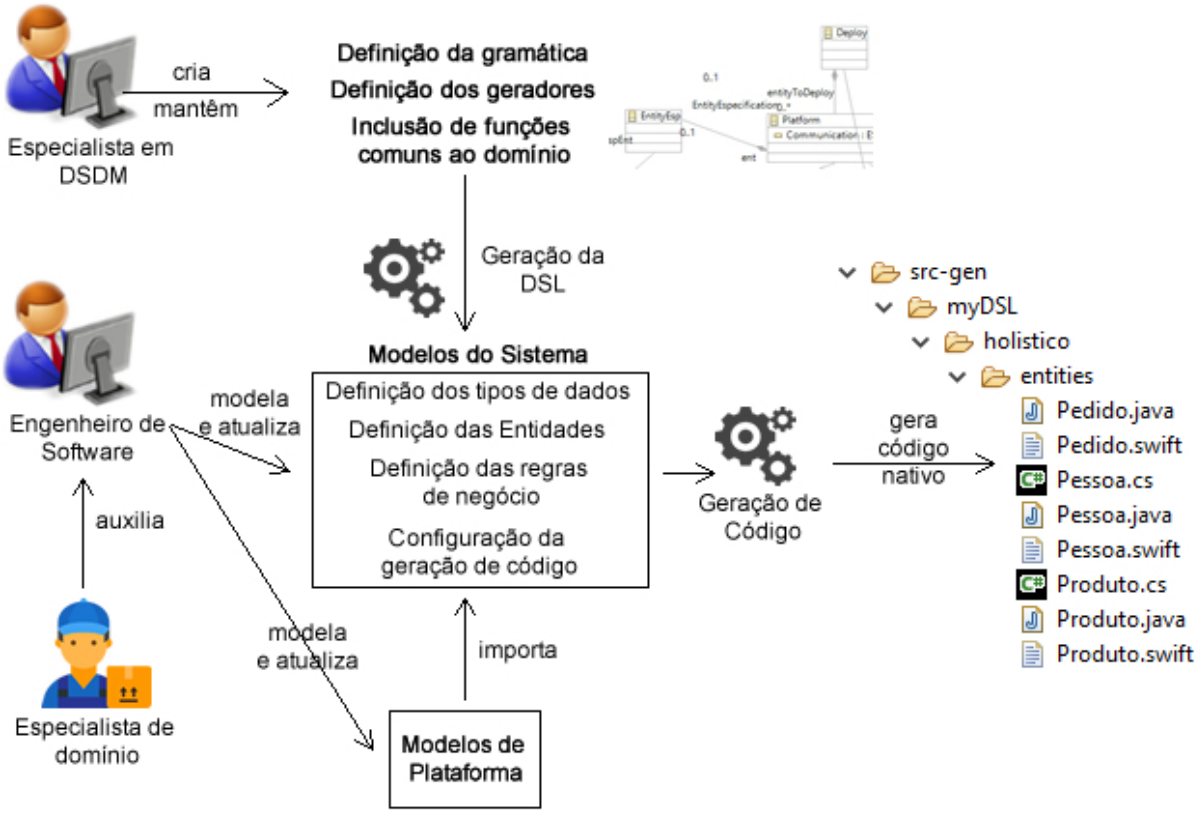

Figura 14 – Modelos de uso da abordagem proposta e seus principais elementos.

é de competência preferencialmente do especialista em DSDM. A Listagem 5.9 apresenta um trecho dessa gramática, destacando a definição dos elementos da abordagem (linha 20), podendo citar a declaração dos tipos primitivos, as entidades, o modelo de *deploy*, os modelos de plataformas, detalhes globais e funções globais. A especificação do modelo de plataforma é definida nas linhas 23 e 24, com a seleção das linguagens consideradas pela abordagem. A gramática completa da GPL encontra-se no Apêndice A. O projeto completo da abordagem incluindo gramática e geradores está disponível em https://github.com/JulianoZ/AbordagemDSDM.

Ainda na Figura 14, o engenheiro de software é responsável por modelar os conceitos do domínio e estender a abordagem com o uso dos modelos de plataformas. Já o especialista de domínio orienta o engenheiro de software nos conceitos pertinentes ao domínio abordado. É importante destacar que os modelos de plataformas podem ser especificados em arquivos separados e importados em novos projetos. Isso auxilia no reúso dos modelos de plataformas e a flexibilidade da abordagem, permitindo sua extensão e a inclusão de novas funcionalidades através da GPL.

```
1 Domainmodel :
2     (elements+=AbstractElement)*;
3
```

```
4  AbstractElement:
5     PackageDeclaration | Type | Import;
6
7  PackageDeclaration:
8      'package' name=QualifiedName '{'
9      (elements+=AbstractElement)*
10     '}';
11 QualifiedName:
12     ID ('.' ID)*;
13
14 Import:
15     'import' importedNamespace=QualifiedNameWithWildcard;
16
17 QualifiedNameWithWildcard:
18      QualifiedName '.*'?;
19
20 Type:
21     DataType | Entity | Deploy | PlatformDecl | GlobalEntity |
       Global;
22
23 PlatformDecl:
24   'platform' name=ID ':' Base=('Java' | 'Swift' | 'CPlusPlus' | '
     CSharp') '{'
25     cmdList+=CommandPlat*
26   '}';
```

Listagem 5.9 – Trecho da gramática utiliza na composição das GPLs".

Destaca-se o fato de que geradores de códigos convencionais possuem o potencial de automatizar grande parte do desenvolvimento, aumentando assim a produtividade. Além desse aspecto, a abordagem proposta aproveita os conceitos similares usados entre as diferentes plataformas para aumentar a produtividade, auxiliando o desenvolvimento de sistemas multiplataforma.

A arquitetura da abordagem multiplataforma permite também a criação de padrões para automação de rotinas através das funções globais, principalmente os detalhes globais. Esses padrões podem ser especificados pelo próprio desenvolvedor para generalizar rotinas para diferentes plataformas, como o caso do detalhe global `DBConnection`. Com esse recurso da abordagem é possível, por exemplo, criar conexões com banco de dados diferentes para cada plataforma e manter o modelo de sistema independente de plataformas, respeitando as especificidades de cada plataforma no processo de geração de código. Os parâmetros genéricos contidos nessa função (`DBConf`) ajudam a configuração para diferentes situações. Como exemplo, essa classe permite a conexão a um BD SQL

para a plataforma web e SQLite para as plataformas móveis.

Em resumo, o objetivo da abordagem desenvolvida é disponibilizar a maior parte das funcionalidades comuns a um domínio para as principais linguagens disponíveis no mercado. Para as funções não previstas pela abordagem, o desenvolvedor poderá incluir a codificação diretamente no modelo de plataforma ou nos modelos do sistema através da linguagem nativa de cada plataforma ou da GPL. Funções que possuem detalhes técnicos de plataformas devem ser implementadas nos modelos de plataformas, através das funções globais ou detalhes globais. Já funções com regras de negócio e/ou comportamentos genéricos para o domínio alvo devem ser especificadas nos modelos do sistema através da GPL. Isso permite o reúso das funções específicas de plataformas em sistemas de diferentes domínios e possibilita alto nível de abstração e generalidade na especificação do sistema para o domínio alvo.

A arquitetura da abordagem permite especificar a maior parte de um sistema multiplataforma utilizando um único ambiente e uma linguagem de alto nível, isolando o desenvolvedor dos detalhes técnicos inerentes a cada plataforma. Também é possível a inclusão de novas plataformas no sistema com menor impacto nos custos do projeto, através da especificação de plataformas.

6 Avaliações

Foram realizadas três avaliações com o objetivo de encontrar evidências sobre as principais contribuições da abordagem, listadas a seguir. Tais contribuições são as mesmas listadas no início do capítulo anterior.

Tabela 6 – Contribuições elencadas para esta tese.

Id	Contribuições
C1	**Modelagem dos conceitos de domínio em alto nível de abstração e em um único ambiente**. A abordagem segue os princípios do DSDM para permitir que o engenheiro de software especifique conceitos de sistema em um nível de abstração acima dos detalhes específicos da plataforma.
C2	**Aproveitamento dos conceitos similares entre as plataformas na modelagem**. Muitos conceitos, como entidades e funções persistentes, precisam ser criados repetidamente, com pequenas diferenças, para cada plataforma. A abordagem usa o DSDM para permitir que eles sejam especificados como modelos e reutilizados em todas as plataformas.
C3	**Redução do custo na configuração de um sistema multiplataforma**. Isso inclui distribuir as funcionalidades entre plataformas e alternar as plataformas que estão sendo usadas no sistema.
C4	**Inclusão de novas plataformas na abordagem com menor impacto no código**. A abordagem oferece ao engenheiro de software a capacidade de incluir novas plataformas no sistema, mesmo que elas não tenham sido inicialmente suportadas. Essa inclusão não tem um grande impacto em termos de esforço.
C5	**Extensão da abordagem através dos modelos de plataformas**. A abordagem permite que as plataformas existentes sejam estendidas ou modificadas, para melhor atender às necessidades de um sistema específico.
C6	**Reutilização de códigos entre domínios**. Os modelos de plataforma suportam conceitos e funções entre domínios a serem especificados, para que possam ser facilmente reutilizados em qualquer domínio.
C7	**Redução do custo de desenvolvimento e manutenção**. O suporte da abordagem à abstração, reutilização e geração de código tem o potencial de reduzir custos durante o ciclo de vida do desenvolvimento.

Para facilitar o entendimento e o objetivo de cada avaliação, foi utilizado a abordagem *Goal Question Metric* (GQM) (BASILI; CALDIERA; ROMBACH, 1994). Essa abordagem ajuda a organizar a análise dos resultados das avaliações e definir de forma mais consistente o seu foco.

6.1 Objetivos da avaliação 1

A primeira avaliação foi a prova de conceito da pesquisa, focando principalmente em verificar se a abordagem pode ser usada para produzir um sistema multiplataforma real. Além disso, as contribuições listadas na Tabela 6 foram analisadas para verificar quais foram atendidas pela abordagem nessa avaliação. Seguindo o padrão GQM essa avaliação possui o seguinte objetivo (**G1**):

- **Objeto (*Process*)**: O objeto é a abordagem proposta nesta tese;
- **Propósito (*Purpose*):** Criar um sistema multiplataforma real e completo, entregando as sete contribuições elencadas na Tabela 6;
- **Qualidade avaliada (*Issue*):** Nesta avaliação, buscou-se avaliar se o objeto (a abordagem) é capaz ou não de cumprir seu propósito; e
- **Ponto de vista (*Viewpoint*):** Nesta avaliação tomou-se o ponto de vista do pesquisador, pois o interesse foi determinar quais aspectos internos da construção da abordagem são responsáveis pela sua capacidade em cumprir o propósito esperado.

Partindo desse objetivo, o sistema multiplataforma foi recriado através da abordagem e sua codificação gerada foi comparada com o sistema previamente desenvolvido (*ad-hoc*), visando responder as seguintes questões de pesquisa (*Question*):

1. **Q1**. Os recursos providos pela abordagem são suficientes para a especificação de um sistema multiplataforma completo? Entre eles:

   Linguagem GPL;

   Modelos de sistema;

   Modelos de plataformas;

   Modelos de *deploy*; e

   Declaração de variáveis, funções globais e detalhes globais;

2. **Q2**. Os códigos gerados realizam as funções esperadas em suas especificações na abordagem? Seja usando a GPL ou linguagem nativa?
3. **Q3**. A abordagem simplifica o desenvolvimento e aumenta a produtividade? e
4. **Q4**. A abordagem consegue entregar as contribuições (Tabela 6) elencadas para esta tese?

Para analisar as questões de pesquisa foram definidas métricas (*Metric* - **M**), visando apresentar resultados mais consistentes para cada questão.

- **M1**. Lista de funções que não puderam ser especificadas pela abordagem. Em caso afirmativo, os motivos pelos quais as funções não funcionaram. Resultados da execução dos códigos gerados;
- **M2**. Lista das codificações de funções que foram especificadas e geradas através da abordagem e não foram executadas com sucesso nos dispositivos reais e de acordo com o esperado;
- **M3**. Diferença entre LOC nos desenvolvimento *ad-hoc* e através do uso da abordagem. Lista de conceitos que puderam ser aproveitados entre as plataformas; e
- **M4**. Lista de contribuições que foram entregues pela abordagem. Lista dos aspectos que confirmaram a entrega de cada contribuição.

A Figura 15 apresenta a relação entre o objetivo, questões e métricas definidas para a avaliação 1 conforme a abordagem GQM.

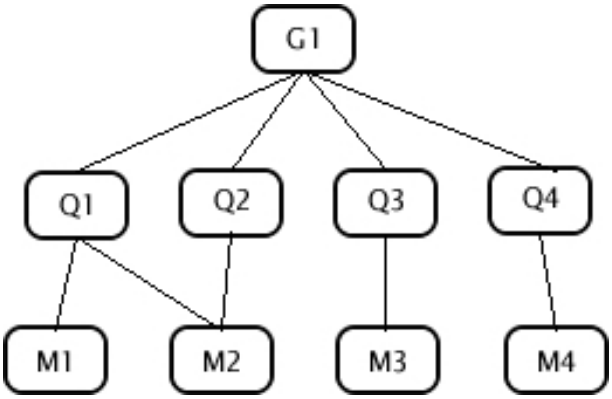

Figura 15 – Relação entre os objetivos, questões e métricas da abordagem GQM para a avaliação 1.

6.2 Objetivos da avaliação 2

A segunda avaliação foi conduzida por cinco especialistas visando reunir opiniões especializadas sobre as contribuições elencadas para esta tese. Os especialistas utilizaram a abordagem na prática, criando funções genéricas através da GPL e gerando código para as plataformas envolvidas na prova de conceito além da inclusão de um *smartwatch*. Ao final, os códigos gerados a partir das especificações genéricas dos especialistas foram executados nos ambientes reais. Essa checagem mostrou que a maior parte dos códigos gerados foram executados com sucesso e pequenos erros apresentados foram ajustados nas especificações da GPL e testados novamente. Além disso, os especialistas participaram de uma entrevista onde foi possível mapear as contribuições da abordagem com os relatos da avaliação. Seguindo o padrão GQM essa avaliação possui o seguinte objetivo (**G1**):

- **Objeto (*Process*)**: O objeto é a abordagem proposta nesta tese;
- **Propósito (*Purpose*):** Analisar as opiniões de especialistas que utilizaram a abordagem para o desenvolvimento multiplataforma, visando achar evidências das contribuições elencadas para essa tese (Tabela 6);
- **Qualidade avaliada (*Issue*):** Nesta avaliação, buscou-se não somente avaliar se o objeto (a abordagem) é capaz ou não de cumprir seu propósito. Buscou-se avaliar mais detalhes de como as contribuições são alcançadas; e
- **Ponto de vista (*Viewpoint*):** Nesta avaliação tomou-se o ponto de vista dos especialistas, pois o interesse foi avaliar os recursos da abordagem no cumprimento de seu propósito através de opiniões externas.

Portanto, diante do objetivo, essa avaliação se concentrou em responder as seguintes questões de pesquisa (*Question*):

1. **Q1**. Quais das contribuições elencadas para esta tese foram entregues segundo os especialistas? e
2. **Q2**. A abordagem tem o potencial de ser usada para o desenvolvimento de sistemas multiplataformas reais?

Para analisar as questões de pesquisa foram definidas métricas (*Metric*), visando apresentar resultados mais consistentes para cada questão. Tais métricas foram:

- **M1**. Evidências apresentadas pelos especialistas para comprovar a entrega de cada contribuição elencada para esta tese; e
- **M2**. Lista de funções que não foram possíveis ser especificadas através da abordagem devido a alguma limitação.

A Figura 16 apresenta a relação entre o objetivo, questões e métricas definidas para a avaliação 2 conforme a abordagem GQM.

6.3 Objetivos da avaliação 3

A última avaliação foi idealizada como uma resposta a uma observação em particular feita durante a segunda avaliação. A maior parte dos participantes, 4 entre os 5 especialistas, apontaram ausência de mecanismos de testes para os códigos gerados através da abordagem. Essa ausência de recurso para testes foi considerada inclusive como uma limitação nas discussões dos resultados na segunda avaliação. Porém, tal limitação

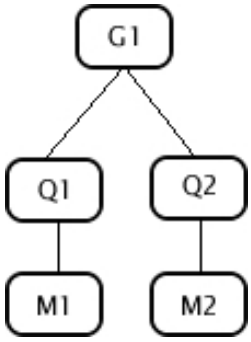

Figura 16 – Relação entre os objetivos, questões e métricas da abordagem GQM para a avaliação 2.

não é da abordagem, pois ela tem flexibilidade para incluir qualquer plataforma como alvo do processo, e isso inclui uma plataforma de testes de unidade.

Para demonstrar tal flexibilidade e confirmar a robustez da abordagem uma terceira avaliação foi conduzida visando acrescentar testes de unidade pelo próprio desenvolvedor através dos modelos da abordagem (*Goal*). Portanto, seguindo o padrão GQM essa avaliação possui o seguinte objetivo (**G1**):

- **Objeto (*Process*)**: O objeto é a abordagem proposta nesta tese;
- **Propósito (*Purpose*):** Criar testes de unidade através dos recursos destinados a extensão, visando analisar a entrega das contribuições elencadas para esta tese (Tabela 6);
- **Qualidade avaliada (*Issue*):** Nesta avaliação, buscou-se avaliar se o objeto (a abordagem) é capaz ou não de incluir novas plataformas (testes de unidade) através dos recursos destinados para esse fim, sem a necessidade de precisar editar gramáticas e geradores da abordagem; e
- **Ponto de vista (*Viewpoint*):** Nesta avaliação tomou-se o ponto de vista do pesquisador, pois o interesse foi avaliar os recursos da abordagem para a inclusão de novas plataformas.

Portanto, diante do objetivo, essa avaliação se concentrou em responder as seguintes questões de pesquisa (*Question*):

1. **Q1**. É possível incluir testes de unidade através dos recursos da abordagem projetados para a sua extensão?
2. **Q2**. É possível criar rotinas de testes de unidade genéricas através da GPL? e

3. **Q3**. Quais das contribuições elencadas para esta tese podem ser comprovadas através dessa avaliação?

As métricas para a analise das questões de pesquisa dessa avaliação são:

- **M1**. Listas de funções específicas de plataformas que puderam ser incluídas nos modelos de plataformas para suporte aos testes de unidade.
- **M2**. Rotinas genéricas especificadas através da GPL cujo códigos gerados foram executados com sucesso em mais que uma plataforma; e
- **M3**. Lista de contribuições que foram entregues pela abordagem nessa avaliação. Lista dos aspectos que confirmaram a entrega de cada contribuição.

A Figura 17 apresenta a relação entre o objetivo, questões e métricas definidas para a avaliação 3 conforme a abordagem GQM.

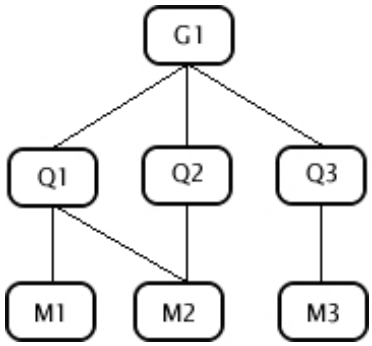

Figura 17 – Relação entre os objetivos, questões e métricas da abordagem GQM para a avaliação 3.

7 Primeira avaliação: prova de conceito

A prova de conceito foi realizada pelos pesquisadores deste trabalho e composta de três etapas. A **primeira** etapa ocorreu quase inteiramente antes do desenvolvimento da abordagem e consistia no desenvolvimento de um sistema multiplataforma para o domínio de comércio eletrônico para restaurantes. Esse sistema envolveu as principais plataformas consideradas e apresentadas nos artigos pesquisados no Capítulo 3. Como a abordagem não existia, o desenvolvimento seguiu um processo *ad-hoc*. Quatro plataformas foram consideradas no desenvolvimento: web, Android, iOS e um dispositivo que explora os conceitos da realidade aumentada (RA) desenvolvido para a plataforma RaspBerry[1]. Esse último foi pesquisado e desenvolvido em paralelo ao desenvolvimento da abordagem, visando simular uma plataforma nova, de modo a testar o suporte da abordagem a novos dispositivos que possam surgir e evidenciando assim o caráter atemporal desta tese. A arquitetura do sistema segue o estilo arquitetural *Model View Controller* (MVC) (BUCANEK, 2009), que é uma escolha comum em muitos sistemas multiplataforma.

A **segunda** etapa ocorreu após o desenvolvimento da abordagem e consistiu em migrar as principais partes do sistema multiplataforma para a abordagem, ou seja, criar os modelos de sistema, de plataforma e de implantação (modelo de *deploy*), conforme descrito na Seção 5.2. Os modelos de sistema e implantação foram criados usando a GPL, e os modelos de plataforma foram criados usando uma combinação entre a GPL e as linguagens nativas das plataformas. A terceira etapa consistiu na análise e discussão dos resultados, considerando os objetivos da avaliação.

7.1 Desenvolvimento do sistema multiplataforma

Esta primeira etapa consistiu no desenvolvimento de um sistema multiplataforma completo, de forma *ad-hoc*, para que os pesquisadores pudessem compreender melhor os aspectos técnicos desse tipo de desenvolvimento. Entre esses aspectos, buscou-se identificar qual o potencial de automação em sistemas multiplataforma e compreender melhor os desafios de forma a conceber maneiras de atender às contribuições elencadas na Tabela 6.

Um dos pontos analisados durante o desenvolvimento foi a distribuição de funcionalidades entre as plataformas, visando atender o maior número de usuários e a exploração dos recursos específicos de cada dispositivo. Para isso, o sistema conta com uma versão web responsável por gerenciar os dados de todas as plataformas envolvidas através de comunicações via *Web Services*. Além disso, possui um painel administrativo usado para

[1] https://www.raspberrypi.org/

gerenciar as seções do restaurante, tais como: produtos, clientes, região para entrega, pedidos, relatórios, entre outras. A Figura 18 apresenta o painel administrativo da versão web. A Figura 19 apresenta a interface da plataforma web para o usuário final, permitindo a realização e a gerência de pedidos por essa plataforma.

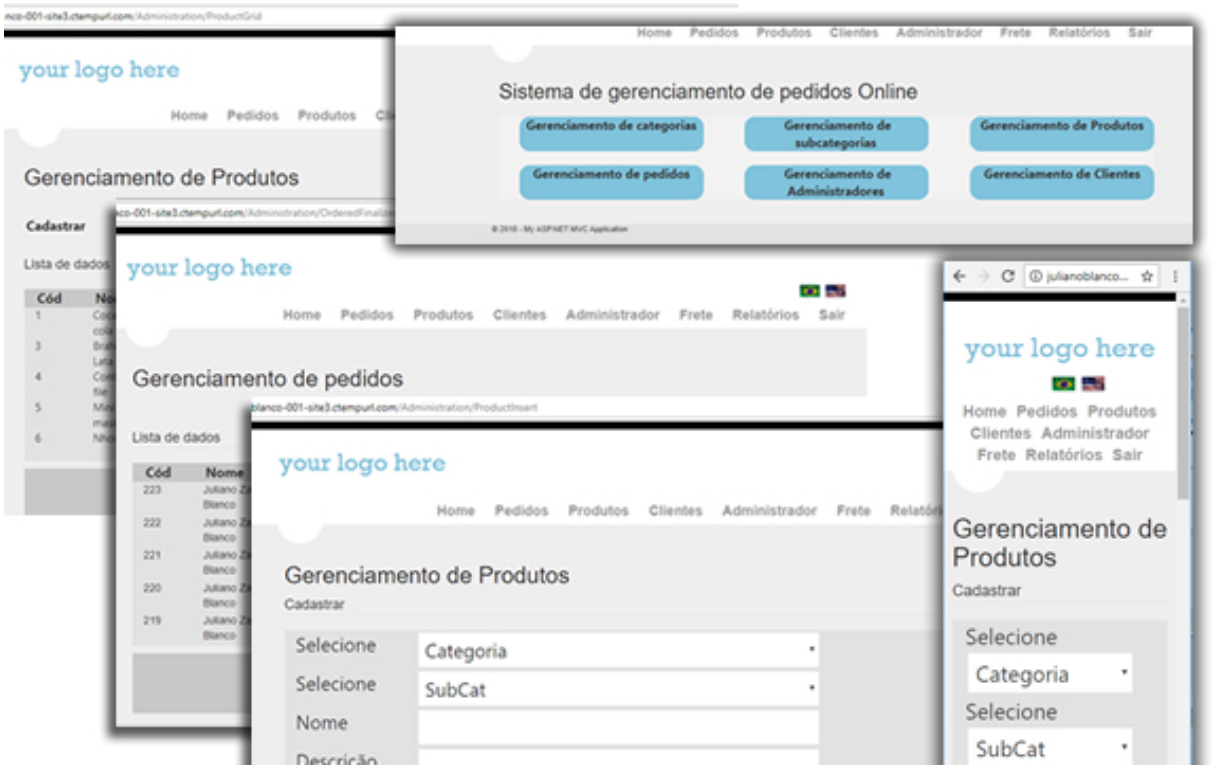

Figura 18 – Painel administrativo do sistema multiplataforma para plataforma web.

A versão móvel do sistema multiplataforma foi desenvolvida para as plataformas Android e iOS. Essa versão é responsável por replicar as principais funções de compras disponíveis na plataforma web para a móvel, visando melhorar a experiência do usuário e aumentar o número de usuários do sistema. Além disso, foi disponibilizada uma função exclusiva para a plataforma móvel que permite o gerenciamento de comandas e a realização de pedidos sem a necessidade de garçom. Essa função conta com serviços de localização (que usa GPS em ambientes abertos mas também outras técnicas como antenas *WIFI*, antenas celular, etc) dos dispositivos móveis e é importante para verificar se o usuário está realmente dentro do restaurante. Tal funcionalidade permite a exploração dos recursos específicos do dispositivo móvel.

A Figura 20 mostra o aplicativo obtendo a localização atual do usuário (Figura 20 A) e apresenta a execução do pedido sem a necessidade de garçom, permitindo informar a mesa em que o usuário se encontra (Figuras 20 B e C).

Uma quarta versão também faz parte do sistema multiplataforma, trata-se de um dispositivo que explora os conceitos da realidade aumentada (RA) através do uso de

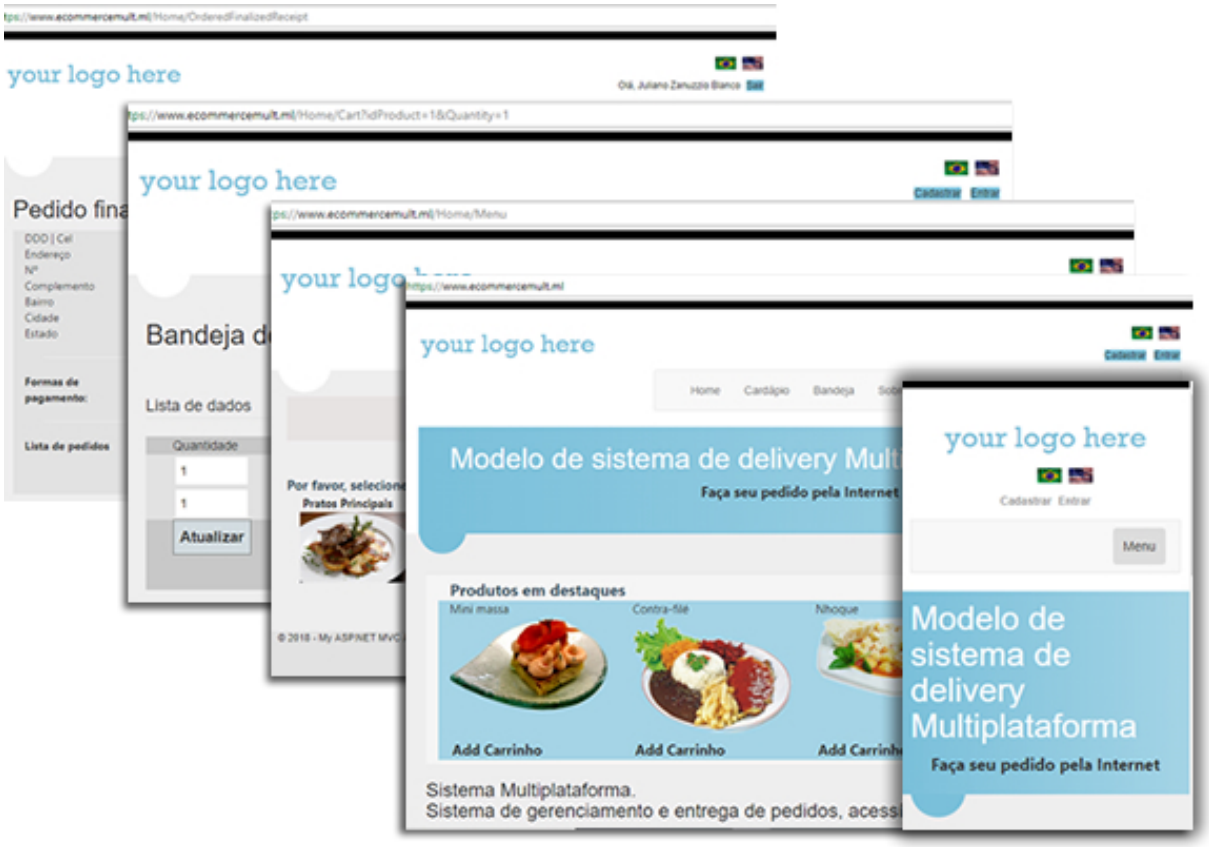

Figura 19 – Versão web para usuários finais do sistema multiplataforma.

projetores convencionais. A ideia desse dispositivo é projetar formas simples (quadrados, círculos, textos) em um cenário (uma prateleira com produtos, por exemplo), de forma a destacar um objeto (um produto do restaurante, por exemplo) com luz e cores, criando um efeito de destaque. Tais formas podem ser modeladas pelo administrador do software, através de um ambiente de modelagem disponível no sistema de RA e apresentado na Figura 21.

Nesse ambiente é possível modelar através dos recursos disponíveis no sistema RA os objetos do mundo real. No destaque do lado esquerdo da Figura 21 é possível visualizar a garrafa de whisky já modelada e no destaque do lado direito a mesma garrafa no mundo real. Está última foi usada para auxiliar a criação da modelagem, responsável em direcionar o foco do projetor a esse objeto.

As modelagens são usadas para adicionar efeitos visuais e apresentar informações aos objetos do mundo real, tais como: alteração de cores, inserção de detalhes nos objetos, iluminação para destaque, entre outros. A Figura 22 apresenta os objetos do mundo real com os efeitos gerados através do direcionamento do foco do projetor pelo sistema de RA (Figura 22 A). O usuário final pode interagir com esse sistema através do aplicativo Android ou iOS, selecionando os objetos do mundo real, conforme a Figura 22 B. Os modelos dos objetos do mundo real são armazenados no sistema web, permitindo o gerenciamento

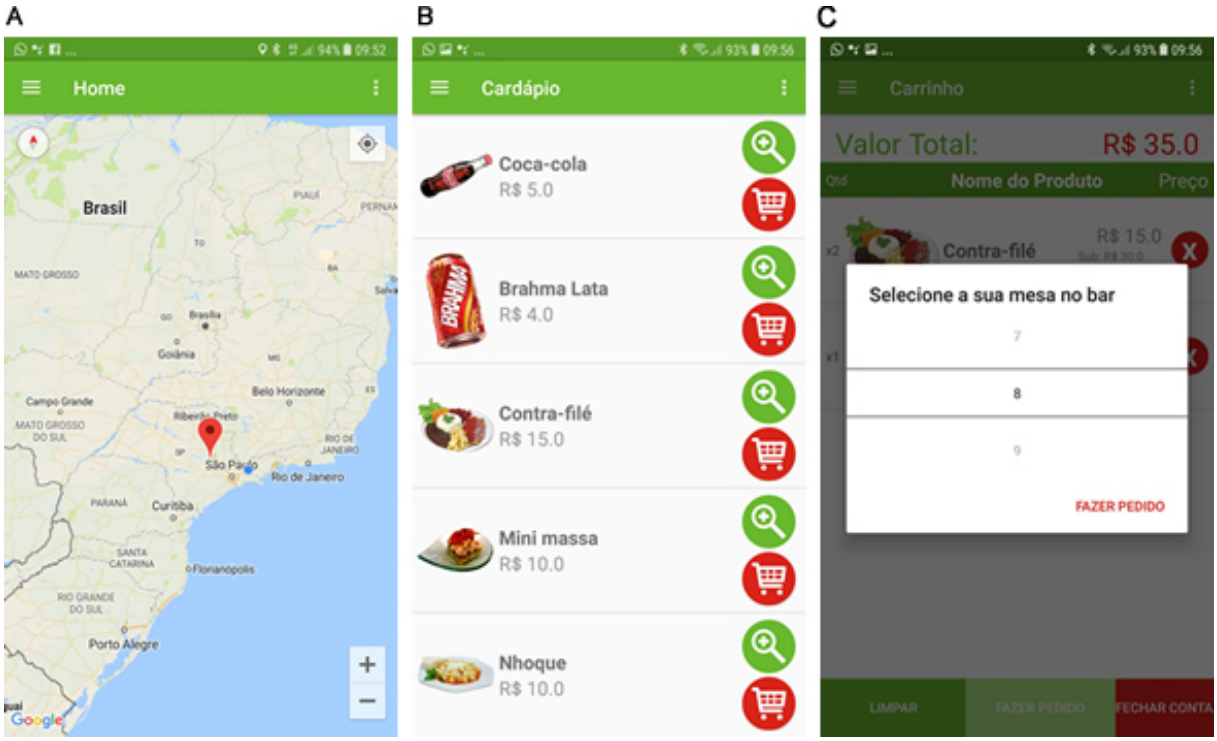

Figura 20 – Versão para dispositivos móveis do sistema multiplataforma. A- Obtenção da localização atual do usuário. B- Visualização do Cardápio. C- Opção de realizar o pedido sem a presença do garçom.

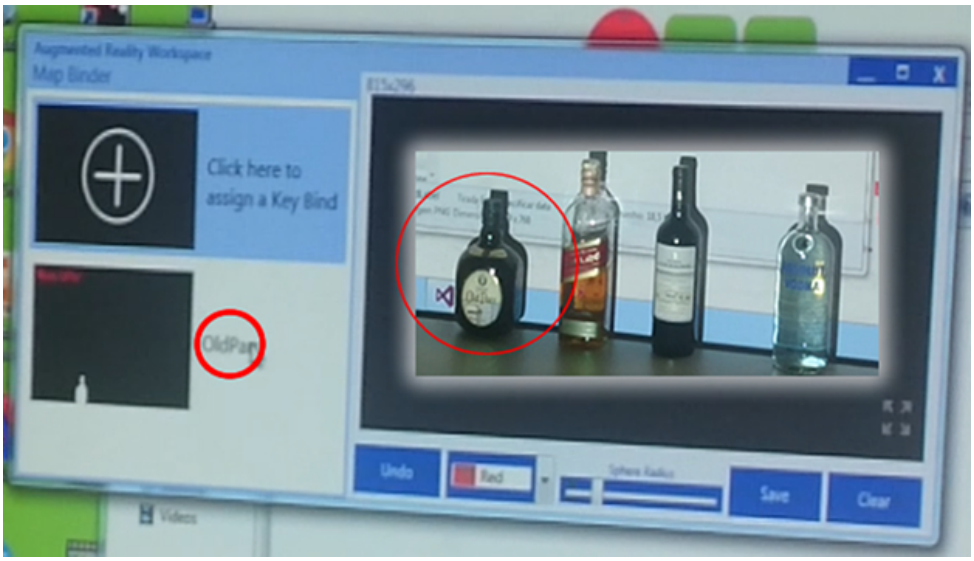

Figura 21 – Ambiente de modelagem dos objetos do mundo real da versão de RA.

do sistema de RA em ambas as plataformas web e móvel.

A Figura 23 apresenta uma visão geral do sistema multiplataforma com todos os dispositivos conectados. Nessa figura é possível verificar como é feita a comunicação entre as plataformas. A versão de RA envia os modelos dos objetos do mundo real da plataforma RaspBerry para o sistema web através do JSON. Esse, por sua vez, permite

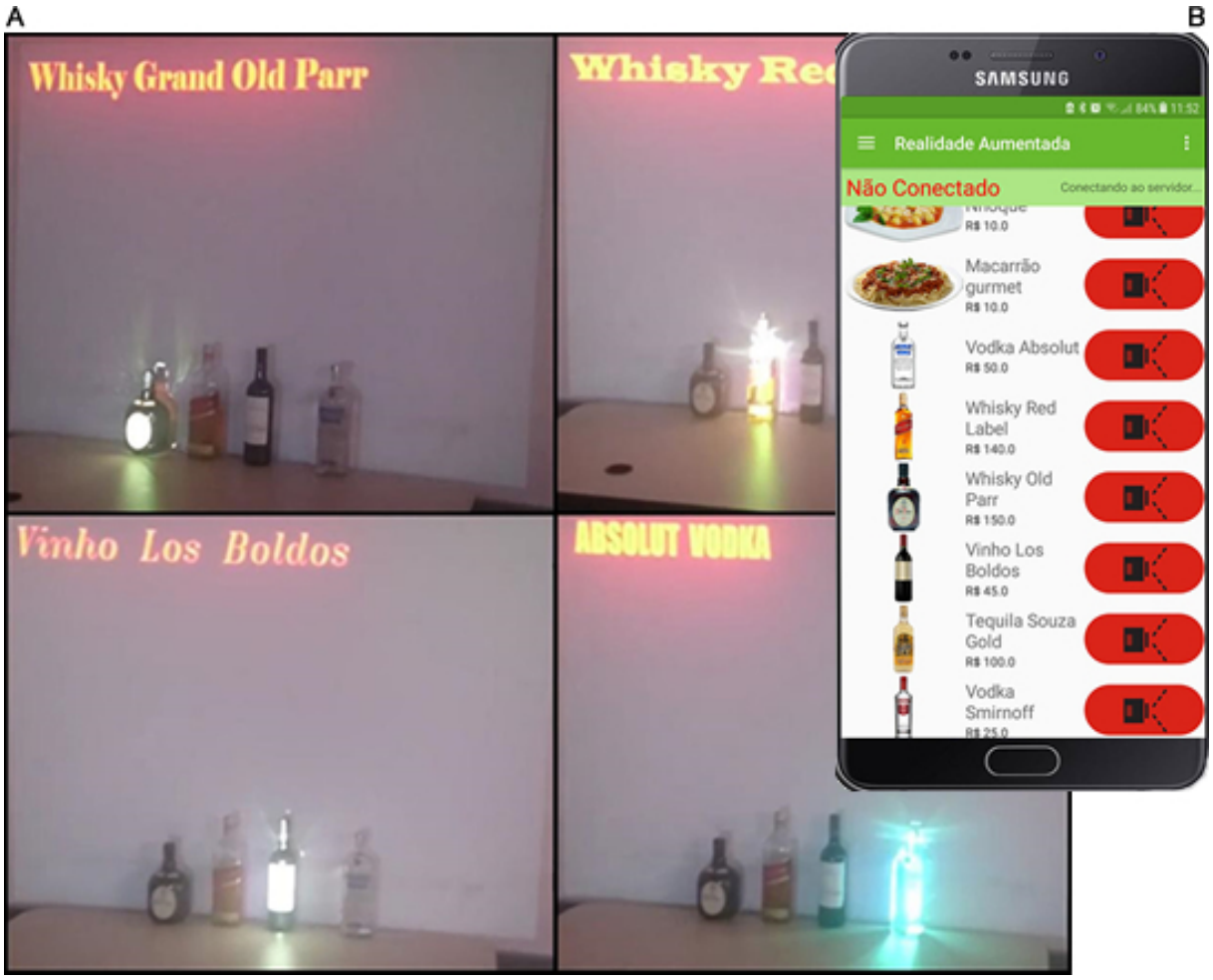

Figura 22 – Dispositivo de RA e aplicativo de controle (*smartphone*). Quando o usuário seleciona um dos produtos listados no aplicativo (botões vermelhos, no lado B da figura), o mesmo produto no mundo real é destacado através do projetor, executado pelo dispositivo de RA (lado A da figura).

que o administrador do restaurante escolha quais os objetos modelados serão utilizados para gerar os efeitos de RA. A versão do sistema para dispositivos móveis (Android ou iOS) recebe os dados dos objetos selecionados pelo administrador do restaurante e apresenta para o usuário final. Dessa forma, o usuário poderá interagir com o sistema através do celular, utilizando assim, os conceitos da RA.

O sistema web pode ser acessado pelo endereço: https://github.com/JulianoZ/teseDoutorado, assim como sua área administrativa. O sistema de realidade aumentada está disponível através do endereço: https://github.com/JulianoZ/AugmentedReality, o aplicativo móvel para a plataforma Android está disponível através do endereço: https://github.com/JulianoZ/AppAndroidComercioEletronicoRA e o aplicativo móvel para a plataforma iOS está disponível através do endereço: https://github.com/JulianoZ/AppiOSComercioEletronicoRA.

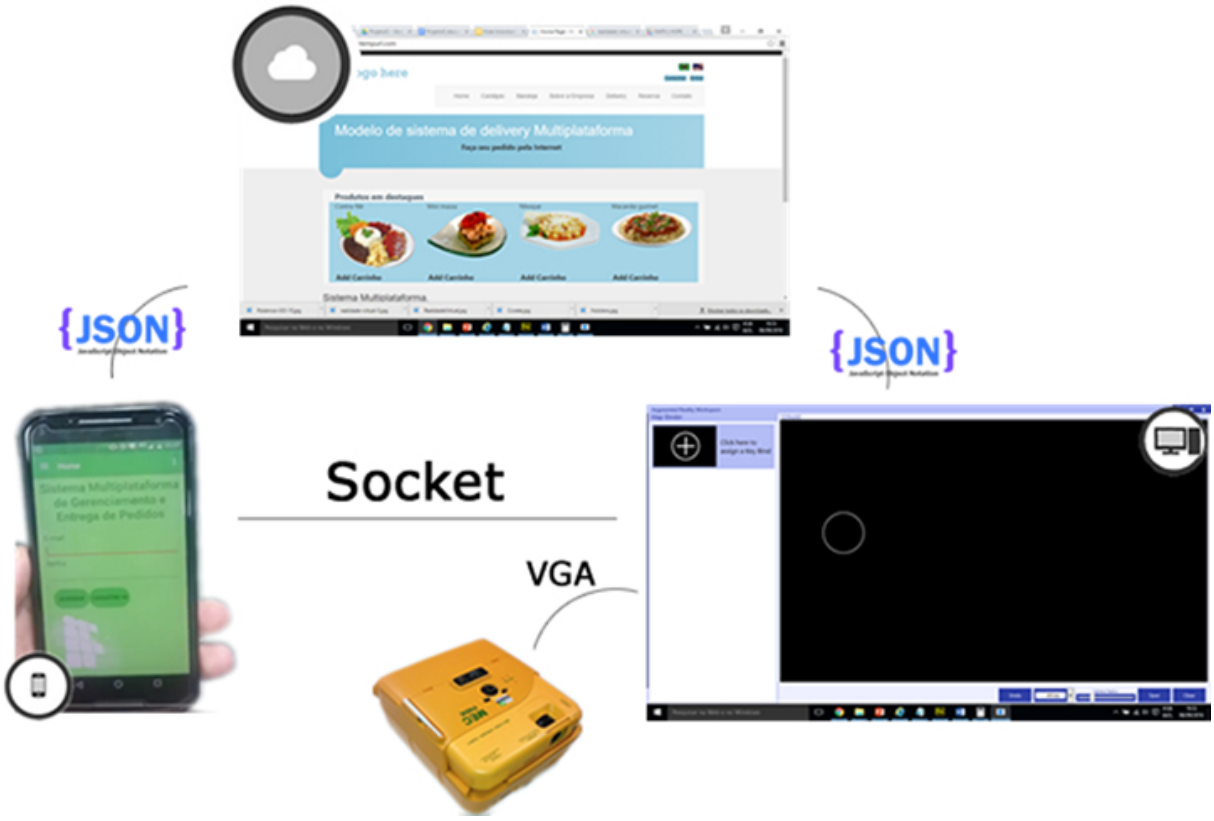

Figura 23 – Comunicação entre as versões do sistema multiplataforma.

7.1.1 Análises do sistema multiplataforma

O sistema multiplataforma concebido para esta tese forneceu base para a estruturação da abordagem, que possui o potencial de automatizar grande parte do processo de desenvolvimento e avaliar as 7 contribuições detalhadas na Tabela 6. Foi estimado que cerca de 79% do sistema desenvolvido na prova de conceito é baseado em funções que podem ser automatizadas para o domínio de comércio eletrônico. Essa estimativa foi realizada através da análise do código fonte, onde foram separadas as funções comuns para o domínio abordado e as regras de negócio específicas.

As funções comuns são consideradas os recursos fundamentais aos sistemas do domínio de comércio eletrônico. Tais funções podem possuir variações entre as realidades operacionais de cada organização, como exemplo, diferenças nas propriedades das entidades usuários, produtos, vendas. Essas diferenças podem dificultar o reaproveitamento de código fonte entre sistemas do mesmo domínio em um processo de desenvolvimento *ad-hoc*, devido ao alto número de alterações que devem ser feitas, em diferentes plataformas e linguagens. Além de, manter o processo de desenvolvimento propenso a erros de criação e manutenção. Já as funções específicas (regras de negócio e detalhes de interfaces) são importantes para a adequação do sistema à realidade operacional de cada organização e são mais difíceis de se reaproveitar, sendo necessária especificá-las em cada projeto.

O motivo da separação do sistema entre funções comuns e específicas foi avaliar a quantidade de esforço necessário para se desenvolver um sistema para este domínio. As funções comuns representam tal esforço pelo fato de serem consideradas em sua maior parte essenciais para o domínio e, que podem ser reduzida consideravelmente com o uso da abordagem. A abordagem proposta tem o potencial de criar as funções comuns, identificadas no sistema, de forma genérica. O segundo motivo da separação foi analisar o reaproveitamento das funções comuns especificadas através da GPL em um sistema previamente desenvolvido para um novo sistema, alterando tais especificações na abordagem.

A Tabela 7 apresenta os detalhes dos cálculos de linhas de códigos (LOC) comuns ao domínio e linhas com regras específicas de negócio. A versão móvel representa as plataformas Android e iOS, junto com a versão web e RA, totalizando as 4 plataformas usadas no estudo. Nessa tabela, os elementos considerados comuns são: classes POJO[2] (*Plain Old Java Objects*) e DAO[3] (*Data Access Object*), Controladores (MVC) que representam a ponte de comunicação entre as camadas de modelos e visualização do estilo arquitetural MVC, Especificidades das Plataformas e Interface. Especificidades de plataformas são detalhes técnicos comuns ao domínio de comércio eletrônico em cada plataforma, como controle de sessão, responsável por transferir dados entre páginas web e recursos de páginas nos dispositivos móveis (*Activity*/android e *ViewController*/iOS). Interface representam os códigos padrões para a camada de visualização em um sistema de comércio eletrônico. Como exemplo, podemos citar: exibição de produtos, campos de busca, organização do carrinho de compras e menu de navegação.

Tabela 7 – Detalhamento dos cálculos das funções comuns e específicas ao domínio do sistema multiplataforma abordado.

Elementos	**Versão Web**	**Versão Móvel**	**Versão RA**
POJO	15 arquivos (LOC: 2700)	4 arquivos (LOC: 640)	1 arquivo (LOC: 34)
DAO	13 arquivos (LOC: 6222)	2 arquivos (LOC: 480)	
Controladores (MVC)	4 arquivos (LOC: 2100)	1 arquivo (LOC: 2100)	LOC: 179
Regras Específicas	LOC: 490	4 arquivos (LOC: 928)	LOC: 320
Interface	LOC: 2520	LOC: 2078	LOC: 226
Detalhes de Interface	LOC: 830	LOC: 630	LOC: 150
Especificidades das Plataformas	Controle de Sessão (LOC: 45) Pagamento (LOC: 133)	Activity/ViewController 9 (LOC:3600) Adapters 2 (LOC: 150)	Mapeamento (LOC: 65) Projetor (LOC: 817)

O restante do sistema é composto de Regras Específicas e Detalhes de Interface

[2] POJO é uma sigla comumente utilizada pela comunidade Java, para se referir a uma classe que tem apenas propriedades, funções de acesso (*getters e setters*) e construtores, sem regras de negócio mais complexas.

[3] DAO representa a camada de acesso e operações com os bancos de dados

com o usuário. Na Tabela 7, as Regras Especificas são responsáveis por adequar o sistema ao modelo de negócio seguido pela empresa, como parcelamento para compras acima de determinado valor. Já os Detalhes de Interface são códigos usados para adequar a interface a identidade visual da organização. Esses dois elementos representam cerca de 21% do sistema desenvolvido, os detalhes desse cálculo são mostrados na Tabela 8.

A Tabela 8 apresenta o total de linhas de códigos comuns (coluna "Comum (LOC)") ao domínio e linhas com regras específicas de negócio (coluna "Específico (LOC)"). Nessa tabela, os elementos "Regras Específicas" e "Detalhes de Interface" (mostrados na Tabela 7) entram no cálculo das linhas de códigos específicos, o restante entra no cálculo dos códigos comuns ao domínio. A coluna Percentual da Tabela 8 representa a relação entre os códigos considerados específicos ("Específico (LOC)") e o LOC restante do sistema (coluna "Comum (LOC)"). A diferença no número de LOC entre as plataformas (coluna "Comum (LOC)" e "Específico (LOC)") representa a necessidade de diferentes funcionalidades entre as plataformas. A plataforma web contém um número maior de funcionalidades, mostrando a importância da distribuição de funcionalidades provido pelo modelo de *deploy* da abordagem. Outra informação importante mostrada na Tabela 8 é o LOC comum (coluna "Comum (LOC)") para o domínio de comércio eletrônico, muitas vezes feito de forma manual. A última linha da tabela mostra o percentual médio de LOC específico (média simples entre os valores da coluna "Percentual") usado para desenvolver o sistema multiplataforma, considerando todas as plataformas.

Tabela 8 – LOC total representando o código comum e específico no domínio de comércio eletrônico.

LOC	**Versão Web**	**Versão Móvel**	**Versão RA**
Comum (LOC)	13720	8898	1321
Específico (LOC)	1320	1558	470
Percentual	9,60%	17,50%	35,50%
Média Geral	21%		

As técnicas de geração de código e desenvolvimento de software dirigido por modelos (DSDM) ajudam a reduzir esse esforço que é comum dentro do domínio e dentro de uma mesma plataforma. No entanto, para esta tese é interessante analisar também a quantidade de esforço comum entre as diferentes plataformas, pois esta é uma das contribuições desta tese.

Como neste caso as versões web, móvel e RA utilizam linguagens diferentes (HTML, CSS, JavaScript, Java, Swift e C#), não existem linhas de código (LOC) que são exatamente as mesmas para todas as plataformas. Mas existe similaridade no código. Por exemplo, existe uma versão diferente da entidade `Product` nas três plataformas. Essas versões são diferentes do ponto de vista sintático, mas do ponto de vista conceitual elas são similares.

Para verificar essa similaridade, foi feita outra análise manual do código desenvolvido, em busca das classes comuns entre as plataformas. A Tabela 9 resume o resultado dessa análise, apresentando o número de LOC que possuem um correspondente similar em outra plataforma.

Tabela 9 – Análise das linhas de código em comum nas diferentes classes.

Plataformas	LOC	Classes comuns
Web e móveis	3658	Order, Product, User, OrderFinalized, OrderDAO, ProductDAO, UserDAO, OrderFinalizedDAO
Web and RA	643	Product, ProductDAO
Móveis e RA	1139	Product, ProductDAO, TCPUser e TCPServer
Web, móveis e RA	924	Product, ProductDAO

Foi constatado que as classes `Order`, `Product`, `User` e `OrderFinalized` e suas respectivas classes DAO são conceitos usados nas plataformas web e *móveis* (Android e iOS). Isso significa que, apesar das diferenças sintáticas, trata-se da mesma classe. Esse foi o ponto de partida da abordagem desenvolvida: conceber uma maneira de possibilitar que o desenvolvedor trate-a de fato como como uma única classe. Para comprovar essa contribuição, essas classes foram incluídas no estudo da próxima seção, assim como outras classes e funções importantes para uma análise consistente das contribuições da abordagem.

7.2 Estudo de caso. Recriando o sistema através da abordagem

O uso da abordagem na prática foi a segunda etapa da prova de conceito e constituiu em recriar as principais partes do sistema multiplataforma apresentado. A Figura 24 mostra um resumo dos modelos criados para o sistema. À esquerda (vermelho) estão as plataformas, à direita (azul) estão os modelos de sistema e no meio (branco) estão os detalhes e funções globais, servindo como uma ponte entre as plataformas e o sistema.

Iniciando com a parte do meio (branco), os detalhes globais (*Global Details*) representam os códigos específicos de plataforma para os sensores GPS (`ControllerGPS`), o gerenciamento dos dados para comunicação com a camada *View* do estilo arquitetural MVC (`ControllerDetailsGrid`) e os detalhes de conexão com o banco de dados (`DBConnection`). As funções globais são usadas para obter a localização do sistema (`GetLocation`), executar operações básicas de CRUD[4] em qualquer objeto, carregar imagens e projetá-las (`GetDiskImages` e `AddImageToCanvas`) para o sistema de RA. O suporte à comunicação entre o dispositivo móvel e o dispositivo RA também foi incluído como funções globais (`SendReceive` e `SocketListener`).

[4] Acrônimo comumente utilizado para definir as quatro operações básicas usadas em Banco de Dados Relacionais (*Create, Read, Update and Delete*).

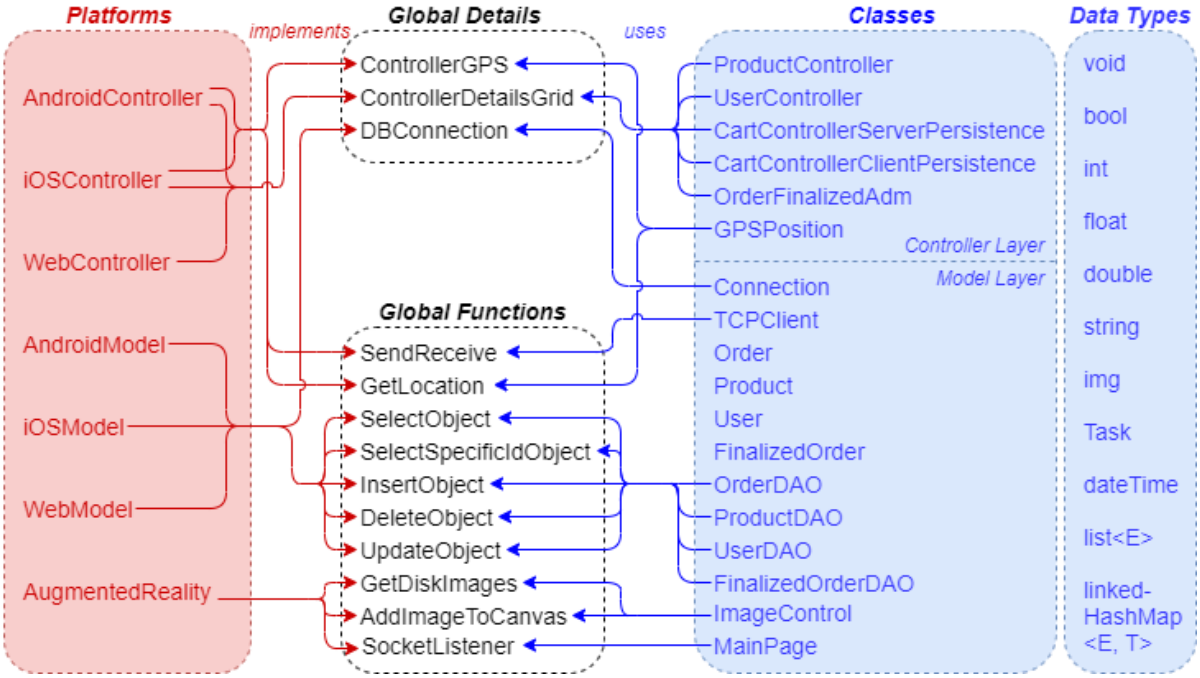

Figura 24 – Modelos criados para a prova de conceito.

No lado direito da Figura 24, em azul, estão os tipos e classes de dados do sistema (independentes da plataforma). Eles representam conceitos para o aplicativo de comércio eletrônico. As setas "*uses*" indicam quais funções globais e detalhes globais são usados por cada classe do sistema. Por exemplo, os objetos de acesso a dados (DAO) usam as funções globais CRUD para inserir, consultar, atualizar e excluir objetos em um serviço de armazenamento persistente. As classes são divididas em duas camadas: *Controller* e *Model*. Seguindo o estilo arquitetural MVC, as classes da camada *Controller* são responsáveis pelo relacionamento entre a camada *View* e *Model*. Existem classes da camada *Controller* para gerenciar produtos, usuários, carrinho de compras, pedidos sendo finalizados e a posição física do dispositivo via GPS. No sistema original (*ad-hoc*) a forma de persistência dos dados do carrinho de compras era realizado no servidor através de um banco de dados relacional (utilizando a classe `CartControllerServerPersistence`). No intuito de testar a distribuição de funcionalidades, durante a migração, foi desenvolvida uma segunda forma de persistência dos dados do carrinho, trata-se de um armazenamento no lado do cliente em memória através da estrutura *Hashmap* (utilizando a classe `CartControllerClientPersistence`). A camada *Model* possui uma classe que representa uma conexão com o serviço de armazenamento persistente (`Connection`), classes para entidades básicas de persistência (`Order`, `Product`, `User` e `FinalizedOrder`), objetos de acesso a dados (classes com sufixo DAO), uma classe para as principais funções do dispositivo de RA (`MainPage`) e uma classe para lidar com as imagens a serem projetadas em objetos do mundo real (`ImageControl`). Finalmente, uma classe `TCPClient`, responsável por enviar comandos entre um dispositivo móvel e o dispositivo de RA via *sockets*. Os tipos de dados são os tipos básicos usados para programar as funções.

No lado esquerdo da Figura 24, em vermelho, estão os modelos de plataformas. Cada dispositivo (Android, iOS e web) possui duas plataformas: uma para a camada "*Model*" e outra para a camada "*Controller*". Cada um implementará um conjunto diferente de funções globais e detalhes globais, conforme indicado pelas setas "*implements*". Isso permite uma implantação mais organizada, facilitando a reutilização futura. Por exemplo, se outro sistema de comércio eletrônico estiver sendo desenvolvido, mas as regras de negócios forem diferentes, o engenheiro de software poderá reutilizar apenas a camada "*Model*" e criar novas implementações para a camada "*Controller*". As plataformas "*Controller*" implementam os detalhes globais para exibir dados como uma grade (`ControllerDetailsGrid`). As plataformas *controller* Android e iOS também implementam detalhes de GPS e localização, bem como a função global para enviar dados através de uma conexão via *sockets*. As plataformas "*Model*" implementam a conexão com o banco de dados e as funções CRUD genéricas, além de algumas regras de negócios específicas de domínio. E a plataforma `AugmentedReality` é responsável pela implementação das funções globais de gerenciamento e exibição de imagens e pelo recebimento de comandos via *sockets* para exibi-las.

A seguir são apresentados exemplos dos principais modelos mostrados na Figura 24. Cada exemplo apresenta um aspecto diferente da abordagem, conforme detalhado na Tabela 10.

Tabela 10 – Exemplos de especificações feitas por meio da GPL e apresentadas neste capítulo.

Exemplos	**Descrição**
Listagem 7.1	Exemplo de implementação de uma função global (`DeleteObject`).
Listagem 7.2	Exemplo de uma classe genérica (`OrderDAO`) instanciando as funções globais para a geração do CRUD e funções que gerenciam o carrinho de compras na camada *Model*.
Listagem 7.3 Figura 25	Exemplo de uma classe genérica (`UserDAO`) instanciando as funções globais para a geração do CRUD e funções para autenticação de usuários na camada *Model*.
Listagem 7.4 Figura 26	Exemplo de uma classe genérica (`ProductController`) da camada *Controller* responsável por obter os dados (incluindo produtos, categorias e subcategorias) persistidos por meio da camada *Model* e enviá-los para a camada *View*.
Listagem 7.5 Figura 26	Exemplo de uma classe genérica (`ProductReducedController`) da camada *Controller* responsável por obter os dados (somente produtos) persistidos por meio da camada *Model* e enviá-los para a camada *View*.

Listagens 7.6, 7.7 e 7.8 Figuras 27 e 27	Exemplo da modelagem do carrinho de compras utilizando dois tipos de persistência (banco de dados e *Hashmap*).
Listagem 7.9 Figura 28	Exemplo de configuração da geração de código do carrinho de compras no modelo de *deploy*.
Listagem 7.10 Figura 29	Exemplo da classe (`OrderFinalizedAdm`) responsável por apresentar os pedidos para o administrador do sistema e realizar as suas mudanças de *status*.
Listagem 7.11	Exemplo da classe genérica `GPSPosition` utilizada para configurar a utilização dos serviços de localização.
Listagem 7.12	Exemplo de classes (`MainPage` e `ImageControl`) que possuem especificações genéricas do software de RA para o domínio de restaurantes.

O desenvolvimento do sistema através da abordagem começou com as especificações da camada *Model* do estilo arquitetural MVC nos modelos de plataformas. Nessa etapa, foram criados os seguintes modelos de plataformas: web, Android e iOS. Para cada modelo de plataforma declarado foram especificados as funções globais para um CRUD completo e regras de negócio para o domínio alvo da camada *Model*. As funções globais estão especificadas na Tabela 11.

Tabela 11 – Funções globais definidas para a camada de modelo.

DBConnection	Responsible for generating the database connections method
InsertObject	Responsible for data persistence
SelectObject	Responsible for return an object list
UpdateObject	Responsible for data update
DeleteObject	Responsible for data delete
SelectSpecificIdObject	Responsible for get a specific data from database

Essas funções foram escritas nas três linguagens consideradas pela abordagem (Java, C# e Swift) e são responsáveis por automatizar o CRUD nas classes utilizadas. A Listagem 7.1 apresenta um exemplo da função global `DeleteObject` sendo implementada na linguagem Java (linhas 2-16), Swift (linhas 19-32) e C# (linhas 35-52), nas plataformas Android (linha 1), iOS (linha 18) e web (linha 34) respectivamente. Após a especificação dos modelos de plataformas foram declarados os seguintes tipos de variáveis que foram usadas no sistema: `int`, `string`, `float`, `double`, `bool`, `dateTime`, `void`, `list<E>`, `linkedHashMap<E, T>`, `img` e `Task`. A declaração dessas variáveis pode ser feita em um arquivo dedicado e importado em todos os novos projetos. O passo seguinte consistiu na especificação das classes e na definição das principais funções pelo desenvolvedor através da linguagem da abordagem.

```
1 platform Android: Java{
```

```
2   implementsGlobal <E> DeleteObject(e: <E>, context: Context):
     string{
3   '''
4   string result;
5   Connection db = Connection.getInstance(c.getApplicationContext
     ());
6   String query = "DELETE FROM $E.name WHERE id=" + e.getId();
7   try{
8      db.getWritableDatabase().execSQL(query);
9      db.close();
10     return="true";
11  }catch(Exception e){
12     return="false";
13  }
14    return result;
15  '''
16  }
17 }
18 platform iOS: Swift{
19  implementsGlobal <E> DeleteObject(e: <E>): string{
20  '''
21    string result;
22    sharedInstance.database!.open()
23    let isDeleted = sharedInstance.database!.executeUpdate("
     DELETE FROM $E.name WHERE id=?", withArgumentsInArray: [e.id])
24    sharedInstance.database!.close()
25    if (isDeleted){
26       result="Data updated into database successfully!"
27    else{
28       result="Error"
29    }
30  return result
31  '''
32  }
33 }
34 platform Web: CSharp{
35  implementsGlobal <E> DeleteObject(e: <E>): string{
36  '''
37    string result;
38    Connection objCon = new Connection();
39    MySqlConnection Conn = new MySqlConnection();
40    Conn = objCon.OpenConnection();
```

```
41      MySqlCommand command = Conn.CreateCommand();
42      command.CommandText = "delete from $E.name WHERE id = " & e.
    id & "";
43      int verify = cmd.ExecuteNonQuery();
44      if (verify != -1){
45         result = "Data deleted into database successfully!";
46      }else{
47         result = "Error";
48      }
49      command.Dispose();
50      return result;
51    '''
52    }
53 }
```

Listagem 7.1 – Implementação da função global DeleteObject nas plataformas Android (linha 1), iOS (linha 18) e web (linha 34).

A Listagem 7.2 mostra a classe `OrderDAO` modelada através da linguagem proposta. Essa classe gerencia a interação do carrinho de compras com a base de dados, além de algumas regras de negócio do sistema multiplataforma e é um conceito comum para as três plataformas: web, Android e iOS. Nas linhas 2 a 6 da Listagem 7.2 estão as declarações das funções globais utilizadas para a interação com a base de dados. Na linha 7 existe a declaração da estrutura *Hashmap* (`products`) para o armazenamento dos dados do carrinho de compras em memória e nas linhas 8-13 operações para obter (`getHashMapProducts`) esses dados e armazená-los (`addHashMapProducts`). Na linha 14 há um método que gerencia a inclusão de itens no carrinho através do uso de um banco de dados, nesse método é possível verificar mais detalhes de como é feita a especificação através da GPL. As instancias de duas classes são mostradas nas linhas 15 e 16, na sequência (linhas 17-24) há exemplos de declarações de variáveis e atribuições de dados. A lista de objetos `listOrder` (linha 25) vem através de uma chamada a função global `SelectObject()` (declarada na linha 3), e é aqui que de fato se concretiza a junção entre esse código independente de plataforma (função global) e um código dependente de plataforma (`listOrder`). Na linha 26 a lista `listOrder` é percorrida através de um laço de repetição e caso existam produtos para um determinado usuário já armazenados no carrinho de compras (condicional da linha 28), os dados são processados (linhas 30-32) e atualizados em um banco de dados (linha 33). Tal atualização é realizado através da chamada da função global `UpdateObject()`, passando o objeto `objO`, já processado, como parâmetro. Caso contrário da condicional (linha 34), os dados do produto são processados (linhas 36-40) e armazenados como um novo item do carrinho (linha 41) através da função global `InsertObject()`. Na linha 47 é mostrado um método para calcular o valor total dos

itens adicionados no carrinho, onde é possível visualizar a função global `SelectObject()` (linha 54) sendo usada novamente. Além do aproveitamento dos conceitos modelados de uma única vez para as três plataformas, alguns detalhes de baixo nível foram abstraídos através da linguagem proposta e são gerados de forma automática através dos geradores específicos para cada linguagem. São exemplos dessas abstrações as interações com as listas de objetos mostrados nas linhas 26 e 55 e as funções globais que adicionam detalhes técnicos de interação com o banco de dados de forma transparente ao desenvolvedor.

```
1  class OrderDAO{
2   usesGlobal <Order> UpdateObject(e: Order): string
3   usesGlobal <Order> SelectObject(ord: string): list<Order>
4   usesGlobal <Order> SelectSpecificIdObject(e: Order, id: int):
    Order
5   usesGlobal <Order> InsertObject(e: Order): string
6   usesGlobal <Order> DeleteObject(e: Order): string
7   products: linkedHashMap<Product, int>
8   operation getHashMapProducts(): linkedHashMap<Product, int>{

9     return products
10  }
11  operation addHashMapProducts(p: Product, quantity: int): void{
12    products.put(p, quantity)
13  }
14  operation OrderInsertCart(idProd: int, quantity: int,
    sectionClient: int): string{
15    objO: Order
16    objProd: Product
17    NewQuant: int
18    result: string
19    listOrder: list<Order>
20    x: int
21    ord: string
22    ord := "order by id ASC"
23    id: int
24    id := idProd
25    listOrder := global SelectObject(ord)
26    For x=0,x<listOrder.size,x++ {
27      objO := listOrder.get(x)
28      If objO.SectionUser == sectionUser && objO.
    Product_idProduct==idProd {
29        If objO.Quantity > 0 {
30          objO.id := objO.id
```

```
31            objO.NewQuant := objO.Quantity + 1
32            objO.ValueTotal := objO.ValueUnit * NewQuant
33            result := global UpdateObject(objO)
34          }Else{
35            objProd = global objProd.SelectSpecificIdObject(
    objProd, id)
36            objO.Product_idProduct := objProd.id
37            objO.ValueUnit := objProd.Price
38            objO.Quantity := quant
39            objO.ValueTotal := (quant * objProd.Price)
40            objO.SectionUser := sectionUser
41            result := global InsertObject(objO)
42          }
43        }
44      }
45      return result
46    }
47    operation CalculateTotalValuePurchase(SessionId: string):
    float{
48      TotalValue: float
49      listOrd: list<Order>
50      objO: Order
51      x: int
52      ord: string
53      ord := "Order by Product_idProduct ASC "
54      listOrd := global SelectObject(ord)
55      For x=0,x<listOrder.size,x++ {
56        objO := listOrd.get(x)
57        If objO.SectionUser == SessionId && objO.StatusFinalized
    == 1 {
58          TotalValue += objO.ValueTotal
59        }
60        return TotalValue
61      }
62    }
63  }
64
```

Listagem 7.2 – Modelagem da classe “OrderDAO” através da abordagem.

Além da classe `OrderDAO`, foram criadas outras classes para a camada modelo do estilo arquitetural MVC através da GPL. O nome dessas classes são apresentadas na Tabela 12.

Tabela 12 – Classes especificadas através da GPL para a camada Modelo.

DAO	POJO
OrderFinalizedDAO	OrderFinalized
OrderDAO	Order
ProductDAO	Product
UserDAO	USer
CategoryDAO	Category
SubCategoryDAO	SubCategory

A Listagem 7.3 apresenta a programação de uma classe genérica (`UserDAO`) instanciando as funções globais para a interação com a base de dados. Nessa classe é possível verificar também o método de autenticação (linha 8) de usuários escrito de forma genérica através da GPL. Esse método (`operation`) foi configurado posteriormente pelo modelo de *deploy* para a geração automática de código para as três plataformas alvo.

```
1  class UserDAO{
2    usesGlobal <User> UpdateObject(e: User): string
3    usesGlobal <User> SelectObject(ord: string): list<User>
4    usesGlobal <User> SelectSpecificIdObject(e: User, id: int):
      User
5    usesGlobal <User> InsertObject(e: User): string
6    usesGlobal <User> DeleteObject(e: User): string
7
8    operation AuthenticationCli(cli: User): list<string>{
9      objCl: User
10     accessData: list<string>
11     listObjCli: list<User>
12     ord: string
13     ord := "Order by id"
14     listObjCli := global SelectObject(ord)
15     x: int
16     status: bool
17     For x=0,x<listObjCli.size,x++ {
18       objCl := listObjCli.get(x)
19       If objCl.email == cli.email && objCl.password == cli.
     password {
20           status := "true"
21           accessData.add(objCli.id)
22           accessData.add(objCli.name)
23           accessData.add(status)
24       }Else{
25           status := "false"
```

```
26            objCl.name := "Erro"
27            accessData.add(objCl.name)
28            accessData.add(status)
29          }
30        }
31        return accessData
32      }
33    }
34
```

Listagem 7.3 – Modelagem da classe "UserDAO" através da abordagem.

A Figura 25 apresenta a interface que utiliza os códigos gerados de autenticação de usuários para as plataformas web (A), Android (B) e iOS (C). Como já informado, o sistema web que usa os códigos gerados pela abordagem pode ser acessado no endereço disponível em https://github.com/JulianoZ/teseDoutorado.

Figura 25 – Autenticação de usuários nas plataformas: web (A), Android (B) e iOS (C) respectivamente.

Além da camada *Model* do estilo arquitetural MVC, a camada *Controller* também foi modelada através da linguagem proposta após a análise dos conceitos similares usados pelas plataformas. Assim, algumas classes que gerenciam o fluxo de dados (camada *Controller*) entre as camadas de *Model* e *View* também são comuns nas três plataformas, como é o caso das classes `ProductController`, `UserController`, `OrderFinalizedAdm`, `CartControllerHashMapPersistence` e `CartControllerServerPersistence`. Para dar suporte às classes da camada *Controller* foram criados modelos de plataformas específicos para conter as funções globais para as três plataformas usadas no sistema da prova de

conceito: `WebController`, `AndroidController` e `iOSController`. Essas plataformas utilizam as mesmas linguagens declaradas para a camada *Model* em cada plataforma: C#, Java e Swift respectivamente.

A classe `ProductController` é responsável por obter os dados persistidos por meio da camada *Model* e enviá-los para a camada *View*. Porém, a interface da plataforma web possui mais espaço em tela do que as plataformas móveis e assim, como forma de diversificar, foram criadas duas classes distintas para esse controle. Uma classe enviando apenas a lista de produtos para a interface usada pelos dispositivos móveis (`ProductReducedController`) e a outra enviando a lista de produtos junto com os dados de categoria e subcategoria (`ProductController`) usada pela plataforma web. A Listagem 7.4 apresenta a classe `ProductController`. Nessa classe é possível ver a utilização do detalhe global `ControllerDetailsGrid` (linha 2) responsável por gerar as funções de inicialização em cada plataforma e enviar a lista de dados para a camada de visualização. As listas de dados são obtidas através da função `populateList()` (linha 3) desenvolvida de forma manual através da GPL. Nas linhas 9, 10 e 11 é possível verificar a obtenção de dados para as três listas enviadas para a interface web, lista de produtos, categorias e subCategorias. Os códigos gerados pela GPL de forma automática para a camada de modelo também podem ser visualizados nessa listagem (linhas 9, 10 e 11), isso mostra como são feitos as comunicações entre as camadas do sistema e o uso dos códigos gerados pela GPL por outras classes dentro do sistema.

```
1  class ProductController{
2  usesGlobalDetails <Product, ProductAdapterConfData,
       ProductControllerList, ProductConfMethod> ControllerDetailsGrid
3      operation populateList(): void{
4       objProd: ProductDAO
5       objCat: CategoryDAO
6       objSubCat: SubCategoryDAO
7       ord: string
8       ord := "Order by name "
9       dataListProduct := global objProd.SelectObject(ord)
10      dataListCategory := global objCat.SelectObject(ord)
11      dataListSubCategory := global objSubCat.SelectObject(ord)

12     }
13  }
```

Listagem 7.4 – Modelagem da classe “ProductController” através da GPL para a plataforma web.

A classe `ProductReducedController` gerencia o fluxo de dados entre as camadas de modelo e visualização para as plataformas móveis, porém envia apenas a lista de pro-

dutos para a interface devido à limitação do tamanho da tela. A Listagem 7.5 apresenta a classe `ProductReducedController` com o mesmo detalhe global (linha 2) usado na classe `ProductController` (Listagem 7.4). A diferença nesse caso é a função `populateList()` que obtém apenas a lista de produtos (linha 7). A quantidade de listas que serão passadas para a camada de visualização podem ser definida nas classes de configuração junto com os outros detalhes enviados como parâmetros para a função `ControllerDetailsGrid`. Dessa forma, os detalhes técnicos inerentes a cada plataforma são abstraídos através da GPL, aumentando o foco do desenvolvedor nos conceitos do sistema.

```
1 class ProductReducedController{
2     usesGlobalDetails <Product, ProductAdapterConfData,
    ProductCrossControllerList,        ProductConfMethod>
    ControllerDetailsGrid
3     operation populateList(): void{
4        objProd: ProductDAO
5      ord: string
6      ord := "Order by name "
7      dataListProduct := global objProd.SelectObject(ord)

8     }
9 }
```

Listagem 7.5 – Modelagem da classe “ProductReducedController” através da GPL para dispositivos móveis.

O resultado das listagens 7.4 e 7.5 são códigos-fonte para a camada *Controller* gerada de forma automática pela abordagem com todos os detalhes técnicos inerentes a cada plataforma. A Figura 26 apresenta a camada de visualização das três plataformas recebendo e organizando a lista de dados na interface, três listas (Categorias, SubCategorias e Produtos) para a versão web e uma lista (Produtos) para a versão móvel (Android e iOS).

Os controladores (camada *Controller*) referentes aos usuários (`UserController`) e aos produtos (`ProductController`), obtém os dados da camada de modelo e entregam para a camada de visualização. Outros controladores precisam obter os dados de uma seção anterior, processá-los e só assim entregá-los à camada de visualização, como é o caso dos carrinho de compras. Para isso a função global `ControllerDetailsGrid` também é usada, porém as classes enviadas como parâmetros são configuradas de modo a receber tais dados. A Listagem 7.6 apresenta como as classes enviadas como parâmetros são configuradas. Na Linha 1 a classe `CartAdapterConf` apresenta os dados que são mostrados nas telas dos dispositivos. Na linha 6, a classe `CartControllerServerPersistenceList` informa a quantidade de listas passadas para a camada de visualização, dessa forma o gerador poderá preparar os ambientes com os detalhes inerentes a cada plataforma. E por último,

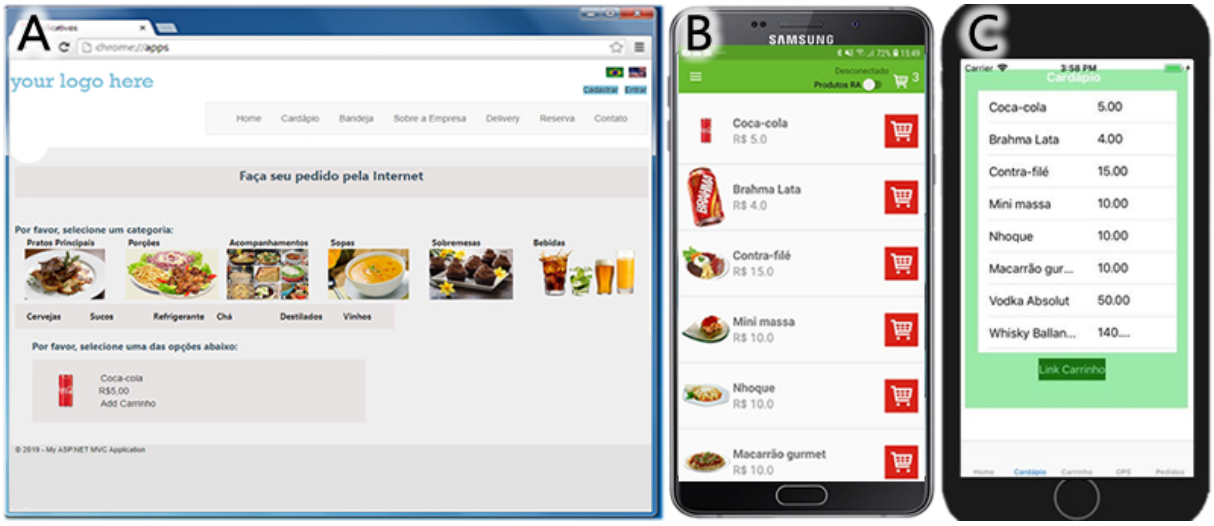

Figura 26 – Tela de produtos nas plataformas: web (A), Android (B) e iOS (C) respectivamente.

a classe `CartConfMethod` (linha 9) configura a criação do método que popula as listas configuradas na última classe. O primeiro e o segundo atributo define o nome da função e o seu tipo respectivamente. Os outros atributos definem quais os dados que são obtidos de uma seção anterior para realizar os processamentos necessários para a seção atual.

```
1  class CartAdapterConf {
2    name: string
3    price: float
4    Quantity: int
5  }
6  class CartControllerServerPersistenceList{
7    Order: string
8  }
9  class CartConfMethod{
10   populateList: string
11   void: string
12   sessionId: int
13   idProd: int
14   quant: int
15 }
```

Listagem 7.6 – Classes de configuração para a função global "CartControllerServerPersistence".

A Listagem 7.7 mostra a modelagem do carrinho de compras de forma genérica para as três plataformas na camada *Controller*. Na Linha 2 é possível ver as classes definidas na Listagem 7.6 sendo passadas como parâmetro para a função global. Essa função

global utiliza os dados das classes passadas como parâmetros para incluir os detalhes técnicos da camada *Controller* inerentes a cada plataforma. Na linha 3 é apresentado o método para popular as listas (`populateList`) passando as três variáveis como parâmetro definidos na classe `CartConfMethod` (Listagem 7.6). O método `OrderListSession` (linha 21) obtém a lista de produtos selecionado pelo usuário e o método `insertDataListOrder` (linha 33) completa os dados da lista anterior para enviar para a camada de visualização. Os métodos `populateList`, `OrderListSession` e `insertDataListOrder` contém regras especificas para a função carrinho de compras. Portanto, qualquer função que necessite obter dados e processar antes de enviar para a camada de visualização pode ser modelado de forma genérica dessa forma.

```
1  class CartControllerServerPersistence{
2    usesGlobalDetails <Order, CartAdapterConf,
     CartControllerServerPersistenceList, CartConfMethod>
     ControllerDetailsGrid
3    operation populateList(sessionId: int, idProd: int, quant: int)
     : void {
4          objOrder: Order
5          objOrder.setProduct_idProduct(idProd)
6          objOrder.setSectionClient(sessionId)
7          objOrder.setQuantity(quant)
8          objOrderDAO: OrderDAO
9          result: string
10         If idProd <> 0 && quant <> 0 {
11       result := objOrderDAO.InsertObject(objOrder)
12         }
13         listOrder: list<Order>
14         ord: string
15         ord := "Order by id"
16         listOrder := global objOrderDAO.SelectObject(ord)
17         listOrderSession: list<Order>
18         listOrderSession := OrderListSession(sessionId, listOrder
     )
19         dataListOrder := insertDataListOrder(listOrderSession)
20   }
21   operation OrderListSession(sessionId: int, listOrder: list<
     Order>): list<Order> {
22     x: int
23     listOrderSession: list<Order>
24     objOrder: Order
25       For x=0,x<listOrder.size,x++ {
26     objOrder := listOrder.get(x)
```

```
27            If objOrder.SectionUser == SessionId {
28                  listOrderSession.add(objOrder)
29              }
30      }
31      return listOrderSession
32  }
33  operation insertDataListOrder(listOrder: list<Order>): list<
    Order> {
34    x: int
35    objProdDao: ProductDAO
36    listOrderProduct: list<Order>
37    objOrder: Order
38    objProduct: Product
39    For x=0,x<listOrder.size,x++ {
40      objOrder := listOrder.get(x)
41          objProduct := global objProdDao.SelectSpecificIdObject(
    objProduct, objOrder.Product_idProduct)
42      objOrder.setName(objProduct.name)
43          objOrder.setPrice(objProduct.price)
44          listOrderProduct.add(objOrder)
45        }
46      return listOrderProduct
47  }
48 }
```

Listagem 7.7 – Modelagem da classe "CartControllerServerPersistence" usada para o gerenciamento do carrinho de compras de forma genérica.

O resultado da Listagem 7.7 é a classe `CartControllerServerPersistence` que opera o carrinho de compras para a camada *Controller* gerada de forma automática para cada plataforma indicada no modelo de *deploy*, para esse caso web, Android e iOS. A Figura 27 apresenta os códigos gerados pela abordagem sendo executados nos dispositivos nativos, exibindo assim os itens armazenados por um usuário no carrinho de compras.

O carrinho de compras da Figura 27 utiliza um banco de dados específico para cada plataforma para o armazenamento dos produtos selecionados pelo usuário. Caso o desenvolvedor prefira utilizar uma outra forma de armazenamento dos dados do carrinho, como armazenamento em memória por exemplo, ele poderá usar a GPL para programar tal rotina. Como exemplo, a Listagem 7.8 mostra a classe `CartControllerHashMapPersistence` que utiliza a estrutura *Hashmap* para armazenamento dos dados no carrinho. Nessa listagem é possível visualizar a chamada de um método da classe `OrderDAO` (Listagem 8, linha 11) usado para o armazenamento dos dados na estrutura *Hashmap* e um outro mé-

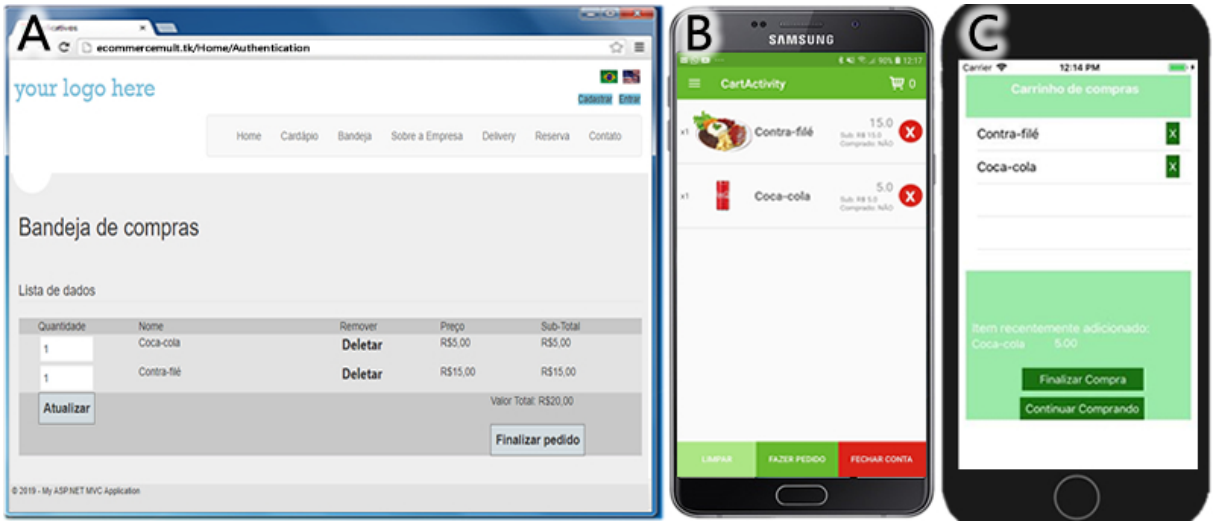

Figura 27 – Tela do carrinho de compras nas plataformas: web (A), Android (B) e iOS (C) respectivamente.

todo (linha 10) usado para obter os produtos selecionados pelo usuário para o carrinho de compras.

```
1  class CartControllerHashMapPersistence{
2    usesGlobalDetails <Order, CartHashMapAdapterConf,
      CartControllerHashMapPersistenceList, CartHashMapConfMethod>
      ControllerDetailsGrid
3    operation populateList(sessionId: int, idProd: int, quant: int)
      : void {
4        objOrderDAO: OrderDAO
5        If idProd <> 0 && quant <> 0 {
6          objProduct: Product
7          objProduct := getProduct(idProd)
8          objOrderDAO.addHashMapProducts(objProduct, quant)
9        }
10       dataListProduct := objOrderDAO.getHashMapProducts()
11   }
12   operation getProduct(id: int): Product{
13       objProduct: Product
14       objProductDao: ProductDAO
15       objProduct := global objProductDao.SelectSpecificIdObject(
      objProduct, id)
16       return objProduct
17   }
```

```
18 }
```

Listagem 7.8 – Modelagem da classe "CartControllerHashMapPersistence" usada para o gerenciamento do carrinho de compras utilizando a estrutura *Hashmap*.

As classes da camada *Controller* (MVC), modeladas através da GPL, foram configuradas através do modelo de *deploy* para a geração de códigos. Nesse modelo define-se por exemplo como será a forma de armazenamento dos dados do carrinho e quais as classes que serão executadas em cada plataforma. Na Listagem 7.9 é possível verificar as plataformas web e iOS utilizando o sistema de persistência de dados em banco (linhas 2 e 3) e o Android utilizando o sistema de armazenamento em memória (linha 4).

```
1 deploy Ecommerce{
2   CartControllerServerPersistence: WebController
3   CartControllerServerPersistence: iOSController
4   CartControllerHashMapPersistence: AndroidController
5 }
```

Listagem 7.9 – Configuração do modelo de *deploy*

O resultado do processo são códigos-fonte para a camada *Controller* gerados de forma automática pela abordagem com todos os detalhes técnicos inerentes a cada plataforma. A Figura 28 apresenta a camada de visualização exibindo os dados fornecidos pela camada *Controller* na seção de carrinho de compras de cada plataforma, no ambiente real onde foram executados testes com os códigos implantados nas plataformas. Nesta foto, é possível ver a tela do carrinho de compras implantado na plataforma web, em um *tablet* Android, em um Apple *iPad* e um celular Android.

Além da distribuição de funcionalidades entre as plataformas e alterações no modo do funcionamento de determinadas rotinas no sistema, o modelo de *deploy* também permite a troca facilitada das plataformas utilizadas pelo sistema. O sistema desenvolvido na prova de conceito possui o gerenciamento dos pedidos recebidos apenas no sistema web. Essa decisão gerencial se deu pelo fato da versão web poder comportar mais informações na tela e poder rodar em diversos dispositivos. Porém, o sistema web precisa usar um servidor que normalmente aumenta o custo financeiro do projeto. Caso os desenvolvedores precisem substituir a plataforma web pela plataforma móvel para gerenciar os pedidos recebidos, os custos dessa migração normalmente são altos. Essa troca de configuração pode ser minimizada com o uso da abordagem proposta. Para tal, foi desenvolvida uma rotina genérica que apresenta aos administradores do sistema os pedidos recebidos pelo restaurante. Dessa forma a substituição entre uma plataforma e outra pode ser realizada através da troca do apontamento da plataforma que irá executar a rotina no modelo de *deploy*. A Listagem 7.10 apresenta a classe `OrderFinalizedAdm` responsável por apresentar os pedidos para o administrador do sistema e realizar a sua mudança de status. Na

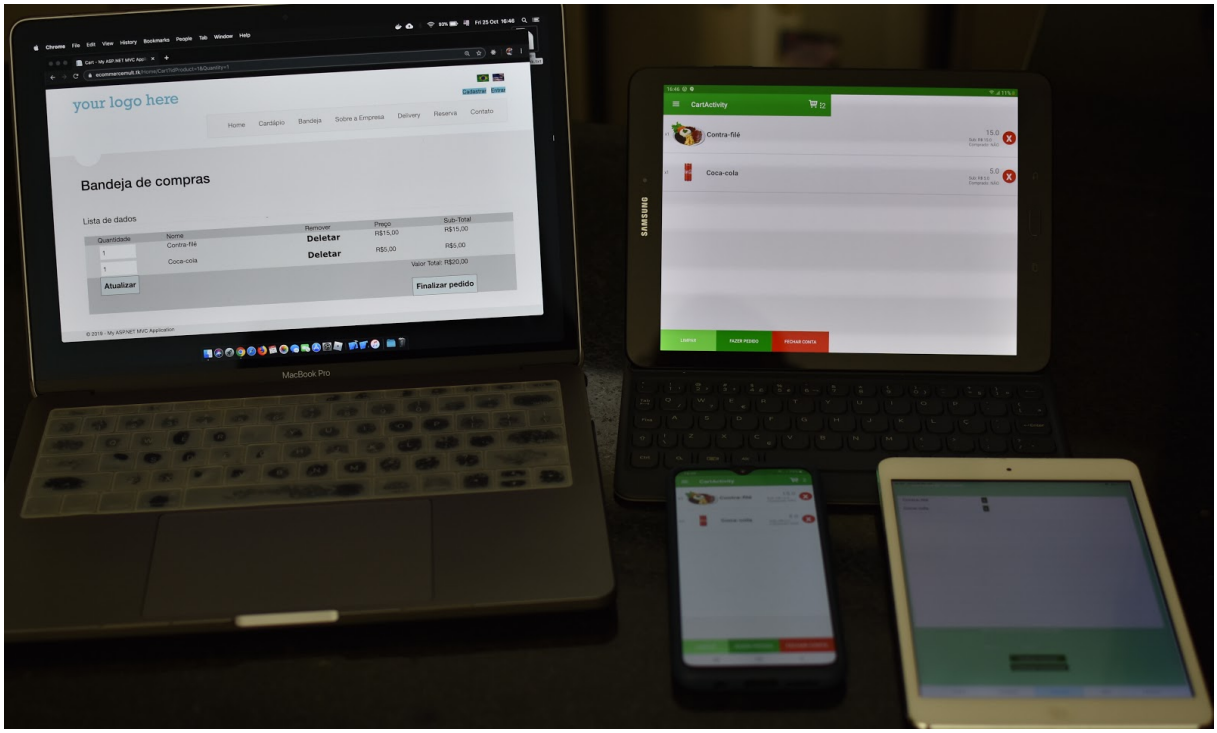

Figura 28 – Tela do carrinho de compras sendo exibida em quatro dispositivos reais: computador *desktop*, dois *tablets* (Android e iOS) e um *smartphone* (Android).

linha 3 o método `populateList` popula a lista de dados para a camada de visualização, na linha 12 o método `insertDataOrderFinalized` insere outros dados às listas, como informações dos clientes, por exemplo. Já nas linhas 26 e 44 estão os métodos responsáveis em alterar o status do pedido para a administração interna. Nessa listagem também é possível ver a utilização da função detalhe global `ControllerDetailsGrid` (linha 2) responsável por gerar as funções de inicialização em cada plataforma e enviar a lista de dados para a camada de visualização. Nesse caso, os detalhes técnicos referentes a cada plataforma ficam implementados nos modelos de plataformas, permitindo a extensão da abordagem sempre que necessário e elevando o nível de abstração da linguagem da abordagem.

```
1 class OrderFinalizedAdm{
2 usesGlobalDetails <OrderFinalized, OrderFinalizedAdapterConf,
    OrderFinalizedControllerList, OrderFinalizedConfMethod>
    ControllerDetailsGrid
3    operation populateList(idOrderedFinalized: int, statusOrdered:
     int): void {
4     objOrderFinalizedDAO: OrderFinalizedDAO
5     listOrderFinalized: list<OrderFinalized>
6     ord: string
7     ord := "Order by id desc"
8     listOrderFinalized := global objOrderFinalizedDAO.
    SelectObject(ord)
```

```
9     adjustStatusOrder(idOrderedFinalized, statusOrdered,
    listOrderFinalized)
10    dataListOrderFinalized := insertDataListOrder(
    listOrderFinalized)
11  }
12  operation insertDataListOrder(listOrderFinalized: list<
    OrderFinalized>): list<Order> {
13    x: int
14    listOrderFin: list<OrderFinalized>
15    objUserDao: UserDAO
16    objUser: User
17    objOrderFinalized: OrderFinalized
18    For x=0,x<listOrderFinalized.size,x++ {
19      objOrderFinalized := listOrderFinalized.get(x)
20      objUser :=  global objUserDao.SelectSpecificIdObject(
    objUser, objOrderFinalized.client_idClients)
21        objOrderFinalized.setUser(objUser)
22        listOrderFin.add(objOrderFinalized)
23    }
24     return listOrderFin
25  }
26  operation adjustStatusOrder(idOrderedFinalized: int,
    statusOrdered: int, listOrderFinalized: list<OrderFinalized>):
    void{
27    result: string
28    If(idOrderedFinalized <> 0){
29      x: int
30      objOF: OrderFinalized
31      objFDao: OrderFinalizedDAO
32      For x=0,x<listOrderFinalized.size,x++ {
33          objOF := listOrderFinalized.get(x)
34          If objOF.idOrderedFinalized == idOrderedFinalized {
35            statusOr: int
36            statusOr := adjustStatus(statusOrdered)
37            objOF.setStatusOrdered(statusOr)
38            result := global objFDao.UpdateObject(objOF)
39          }
40        }
41
42    }
43  }
44  operation adjustStatus(CurrentStatusOrder: int): int{
```

```
45        x: int
46        For x=0,x<4,x++ {
47         If CurrentStatusOrder == x {
48           CurrentStatusOrder += 1
49        }}
50        return CurrentStatusOrder
51    }
52 }
53
```

Listagem 7.10 – Classe genérica dos pedidos finalizados para administração interna do restaurante.

A classe genérica apresentada na Listagem 7.10 pode ser executada em qualquer plataforma especificada no modelo de *deploy*. Isso faz com que a troca de configuração do sistema multiplataforma ocorra a qualquer momento e de forma simplificada, bastando para isso alterar o apontamento da plataforma que a classe irá rodar no modelo de *deploy*. A Figura 29 apresenta o uso do código gerado sendo executados nas plataformas web (A), Android (B) e iOS (C).

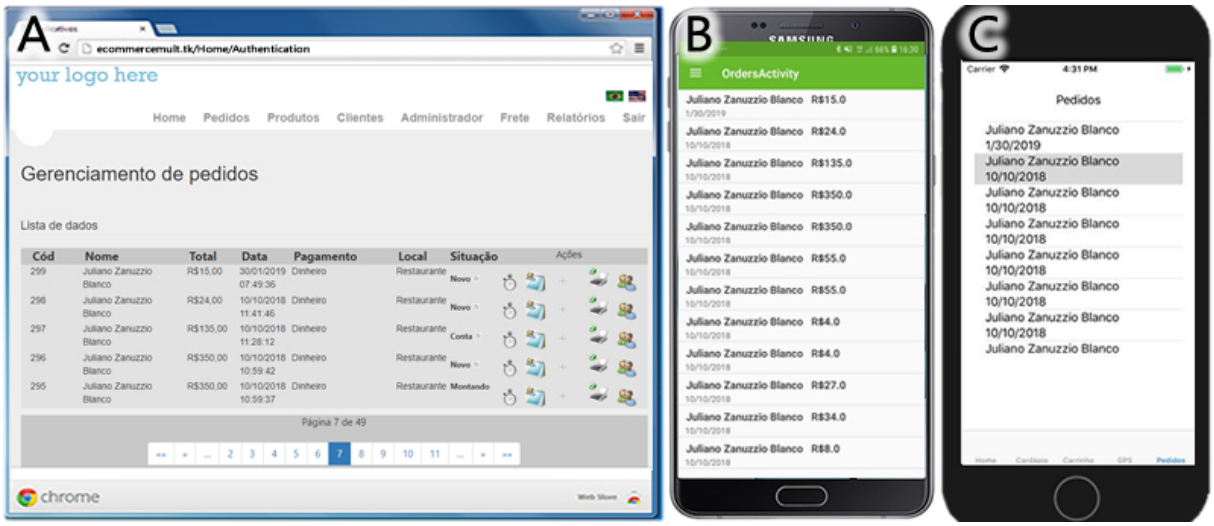

Figura 29 – Tela de pedidos nas plataformas: web (A), Android (B) e iOS (C) respectivamente.

Os modelos de plataformas podem realizar a inclusão de recursos específicos dos dispositivos. São exemplos desses recursos acelerômetro, GPS, giroscópio, entre outros. Dessa forma, foram incluídas rotinas para obtenção da localização física dos dispositivos móveis Android e iOS através da especificação de funções globais. A Listagem 7.11 apresenta a classe genérica `GPSPosition` utilizada para configurar a utilização dos dados de localização. Na linha 3 tem-se o detalhe global `ControllerGPS` usado para inserir os detalhes técnicos para a solicitação da permissão de uso dos recursos de localização.

A linha 4 apresenta a função global `getLocation` usada para adicionar os dados de localização na lista `dataRealLocationList` (linha 2) através da classe de configuração `GPSPositionGetDataLocation` (linha 13). Essa classe de configuração permite a adição dos parâmetros de localização disponíveis na biblioteca nativa de cada plataforma. Como exemplo, para a biblioteca utilizada no estudo de caso da abordagem, poderíamos adicionar os parâmetros `provider`, `accuracy`, `altitude`, além dos já utilizados `latitude` e `longitude` (linhas 14 e 15 da Listagem 7.11). Por fim, a função `useData` é utilizada para manipular a lista `dataRealLocationList` e obter os dados de localização.

```
1  class GPSPosition{
2    dataRealLocationList: list<string>
3    usesGlobalDetails <GPSPositionGetDataLocation> ControllerGPS
4    usesGlobal <GPSPositionGetDataLocation> getLocation(): list<
      string>
5    operation useData(): void{
6      x: int
7      data: string
8      For x=0,x<dataRealLocationList.size,x++ {
9        data := dataRealLocationList.get(x)
10     }
11   }
12 }
13 class GPSPositionGetDataLocation{
14   latitude: string
15   longitude: string
16 }
17
```

Listagem 7.11 – Classe genérica para obtenção da localização física do dispositivo móvel.

Os detalhes técnicos referentes à obtenção dos dados de localização foram incluídos nos modelos de plataformas, de forma que esse recurso possa ser utilizado de maneira simplificada e genérica. A inclusão de novos recursos nos modelos de plataformas os torna mais completos e pode reduzir o custo de novos desenvolvimentos de forma considerável.

A plataforma *AugmentedReality* (dispositivo que explora a RA) foi inserida ao projeto usando a GPL e a programação C# para implementação dos modelos de plataformas. Para essa plataforma foram inseridas funções globais específicas para a RA e instanciadas através da classe genérica `MainPage` desenvolvida com a linguagem proposta. A Listagem 7.12 apresenta a classe `MainPage` (linhas 1-36) que possui o código genérico do software de RA para o domínio de restaurantes e também a classe `ImageControl` (linhas 37-40), responsável por usar as funções globais `GetDiskImagesAsyn` (linha 38) e `AddImgtoCanvasAsync` (linha 39). Tais funções foram implementadas nos modelos de

plataformas *AugmentedReality* e são usadas para obter as imagens armazenadas no *RaspBerry* e projetá-las sobre os objetos do mundo real, respectivamente.

No exemplo da Listagem 7.12, são projetadas imagens a serem sobrepostas às garrafas de bebidas reais em uma prateleira de um restaurante, a cada intervalo de tempo (linhas 21-34). As projeções tem o objetivo de destacar as garrafas e inserir informações, trazendo assim um foco aos objetos do mundo real e caracterizando a realidade aumentada. Essa classe foi configurada no modelo de *deploy* para gerar código para a plataforma *AugmentedReality*.

```
1  class MainPage{
2    list_imgs: list<img>
3    teste: img
4    imageControl: ImageControl
5    time: int
6    time := 2000
7    operation MainPage(){
8      this.InitializeComponent()
9    }
10   operation AsyncTaksAsync(): void{
11     c: string
12     storageFolder: StorageFolder
13     storageFolder := KnownFolders.PicturesLibrary
14     folderName: string
15     folderName := "RALIBRARY"
16     storageFolder := storageFolder.GetFolderAsync(folderName)
17     list_imgs := imageControl.GetDiskImagesAsync(storageFolder)
18     contador: int
19     contador := 0
20     getImg: img
21     While true {
22       If contador < list_imgs.size {
23         getImg := list_imgs[contador]
24         imageControl.AddImgtoCanvasAsync(canvas, getImg)
25       }Else{
26         contador := 0
27         If contador>0 {
28           getImg := list_imgs[contador]
29           imageControl.AddImgtoCanvasAsync(canvas, getImg)
30         }
31       }
32       contador++
33       Task.Delay(time)
```

```
34      }
35    }
36 }
37 class ImageControl{
38    usesGlobal GetDiskImagesAsync(picturesFolder: StorageFolder):
       list<img>
39    usesGlobal AddImgtoCanvasAsync(canvas: Canvas, image_: img):
       Task
40 }
41
```

Listagem 7.12 – Classe genérica para a plataforma de RA através da abordagem.

7.3 Resultados e Discussão

Com a abordagem desenvolvida e em execução, a terceira etapa da prova de conceito foi realizada, visando avaliar as sete contribuições elencadas para esta tese.

Constatou-se, na etapa anterior, que foi possível migrar toda a estrutura, comportamento e regras de negócio da aplicação original para a abordagem, utilizando a GPL. Esse esforço incluiu suporte ao estilo MVC, persistência de dados, comunicação via *sockets* e a lógica do comércio eletrônico. Dessa forma, constatou-se que a GPL consegue fornecer um único ambiente para modelar os conceitos de domínio de forma independente de plataforma. Tal evidência é um indício de que a abordagem alcançou a contribuição **C1** (modelagem dos conceitos de domínio em alto nível de abstração e em um único ambiente). Além disso, não houveram funções que não puderam ser especificadas pela abordagem, atendendo a **M1** e, também, não houveram funções que foram especificadas e geradas através da abordagem e não foram executadas com sucesso nos dispositivos reais e de acordo com o esperado, atendendo a **M2** definida no Capítulo 6 para essa avaliação.

A **M4** (Lista de contribuições que foram entregues pela abordagem) é apresentada e discutida ao longo dessa seção de resultados, apresentando evidências de grande parte das contribuições (**C**) elencadas para esta pesquisa.

A Tabela 9, apresentada no inicio do capítulo, mostrou as classes comuns entre as plataformas utilizadas no sistema multiplataforma desenvolvido. Para essas classes, foram gerados os códigos correspondentes para as quatro plataformas através das especificações realizadas na abordagem e inseridos nos ambientes de desenvolvimento para a realização de testes. Tais códigos foram compilados e testados nos ambientes de produção de cada plataforma para validar a geração correta do código fonte. Assim como apresentado na seção anterior (Seção 7.2), os códigos gerados foram executados com sucesso e puderam ser utilizados para realizar algumas análises da abordagem.

A Tabela 13 mostra um comparativo do total de linhas gastas para modelar, através do processo *ad-hoc* (mostrados na Seção 7.1) e através da abordagem, os conceitos semelhantes entre as plataformas usadas no sistema. Nessa tabela é possível visualizar a redução no custo de desenvolvimento que a abordagem proporcionou, aproveitando os conceitos semelhantes. Essa tabela apresenta também os dados referente a **M3** (Diferença entre LOC nos desenvolvimento *ad-hoc* e através do uso da abordagem e lista de conceitos que puderam ser aproveitados entre as plataformas) definida para essa avaliação. O resultado mostrou que há um total de 215 LOC gastas para modelar as classes semelhantes entre as plataformas web e móvel (Android e iOS) por meio da GPL, contra as 3658 linhas gastas no desenvolvimento de forma *ad-hoc*. Essa diferença representa uma redução de cerca de 17 vezes ao usar a abordagem, comprovando a entrega da contribuição **C7** (redução de custos de desenvolvimento e manutenção) pela abordagem.

Nota-se que além da redução de esforço dentro de uma mesma plataforma através da abordagem, foi possível também reutilizar o esforço entre plataformas diferentes, indicando a entrega da contribuição **C2** (aproveitamento dos conceitos similares). No exemplo da Tabela 13, as 3658 linhas de código referentes ao que é comum entre as plataformas web e móvel foi modelada de uma única vez, de forma genérica e independente de plataforma através da GPL. A modelagem por meio da abordagem considerou que os modelos de plataformas já estavam preparados e não foram contabilizadas. Uma outra análise considerando os modelos de plataformas é apresentada a seguir.

Tabela 13 – Análise do número de LOC dos conceitos comuns especificadas através da GPL.

Plataformas	**LOC**	**Classes comuns**
Web e Móvel	*Ad-hoc*: 3658 Abordagem: 215	Order, Product, User, OrderFinalizaded, OrderDAO, ProductDAO, UserDAO, OrderFinalizadedDAO
Web e RA	*Ad-hoc*: 643 Abordagem: 41	Product, ProductDAO
Móvel e RA	*Ad-hoc*: 1139 Abordagem: 112	Product, ProductDAO, TCPUser e TCPServer
Web, Móvel e RA	*Ad-hoc*: 924 Abordagem: 41	Procuct, ProductDAO

A Tabela 14 mostra uma comparação em termos de LOC (linhas de código), sem e com a abordagem, excluindo a camada *View* do estilo arquitetural MVC, pois ela não foi implementada na prova de conceito. Os dois zeros na tabela são explicados da seguinte forma: 1) não há código independente de plataforma sem a abordagem, pois ele está escrito inteiramente na linguagem da plataforma e não pode ser reutilizado nas outras plataformas (plataformas Android, iOS e web usam linguagens diferentes na prova de conceito); e 2) também não há código independente de plataforma escrito em uma linguagem nativa com

a abordagem, pois isso não faz sentido.

Tabela 14 – Análise da LOC para a prova de conceito.

		Sem a abordagem	Com a abordagem	
			GPL	Nativo
Código específico de plataforma	Android	1204	16	207
	iOS	1503	16	157
	Web	1012	10	97
	AR	195	55	89
Código independente de plataforma		0	238	0
Total		3719	335	550
			885	

Ao comparar o tamanho total das duas versões do sistema (última linha da Tabela 14), há uma grande redução em termos de LOC. Isso é esperado, pois as funções CRUD se tornaram genéricas e reutilizadas para todas as classes persistentes. Essa observação indica que a contribuição **C7** (redução de custos de desenvolvimento e manutenção) está sendo entregue pela abordagem. Essa evidência também é usada como resposta a **M3**, definida no Capítulo 6 para essa avaliação.

A análise LOC também indica que a reutilização de conceitos semelhantes entre plataformas através da modelagem é possível com a abordagem (contribuição **C2**). Cada classe foi implantada em duas ou três plataformas. Isso também contribuiu para a redução do LOC.

Finalmente, a análise do LOC indica que a abordagem leva a um melhor foco nos conceitos do domínio. Embora todas as 3719 LOC do sistema original combinem conceitos genéricos de domínio e detalhes de plataforma, com a abordagem existem 238 linhas de código que correspondem exclusivamente a conceitos de alto nível. As demais linhas de código, criadas usando a GPL e o código nativo, concentram-se apenas nos detalhes da plataforma. Essa separação ajuda a aumentar o nível de abstração no qual o engenheiro de software pode trabalhar (contribuição **C1**).

Além das evidências para as contribuições **C1**, **C2** e **C7**, a prova de conceito mostrou que é possível integrar novas plataformas, como o caso do sistema para o dispositivo de RA. Tal sistema pôde ser especificado em alto nível através da GPL e suas dependências implementadas nos modelos de plataformas, o qual poderá ser reutilizados por outras classes. Dessa forma, constatou-se evidência para a **C4** na abordagem, referente à inclusão de novas plataformas na abordagem com menor impacto no código.

Outra evidência constatada é a distribuição de funcionalidades, apresentada de duas formas durante a prova de conceito. A primeira foi nas Listagens 7.7 e 7.8, onde foram especificadas, através da GPL, duas formas diferentes de persistência para a função carrinho de compras (banco de dados e memória). A troca entre as formas de persistência é

feita diretamente no modelo de *deploy*. A segunda forma é a escolha de qual plataforma irá executar uma determinada função apenas configurando o modelo de *deploy*. O exemplo mostrado, através da Listagem 7.10, mostra os pedidos recebidos sendo gerados para outras plataforma, além da web. Esses exemplos mostram indícios da contribuição **C3** (redução do custo na configuração de um sistema multiplataforma).

As implementações nos modelos de plataformas promovidas para essa avaliação podem ser reutilizadas em sistemas do mesmo ou diferentes domínios. São os casos das implementações da rotinas de CRUD, GPS, gerenciamento de dados para a camada *Controller*, entre outros. Tais implementações apontam evidências para a possibilidade de reutilização de códigos entre domínios, referente a **C6**.

As especificações feitas por meio da GPL da avaliação apresentada neste capítulo está disponível no endereço `https://github.com/JulianoZ/GPLAbordagemDSDM`, assim como toda a codificação gerada de forma automática para as linguagens Java, C# e Swift.

7.3.1 Ameaças à validade

Existem alguns aspectos que devem ser discutidos em relação a possíveis ameaças da validade nos estudos.

A prova de conceito foi realizada pelos pesquisadores, o que poderia ter introduzido viés. Para reduzir esse viés, nenhuma opinião ou dados relacionados à curva de aprendizado foram levados em consideração neste estudo. Somente a viabilidade da abordagem, seus modelos e a capacidade de incluir uma plataforma anteriormente não suportada foram testados na prova de conceito.

Além disso, toda codificação gerada foi testada nos IDEs nativos de cada plataforma e todo código foi executado sem apresentação de erros. Durante a avaliação, alguns erros de execução foram apontados nos IDEs nativos. Para esses casos, a especificação GPL era corrigida e os códigos gerados novamente, até a execução correta nos ambientes nativos.

8 Segunda avaliação: análise por especialistas

Uma segunda avaliação foi conduzida com especialistas (**E**) visando avaliar as contribuições da abordagem multiplataforma proposta nesta tese. Para esse estudo foram selecionados especialistas com grande experiência em tecnologias web e móveis (no mínimo 10 anos) e desejável experiência com *frameworks* multiplataforma para servir de base de comparação. Foram selecionados 5 especialistas com esses critérios atuantes em empresas e/ou academia que são conhecidos do autor da tese. Entre os especialistas selecionados, dois deles atuam em empresas e possuem ampla experiência na área de desenvolvimento de software (**E1 e E2**). Outros dois possuem experiência basicamente acadêmica em pesquisa, sendo um doutorando e outro professor doutor em um centro de tecnologia em computação (**E3 e E4**). O quinto especialista é doutor com experiência na indústria de computação na área de programação e atualmente professor e pesquisador em um centro de tecnologia (**E5**). Todos eles possuem experiência em diversas linguagens de programação, arquiteturas de sistemas e banco de dados.

A avaliação consistiu em três fases. Na primeira, o especialista precisava aprender o modo de funcionamento da abordagem através de uma série de materiais disponíveis em um repositório compartilhado. Entre os materiais, estavam um guia de orientação inicial, um documento introdutório contendo de forma ordenada os passos para a utilização da abordagem (*OverView*), alguns vídeos explicativos, envolvendo a instalação, uso e detalhes técnicos, um arquivo com a documentação oficial da abordagem, um arquivo contendo as tarefas e um questionário final usado como roteiro para a entrevista, aplicado após o termino da avaliação. Os arquivos com os materiais da avaliação podem ser acessados através do endereço virtual https://drive.google.com/drive/folders/1Kvv8p5tR_konbjYrzsk2BkDhZF6KBPP0[1], da forma que foram disponibilizados para os especialistas.

Ainda na primeira fase, os especialistas tiveram acesso aos executáveis da abordagem contendo trechos de códigos exemplificando cada modelo junto com seus respectivos comentários explicativos no código. Entre os trechos, estão códigos referentes aos modelos de *deploy*, modelo de plataformas, declaração de funções globais e variáveis e por último a modelagem GPL, com exemplos dos conceitos do domínio de comércio eletrônico modelados.

Após o entendimento da abordagem (Fase 1), o especialista anotou o tempo gasto e

[1] Alternativa para acesso do material da avaliação: https://github.com/JulianoZ/MateriaisExperimentoEspecialista.

se direcionou para a segunda fase da avaliação. A avaliação foi orientada a tarefas, visando representar o desenvolvimento de diferentes partes de um sistema multiplataforma. As especificações de cada tarefa (**T**) estão descritas a seguir:

T1. Essa tarefa explora as questões da camada *Model* da arquitetura MVC, solicitando aos especialistas a modelagem de funções específicas de persistência e regras de negócio de forma genérica. A configuração da geração de código no modelo de *deploy* também foi incluída nessa tarefa;

T2. Nessa tarefa, o especialista foi motivado a fazer com que certas funções executadas em uma plataforma específica sejam executadas também em outras através do modelo de *deploy*. A troca na forma de funcionamento de certas funções também foi solicitada nessa tarefa, através de ajustes no modelo de *deploy*. E, por fim, solicitou-se ao especialista a exploração do recurso GPS na camada *Contoller* (MVC) e o gerenciamento e a geração de código no modelo de *deploy*;

T3. O especialista foi solicitado a estender o atendimento da abordagem para outros bancos de dados (BD) não disponíveis nos modelos de plataformas. Para tal, foi necessário criar novas funções globais e implementar as regras de conexão nos modelos de plataformas, deixando assim a abordagem preparada para lidar com novos BD de forma genérica por meio da GPL;

T4. O especialista foi motivado a criar um novo projeto para um domínio diferente do abordado nos exemplos apresentados na abordagem. Todas as funções já implementadas nos modelos de plataformas, assim como os códigos genéricos modelados na GPL, puderam ser reutilizados do domínio anterior. A ideia é analisar o ganho no desenvolvimento através do reúso de códigos; e

T5. Na última tarefa, o especialista precisava declarar a plataforma *smartwatch* na abordagem para a inclusão no sistema. Para essa nova plataforma, foi solicitado ao especialista implementar a função global de exibição de produtos especificada para as plataformas Android e iOS. Após a implementação da função, o especialista configurava a geração de código no modelo de *deploy* e analisava o código gerado para a plataforma *smartwatch*.

O raciocínio para a definição dessas tarefas foi o seguinte: tentou-se estimular a realização de tarefas típicas de um desenvolvimento multiplataforma, com base na experiência dos pesquisadores do grupo de pesquisa. Tentou-se também cobrir todas as contribuições esperadas da abordagem, de forma a utilizar todos os seus recursos disponíveis. A tabela 15 apresenta as contribuições exploradas por cada tarefa.

Tabela 15 – Contribuições exploradas por cada tarefa na avaliação.

Tarefas	Contribuições
T1	**C2**. Aproveitamento dos conceitos similares entre as plataformas na modelagem. **C1**. Modelagem em alto nível de abstração e em um único ambiente. **C7**. Redução do custo do desenvolvimento multiplataforma.
T2	**C3**. Alteração das plataformas utilizadas no sistema com baixo custo e distribuição de funcionalidades entre as plataformas. Junto com as **C1** e **C7**.
T3	**C5**. Extensão da abordagem através dos modelos de plataformas. Junto com as **C1** e **C7**.
T4	**C6**. Reúso das funções globais e conceitos em outro domínio. Junto com as **C1** e **C7**.
T5	**C4**. Inclusão de novas plataformas no sistema com menor impacto no custo. Junto com as **C1** e **C7**.

Além das contribuições apontadas para cada tarefa na Tabela 15, também existem as contribuições comuns a mais que uma tarefa, que são a **C1** e a **C7**. Essas contribuições são cobertas indiretamente por todas as tarefas. As tarefas foram modeladas seguindo o estilo arquitetural MVC, para manter a organização da avaliação e utilizar a estratégia empregada no sistema de comércio eletrônico para o domínio de restaurante da prova de conceito.

Ao final da avaliação (Fase 3) cada especialista (**E**) foi submetido a uma entrevista detalhada para obter dados que ajudam a entender os aspectos prós e contras da abordagem. O projeto da avaliação contendo os detalhes da entrevista foi submetido para avaliação junto à plataforma Brasil[2] e aprovada para a execução. Cada especialista precisou assinar um termo de consentimento que continha explicações sobre o projeto e os termos legais para a participação da pesquisa. As entrevistas foram orientadas através de um roteiro com 11 questões (**Q**). Cada questão explorava aspectos importantes para análise da abordagem, que serviram de base para verificar se as contribuições elencadas para esse projeto foram alcançadas e em qual nível.

8.1 Observações dos especialistas

Nesta seção são apresentadas as observações feitas pelos especialistas. Nas discussões que se seguem, todas as contribuições (**C**) elencadas na Tabela 6 estão demarcadas nas observações dos especialistas. **C+** aponta evidências positivas em relação a contribuição indicada e **C-** aponta suas limitações. As observações dos especialistas foram utilizadas para apontar indícios prós e contras sobre as contribuições elencadas para esta tese.

[2] http://plataformabrasil.saude.gov.br/

A análise das evidências foi feita com base na leitura das transcrições das entrevistas, e foi realizada em par pelos pesquisadores desta tese (orientador e aluno). Os resultados foram confrontados e discutidos até chegar em um senso comum. Além disso, as análises das observações foram realizadas em duplicidade por um dos pesquisadores. Salienta-se que cada contribuição só é somada (**C+** ou **C-**) se os argumentos forem distintos em cada questão (**Q**). Tais cuidados foram tomados visando evitar de contabilizar evidências no caso de pessoas que são repetitivas em sua fala.

A primeira questão (**Q1**) foi: **Qual a sua opinião sobre a curva de aprendizado da abordagem?** Essa questão foi de múltipla escolha com as seguintes alternativas: 1) Muito fácil de aprender. Precisei olhar rapidamente a documentação; 2) Fácil de aprender o funcionamento. Precisei me ater um pouco à documentação; 3) Tempo mediano para aprender. Precisei analisar os exemplos com pouco de atenção; 4) Difícil de aprender. Precisei ficar um tempo razoável analisando a documentação; e 5) Muito difícil o entendimento. A documentação não foi suficiente para o entendimento devido à complexidade da abordagem.

Essa questão obteve informações sobre a curva de aprendizado da arquitetura da abordagem, envolvendo os modelos de *deploy* e plataforma, as declarações (variáveis e funções globais) e a GPL, representando a primeira fase da avaliação. A Tabela 16 apresenta o tempo médio para o entendimento da abordagem e as respectivas opções e respostas dos especialistas sobre essa questão.

Tabela 16 – Curva de aprendizado da abordagem.

E	**Respostas**
E1	**Entendimento da abordagem**: 6 horas. **Parciais por Tarefa (T)**: **T1**- 1h00, **T2**- 2h20, **T3**- 0h40, **T4**- 0h45 e **T5**- 1h30. **Resposta**: Fácil de aprender, mas teve que olhar a documentação. **Justificativa**: Pelo fato de ser uma arquitetura nova, porém intuitiva. Após assistir o vídeo sobre a arquitetura foi tranquilo entender o funcionamento da abordagem e começar a trabalhar.
E2	**Entendimento da abordagem**: 4 horas. **Parciais por Tarefa (T)**: **T1**- 0h40, **T2**- 3h40, **T3**- 0h45, **T4**- 0h30 e **T5**- 4h00. **Resposta**: Tempo mediano para aprender. Precisei analisar os exemplos com pouco de atenção para entender a estrutura. **Justificativa**: Precisei dar uma certa atenção a declaração do modelo de *deploy*, no direcionamento das classes para as plataformas visando a geração das linguagens, assim como o entendimento das funções globais e os seus detalhes.

E3	**Entendimento da abordagem**: 7 horas. **Parciais por Tarefa (T)**: **T1**- 0h25, **T2**- 1h20, **T3**- 0h35, **T4**- 0h25 e **T5**- 1h30. **Resposta**: Fácil de aprender o funcionamento. Precisei me ater um pouco na documentação. **Justificativa**: Achei a arquitetura bem intuitiva e interessante pelo fato de haver separações entre as partes de modelagem genérica (GPL) e as partes com códigos específicos de plataformas (**C1+**).
E4	**Entendimento da abordagem**: 8 horas. **Parciais por Tarefa (T)**: **T1**- 2h30, **T2**- 2h20, **T3**- 2h30, **T4**- 0h30 e **T5**- 2h00. **Resposta**: Tempo mediano para aprender a abordagem e realizar as atividades com sucesso. **Justificativa**: A documentação disponibilizada ajudou bastante, consegui ver que possui a geração automática então já fui fazendo de forma cuidadosa pois sabia que era um código genérico e que fazia acesso aos detalhes nos modelos de plataformas.
E5	**Entendimento da abordagem**: 7 horas. **Parciais por Tarefa (T)**: **T1**- 2h00, **T2**- 2h55, **T3**- 1h30, **T4**- 1h30 e **T5**- 1h00. **Resposta**: Tempo mediano para aprender, precisei testar e olhar os exemplos na abordagem, mas logo entendi a separação entre os códigos genéricos e específicos. **Justificativa**: Os exemplos da própria abordagem foram suficiente para aprender. A documentação está boa, mas faltou a documentação dos exemplos da abordagem. Logo entendi a separação entre os códigos genéricos e específicos. (**C1+**).

Em resumo, segundo os especialistas, o tempo de aprendizado foi baixo para os especialistas **E1** e **E3** e mediano para o restante dos entrevistados. O detalhe que teve maior percepção pelos especialistas foi a separação entre a programação específica de plataforma (modelos de plataformas) e a programação genérica (GPL) (**E3**, **E4** e **E5**).

A segunda questão (**Q2**) foi: **Qual é a sua opinião sobre a linguagem de programação (GPL) usada na abordagem? A linguagem é fácil ou difícil de usar? É intuitivo (A)? Quão complexa é a linguagem (B)?** Nessa questão os especialistas apontaram aspectos referentes ao aumento do nível de abstração e a facilidade de se pensar de forma multiplataforma em um único ambiente. A Tabela 17 apresenta os relatos que evidenciaram essa questão e indicaram que a abordagem tem o potencial de aumentar a abstração na modelagem dos conceitos do domínio e tratar o desenvolvimento em um único ambiente (Item B, **E1**, **E2**, **E3** e **E4**).

Outro ponto percebido através da **Q2** foi a redução da percepção de complexidade da linguagem pelos especialistas através do isolamento dos detalhes técnicos de cada plataforma nos modelos de plataformas (Item B, todos os **E**). O ganho de produtividade com o uso da abordagem foi apontado por dois especialistas (**E3** e **E5**) como a longo prazo, caso o desenvolvedor precise implementar de forma manual todas as funções globais para diferentes plataformas. Inclusive, por se tratar de um editor simples, a implementação das funções globais na abordagem pode ser muito custosa, já que ela não possui muitos recursos (**E5**). De fato, a abordagem tem ganhos a longo prazo caso as funções globais precisem ser todas implementadas e realmente a abordagem não é o local mais indicado para implementá-las, uma vez que elas precisam ser criadas e testadas em seus ambientes nativos para melhorar a confiabilidade. Mas uma vez implementadas as funções de forma adequada na plataforma, as funções podem prover ganhos consideráveis no desenvolvimento multiplataforma, conforme os relatos do **E4** e **E5**. Adicionalmente, caso sejam utilizados os *templates* Velocity, tem-se um ganho adicional devido à capacidade de geração de código mais customizável.

Continuando com a analise do **E5**, ele apontou alguns recursos que não conseguiu utilizar na GPL, assim como: Interface java, classes abstratas, classes e métodos estáticos, modificadores de acesso para métodos e atributos (*public, private e protected*) e modificador final. Tais recursos não foram implementados na gramática da GPL, por se tratar de um protótipo, mas eles estão inseridos como trabalhos futuros.

Tabela 17 – Avaliação da programação GPL: facilidade (**A**), intuitividade (**A**) e complexidade (**B**).

E	**Respostas**
E1	**A**. Foi uma linguagem intuitiva, uma vez que já conhecia um pouco de *Swift* e Java. **B**. Complexidade baixa, devido ao fato de ter os modelos de plataformas com os detalhes técnicos de cada dispositivo, fez com que a linguagem GPL tenha apenas as regras de negócio e conceitos do domínio (**C1+**).

E2	**A**. A Linguagem é parecida com Java e para mim foi intuitiva. Senti um pouco de dificuldade com *operation*. Pois tem algumas declarações sem vírgula e a pontuação é um pouco diferente. Faltou um pouco de referência na sintaxe da linguagem (**C1-**). **B**. A complexidade é baixa devido a maior parte dos detalhes se concentrarem nos modelos de plataformas (**C1+**). Poderia ter algumas melhorias, assim como formatação dos códigos gerados, com melhor endentação para ficar mais legível e casos de testes para validação. Seria interessante também, acrescentar mais recursos para a GPL, como validação de sintaxe e *code completion*, tanto nos modelos de plataformas como na programação da GPL (**C7-**). Mas, dá para entender bem o código gerado e acrescentar novos detalhes e funções, bem legível. É importante pois ele já gera muita coisa pronta no sistema (**C4+** e **C5+**).
E3	**A**. Achei que a curva de aprendizado da GPL não é tão alta pelo fato de utilizar padrões de sintaxes próximas das linguagens que domínio e também palavras reservadas intuitivas para declarar modelos, variáveis e comandos de programação. **B**. A abordagem aumenta o nível de abstração e reduz consideravelmente as complexidades. Isso pode ajudar a longo prazo a definição melhor dos conceitos do domínio, trabalhando a modelagem do sistema (**C1+**). A abordagem separa o desenvolvimento entre os vários níveis de abstração envolvidos no processo. Porém, acredito que o ganho venha a longo prazo, com os modelos de plataformas e as funções globais quase todas implementadas (**C7-**). Isso pode ajudar a longo prazo, a ideia de poder pensar em outro nível de abstração (**C1+**).
E4	**A**. Achei a linguagem intuitiva por ter similaridade com a linguagem Scala[3] e isso ajudou muito. **B**. Uma vez que adiciona os detalhes técnicos nos modelos de plataformas a GPL fica com alto nível de abstração (**C1+**). Eu tenho problema em perder tempo em tarefas braçais e recorrentes. Uma ferramenta como essa, pode resolver parte desses problemas, reduzindo o esforço braçal dessas atividades. (**C7+**).

[3] https://docs.scala-lang.org/pt-br/tour/tour-of-scala.html

E5	**A**. Linguagem intuitiva. Senti falta de alguns detalhes de programação que não foi possível inserir na GPL, vi que herança está disponível, mas Interface java, classes abstrata, classes e métodos estáticos, modificadores de acesso para métodos e atributos (*public, private, protected*) e final não consegui usar (**C1-**). **B**. A complexidade foi baixa. Se as funções globais estiverem prontas (**C7-**), isso iria aumentar muito a produtividade (**C7+**). Se a pessoa tiver que implementar todas as funções globais a produtividade pode cair muito. Inclusive se tiver que programar no editor, que não possui nenhuma verificação de erro, isso pode ser uma limitação. As pessoas vão querer criar no próprio IDE ou vão precisar implementar nos ambientes de produção, testar e depois passar para os modelos de plataformas e nesse caso o ganho pode ser a longo prazo (**C1-**).

A questão **Q3** foi: **Qual é a sua opinião sobre os modelos de plataformas fornecidos pela abordagem? É possível incluir todos os detalhes técnicos por meio de funções globais (A)? É possível reutilizar detalhes técnicos em sistemas de diferentes domínios (B)? É possível estender a abordagem por meio de modelos de plataforma (C)?** Essa questão avaliou os modelos de plataformas em três pontos, o primeiro (**A**) é a capacidade dos modelos em conseguir implementar funções com a maior parte dos detalhes técnicos das plataformas necessários para um sistema comercial. O segundo ponto (**B**) verificou se os modelos de plataformas são facilmente reutilizados em diferentes domínios e o último ponto (**C**) verificou qual o nível de extensão da abordagem permitido pelos modelos de plataformas. A Tabela 18 apresenta as respostas dos especialistas.

Tabela 18 – Avaliação dos modelos de plataforma. É possível a: inclusão dos detalhes técnicos (**A**), reutilização dos modelos (**B**) e extensão da abordagem(**C**).

E	**Respostas**
E1	**A**. Através das avaliações, acredito que seja possível inserir todos ou a maior parte dos detalhes técnicos, assim como foi feito com o GPS (**C5+**). **B**. É possível reaproveitar os códigos dos modelos de plataformas em outros domínios assim como foi feito na tarefa 4 (**C6+**). **C**. A abordagem é aberta para adição de novas funções. Como o caso da conexão SQL, nós fizemos com que a abordagem pudesse suportar um outro tipo de banco de dados, além do que já estava definido através das funções globais nos modelos de plataformas (**C4+** e **C5+**).

E2	**A**. Achei os modelos de plataformas bons, ainda mais pelo fato de ser genérica, você passa o objeto e a abordagem "se virá" para gerar. Acho que consegue adicionar qualquer função técnica nos modelos de plataformas. Se funcionou o exemplo do GPS no Android e iOS, então acredito que funcionária para qualquer outro recurso (**C4+** e **C5+**). Com a minha experiência em Flutter[4] (google), tem coisas que funciona no Android e que no iOS precisaria de novas configurações para funcionar. Como o Flutter gera códigos prontos sem possibilidade de mexer do fonte gerado, isso pode ser um problema do *framework* que a sua abordagem não teria, pelo fato de poder estender os modelos de plataformas e trabalhar com o código legível gerado. Isso permite com que o desenvolvedor possa customizar a geração de código melhorando o desempenho e incluindo outros detalhes (**C5+**). **B**. Acredito que é possível sim reaproveitar os códigos e é bem prático devido ao fato que é genérico (**C6+**). Você consegue alterar detalhes de especificidades de forma bem tranquila, bastando para isso informar a entidade atual. **C**. É possível estender e pareceu muito simples fazer essa inclusão de novos detalhes. Foi simples fazer a conexão e uso de outros bancos e a inclusão de outros detalhes técnicos, você só troca os detalhes técnicos nas funções globais e pode utilizar a mesma linguagem ou a lógica modelada na GPL (**C5+** e **C7+**). Porém, os códigos implementados nas funções globais devem seguir a sintaxe de cada plataforma e isso é melhor desenvolvido nos ambientes nativos de produção. Programar diretamente na abordagem não é o melhor lugar devido a falta de recursos, como testes e *code completion* , tanto nos modelos de plataformas como na programação da GPL (**C1-**).

[4] https://flutter.dev/

E3	**A**. Acredito que poderia adicionar a maior parte dos detalhes técnicos (**C5+**). Porém, talvez teria algumas limitações em geração de códigos em sistemas de grande escala, pensando em toda ambientação do sistema junto às bibliotecas e dependências (**C1-** e **C3-**). Precisaria talvez ter algumas etapas de testes incluídas, podendo incluir etapas de validação (**C5-**). **B**. Os modelos de plataformas podem ser reutilizados para outros domínios (**C6+**). Partindo do princípio que se use o estilo arquitetural MVC, é possível trocar o domínio e usar. Principalmente domínios vinculados a sistemas de informação. **C**. A abordagem abre portas para poder evoluir um sistema sem ficar preso em uma *framework* específico (**C5+**) e oferece uma gama muito grande de possibilidades aumentando a produtividade (**C7+**) a longo prazo (**C7-**). Podemos inclusive gerenciar papéis (**C1+**) no desenvolvimento de uma forma melhor com a abordagem e ter benefícios no futuro. Partindo do princípio que cada vez temos software que precisa lidar com diferentes plataformas a extensão é muito importante e fundamental para a evolução livre do sistema. Hoje é difícil desenvolver algo homogêneo e precisamos adequar o sistema as novas tecnologias/plataformas que estão surgindo e a extensão provida pela abordagem pode ajudar bastante nisso (**C4+**, **C5+**).
E4	**A**. Acredito que seja possível adicionar todos os detalhes técnicos nos modelos de plataformas. Uma vez que consegui, através dos exemplos aplicados, inserir detalhes técnicos e fazer funcionar com êxito e dessa forma não consigo pensar em um cenário que que não pudesse ser implementados (**C4+** e **C5+**). A arquitetura me deu liberdade para inserir os detalhes e funções nos modelos de plataformas sem problemas. **B**. Acho que é possível reutilizar e reaproveitar as funções globais dos modelos de plataformas em outros domínios, levando em consideração o exemplo do `SelectObject` que usamos nas tarefas (**C6+**). **C**. É possível estender a abordagem e acho inclusive que é um dos pontos fortes do projeto. Na tarefa inserimos uma nova conexão e trabalhamos com outro banco de forma fácil por usar linguagens conhecidas (**C4+** e **C5+**).

E5	**A**. Aparentemente é possível adicionar todos os detalhes, não consigo pensar em algo que não (**C4+** e **C5+**). **B**. Acredito que os modelos de plataformas podem ser reutilizados em sistemas de diferentes domínios sem problemas (**C6+**). **C**. É possível estender a abordagem da forma como foi construída e acho que não é difícil. Achei complicado por ser a primeira vez, mas depois foi tranquilo (**C4+** e **C5+**). Se o desenvolvedor precisar implementar muitas funções globais vai complicar a produtividade (**C7-**), pois não vai conseguir ver os erros porque ele não consegue testar (**C5-**).

A terceira questão permitiu avaliar de forma detalhada os modelos de plataformas da abordagem. De forma geral, os especialistas apontaram de forma favorável a possibilidade de inclusão dos detalhes técnicos das plataformas aumentando o nível de abstração da GPL (Item A, todos os **E**). Apontaram também a capacidade de reúso e extensão dos modelos de plataformas em sistemas de diferentes domínios (Item B, todos os **E**), mostrando evidências dessas contribuições (**C6+**) providas pelas abordagem. Além disso, outro benefício apontado foi o gerenciamento das funções da equipe de desenvolvimento (**E3**, item C), podendo apresentar melhor separação de tarefas de uma equipe e ganhos de produtividade a longo prazo. O **E5** apontou a dificuldade em implementar as funções globais diretamente na abordagem devido a falta de recursos para isso, confirmando a discussão realizado na **Q2**.

A questão **Q4** é um complemento da questão anterior com a diferença de analisar as funções globais, os parâmetros genéricos e a programação de customização Velocity dos modelos de plataforma. A definição da **Q4** foi: **O que você achou das funções globais e parâmetros genéricos, usados em conjunto com os modelos Velocity, para criar funções dinâmicas nos modelos de plataformas? Eles podem automatizar a geração de código (A)? Eles permitem a criação de funções dinâmicas durante a codificação (B)? Eles têm potencial para criar código personalizado para diferentes entidades (B)?** Tal análise foca em dois pontos, nível de automatização da geração de código (**A**) através das funções globais e criação de funções customizadas para diferentes entidades em tempo de geração de código (**B**), através dos parâmetros genéricos e programação Velocity. A Tabela 19 apresenta os principais trechos das respostas dos especialistas.

Tabela 19 – Avaliação dos recursos dos modelos de plataformas. Capacidade de: automatizar a geração de código (**A**) e criar funções customizadas (**B**).

E	**Respostas**

E1	**A**. Por ser funções dinâmicas, podemos usá-las para generalizar (aumento de nível de abstração da GPL (**C6+**)) a programação GPL e assim aumentar a produtividade do desenvolvimento (**C7+**). **B**. Achei bem interessante os recursos das funções globais, pois reduz o custo do desenvolvimento (**C7+**) através do reúso de código (**C6+**). Reutilizamos várias funções globais (**C2+**) na GPL e com isso automatizamos e reduzimos o tempo do desenvolvimento. Códigos gerados melhora muito a produtividade pelo fato de usar um local único para desenvolver. A abordagem permite reutilizar funções globais prontas para gerar código para diferentes plataformas e entidades (**C6+**) e usar o mesmo algoritmo genérico (GPL) para gerar para diferentes plataformas (**C2+**).
E2	**A**. Ele gera código customizado em tempo de criação de código (**C6+** e **C5+**), atendendo a diferentes entidades e sistemas e para quem tem experiência já vai vendo como está ficando, dá para ter uma ideia se está correto ou não. Dessa forma conseguimos ter grande produtividade (**C7+**). **B**. A programação Velocity dentro dos modelos de plataformas trata os parâmetros genéricos e tem o potencial de criar funções personalizadas. Esse recurso é bem interessante (**C5+** e **C6+**). A lógica fica separada e permite tratar os conceitos de forma isolada dos detalhes técnicos (**C1+**). É possível manter a lógica em alto nível na GPL, favorecendo a sua utilização em diferentes plataformas (**C2+**) e os detalhes isolados nos modelos de plataformas.
E3	**A**. Ela pode automatizar a criação de códigos e agilizar a criação de funções com grande número de atributos (**C7+**). Funções que precisam tratar muitos atributos de qualquer forma são tarefas normalmente fáceis, porém exigem grande concentração para evitar erros se tornando trabalhosas e esse pode ser um dos ganhos das funções globais. **B**. Permite a criação de funções dinâmicas pelo fato de usar a programação Velocity, para customizar códigos em tempo de geração de código (**C5+** e **C6+**), podendo agilizar o desenvolvimento (**C7+**). A abordagem pode abrir portas fazendo necessário especialistas de diferentes linguagens, podendo assim incluir mais plataformas na abordagem (**C4+**). Definindo o tipo genérico, ele pode gerar funções customizadas (**C5+** e **C6+**). A abordagem permite com que um arquiteto de software possa organizar o desenvolvimento em alto nível através da GPL (**C1+**). A abordagem pode abrir portas, fazendo necessário especialistas de diferentes linguagens (**C7-**), podendo assim incluir mais plataformas na abordagem.

E4	**A**. É possível automatizar através das funções globais e sem dúvida ganhar tempo (**C7+**). Atualmente, todos os processos de criação de software que vamos desenvolver, passa por uma geração de código, braçal, mecânico e recorrente, portanto a abordagem possui recursos para resolver esse problema a longo prazo. Eu diria que seria uma curva de ganho, quanto maior o número de projetos utilizando a plataforma, mais iria achatar essa curva, gerando benefício (**C7-**). Mesmo desenvolvimento uma vez só, o projeto vai gerar muita manutenção e dessa forma acredito que valha o esforço para o uso da plataforma (**C7+**). **B**. Customizando as funções acabamos tendo o benefício de criar funções dinâmicas que servem para diferentes entidades e classes (**C6+**). Possui o potencial de criar códigos customizados devido a arquitetura da plataforma.
E5	**A**. Ela consegue automatizar a criação de código (**C7+**). Gostei de ver essa aplicação. **B**. Permite a criação de funções dinâmicas e códigos customizados (**C4+** e **C5+**). A função global faz bastante sentido se eu for aplicar para multiplataforma, mas ainda faz sentido se for aplicar para uma plataforma específica. Isto é, para uma única plataforma tem o ganho da geração de código customizado (**C6+**) e para várias plataformas esse ganho se maximiza (**C7+**). De qualquer forma, o código das funções globais precisam ser feitos e testados nos IDEs específicos de cada plataforma antes (**C1-** e **C7-**).

Conforme a Tabela 19 é possível constatar algumas contribuições elencadas para a abordagem (**C2**, **C4**, **C5**, **C6** e **C7**) através das funções globais e seus recursos. Segundo os especialistas, as funções globais permitem a implementação de métodos e programações genéricas, permitindo a evolução da abordagem pelos próprios desenvolvedores (Item A, todos os **E**). Além disso, é possível manter a GPL genérica com poucos detalhes técnicos, mantendo o desenvolvedor mais focado nos conceitos do domínio (Item B, **E2** e **E3**).

A questão **Q5** foi: **O que você achou do modelo de *deploy*? A distribuição de funcionalidade (A). A troca ou inclusão de plataformas ao sistema, alternando plataformas para algumas funções, como o recebimento de pedidos nos exemplos (B), e o gerenciamento da geração de código por meio desse modelo (C).** A Tabela 20 apresenta as respostas dos especialistas para essa questão.

Tabela 20 – Avaliação do modelo de *deploy*. Distribuição de funcionalidades (**A**), a troca de plataformas no sistema (**B**) e o gerenciamento da geração de código (**C**).

E	**Respostas**

E1	**A**. A distribuição de funcionalidades é prática e fácil. Basta declarar e automaticamente ele já gera. O gerenciamento das funções fica bem facilitada. É possível visualizar de forma fácil para onde cada classe é direcionada e gerenciar a distribuição de funcionalidades (**C3+**). **B**. Os códigos são gerados de forma automática e isso facilita. Podemos realocar facilmente as funções a outras plataformas (**C3+**), desde que elas estejam preparadas para isso na GPL e nos modelos de plataformas (**C3-**). **C**. Fácil de gerenciar e administrar (**C3+**).
E2	**A**. Podemos usar as classes da GPL e direcioná-las para as plataformas específicas. É possível separar muito bem o que vai em cada plataforma. (**C2+** e **C3+**). **B**. A troca de plataformas e funções são bem facilitadas com o modelo de *deploy* (**C3+**). Podemos inclusive inserir outras plataformas (**C4+**). O Flutter[5] é uma ferramenta que trabalho, nele não é possível escolher o que vai ser gerado para o Android e iOS. Todos os códigos são gerados para ambas as plataformas, prejudicando um gerenciamento mais fino. **C**. Atualiza a GPL implementa a funcionalidade (*operation*) e facilmente configura o *deploy* e isso já mostra qual linguagem gerou e é possível enxergar o código gerado para verificar erros (**C3+**). O fato de poder usar o estilo arquitetural MVC é possível organizar muito bem o código. Dá uma visão muito boa de sistema multiplataforma.
E3	**A**. Eu achei muito simples, bem interessante, pois podemos deixar tudo em um único bloco e gerenciar de forma mais fácil o sistema multiplataforma . Fazendo a distribuição de funções entre as plataformas de forma dinâmica (**C3+**), além de poder delegar uma mesma classe para várias plataformas (**C2+**). **B**. A troca de plataformas pode ser feito de forma facilitada (**C3+** e **C4+**). **C**. O gerenciamento da geração de códigos fica simples pois o desenvolvedor consegue concentrar essa atividade em um único local, isto é, no modelo de *deploy* (**C3+**). No modelo de *deploy* pode ter alguém responsável em gerenciar em alto nível o desenvolvimento multiplataforma, focando melhor nos conceitos (**C1+**) e tendo uma melhor definição dos papéis das pessoas no desenvolvimento.

[5] https://flutter.dev/

E4	**A**. A questão do *deploy* foi uma das partes que fiquei mais feliz em ver. Basta acrescentar no *deploy* e a solução irá gerar os códigos para as plataformas que quisermos, isso facilita muito a distribuição de funcionalidades, a visibilidade da onde cada classe vai facilita muito o gerenciamento multiplataforma (**C3+**). **B**. Tranquilo para substituir uma plataforma no sistema, é preciso colocar os detalhes nos modelos de plataformas e depois basta fazer o apontamento. Só preciso implementar os modelos de plataformas, depois disso a troca é muito simples e rápida. A troca da plataforma A para a B é feito de forma muito natural e intuitiva (**C3+**). **C**. Muito bom o gerenciamento da geração de código para um sistema multiplataforma (**C3+**). Parece ser bom o ganho de tempo e o gerenciamento (**C7+**). A não necessidade de se preocupar com a programação braçal e recorrente é muito bom (**C1+**).
E5	**A**. É bem simples de realizar a distribuição de funcionalidades, muito fácil de ler e gerenciar as classes e plataformas (**C3+**). Desde que as funções globais estejam prontas e especificadas para cada plataforma (**C7-**). **B**. Desde que as funções estiverem prontas para as plataformas alvo, é só alterar nos modelos de *deploy*. Torna a troca e a inclusão entre as plataformas uma tarefa simples (**C3+** e **C4+**), mas é necessário implementar todas as especificidades nos modelos de plataformas para a nova plataforma e isso pode dar um certo trabalho. Os códigos deverão ser testados nos IDEs nativos antes de passar para a abordagem e isso pode gerar um custo extra (**C5-**). **C**. Achei que fica muito simples para gerenciar depois de tudo implementado. (**C3+**).

De acordo com as respostas dos especialistas na **Q5**, o modelo de *deploy* foi um dos aspectos apontados como mais interessantes da abordagem, pelo fato de permitir um grande número de possibilidades no gerenciamento de sistemas multiplataforma (Item A, todos os **E**). Entre as principais possibilidades apontadas pelos especialistas estão a distribuição de funcionalidades que permite explorar melhor os recursos de cada dispositivo (Item A, todos os **E**), a possibilidade de utilizar formas distintas de funcionamento para funções em plataformas semelhantes (Item B, **E1** e **E3** na Tabela 22), a inclusão de novas plataformas reutilizando as modelagens prontas (Item B, todos os **E**) e a visibilidade em alto nível das funções que serão executadas em cada plataforma do sistema (Item C, **E3** e **E4**).

Ainda tratando a **Q5**, os especialistas **E1** e **E5** apontaram facilidade no gerenciamento da geração de código e inclusão de novas plataformas desde que as funções globais estejam especificadas. Segundo eles, caso não existam funções globais para uma nova plataforma, todo o desenvolvimento deverá ser realizado nos ambientes nativos de produção,

fato que pode prejudicar a produtividade. A abordagem permite a sua extensão através das linguagens nativas implementadas nos modelos de plataformas e a inclusão de uma nova plataforma segue essa regra. Para isso, é necessário adicionar a codificação nativa e já previamente testada nos IDEs oficiais e em seguida incluir as diretrizes do Velocity, nos modelos de plataformas, para garantir confiabilidade na geração de código e funções globais genéricas respectivamente. Após esses passos, pode-se aproveitar os conceitos similares modelados na GPL para várias plataformas e o gerenciamento da geração de código nos modelos de *deploy*. Dessa forma, é possível garantir níveis de produtividade crescente a curto, médio e longo prazo, conforme o relato do **E4** (Item A) e **E5** (Item B) na Tabela 19 e **E4** na Tabela 21. Segundo eles, o Velocity tem o potencial de aumentar a produtividade através da geração de funções similares para diferentes entidades na mesma plataforma, gerando ganhos a curto prazo. A implementação das funções globais com o uso do Velocity em todas as plataformas declaradas na abordagem tem o potencial de aumentar a produtividade a longo prazo.

A sexta questão (**Q6**: **Você acredita que a abordagem pode reduzir os custos do desenvolvimento multiplataforma?**) obteve dos especialistas suas percepções sobre o real ganho em produtividade e redução de custos em se utilizar a abordagem. Essa questão foi de múltipla escolha com as seguintes alternativas: 1) Reduz muito; 2) Reduz consideravelmente para tornar a abordagem viável; 3) Reduz o custo do desenvolvimento, mas não vale a pena utilizá-lo; e 4) Não reduz o custo do desenvolvimento.

Em resumo, uma parte dos especialistas (**E3** e **E4**) informaram que a abordagem tem um grande potencial a longo prazo, caso seja necessário implementar os modelos de plataformas. E, um ganho de produtividade considerável, quando os modelos de plataformas já estiverem parcialmente completos (**E1**) ou se tratando de sistemas mais complexos ou desenvolvimentos recorrentes (**E5**). Outro aspecto questionado foi a confiabilidade dos códigos gerados (**E2**), que se deve em grande parte à maturidade das funções globais implementadas nos modelos de plataformas. A Tabela 21 apresenta as respostas dos especialistas para essa questão.

Tabela 21 – Redução de custos e aumento de produtividade com o emprego da abordagem.

E	**Respostas**
E1	Acredito que pode reduzir muito o desenvolvimento e manutenção de um sistema multiplataforma (**C7+**), principalmente se os modelos de plataformas já estiverem parcialmente completos (**C7-**).

E2	Pode reduzir consideravelmente os custos do desenvolvimento de um sistema multiplataforma (**C7+**). Desde que os códigos gerados sejam confiáveis (**C7-**), isto é, a abordagem deve gerar códigos de forma fiel a modelagem feita através da GPL (**C1-**). O código gerado precisa estar otimizado, garantindo desempenho para cada plataforma.
E3	A longo prazo (**C7-**) acredito que sim (**C7+**). Teremos um trabalho de desenvolvimento mais rápido uma vez que os modelos de plataformas estiverem montadas e prontas. Depois de pronto os modelos de plataformas o desenvolvimento pode ser muito mais rápido com a GPL, inclusive pelo fato de incluir novas plataformas ao sistema (**C4+**). Em situações onde preciso transformar um código em outro a abordagem pronta pode reduzir muito a troca de plataformas (**C3+** e **C7+**).
E4	Sem dúvida reduz muito a longo prazo (**C7-**). Eu diria que reduz muito a longo prazo e a curto prazo reduz (**C7+**) por se tratar de uma ferramenta multiplataforma, mas com certeza a longo prazo os ganhos e possibilidades são muito boas. Reduz quando for a primeira vez que estiver utilizando, pelo emprego do Velocity, e reduz muito quando já se utilizou o projeto algumas vezes (**C7+**).
E5	Reduz consideravelmente (**C7+**). Depende da aplicação, algo relativamente simples ou se não vai precisar desenvolver outros aplicativos talvez não valha a pena usar, devido às questões de aprendizado da abordagem (**C7-**). Mas algo mais complexo, que de manutenção e que precise expandir ou nos casos de precisar desenvolver vários aplicativos, acho que valha a pena, nesse caso exploraria as principais contribuições da abordagem.

As limitações observadas pelos especialistas na questão **Q6** durante a avaliação foram duas. A primeira trata da confiabilidade dos códigos gerados (**E2**) e a segunda se refere ao trabalho inicial em implementar as funções nos modelos de plataformas (**E3 e E5**). Essa ação pode demandar um tempo maior do que os benefícios providos pela abordagem. Porém, essas limitações podem ser parcialmente solucionadas com o uso de modelos de plataformas colaborativos, disponíveis em algum repositório de códigos compartilhados, GitHub[6] por exemplo. Dessa forma, além de contar com implementações confiáveis a abordagem poderá disponibilizar uma grande quantidade de funções prontas, evitando o trabalho inicial de implementação e focando o desenvolvendo na modelagem dos conceitos em alto nível. Tais limitações serão tratadas novamente na Seção 8.2.

A questão **Q7** tratou a manutenção de um sistema desenvolvido através da abordagem e foi a seguinte: **Como você avalia a manutenção de um sistema multiplataforma desenvolvido com a abordagem? Você conseguiu alterar a modelagem**

[6] https://github.com

(GPL) e reutilizar o código gerado? Por exemplo, mudar uma variável de *float* para *double* ou mudar a maneira como o carrinho de compras armazena informações (em um banco de dados ou na memória)? A Tabela 22 apresenta as respostas dos especialistas para essa questão. No geral, a manutenção foi bem avaliada (Todos os **E**), principalmente pelo fato das alterações serem replicadas nos códigos de cada plataforma de forma automática.

Assim como apontado pelo **E2**, pequenos ajustes, como alteração de tipos, podem ser um problema em sistemas multiplataforma nativos, uma vez que é necessário trocar todas as dependências de forma manual. Com a abordagem esse trabalho é minimizado, além de reduzir os riscos de possíveis erros na codificação. O especialista **E5** colocou como um problema na manutenção o uso de muitas funções globais em um sistema, uma vez que as suas atualizações podem ser muito custosas por se tratarem de codificações mais complexas. Depois apontou que talvez as atualizações em funções globais só ocorram quando alguma especificidade técnica foi descontinuada ou alterada. Esse fato realmente se aplica e nesse caso, as alterações de funções globais não impactariam muito a manutenção dos conceitos do domínio do sistema, que ficam isolados na GPL.

Tabela 22 – Análise da manutenção de um sistema através da abordagem.

E	**Respostas**
E1	Eu acredito que haja redução de custos de manutenção com o uso da abordagem. As alterações podem ser feitas na própria GPL (**C2+**) e dessa forma, o custo de manutenção pode ser minimizado (**C7+**).
E2	É possível facilmente alterar a GPL e alterar os modelos de *deploy* (**C7+**). A manutenção em geral pode ser um problema em sistemas multiplataforma. Uma alteração de tipos por exemplo é feita de forma automática entre as plataformas. Os tipos e suas dependências são alterados com a sintaxe correta para cada plataforma e isso é um grande ganho (**C7+**).
E3	A abordagem permite realizar a manutenção e a atualização de uma forma tranquila (**C7+**). Um exemplo foi a tarefa onde trocamos a forma de funcionamento do carrinho de compras (persistência para armazenamento em memória). Os globais inclusive podem agilizar bastante a manutenção pelo fato de já conter as especificidades de cada plataforma (**C7+**).
E4	Acho bom pois a alteração é fácil e rápida, é o que o programador procura quando está sendo desenvolvida algo para multiplataforma (**C7+**). Basta salvar as alterações e os códigos para multiplataforma já são atualizados e de forma nativa e isso gera grandes ganhos (**C2+**). E para conseguir isso o desenvolvedor usa uma linguagem de alto nível que melhora muito a concentração nos conceitos do domínio (**C1+**).

E5	Quanto mais funções globais forem usadas, mais trabalho poderá dar a manutenção, pois o desenvolvedor terá mais funções globais para ser alteradas, aumentando a complexidade (**C7-**). Mas se as funções globais não precisarem ser alteradas e a manutenção se concentrar apenas na GPL ela resultará em ganhos em relação a manutenção de forma manual, usando os IDEs. Outro ponto é que acho que as funções globais só alteram quando há descontinuidade em alguma API e nesse caso seria uma atualização pontual em uma plataforma específica, reduzindo o trabalho. Se a manutenção estiver só na GPL é tranquilo (**C2+** e **C7+**).

A oitava questão (**Q8: Como você avalia os benefícios proporcionados pela abordagem no desenvolvimento de sistemas multiplataforma?**) tratou sobre os benefícios observados pelos especialistas com a utilização da abordagem, apresentados na Tabela 23. Entre os benefícios citados pelos especialistas, os principais são o potencial em redução dos custos de desenvolvimento (**E1** e **E3**), melhor gerenciamento de um sistema multiplataforma (**E4** e **E5**), modelagem em ambiente único (**E1**, **E2** e **E5**), maior facilidade para modelar os conceitos através de uma linguagem de nível mais elevado (**E2** e **E3**) e a evolução da abordagem de forma independente do fabricante (**E4**). Os especialistas **E3** e **E5** acharam interessante a arquitetura da abordagem, uma vez que facilita a organização. O **E3** relatou, inclusive, que a separação de tarefas de uma equipe de desenvolvimento pode ser beneficiada através da abordagem.

Tabela 23 – Benefícios providos pela abordagem.

E	**Respostas**
E1	O fato de utilizar um ambiente único (**C1+**) para gerenciar um sistema multiplataforma facilita muito as questões gerenciais do projeto (**C3+**). A abordagem provê benefícios bons e conseguimos a redução considerável de custo no desenvolvimento, uma vez que temos uma grande gama de plataformas atualmente e que devem ser considerados no desenvolvimento (**C7+**).
E2	Avalio bem a abordagem, um código único e gera para todos os outros dispositivos (**C2+**). Não precisa dominar outras plataformas (**C1+**), se as funções globais estiverem implementadas (**C7-**). Se a abordagem me garante desempenho, não preciso me preocupar e nesse caso o ganho é bem alto.

E3	Aumenta a produtividade (**C7+**) quando a abordagem já estiver madura e os modelos de plataformas "já completos". Também melhora o gerenciamento dos papéis dos desenvolvedores no processo (**C1+**). A abordagem ajuda também na documentação. A GPL pode ser usada como documentação inclusive (**C1+**). Inclusive a GPL pode substituir a documentação e como ela é atualizada, a documentação é atualizada automaticamente também. Quanto maior o nível de abstração, melhor para gerir um sistema grande.
E4	Acredito que podem ser os mesmos do que outros *frameworks*, a diferença é o ponto de vista atemporal. O desenvolvedor pode implementar soluções e saídas sem dependência dos fabricantes (**C5+**). Ele pode evoluir o *framework* de acordo com a necessidades dos sistemas produzidos.
E5	A organização que a abordagem oferece é bem interessante (**C1+**). Está tudo em um arquivo único e isso facilita. A GPL poderia ser separada para melhorar ainda mais a organização e gerência, colocando cada parte do sistema em arquivos separado (**C7-**).

As respostas para as questões **Q9** (**Avalie sua experiência com a abordagem em uma escala de 1 a 10, onde 1 significa uma experiência muito ruim e 10 significa uma experiência muito boa.**) e **Q10** (**Você tem outra ideia para reduzir custos no desenvolvimento multiplataforma?**) estão na Tabela 24.

Tabela 24 – Avaliação da abordagem (Notas de 1 a 10) e novas ideias para reduzir custos para o desenvolvimento multiplataforma.

E	**Notas da experiência e Pontos a melhorar**
E1	Nota: 9. Manter mais genérico os modelos de plataformas (**C5-**) da abordagem para garantir melhor reúso (**C6-**), é necessário copiar e colar os modelos de plataformas em outros projetos. Eles deveriam ficar em um local isolado na abordagem para ser atualizado e importado. Senti falta de um local para teste de algoritmo (**C5-**), alguma forma de testar os códigos da GPL antes de inserir nos ambientes nativos. Da forma que está, fica complicado saber se o algoritmo não possui erros. Outra questão são os padrões autorais que a abordagem permite para generalizar a GPL, principalmente na camada *Controller* (MVC). Elas possuem uma curva de aprendizado relativamente alta (**C1-**, **C2-** e **C7-**), pois muda o padrão habitual de desenvolvimento, mas pode gerar ganhos futuros (**C7+**).

E2	Nota: 8. Seria necessário otimizar o editor da GPL, identificando erros de sintaxe e semântica e também melhorar a endentação. Recursos como *code completion* também seriam bem úteis (**C7-**). Outro ponto importante é a garantia de funcionamento e desempenho que a abordagem precisa prover (**C5-**), talvez alguns recursos de teste de código. Se isso for bom, não preciso me preocupar com esses detalhes e nesse caso o ganho é bem alto (**C7+**). No caso do iOS o aplicativo não pode ter nenhum vazamento de memória ou qualquer detalhe que reduza o desempenho, caso contrário a Apple não publica. E o usuário fica dependente dessa questão em um *framework* fechado (caixa preta). Portanto em uma abordagem aberta como essa, isso pode ser melhor gerenciado, pois você pode editar a geração de código nos modelos de plataformas e corrigir determinados problemas (**C5+**).
E3	Nota: 10. A ideia é muito boa, desenvolver um modelo e a partir daí gerar para vários ambientes diferentes (**C1+** e **C2+**), mas precisa melhorar em alguns pontos. A validação do código gerado pode ser uma limitação. A inclusão de alguma atividade para validação dos códigos gerados pode ajudar, pois isso é importante para garantir o correto funcionamento nas plataformas (**C5-**). O fato de testar o código apenas nos ambientes de produção de forma frequente pode reduzir o potencial de produtividade da abordagem (**C7-**).
E4	Nota 9. A geração da *View* é um ponto muito importante e poderia ser inserida na abordagem (**C1-**). A geração genérica de interface poderia garantir um grande ganho para abordagem como trabalho futuro. A *View* é algo delicado, pois depende muito da identidade visual da marca, mas seria interessante a geração genérica da interface para a customização detalhada pelo desenvolvedor nos ambientes nativos (IDE) de cada plataforma.
E5	Nota 9. É difícil programar as funções globais nos modelos de plataformas dentro da abordagem. O editor não oferece nenhum suporte para realizar essa tarefa, como testes, validação de sintaxe, entre outros (**C1-** e **C7-**). Se a ideia não é trabalhar de forma rotineira as questões de implementações e atualizações das funções globais, a abordagem oferece uma boa estrutura para se trabalhar através de uma linguagem que realmente eleva o nível de abstração (**C1+**). Mas se tiver que fazer implementações e atualizações das funções globais diretamente na abordagem a produtividade e a confiabilidade reduz bastante, não tem a facilidade de um IDE auxiliando a programação.

As notas dos especialistas referente à experiência em utilizar a abordagem foram no geral altas, devido à percepção do potencial de benefícios para o desenvolvimento multiplataforma discutidos nas questões anteriores. Porém, foram apontadas algumas limitações (**L**) para a abordagem. A melhoria de tais limitações é fundamental para alcançar uma abordagem multiplataforma que realmente possa prover ganhos e será priorizada para os trabalhos futuros. Cada limitação foi relacionada diretamente com uma ou mais contribuições (**C**) elencadas para a abordagem, como pode ser visualizado na lista que se segue.

- **L1**. Necessidade de testes de código na própria plataforma. Incluindo testes de validação no editor e testes nos códigos gerados (**E1, E2, E3 e E5**). Essa limitação está relacionada com a **C1**;
- **L2**. Resultados somente a longo prazo devido ao fato da implementação e atualização de funções globais (**E3 e E5**). Essa limitação está relacionada com a **C2**, **C3** e **C4**;
- **L3**. Confiabilidade dos códigos gerados (**E2 e E5**). Essa limitação está relacionada com a **C1**;
- **L4**. Garantia de desempenho dos códigos gerados da abordagem (**E2 e E5**). Essa limitação está relacionada com a **C1** e **C5**;
- **L5**. Necessidade de entender um novo padrão de programação, que pode ter uma curva de aprendizado acentuada (**E1**). Essa limitação está relacionada com a **C1**;
- **L6**. Geração da camada de visualização (**E4**). Essa limitação está relacionada com a **C1** e **C7**;
- **L7**. Uso de alguma ferramenta gerencial para auxiliar a modelagem visual dos conceitos de domínio, como exemplo *Astah Professional* (**E3**). Essa limitação está relacionada com a **C1**; e
- **L8**. Necessidade de recorrer às linguagens nativas e ao IDE de cada plataforma. Essa limitação está relacionada com a **C1** e **C7**.

As limitações da abordagem apontadas pelos especialistas são discutidas na Seção 8.2, junto com as contribuições relacionadas a elas. Nessa seção, apresenta-se também um resumo dos resultados.

Para a última questão (**Q11**: **Avalie cada uma das contribuições em uma escala de 1 a 10.**) foram anotadas apenas evidências (**C+** ou **C-**) não citadas em outras questões. A Tabela 25 apresenta os resultados.

Tabela 25 – Notas dos especialistas para o atendimento de cada contribuição elencada para essa tese.

E	**Notas e Justificativas**
E1	**C1**- Nota: 10. **C2**- Nota: 10. **C3**- Nota: 10. **C4**- Nota: 8. Necessário atualizar os modelos de plataformas com detalhes técnicos de plataformas (**C4-**). **C5**- Nota: 9. Necessário atualizar os modelos de plataformas com detalhes técnicos implementando as funções específicas (**C5-**). **C6**- Nota: 8. **C7**- Nota: 9.
E2	**C1**- Nota: 9. **C2**- Nota: 10. **C3**- Nota: 7. Desenvolver um testador de código mais simples. **C4**- Nota: 9. Deu para se ter uma boa ideia de como realizar a inclusão de novas plataformas. **C5**- Nota: 8. Pode incluir funções genéricas para outras plataforma estendendo a abordagem (**C5+**). Tem um custo maior no momento de implementação de novos detalhes técnicos, mas a longo prazo a produtividade aumenta. **C6**- Nota: 9. **C7**- Nota: 8.
E3	**C1**- Nota: 8. O nível de abstração quando sobe muito talvez precise de uma ferramenta para ajudar a modelar melhor o domínio, como o Astah Professional. **C2**- Nota: 10. Realmente ficou bem claro que conseguimos pensar no multiplataforma e aproveitar os conceitos similares. **C3**- Nota: 10. É bem fácil trocar e gerenciar a troca de funções e plataformas. **C4**- Nota: 9. Segue a ideia de herança, podemos incluir novas plataformas e especificar os detalhes durante o desenvolvimento. **C5**- Nota: 9. Tem um custo maior no momento de implementação de novos detalhes técnicos, mas a longo prazo a produtividade aumenta. **C6**- Nota: 6. Talvez precisaria de uma ferramenta para ajudar a gerenciar o domínio melhor (**C6-**). Pensando em uma escala maior e um processo maior para modelar melhor o domínio. **C7**- Nota: 8.

E4	**C1**- Nota: 9. Sem dúvida é possível modelar o sistema todo em um único ambiente. O trabalho ficou bem conciso e muito bem organizado melhorando o gerenciamento do sistema. **C2**- Nota: 10. **C3**- Nota: 9. Faltou tratar a interface. **C4**- Nota: 10. Ficou desacoplado a inclusão de novas plataformas ao sistema, basta implementar os globais nas linguagens originais nativas e o código genérico é sempre o mesmo para qualquer plataforma. **C5**- Nota: 10. A extensão é feita de forma independente gerando liberdade aos desenvolvedores estender a ferramenta. **C6**- Nota: 9. Reutilização de funções ficou bem legal. Tudo que é possível reutilizar que já foram previamente desenvolvidos é ótimo. **C7**- Nota: 9.
E5	**C1**- Nota: 10. **C2**- Nota: 10. Alguns pontos não estão implementadas, modificadores de acessos por exemplo. **C3**- Nota: 8. Fácil desde que esteja tudo implementado. **C4**- Nota: 9. Se tiver as funções globais prontas. **C5**- Nota: 10. **C6**- Nota: 10. **C7**- Nota: 9.

8.2 Discussão dos resultados

De forma geral, os especialistas apontaram aspectos interessantes e promissores em relação a abordagem com algumas ressalvas que serão melhor discutidas na próxima seção. Vale ressaltar que a avaliação foi de ordem qualitativa, onde cada especialista disponibilizou no mínimo 12 horas para o entendimento e as tarefas (Tabela 26). O tempo despendido com a avaliação, combinado com a ampla experiência em desenvolvimento de software dos especialistas, contribuíram para uma análise mais profunda da abordagem e que renderam aprofundadas discussões durante as entrevistas. Tais discussões ajudaram a identificar com maior clareza os principais aspectos prós e contras da abordagem. Além disso, a concentração nas questões do desenvolvimento multiplataforma permitiu a troca de experiências sobre as melhores soluções disponíveis no mercado e novas ideias para acrescentar a esta tese.

Entre os aspectos prós e contras da abordagem obtidos através das entrevistas com os especialistas, estão as evidências das contribuições da abordagem contrapondo com as limitações apontadas e discutidas pelos especialistas.

Tabela 26 – Tempo gasto por cada especialista em aprender a abordagem e em cada tarefa

Especialista	**Aprendizagem**	**T1**	**T2**	**T3**	**T4**	**T5**	**Total**
E1	6h00	1h00	2h20	0h40	0h45	1h30	**12h15**
E2	4h00	0h40	3h40	0h45	0h30	4h00	**13h35**
E3	7h00	0h25	1h20	0h35	0h25	1h30	**11h15**
E4	8h00	2h30	2h20	2h30	0h30	2h00	**17h50**
E5	7h00	2h00	2h55	1h30	1h30	1h00	**15h55**
Média	**6h24**	**1h19**	**2h31**	**1h12**	**0h44**	**2h00**	**14h10**

8.2.1 Evidências e limitações apontadas pelos especialistas em relação as contribuições da abordagem desta tese (**M1**)

Cada tarefa da avaliação tem o objetivo de avaliar uma ou mais contribuições e as questões da entrevista tem o objetivo de obter as percepções dos especialistas sobre tais contribuições, respondendo a métrica **M1** definida para essa avaliação no Capítulo 6. Dessa forma, a Tabela 27 apresenta as vezes que cada contribuição (listadas novamente abaixo) foi relatada pelos especialistas e em quais questões elas foram identificadas. Salienta-se que mesmo que o especialista aponte a mesma contribuição mais de uma vez usando o mesmo argumento e na mesma resposta, contabilizamos apenas uma.

- **C1**: Modelagem em alto nível de abstração e em um único ambiente;
- **C2**: Aproveitamento dos conceitos similares entre as plataformas na modelagem;
- **C3**: Redução do custo na configuração de um sistema multiplataforma;
- **C4**: Inclusão de novas plataformas na abordagem com menor impacto;
- **C5**: Extensão da abordagem através dos modelos de plataformas;
- **C6**: Reúso de códigos em outros domínios; e
- **C7**: Redução do custo de desenvolvimento e manutenção.

Tabela 27 – Prós (**C+**) e Contras (**C-**) dos Especialistas (**E**), relatados através das Questões (**Q**) da entrevista, em relação as Contribuições (**C**) da abordagem desta tese.

C	**E1**	**E2**	**E3**	**E4**	**E5**

C1+	**Q2**: 2 **Q8**: 1	**Q2**: 1 **Q4**: 1 **Q8**: 1	**Q1**: 1 **Q2**: 2 **Q3**: 1 **Q4**: 1 **Q5**: 1 **Q8**: 2 **Q10**: 1	**Q2**: 1 **Q5**: 1 **Q7**: 1	**Q1**: 1 **Q2**: 1 **Q8**: 1 **Q10**: 1
C1-	**Q10**: 1	**Q2**: 1 **Q3**: 1 **Q6**: 1	**Q3**: 1	**Q10**: 1	**Q2**: 2 **Q4**: 1 **Q10**: 1
C2+	**Q4**: 2 **Q7**: 1	**Q4**: 1 **Q5**: 1 **Q8**: 1	**Q5**: 1 **Q10**: 1	**Q7**: 1	**Q7**: 1
C2-	**Q10**: 1	- - -	- - -	- - -	- - -
C3+	**Q5**: 3 **Q8**: 1	**Q5**: 3	**Q5**: 3 **Q6**: 1	**Q5**: 3	**Q5**: 3
C3-	**Q5**: 1	- - -	**Q3**: 1	- - -	- - -
C4+	**Q3**: 1	**Q2**: 1 **Q3**: 1 **Q5**: 1	**Q3**: 1 **Q4**: 1 **Q5**: 1 **Q6**: 1	**Q3**: 2 **Q5**: 1	**Q3**: 2 **Q4**: 1 **Q5**: 1
C4-	**Q11**: 1	- - -	- - -	- - -	- - -
C5+	**Q3**: 2	**Q2**: 1 **Q3**: 3 **Q4**: 2 **Q10**: 1 **Q11**: 1	**Q3**: 3 **Q4**: 2	**Q3**: 2 **Q8**: 1	**Q3**: 2 **Q4**: 1
C5-	**Q10**: 2 **Q11**: 1	**Q10**: 1	**Q3**: 1 **Q10**: 1	- - -	**Q3**: 1 **Q5**: 1
C6+	**Q3**: 1 **Q4**: 3	**Q3**: 1 **Q4**: 2	**Q3**: 1 **Q4**: 2	**Q3**: 1 **Q4**: 1	**Q3**: 1 **Q4**: 1
C6-	**Q10**: 1	- - -	**Q11**: 1	- - -	- - -
C7+	**Q4**: 2 **Q6**: 1 **Q7**: 1 **Q8**: 1 **Q10**: 1	**Q3**: 1 **Q4**: 1 **Q6**: 1 **Q7**: 2 **Q10**: 1	**Q3**: 1 **Q4**: 2 **Q6**: 2 **Q7**: 2 **Q8**: 1	**Q2**: 1 **Q4**: 2 **Q5**: 1 **Q6**: 1 **Q7**: 1 **Q8**: 1	**Q2**: 2 **Q4**: 2 **Q6**: 1 **Q7**: 1

C7-	**Q6**: 1 **Q10**: 1	**Q2**: 1 **Q6**: 1 **Q8**: 1 **Q10**: 1	**Q2**: 1 **Q3**: 1 **Q4**: 1 **Q6**: 1 **Q10**: 1	**Q4**: 1 **Q6**: 1	**Q2**: 1 **Q3**: 1 **Q4**: 1 **Q5**: 1 **Q6**: 1 **Q7**: 1 **Q8**: 1 **Q10**: 1

As questões na Tabela 27 são importantes para identificar em quais partes da abordagem cada contribuição foi percebida e comentada. Esses dados foram usados para analisar com maior consistência as evidências de cada contribuição. O total de apontamentos prós (**C+**) e contras (**C-**) dos especialistas para cada contribuição (**C**) da abordagem desta tese é apresentado na Figura 30.

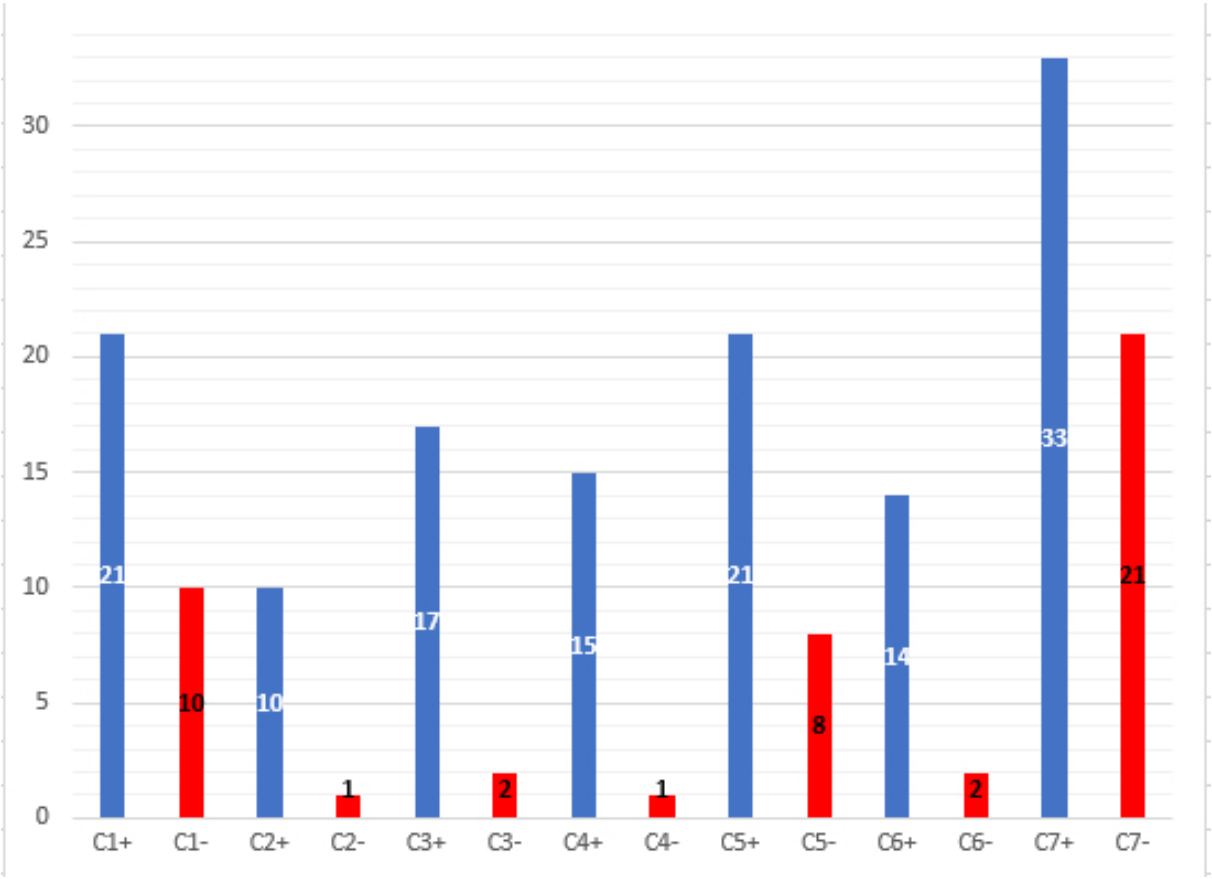

Figura 30 – Total de apontamentos Prós (**C+**) e Contras (**C-**) dos especialistas para cada contribuição (**C**) da abordagem desta tese.

Considerando os dados da Tabela 27, os tópicos que se seguem tratam sobre as evidências positivas e limitações de cada contribuição elencada para esta tese.

8.2.1.1 Análises da **C1**

É possível afirmar que cerca de 33% das evidências para a **C1** apareceram na questão referente à GPL (**Q2**). Nessa questão, é possível verificar que a maior parte dos especialistas (**E1**, **E2**, **E3** e **E4**) conseguiu identificar o aumento do nível de abstração e o isolamento dos detalhes técnicos nos modelos de plataformas (Tabela 17, item B). Esses aspectos são importantes para manter uma abordagem genérica o suficiente para garantir o aproveitamento de algoritmos entre diferentes plataformas e permitir a extensão da abordagem pelos desenvolvedores.

Embora a maior parte das evidências para a **C1** esteja na **Q2**, é possível verificar apontamentos para essa contribuição ao longo de quase todas as questões. Isso acontece porque a **C1** trata das características mais gerais da abordagem, como o aumento de nível de abstração e a modelagem em um único ambiente.

Além das evidências em pró da **C1**, apontadas pelos especialistas, também foram relatadas limitações (**C1-**) que podem ser associadas a essa contribuição. Baseado nas limitações (**L**) apontadas na seção anterior, podemos relacionar as limitações **L1**, **L3**, **L4**, **L5**, **L6** e **L7** a **C1** em alguns pontos específicos, discutidos na sequência.

A **L1** trata a ausência de mecanismos de testes e poucos recursos de validação da GPL. A maior parte dos especialistas (**E1, E2, E3 e E5**, apontados na Tabela 24) sentiram falta da possibilidade de testar os algoritmos genéricos (GPL) na abordagem, assim como os códigos gerados. O **E2** enfatizou também na **Q2** que seria interessante acrescentar validações de sintaxe e recursos como *code completion* no editor GPL. Atualmente o editor possui um mecanismo que valida o código de acordo de acordo com as regras estabelecidas na gramática da GPL e possui recursos para completar a programação em alguns pontos da linguagem. É evidente que esses recursos precisariam ser aprimorados caso fosse feito o lançamento de um produto final e isso faz parte de nossos trabalhos futuros. Porém, por se tratar de um protótipo, as prioridades se concentraram na concepção de uma arquitetura que pudesse atender às contribuições elencadas. O mesmo se aplica para os testes dos códigos gerados. Para esse caso, poderia ser resolvido com a criação de uma plataforma dedicada para testes, com funções globais genéricas que automatizam a geração de testes de unidade para serem inseridos e testados nos IDEs nativas de cada plataforma. Cada IDE poderá utilizar o *framework* oficial para testes, como o JUnit[7] para Java por exemplo. Essa possibilidade foi explorada posteriormente, em uma terceira avaliação, apresentada no Capítulo 9.

detalhes de implementação do dia-a-dia, como code completion, menus já preparados para a plataforma

A **L3** trata a confiabilidade dos códigos gerados e é apontada pelos especialistas

[7] https://junit.org/junit5/

E2 na Tabela 21 e **E5** na Tabela 24. O **E2** aborda a confiabilidade no sentido de gerar códigos fiéis aos modelados na GPL, respeitando as questões de sintaxe e semântica nos códigos gerados. Segundo as avaliações realizadas na Seção 7.2, as características semânticas dos algoritmos desenvolvidos através da GPL se refletem de forma fiel nas linguagens alvo, conforme a Figura 28. Em todos os testes realizados, não houve discrepância, em termos de execução, entre o código modelado na GPL e o código gerado. O **E5** se refere a confiabilidade em se implementar funções diretamente nos modelos de plataformas. Segundo ele, a abordagem não fornece recursos avançados de *code completion* e menus já preparados para a plataforma, o que pode contribuir para o aumento de erros e reduzir a confiança dos códigos especificados na GPL. Embora esteja previsto nos trabalhos futuros a melhora de tais recursos, os IDEs nativos de cada plataforma ainda são os melhores lugares para a criação e teste de funções antes de transferir para os modelos de plataformas da abordagem. Por fim, o especialista **E2** na Tabela 21 aponta que se os códigos gerados forem confiáveis a abordagem teria redução dos custos do desenvolvimento. E, o **E5** na Tabela 24, aponta que se não for necessário trabalhar de forma rotineira as questões de implementações e atualizações das funções globais, a abordagem realmente eleva o nível de abstração.

A **L4** se refere à questão do desempenho dos códigos gerados pela abordagem e foi apontada pelo **E2** na Tabela 24. Essa limitação está relacionada a dois pontos: a qualidade dos algoritmos genéricos (GPL) modelados pelo desenvolvedor e a qualidade das funções globais implementadas nos modelos de plataformas. Portanto, o desempenho depende da qualidade do programador que está utilizando a abordagem. As funções globais implementadas nos modelos de plataformas também interferem no desempenho dos códigos gerados. Essas últimas, assim como já mencionado, precisam ser previamente testadas nas plataformas específicas e sua qualidade semântica irá garantir desempenho.

A **L5** foi relatada pelo especialista **E1** na Tabela 24 e trata da dificuldade que sentiu em criar padrões para modelar de forma genérica as classes da camada *Controller* e *Model* do estilo arquitetural MVC, chamado por ele de "padrões autorais". O especialista se refere ao uso de detalhes globais, descritos na Seção 5.2.1 e Listagem 5.2, que segundo ele são um padrão não usual para criação de código e por esse motivo a dificuldade em se entender seu funcionamento nos primeiros contatos. Embora exista uma certa dificuldade em seu entendimento inicialmente, o potencial de aumento na produtividade compensa a longo prazo, segundo o **E1** na Tabela 24.

O **E4** sentiu falta do suporte para a camada de visualização pela abordagem (**L6**), relatado na Tabela 24. Esse recurso foi previsto, mas devido às restrições de prazos do projeto desta tese as avaliações não abordaram a camada de visualização. A camada de visualização pode ser configurada através dos mesmos padrões discutidos para a camada *Controller*. Nesse caso, poderiam ser criadas classes para definir os detalhes de interfaces,

assim como formatos, cores, informações, quantidade de dados, elementos de exibição (menu, logo, banner) entre outros detalhes. Através de uma função global, esses detalhes poderiam ser lidos e os arquivos para a camada de visualização criados. A ideia é criar arquivos com os recursos mínimos para essa camada e deixar com que a arte final possa ser especificada com mais detalhes nos IDEs nativos de cada plataforma. Segundo o **E4** na Tabela 24, essa prática já traria ganhos consideráveis para a produtividade das interfaces do sistema.

A **L7** foi apontada pelo **E3** na Tabela 24 e se refere ao aumento do nível de abstração proporcionada pela GPL. Segundo o especialista, quando isso ocorre, os desenvolvedores podem requerer um software específico para visualizar melhor a modelagem do projeto, como o *Astah professional* por exemplo. Segundo o especialista, esse software permitiria uma visualização melhor da relação entre as classes e métodos genéricos modelados pela GPL. Porém, como toda a lógica do sistema é modelada através da GPL (classes, atributos, variáveis, métodos, regras de negócio, acesso a funções globais, entre outros), a forma textual foi a escolha com maior prioridade para a abordagem. Como uma forma de melhorar a visão geral das relações entre artefatos do sistema, pode-se incluir a modelagem visual na abordagem no futuro. Uma das funções do modelo de *deploy* é auxiliar a geração de códigos para diferentes plataformas, segundo os especialistas (Tabela 20, todos os **E** no item A), por meio de uma visão mais favorável da distribuição de funcionalidades entre as plataformas do sistema.

A **L8** também foi relacionada à **C1** pelo fato de que a necessidade de se implementar os modelos de plataformas utilizando os IDEs de cada plataforma possa ir de encontro com o desenvolvimento em um único ambiente proposto pela abordagem. Esse fato é verdadeiro nas fases iniciais do uso da abordagem. Porém, essa limitação pode ser reduzida à medida que os modelos de plataformas são implementados ou fazendo reúso de modelos de plataformas já implementados, como relatado pelos especialistas **E1**, **E3** e **E4** na Tabela 21.

8.2.1.2 Análises da **C2**

Essa contribuição foi identificada 10 vezes durante as entrevistas e cerca de 30% delas apareceram na **Q4** que trata as funções globais. Tal indicativo reforça a ideia de que as funções globais possuem o potencial de isolar as especificidades de plataformas, permitindo uma programação mais genérica, segundo relatos dos **E1** (Item A) e **E2** (Item B) na Tabela 19. Outros 30% dos apontamentos para a **C2** apareceram na **Q7**, mostrando vantagens do aproveitamento de código na manutenção do sistema, onde as alterações são realizadas em alto nível na GPL (**E1** e **E4** na Tabela 22). Em menor número (20%), apareceram na **Q5**, que trata os modelos de *deploy*, indicando a percepção dos especialistas em reaproveitar os algoritmos genéricos para diferentes plataformas (**E2** e **E3** na Tabela

20). O percentual restante ficou dividido entre a **Q8** com o **E2** (10%) e **Q10** com o **E3** (10%), mostrando o aproveitamento de código genérico como benefício da abordagem.

A limitação relacionada à **C2** é a **L2**, destacada pelo **E3** nas Tabelas 17 (Item B) e 23 e pelo **E5** na Tabela 17 (Item B). A **L2** está relacionada à **C2** pelo fato do aproveitamento de código só ser possível se os modelos de plataformas estiverem prontos para as plataformas especificas. Segundo os especialistas (**E3** e **E5**), caso seja necessário criar todas as funções globais para um sistema multiplataforma, a produtividade da abordagem fica comprometida. Ainda mais pelo fato dessas funções precisarem ser criadas e testadas nos IDEs nativos, para aumentar a confiabilidade dos códigos.

De fato, o posicionamento dos especialistas faz sentido se a cada novo projeto não fosse possível importar os modelos de plataformas parcialmente completos. A arquitetura da abordagem foi modularizada para facilitar a importação e reúso dos modelos de plataformas, parcialmente completos, em novos projetos. O termo "parcialmente" foi usado devido à probabilidade de ter algo específico de domínio a ser implementado nos modelos de plataformas em novos projetos. Dessa forma, através da importação de modelos de plataformas parcialmente prontos, a produtividade provida pela abordagem pode ser potencializada, segundo o **E5** na Tabela 22, **E1**, **E3** e **E4** na Tabela 21 e **E5** na Tabela 19 (Item B). Inclusive, durante as entrevistas, o **E1** na Tabela 24 sugeriu manter os modelos de plataformas em repositórios colaborativos, para que os mesmos possam ser completados pela comunidade e usados em qualquer sistema. Dessa forma, eliminaria parcialmente os custos apontados pelos especialistas.

A solução de importação dos modelos de plataformas parcialmente prontas na abordagem também pode ser aplicada para **C3** e **C4** e resolve partes das limitações apontadas para elas.

8.2.1.3 Análises da **C3**

A **C3** foi apontada de forma unânime na **Q5**, pelo fato dessa questão tratar os modelos de *deploy*. Os especialistas apontaram vantagens em poder utilizar um modelo específico para configurar a geração de código, conforme os relatos com apontamentos para a **C3** na Tabela 20. Segundo eles, o modelo de *deploy* permite o gerenciamento de um sistema multiplataforma em alto nível, possibilitando a distribuição de funcionalidades e a troca de plataformas de forma facilitada.

Além dos relatos em pró, também foram citadas limitações para a **C3** pelos especialistas. A **L2** também está relacionada a **C3**, assim como a **C2**. Conforme os especialistas **E3** nas Tabelas 17 (Item B) e 23 e pelo **E5** na Tabela 20 (Item A), a implementação das funções dos modelos de plataformas precisa estar parcialmente completa para poder explorar as facilidades de gerenciamento do modelo de *deploy*. Tais gerenciamentos incluem a distribuição de funcionalidades entre os dispositivos e a troca das plataformas

usadas no sistema. Caso seja importado um modelo de plataforma com grande parte das implementações prontas, o gerenciamento de um sistema multiplataforma pelo modelo de *deploy* trará aumento de produtividade, segundo o relato de todos os especialistas na Tabela 20.

Outra limitação associada à **C3** foi relatada pelo **E3** na Tabela 18, indicando dúvidas em relação ao atendimento de sistemas de grande porte devido às bibliotecas e dependências. A avaliação realizada no Capítulo 7 (Seção 7.2) recriou um sistema multiplataforma de médio porte e os códigos gerados implementados nos dispositivos reais. Para esse caso, a abordagem não apresentou problemas em relação às dependências, umas vez que as bibliotecas utilizadas não estavam descontinuadas. Assim, os IDEs conseguiram importar tais dependências e não houve erros no sistema.

8.2.1.4 Análises da **C4**

A **C4** aparece predominantemente nas questões **Q3** (45%) e na **Q5** (25%). A **Q3** verifica a capacidade de extensão e o reúso dos modelos de plataformas, pontos fundamentais para a inclusão de novas plataformas aos sistemas desenvolvidos através da abordagem. Segundo os relatos nas respostas da **Q3** (Item A) (Tabela 18), os especialistas alegaram a possibilidade de inclusão de qualquer nova função na abordagem. Sendo assim, qualquer plataforma que possa ser desenvolvida em uma das linguagens disponíveis na abordagem pode ser incluída e fazer parte do sistema.

A **Q5** trata os modelos de *deploy* (Tabela 20). Nesse modelo é configurada a geração de código para novas plataformas. Embora a inclusão de novas plataformas é realizada no modelo de *deploy*, os especialistas (**E1**, **E4** e **E5** na Tabela 20) relatam a necessidade de preparar a abordagem para essa nova plataforma. Tal preparação inclui a implementação das funções globais nos IDEs específicos, antes de receber as diretrizes do Velocity nos modelos de plataformas e se tornar funções genéricas, com o potencial de geração de códigos para diferentes classes.

Portanto, a inclusão de novas plataformas na abordagem com menor impacto no código do sistema, segundo os relatos dos especialistas **E1**, **E4** e **E5** na **Q5** (Tabela 20), pode ocorrer de fato no reúso da programação genérica (GPL) e no reúso dos modelos de plataformas. Para esse último, a redução do impacto é verdadeira apenas se os modelos de plataformas estiverem parcialmente completos, assim como descrito para a **C2** e **C3**. Caso todas as implementações dos modelos de plataformas para a nova plataforma precisem ser feitas do início, a **C4** terá a **L2**, segundo o especialista **E1** na Tabela 25. Nesse caso, ainda haveria o aproveitamento da **C2** e somente após as implementações dos modelos de plataformas os desenvolvedores poderiam explorar o potencial máximo de redução de custos da abordagem. O potencial máximo de redução de custos se enquadra no reúso dos modelos de plataformas parcialmente prontos, permitindo ao desenvolvedor modelar

a GPL em alto nível e configurar a geração de códigos no modelo de *deploy*.

8.2.1.5 Análises da **C5**

A extensão da abordagem (**C5**) é comentada preferencialmente na **Q3** (60%), que trata os modelos de plataformas. Os especialistas tiveram que estender a abordagem em uma das tarefas, e isso forneceu base para opinar de forma consistente a respeito desse recurso. Segundo os relatos na Tabela 18 dos **E1** e **E3** (Item C) e **E2**, **E4** e **E5** (Itens A e C), a extensão facilitada da abordagem pode garantir a evolução de um sistema multiplataforma de forma independente do fabricante. Outro detalhe apontado foi o fato do desenvolvedor poder ajustar o desempenho dos códigos gerados, permitindo o aprimoramento da abordagem e a customização da geração de código (**E2** na Tabela 18, item A).

Além da **C1**, a limitação **L4**, que trata o desempenho dos códigos gerados, também está relacionada à **C5** e as justificativas entre as contribuições são bem próximas. Segundo o **E2** na Tabela 24, um ponto importante é a garantia de funcionamento e desempenho que a abordagem precisa prover, se isso for bom o ganho é bem alto. Para essa limitação podemos reiterar as explicações para a **C1** na Seção 8.2.1.1, onde o desempenho é apontado como uma dependência da qualidade do programador que está utilizando a abordagem. A qualidade das funções globais implementadas também interfere no desempenho dos códigos gerados. Como já foi frisado, elas precisam ser implementadas e testadas nos IDEs nativos antes de ser migradas para a abordagem para aumentar desempenho.

Ainda na questão do desempenho e confiabilidade dos códigos, o **E2** (Tabela 18, item A) relatou um comparativo entre a abordagem proposta e o *framework* multiplataforma Flutter[8], que ele utiliza. Segundo o **E2**, em alguns casos são necessários configurações distintas para executar certas funções no android e iOS de forma correta. Como exemplo dessas funções, estão a conversão de documentos (Html para PDF) ou funções que exijam permissões dos usuários, como o uso de câmeras e GPS. No caso do iOS é exigido mais etapas de permissões durante esse processo. Para esses casos, o fato do Flutter não permitir a modelagem de forma distinta entre as plataformas e nem a edição dos códigos-fonte gerados pode acarretar em problemas em sua execução. A abordagem proposta permite adicionar formas diferentes de se realizar uma determinada ação, devido à configuração da geração de códigos no modelo de *deploy*, além de permitir a edição das funções específicas nos modelos de plataformas e no código gerado. Dessa forma, o desenvolvedor poderá configurar essas diferenças no modo de implementação entre as plataformas e evitar erros ou mal funcionamento.

Outra vantagem na possibilidade de extensão da abordagem através dos modelos de plataformas é a adequação de certos códigos em casos de problemas na publicação dos

[8] https://flutter.dev/

aplicativos nas respectivas lojas. Segundo o **E2** na Tabela 24, a Apple é muito criteriosa na publicação de aplicativos em sua plataforma, não permitindo nenhum vazamento de memória ou qualquer outro detalhe técnico que reduza o desempenho do aplicativo em seu SO. Dessa forma, o desenvolvedor precisa confiar nos códigos gerados pelos *frameworks* multiplataforma, pois se eles criarem algo com algum problema que impeça a sua publicação, a sua resolução pode se tornar um grande desafio.

Além da **L4**, alguns especialistas (**E1**, **E2** e **E3** na Tabela 24 e **E3** e **E5** na Tabela 18) relataram a questão da ausência de mecanismos de testes para a inclusão de novas funções nos modelos de plataformas, afetando a **C5**. Diferentemente da análise de testes para a **C1** (Seção 8.2.1.1), a implementação das funções globais nos modelos de plataformas precisam ser desenvolvidas e testadas previamente nos IDEs nativos, assim como já mencionado. A codificação dessas funções diretamente na abordagem pode comprometer a confiabilidade e aumentar as chances de erros. Atualizações simples dos modelos de plataformas podem ser realizadas diretamente na abordagem, já o restante deve ser nos ambientes nativos. Embora exista a questão do uso de outros ambientes, a ideia é sempre realizar o reúso dos modelos de plataformas parcialmente prontos. Isso minimiza o trabalho de implementação dos modelos de plataformas e atualizando-os apenas com a necessidade de alguma extensão na abordagem, assim como relatado pelo **E4** na Tabela 23.

8.2.1.6 Análises da **C6**

Os apontamentos para a **C6** aparecem, em menor parte, na **Q3** (35%), que trata os modelos de plataformas, e o restante (65%) na questão que trata seus recursos (**Q4**). Na **Q3**, segundo os relatos de todos especialistas no item B da Tabela 18, uma vez implementada e adicionada as diretrizes do Velocity nas funções globais, os modelos de plataformas ficam genéricos o suficiente para o atendimento a outros domínios. Dessa forma, é possível aumentar o reúso das funções dos modelos de plataformas da abordagem.

Na **Q4**, todos os especialistas relataram (Itens A e B na Tabela 19) que os códigos gerados são customizados em tempo de geração de códigos nas funções globais e devido a isso, os modelos de plataformas podem ser utilizados por diferentes entidades e domínios do problema.

O reúso dos códigos genéricos e dos modelos de plataformas de um sistema para outro, neste estudo, ficou limitado às camadas *Model* e *Controller* do estilo arquitetural MVC na abordagem. Uma vez que a abordagem não suporta a modelagem da camada de visualização. Essa limitação (**L6**), também relacionada a **C1**, foi apontada pelo **E4** na Tabela 24. Conforme explicado no tópico de tratamento da **C1** (Seção 8.2.1.1), a camada de visualização está incluída como trabalhos futuros.

8.2.1.7 Análises da **C7**

A última contribuição elencada (**C7**) foi a mais apontada pelos especialistas, talvez pelo fato de ser a mais genérica e ter um efeito "guarda-chuva" em relação às demais. As outras contribuições podem gerar redução de custos no projeto, mas tem outros aspectos que não são cobertos pelas contribuições elencadas e que também podem reduzir custos, motivo pelo qual ela foi incluída como uma contribuição da abordagem. Nesta tese, entende-se redução de custo em termos financeiro e de desenvolvimento. Para esse último, está incluído a redução do tempo do desenvolvimento, manutenção e o melhor gerenciamento dos profissionais envolvidos. O gerenciamento dos profissionais foi relatado pelo **E3** na Tabela 23, no trecho "Também melhora o gerenciamento dos papéis dos desenvolvedores no processo". Porém, para uma possível redução do número de profissionais, os modelos de plataformas teriam que estar quase completos e/ou com poucas implementações pendentes. Caso isso ocorra, os desenvolvedores terão pouco ou nenhum contato com as linguagens nativas nos modelos de plataformas e assim poderão trabalhar quase que exclusivamente com a GPL. Tal dedução foi inferida diante dos relatos de aumento de produtividade para os casos de reutilizar modelos de plataformas parcialmente prontos, segundo os especialistas **E5** em **Q2** (Item B) e **Q5** (Item A e B), **E4** na **Q5** (Item B e C) e **E2** na **Q3** (Item C).

A **C7** foi citada como benefício devido ao uso da GPL (**E4** e **E5** na **Q2**, item B), dos modelos de plataformas (todos especialistas na **Q4**, item A), na estrutura da abordagem de forma geral (todos especialistas na **Q6**), na manutenção (todos especialistas na **Q7**) e colocado como um dos benefícios principais pelos **E1** e **E3** na **Q8**.

Como pode ser observado, existem evidências de redução de custos indicadas pelos especialistas nos principais modelos da abordagem, porém a falta de suporte para a camada de visualização é uma limitação (**L6**) que afeta essa contribuição, além da **C1**. Isso porque a interface deve ser confeccionada em cada IDE e projetada para se comunicar com a camada *Controller* gerada de forma automática pela abordagem. Como já informado na Seção 8.2.1.1, o suporte à camada de visualização está incluído nos trabalhos futuros. Um dos benefícios do suporte de interfaces pela abordagem é a geração de códigos compatíveis e prontos para se comunicar com as outras camadas. Dessa forma, o desenvolvedor não precisaria analisar a estrutura de comunicação entre as camadas para produzir as interfaces, diferente do que precisa ser feito atualmente.

Além da **L6**, a **L8** também foi relacionada à **C7** pelos especialistas, pelo fato de implementar as funções dos modelos de plataformas, já discutido em outras contribuições. Os especialistas **E1**, **E3** e **E4** na Tabela 21 relatam ganhos de produtividade condicionado ao reúso de um modelo de plataforma parcialmente pronto. Porém, caso precise recorrer às linguagens nativas e ao IDE de cada plataforma para implementar todos os modelos de plataformas, a redução de custos provida pela abordagem pode ser prejudicada. Esse

ponto é apontado também como limitação pelo **E5** na maior parte das questões. Na Tabela 24, os indícios contrários à **C7** são diversificados e em sua maior parte também já foram discutidos nas seções anteriores, exceto o relato do **E3**, que aponta limitações para a necessidade de testar os códigos gerados apenas nos ambientes nativos. De fato, existe essa necessidade, mas a detecção de erros ou desempenho insatisfatório dos códigos gerados podem ser ajustadas na própria abordagem e transformar essa limitação em vantagem, conforme relato do **E2** na Tabela 18 (Item A).

O **E3** (Tabela 19, item B) aponta a necessidade de especialistas de diferentes linguagens para a abordagem. Mas, na mesma questão (Item A) o **E3** indica o aumento de produtividade através do reúso das funções globais e melhora no gerenciamento dos papeis da equipe na Tabela 23.

Além da análise das evidências sobre as contribuições apontadas pelos especialistas que responde a métrica **M1**, a métrica **M2** também foi analisada nessa avaliação. Essa última métrica se referente à funções que não foram possíveis ser especificadas através da abordagem devido a alguma limitação, portanto, conforme os relatos dos próprios especialistas não houveram evidências dessas funções nas tarefas desenvolvidas.

8.2.2 Limitações

O segundo estudo apresentado para esta tese trouxe evidências a favor e contra todas as sete contribuições elencadas para essa abordagem. A quantidade de evidências a favor é maior, o que indica que a abordagem é bem-sucedida em alcançar seus objetivos.

Os estudos também descobriram oito limitações importantes, conforme discutido nas seções anteriores. Elas estão resumidos novamente e foram classificados em uma ordem decrescente de gravidade:

- **L8**. Necessidade de recorrer às linguagens nativas e ao IDE de cada plataforma;
- **L2**. Resultados somente a longo prazo devido ao fato da implementação e atualização de funções globais (**E3 e E5**);
- **L1**. Necessidade de testes de código na própria plataforma, incluindo testes de validação no editor e testes nos códigos gerados;
- **L6**. Geração da camada de visualização (**E4**);
- **L5**. Necessidade de entender um novo padrão de programação;
- **L3**. Confiabilidade dos códigos gerados;
- **L4**. Garantia de desempenho dos códigos gerados da abordagem; e

- **L7**. Uso de alguma ferramenta para auxiliar a modelagem visual dos conceitos de domínio.

A **L8** é a mais fundamental. Inicialmente, a abordagem deveria oferecer suporte a um ambiente único para a criação do software, mas no final ficou claro que os IDEs das plataformas nativas precisam ser usados ao criar as implementações para as plataformas. À medida que o tempo passa e as plataformas são implementadas e testadas, há menos necessidade de recorrer aos IDEs nativos. Essa também é a essência da limitação **L2**.

L1 e **L6** também são importantes, mas eles podem ser resolvidos fornecendo novas plataformas especificamente para essas tarefas. Em relação à **L1**, é possível criar funções globais para teste, para que casos de teste independentes de plataforma possam ser especificados na GPL. Em relação à **L6**, funções e detalhes globais podem ser usados para representar conceitos comuns de interface, como manipulação de eventos e gerenciamento de estado. Pode até ser possível criar plataformas para interfaces específicas de domínio, com base no fato de que muitos sistemas do mesmo domínio compartilham interfaces semelhantes. A plataforma de testes foi explorada em uma terceira avaliação, e a camada *View* é assunto de trabalhos futuros.

L5 também é mencionada como uma limitação importante. O conceito de funções globais e detalhes globais requerem o aprendizado de uma nova técnica de programação.

L3 e **L4** são o outro lado da flexibilidade fornecida pela abordagem. O engenheiro de software tem mais controle sobre o código gerado, mas também é responsável por garantir que esteja correto e adequado. No Xamarin, por exemplo, que é uma solução comercial, o desenvolvedor pode confiar que o fabricante gastou mais tempo testando as implementações, mas por outro lado ele fica completamente dependente do fabricante.

Por fim, **L7** foi citado por um dos especialistas como uma possibilidade de aprimoramento. Essa limitação não foi percebida na prática, no entanto, uma vez que a linguagem textual foi reconhecida como intuitiva e fácil de entender.

8.2.3 Ameaças à validade

Existem alguns aspectos que devem ser discutidos em relação a possíveis ameaças à validade nos estudos.

Encontrar especialistas foi uma tarefa muito difícil, pois exigia muitas horas de trabalho voluntário. Os pesquisadores recorreram a seus contatos na academia e na indústria e foi possível encontrar cinco especialistas que cumprem os requisitos (mínimo de 10 anos de experiência em desenvolvimento móvel e web e alguma experiência em desenvolvimento multiplataforma). No entanto, essa amostragem não foi ideal, pois os participantes conheciam os pesquisadores e poderiam ter fornecido respostas tendenciosas. No entanto,

a quantidade de observações negativas feitas pelos especialistas foi considerável e indica que o viés pode não ter sido muito forte. Também indica que o nível de conhecimento deles era adequado o suficiente para permitir identificar problemas reais, indo além de uma análise superficial.

Também foi muito difícil definir tarefas para os especialistas. Todo o estudo levou em média 14 horas de trabalho de cada especialista. Eles não foram realizados ininterruptamente, pois os especialistas trabalhavam uma pequena quantidade de tempo todos os dias. Levou vários dias para que todos os especialistas concluíssem todas as tarefas, portanto eles ficaram em contato com a abordagem por um longo tempo. Idealmente, as tarefas devem ser mais longas, para que os especialistas possam ter uma compreensão mais profunda de todos os seus detalhes. No entanto, isso não foi possível porque a disponibilidade dos especialistas para o estudo era apenas parcial.

Ainda com relação às tarefas, cobriu-se diferentes cenários para exercitar as diferentes contribuições da abordagem. Por exemplo, a tarefa **T3** solicitou que os especialistas modificassem uma plataforma existente para dar suporte a um novo banco de dados e a **T5** pediu que os especialistas incluíssem uma nova plataforma (android *Wear*). No final, foi possível coletar evidências sobre todas as contribuições. No entanto, nenhuma tarefa exigia que um sistema inteiro fosse desenvolvido do zero. Provavelmente, isso poderia levar à identificação de mais benefícios e limitações, mas, novamente, não era possível devido à disponibilidade parcial dos especialistas.

Outra ameaça está relacionada a um possível fator externo. Todos os especialistas identificaram ganhos de produtividade, mas acreditamos que parte desses ganhos possa ter se originado nos modelos Apache Velocity, e não na abordagem. Embora esses modelos do Apache Velocity por si só não promovam a reutilização e o desenvolvimento entre plataformas, eles podem reduzir bastante o esforço de programação, especialmente no domínio CRUD. Havia algumas tarefas que não envolviam o uso de tais modelos, mas sua presença no estudo pode ter influenciado as respostas dos especialistas, que atribuíram 100% dos ganhos de produtividade à abordagem, enquanto, na realidade, uma fração disso provavelmente resultou do uso do Velocity.

Uma das limitações (**L5**) refere-se à curva de aprendizado referente à abordagem, em particular o uso de funções globais e detalhes globais. Os especialistas conseguiram concluir as tarefas, mas tinham exemplos a seguir, portanto, o aspecto da aprendizagem não foi completamente abordado pelos estudos.

Por fim, o segundo estudo teve forte subjetividade, em dois aspectos. Opiniões de especialistas são subjetivas. Para reduzir essa subjetividade, selecionamos cinco especialistas com experiência, para aumentar a confiança de que suas opiniões não são isoladas ou baseadas em suposições. O segundo aspecto subjetivo foi a análise das opiniões dos especialistas, realizada pelos pesquisadores. Para reduzir a subjetividade em relação a esse

aspecto, dois pesquisadores (aluno e orientador) conduziram a análise separadamente e em seguida, foi alcançado um consenso na discussão.

8.2.4 Considerações finais

Nessa avaliação definiu-se um conjunto de tarefas que visou motivar especialistas a explorar os principais aspectos da abordagem proposta. Tal avaliação foi conduzida para coletar, através de uma entrevista, informações consistentes que ajudaram a identificar evidências a favor e contra em relação às contribuições elencadas para esta tese. Os principais pontos das entrevistas foram discutidos nesse capítulo, permitindo alcançar algumas conclusões sobre a abordagem.

Os resultados e relatos obtidos após a execução das avaliações apresentaram indícios positivos em relação à abordagem, apesar de terem sido apontadas algumas limitações. Uma das principais se refere à implementação das funções globais nos modelos de plataformas. Os especialistas foram unânimes em responder que a definição dos detalhes técnicos para as plataformas precisam estar parcialmente completos para poder explorar o potencial de produtividade da abordagem. Alguns especialistas também alegaram que, mesmo sem os modelos de plataformas prontos, a abordagem ainda oferece certo ganho em produtividade.

A implementação das funções dos modelos de plataformas nos IDEs nativos pode ser uma limitação ao ganho de produtividade a curto prazo. Por essa razão um dos trabalhos futuros é motivar a comunidade a ajudar a popular esses modelos com as especificidades de cada plataforma em repositórios colaborativos.

Em suma, os relatos dos especialistas apresentaram resultados satisfatórios e mostraram que a arquitetura da abordagem pode auxiliar, de fato, com o aumento de produtividade na confecção de sistemas multiplataforma. Conforme apresentado durante a avaliação, a abordagem foi modularizada e sua arquitetura projetada para superar os desafios do desenvolvimento multiplataforma. Embora existam limitações, as principais contribuições elencadas para esta tese foram alcançadas em diferentes níveis. Entre elas a extensão da abordagem, a inclusão de novas plataformas, a redução do custo na configuração e o aproveitamento dos conceitos similares. Tais contribuições são fundamentais para tornar a abordagem desta tese distinta em relação aos trabalhos relacionados na indústria e academia pesquisados.

Destaca-se também que a abordagem conseguiu considerar o desenvolvimento de sistemas para dispositivos móveis, web e RA, através de uma arquitetura expansível pelo desenvolvedor. Conforme apresentado nos trabalhos relacionados, são poucas as abordagens que permitem o desenvolvimento para outras plataformas além dos móveis. As que oferecem não são preparadas para a evolução pelos desenvolvedores, gerando as limita-

ções discutidas ao longo da pesquisa. Entre elas, desenvolvedor "preso" às liberações de funções pelos fabricantes, riscos da abordagem ser descontinuada, não permitem ajustes no desempenho dos códigos gerados e não possuem um gerenciador fácil para alterar o modo de funcionamento de certas funções entre os dispositivos.

Soluções para o desenvolvimento multiplataforma vêm sendo pesquisadas há mais de dez anos e novos trabalhos surgem com certa frequência na indústria e na academia. Esse cenário mostra que ainda existem muitos campos de estudos nessa área e que as soluções existentes ainda estão imaturas e em um estágio preliminar (CHEN et al., 2019). Portanto, os trabalhos futuros e aprimoramentos identificados ao longo desta pesquisa são fundamentais para o avanço dessa linha de pesquisa visando uma contribuição mais consistente para a comunidade.

9 Terceira avaliação: plataforma de testes de unidade

Conforme discutido anteriormente, a abordagem multiplataforma concebida para esta tese foi estruturada visando, entre outros aspectos, a sua extensão pelos próprios desenvolvedores. Essa característica foi avaliada na prova de conceito (Capítulo 7) e avaliação por especialistas (Capítulo 8). Cada uma dessas avaliações mostraram a inclusão de novas funções e novas plataformas pelos usuários finais, podendo citar exemplos como: inclusão de mecanismos de persistência em memória e disco (Capítulo 7), inclusão da plataforma de RA (Capítulo 7), mecanismos de conexão com outros banco de dados (Capítulo 8), inclusão da plataforma *smartwatch* (Capítulo 8), entre outros.

Na avaliação conduzida por especialistas, uma das principais limitações citadas foi a ausência de mecanismos para testar as especificações genéricas feitas através da GPL. De fato tais especificações não podem ser testadas diretamente, pois o código da GPL não é executado, e sim utilizado como base para geração de código. Mas é possível criar casos de testes de unidade genéricos para os códigos gerados e executá-los nos IDEs nativos. Essa opção não foi visualizada pelos especialistas, embora seja um dos diferenciais desta tese. Portanto, neste capítulo é apresentada a inclusão de testes de unidade genéricos para as principais plataformas envolvidas na prova de conceito. A inclusão dos testes de unidade visa demonstrar que essa limitação apontada pelos especialistas não é da abordagem em si, e que isso poderia ser resolvido, trazendo assim mais indícios da robustez da abordagem.

9.1 Estudo de caso. Inclusão de testes de unidade na abordagem

O teste de unidade (HUIZINGA; KOLAWA, 2007) constitui na análise da menor parte testável de um sistema. A estratégia adotada neste estudo foi a seguinte: para cada função do tipo "*assert*" existente nos *frameworks* de teste que se deseja usar, como por exemplo, `assertEquals`, que é o tipo mais comum, deve-se criar uma função global. Cada plataforma irá prover uma implementação dessa função para sua linguagem, considerando-se detalhes como assinatura do método, ordem dos parâmetros, entre outros. Detalhes como anotações de código podem ser inseridos através dos detalhes globais. Então, basta utilizar a função global ao se criar o casos de teste na GPL.

A utilização do teste de unidade na abordagem foi dividida em quatro etapas. A primeira etapa foi a declaração de duas funções globais para dar suporte à geração de código. A Listagem 9.1 apresenta um exemplo que define um detalhe global (`UnitTestAnnotation`, linha 1) para anotações de testes de unidade, e uma função glo-

bal, chamada `globalAssertEquals()` (linha 2), que possui detalhes técnicos referentes ao uso do `assertEquals` para cada plataforma. Os detalhes técnicos referentes à inclusão dos testes de unidade ou qualquer nova funcionalidade na abordagem são considerados extensões, projetadas para serem feitas pelos desenvolvedores através dos modelos de plataformas. Para tal, não é necessária nenhuma modificação no gerador ou gramática da abordagem, considerados caixa preta.

```
1 globalDetails UnitTestAnnotation
2 global globalAssertEquals(expected: double, actual: double, msg:
    string): void
```

Listagem 9.1 – Funções globais para o suporte aos testes de unidade.

O detalhe global `UnitTestAnnotation` é utilizado para incluir as anotações exigidas pelas linguagens C# e Java. Como a linguagem Swift não exige anotações a sua implementação nos modelos de plataformas é vazia.

A Listagem 9.2 mostra a implementação de uma plataforma para testes de unidade para dispositivos iOS. Nessa listagem, é apresentada a segunda etapa do processo, que consiste na declaração das plataformas e a implementação das funções globais. Neste caso, está sendo declarada a plataforma `UnitTestingiOS`, como pode ser visto na linha 1. A implementação do detalhe global (`UnitTestAnnotation`) fica vazia (linha 3), pois a linguagem Swift não requer anotações para os casos de teste. Nesta listagem é possível ver também a implementação da função global `globalAssertEquals` (linhas 5 a 9).

```
1  platform UnitTestingiOS: Swift {
2    implementsGlobalDetails UnitTestAnnotation{
3      ,,,,,,
4    }
5    implementsGlobal globalAssertEquals(expected: double, actual:
      double, msg: string): void{
6    ,,,
7    XCTAssertEqual(expected, actual, msg);
8    ,,,
9    }
10 }
```

Listagem 9.2 – Declaração do modelo de plataforma para teste de unidade na linguagem Swift.

A Listagem 9.3 mostra a declaração de uma plataforma de testes para a plataforma Java. Neste caso, o detalhe global `UnitTestAnnotation` é implementado (linhas 2-4) de forma a inserir a anotação `@Test`, requerida pelo *framework* JUnit[1]. A implementação da função global `globalAssertEquasl` (linha 5-9) traz a versão Java dessa função.

[1] https://junit.org/junit5/docs/current/user-guide/

```
platform UnitTestingAndroid: Java {
  implementsGlobalDetails UnitTestAnnotation{
  '''@Test'''
  }
  implementsGlobal globalAssertEquals(expected: double, actual:
   double, msg: string): void{
    '''
    assertEquals(expected, actual, msg);
    '''
  }
}
```

Listagem 9.3 – Declaração do modelo de plataforma para teste de unidade na linguagem Java.

Uma vez declaradas e implementadas as funções globais com os detalhes de plataformas, o desenvolvedor pode especificar de forma genérica os testes de unidade através da GPL. A especificação dos testes de unidade é a terceira etapa do processo, que consiste em preparar a lógica dos testes e utilizar as funções globais previamente implementadas. A Listagem 9.4 apresenta a classe `BankAccountTest` usada para testar um programa responsável por receber o nome e o saldo de um cliente de um banco financeiro e realizar operações de crédito e débito em conta. Essa classe utiliza a função global `globalAssertEquals` (linha 2) e passa os parâmetros: valor esperado (`expected`), valor atual (`actual`) e a mensagem de erro (`msg`), exigidos na sua declaração. Na linha 3 pode ser visualizada a chamada do detalhe global `UnitTestAnnotation`, responsável por adicionar as anotações pertinentes aos testes quando necessário. O método `balanceTest` (linha 4) contém a especificação genérica do teste de unidade desenvolvido através da GPL. Nesse método um saldo é atribuído para o cliente (linha 10), assim como o valor do débito (linha 11) e o valor esperado para o novo saldo (linha 12). Após as especificações, a função global `globalAssertEquals` é chamada (linha 21) e os parâmetros passados.

```
class BankAccountTest{
  usesGlobal  globalAssertEquals(expected: double, actual: double
   , msg: string): void{}
  usesGlobalDetails UnitTestAnnotation
  operation balanceTest(): void{
    m_customerName: string
    m_customerName := ''Juliano''
    beginningBalance: double
    debitAmount: double
    expected: double
    beginningBalance := 11.99
    debitAmount := 4.55
```

```
12      expected := 7.44
13      account :BankAccount
14      account.setM_customerName(m_customerName)
15      account.setM_balance(beginningBalance)
16      account.Debit(debitAmount)
17      actual : double
18      actual := account.getM_balance()
19      msg: string
20      msg := ‘‘Account not debited correctly’’
21      global globalAssertEquals(expected, actual, msg)
22    }
23  }
```

Listagem 9.4 – Especificação genérica do teste de unidade através do GPL.

Por fim, a quarta etapa é a especificação da geração de código no modelo de *deploy*. Nesse modelo a classe `BankAccountTest` é configurada para as plataformas web, Android e iOS. O resultado são códigos completos para os testes de unidade nas três linguagens especificadas, prontas para ser executadas nos IDEs de cada plataforma.

Além do teste de igualdade, outros tipos de testes podem ser especificados através das funções globais, bastando para isso adicionar os comandos específicos para cada plataforma nos modelos de plataformas.

Os comandos de asserções normalmente aceitam diferentes quantidades e tipos de parâmetros, com detalhes diferentes de retorno de acordo com a necessidade, além dos parâmetros já apresentados. Como exemplo, existe um parâmetro que ignora ou não o tamanho da caixa de texto para verificar a igualdade em uma comparação e o *CultureInfo* que apresenta a data de acordo com a região. Também deve existir uma versão diferente para cada variante dos comandos de asserção. Por exemplo, em JUnit existe um método `assertEquals` para parâmetros do tipo `double`, um método `assertEquals` para parâmetros do tipo `int`, e assim por diante. Outras linguagens também podem ter suas variantes. Para esses casos, as quantidades e tipos de parâmetros deveriam ser especificados na própria GPL. Para evitar que o desenvolvedor precise declarar diferentes funções globais, a abordagem oferece, além da flexibilidade das funções e detalhes globais, a possibilidade de utilizar *templates* do Apache Velocity para gerar código customizado (ver Seção 5.2.2). Isso permite que seja declarada uma única função global, a ser utilizada pelo desenvolvedor, e os detalhes sejam tratados nas implementações.

A Listagem 9.5 apresenta o uso da função global `unifiedGlobalAssertEquals`, onde as especificações dos parâmetros são definidas através de uma classe de configuração na GPL. Na declaração dessa função (linha 2) é possível visualizar a classe genérica `E`, usada para a passagem dos parâmetros. Na linha 6, a implementação da função

`unifiedGlobalAssertEquals` é apresentada, utilizando a plataforma iOS como exemplo. Nessa implementação é possível visualizar a classe genérica `E` sendo passada como parâmetro e o Velocity sendo usado para percorrer as suas propriedades (linha 8), deixando o uso dos parâmetros dinâmicos. O uso da função global pode ser visto na GPL (linha 16), com a mesma lógica do método `balanceTest()` apresentado na listagem anterior (Listagem 9.4 - linhas 4-22) e substituída por reticência. A diferença está na classe de configuração `TestConfiguration` (linhas 22-27), onde é possível especificar diferentes parâmetros para as operações de testes de acordo com a necessidade.

```
1  A) Global function declaration
2  global <E> unifiedGlobalAssertEquals(): void
3
4  B) Implementation in the platform model
5  platform UnitTestingiOS: Swift {
6    implementsGlobal <E> unifiedGlobalAssertEquals(): void{
7    ’’’
8    XCTAssertEqual(#foreach($a in $E.attributes)${a.name}#if(
       $foreach.hasNext) ,#end#end);
9    ’’’
10   }
11 }
12
13 C) Generic specification in GPL
14 class BankAccountTestConfiguration{
15   usesGlobalDetails UnitTestAnnotation
16   usesGlobal <TestConfiguration> unifiedGlobalAssertEquals():
       void{
17     m_customerName: string
18     ...
19     msg := ‘‘Account not debited correctly’’
20   }
21 }
22 class TestConfiguration{
23   expected: double
24   actual: double
25   ignoreCase: bool
26   msg: string
27 }
```

Listagem 9.5 – Especificação genérica do teste de unidade através do GPL usando uma classe para configuração dos parâmetros.

Após a configuração do modelo de *deploy* os códigos dos testes são gerados para as

plataformas declaradas. A Figura 31 apresenta os códigos gerados através da abordagem sendo executado nas plataformas iOS, web e Android respectivamente. Para esse teste foi adicionado um erro proposital no método responsável em debitar o valor do saldo. Portanto em todas as plataformas foi acusado o mesmo erro (*Account not debited correctly ==> expected: <7.44> but was: <16.54>*), como pode ser visualizado nos destaques em vermelho.

9.2 Resultados e Discussão

Visando manter a consistência nas avaliações, esta seção apresenta o mapeamento desta avaliação com as contribuições elencadas para esta tese, conforme a métrica **M3** definida para essa avaliação no Capítulo 6.

Assim como apresentado na avaliação 1 e 2, a GPL permitiu especificações em alto nível e em um único ambiente, mostrando indícios para a contribuição **C1**. Além disso, se mostrou genérica suficiente para especificar rotinas de testes de modo multiplataforma, permitindo o aproveitamento dos conceitos similares entre as plataformas (**C2**).

A inclusão dos detalhes técnicos referentes aos testes de unidade nos modelos de plataforma mostrou a capacidade de expansão da abordagem pelos desenvolvedores. Indicando que a abordagem alcançou também a contribuição **C5**, referente à extensão da abordagem através dos modelos de plataformas. As funções globais e detalhes globais implementados nos modelos de plataformas podem ser reutilizados em outras rotinas de testes de unidade de forma independente de plataforma, mostrando evidências também para a contribuição **C6** (Reutilização de códigos entre domínios).

Por fim, com os modelos de plataformas implementados junto com os comandos de asserções, a especificação de novas rotinas de testes na GPL terão um custo de desenvolvimento e manutenção menores (**C7**). Isso ocorre porque, além das rotinas serem especificadas em alto nível, elas são genéricas e podem ser usadas por qualquer plataforma declarada na abordagem. Embora os códigos gerados foram testados com sucesso nos IDEs nativos, as bibliotecas de testes referente a cada plataforma (*imports*) foram adicionados à mão posteriormente. Isso se fez necessário porque a abordagem ainda não tem suporte para adicionar *imports* e *namespaces* nas implementações das plataformas, mas isso será corrigido e está incluído como trabalhos futuros.

Vale ressaltar também que o estudo de inclusão de teste de unidades está em uma fase inicial. Não foram feitos testes mais aprofundados, com desenvolvedores ou sistemas maiores. Foi mais uma prova de que de fato a abordagem tem suporte a extensão pelos desenvolvedores de diferentes tipos de funcionalidades. Fato que torna os usuários finais da abordagem independentes das liberações dos fabricantes para utilizar uma nova funcionalidade.

Além da métrica **M3**, essa avaliação analisou evidências em respostas as métricas **M1** e **M2**. Para a **M1**, referente às listas de funções específicas de plataformas que foram incluídas, as funções foram: `UnitTestAnnotation`, `globalAssertEquals` e `unifiedGlobalAssertEquals` para as plataformas Android, iOS e web. Referente as rotinas genéricas especificadas através da GPL (**M2**), podemos citar a classe `BankAccountTest`. Tal classe foi especificada de forma genérica através da GPL com rotinas para testar o funcionamento correto das funções pertinentes a classe `BankAccount`. Além da especificação, os códigos foram gerados e executados nos ambientes reais (Android, iOS e web), comprovando a eficiência da abordagem.

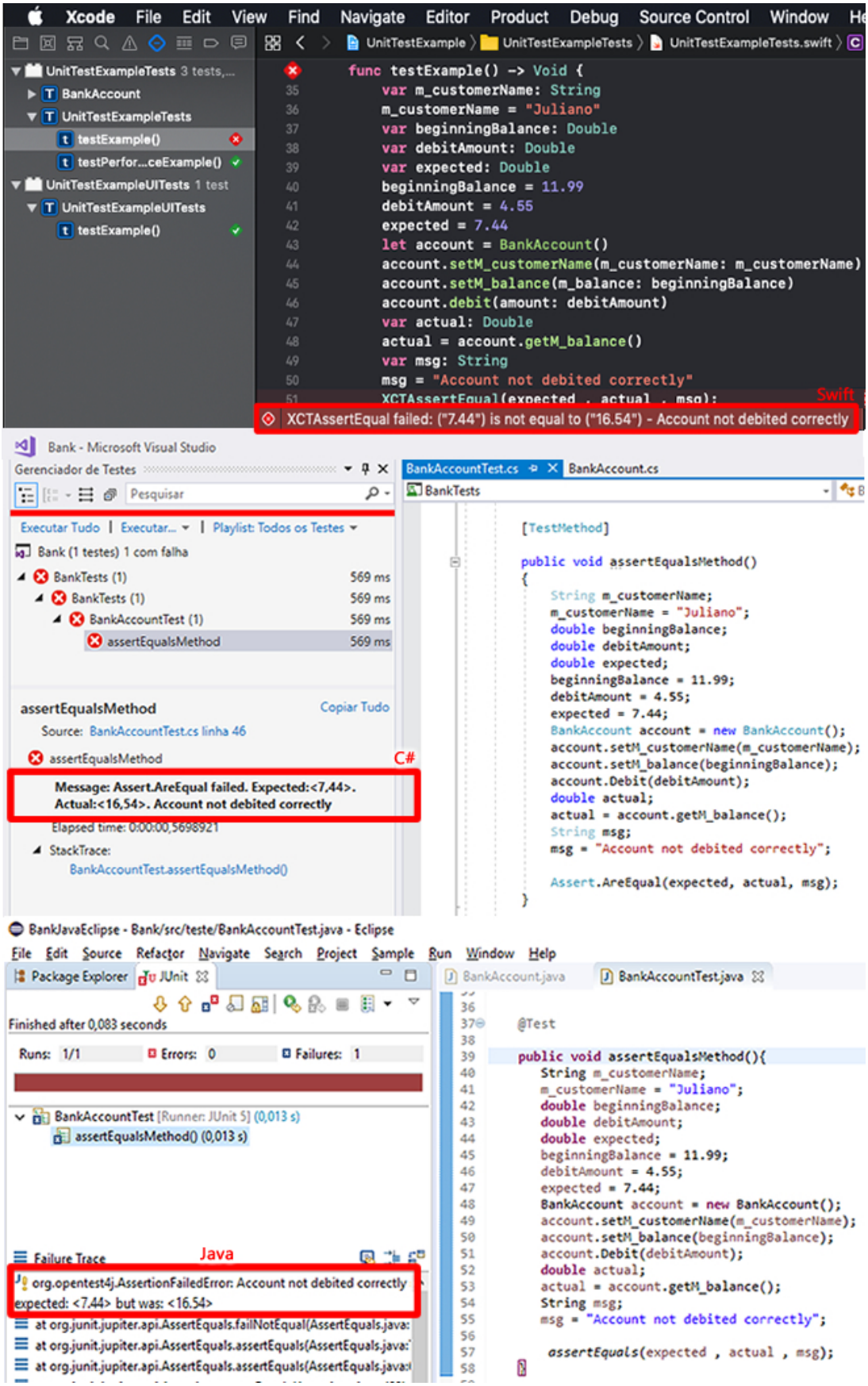

Figura 31 – Códigos gerados através da abordagem sendo executado nas plataformas iOS, web e Android.

10 Considerações Finais

As soluções para o desenvolvimento multiplataforma são investigadas há muito tempo, e novas pesquisas surgem frequentemente na indústria e na academia. Existem muitas áreas abertas para pesquisa, pois as soluções existentes ainda não estão maduras o suficiente (CHEN et al., 2019).

Nesse cenário, este trabalho apresenta uma abordagem para o desenvolvimento multiplataforma, onde a ideia principal é manter o desenvolvimento em um ambiente de modelagem único e independente de plataformas, e usar geradores para produzir código executável. A abordagem aproveita os conceitos semelhantes entre as plataformas e facilita a inclusão e a troca de plataformas usadas pelo sistema. Como consequência, os desenvolvedores podem se beneficiar da reutilização fornecida pelos geradores de código e ainda ter controle para personalizar, estender ou incluir novas plataformas. Além da possibilidade de gerenciar a distribuição de funcionalidades genéricas entre as plataformas consideradas por cada sistema.

A abordagem teve como objetivo sete contribuições, selecionadas após uma análise das soluções disponíveis na indústria e na academia. O principal diferencial desta tese é a junção dessas sete contribuições. Existem soluções tanto na academia quanto na indústria que entregam contribuições similares, mas de forma separada. Até onde foi pesquisado, nenhum trabalho aborda o tema com a ótica adotada aqui. Seguindo uma visão mais holística, acreditamos que uma solução para o desenvolvimento multiplataforma deve considerar um sistema multiplataforma como um sistema, e não várias versões de um sistema.

10.1 Contribuições da pesquisa

Três estudos foram realizados para avaliar a abordagem. Houve algumas limitações associadas à abordagem, mas no geral as evidências coletadas favorecem as contribuições.

Em particular, os estudos demonstraram a capacidade de estender ou modificar plataformas existentes, o que não é uma tarefa projetada aos usuários das ferramentas pesquisadas. Isso foi feito, em um primeiro momento, pelos pesquisadores, mas também foi feito pelos especialistas, que com sucesso incluíram plataformas como *smartwatch* e um SGBD diferente. Além da inclusão da plataforma *smartwatch*, os especialistas estenderam a abordagem adicionando funções como exibição e seleção de produtos nos modelos de plataformas. Também foi criado um novo tipo de dispositivo (RA) e o incluímos na abordagem com sucesso. Nas ferramentas pesquisadas, essas tarefas seriam muito difíceis

ou impossíveis nos casos das ferramentas comerciais, pois a modificação de plataformas e a inclusão de novas plataformas dependem do fornecedor da ferramenta, e não do desenvolvedor. Os estudos também demonstraram a distribuição de funcionalidades entre plataformas, o que também é um recurso não encontrado nas ferramentas pesquisadas. Essas ferramentas, em sua maior parte, consideram que cada solução individual é igual ou semelhante, em termos de funcionalidade quando se trata de aproveitamento de código e nenhuma fornece formas para gerenciar a distribuição de funcionalidades. A GENEXUS (MARINHO; RESENDE, 2015) por exemplo, permite a criação de versões diferentes do aplicativo para plataformas distintas, porém é necessário criar a especificação completa em locais diferentes na ferramenta, prejudicando o aproveitamento de códigos. Esses recursos fornecem à abordagem uma forma distinta da academia/indústria de como tratar o problema do desenvolvimento multiplataforma. O atendimento das sete contribuições elencadas para esta tese fornecem uma visão holística do desenvolvimento multiplataforma e dá autonomia para o desenvolvedor configurar a abordagem de acordo com a sua necessidade.

Além das três contribuições apontadas como exclusivas, a abordagem é considerada holística por também atender a outras quatro. A listagem a seguir apresenta todas as contribuições tratadas por essa tese:

- Modelagem dos conceitos de domínio em alto nível de abstração e em um único ambiente;
- Aproveitamento dos conceitos similares entre as plataformas na modelagem;
- Redução do custo na configuração de um sistema multiplataforma. Isso inclui distribuir as funcionalidades entre plataformas e alternar as plataformas que estão sendo usadas no sistema;
- Inclusão de novas plataformas na abordagem com menor impacto no código;
- Extensão da abordagem através dos modelos de plataformas;
- Reutilização de códigos entre domínios; e
- Redução do custo de desenvolvimento e manutenção.

As evidências obtidas em relação a essas contribuições corroboram a tese aqui defendida: é possível abordar de maneira única os diferentes desafios do desenvolvimento multiplataforma, adotando uma solução mais holística que trata o desenvolvimento como um esforço unificado.

10.2 Limitações

Os estudos também mostraram que atualmente não é possível abandonar completamente os IDEs nativos das plataformas. A inclusão de novos recursos nos modelos de plataformas da abordagem, na maior parte dos casos, precisa ser criada e testada nos ambientes nativos. Essa observação contraria o objetivo principal da abordagem de usar um único ambiente para criar o software. Esse problema tende a ser reduzido com o passar do tempo e os modelos de plataformas se tornarem mais completos para o reúso. Por essa razão os resultados podem aparecer de forma mais satisfatória somente a médio ou longo prazo, quando o uso da abordagem estiver em estágios mais avançados e houver uma quantidade razoável de implementações prontas nos modelos de plataformas.

A necessidade de entender um novo padrão de programação também apareceu como um fator limitante na avaliação com especialistas. De fato, uma nova forma de desenvolver sistemas precisa ser entendida com a abordagem, mas esse aspecto tende a trazer vantagens a médio e longo prazo, segundo os próprios especialistas.

Outro aspecto limitante indicado pelos especialistas é a confiabilidade dos códigos gerados, que podem ser superadas com o uso de testes de unidade, implementados para a terceira avaliação.

10.3 Trabalhos futuros

Futuros trabalhos estão sendo planejados para superar as limitações identificadas nos estudos, conforme descrito na Seção 8.2.2. Em particular, planejamos realizar mais estudos sobre testes multiplataforma com base no que foi alcançado até então. Espera-se que a abordagem suporte a especificação de testes mais avançados sem exigir modificações em seus elementos. Também espera-se usar a abordagem para gerar códigos também para a camada *View*. Esse é um problema mais complexo, mas é provável que dentro de um escopo mais fechado (exemplo: telas de um sistema *e-commerce*) isso possa ser viável e trazer benefícios.

Algumas possibilidades práticas de melhoria também foram identificadas. Um deles é o uso de conceitos de Programação Orientada a Objetos nos modelos de plataforma, permitindo a herança entre os modelos de plataforma e facilitando a reutilização e o gerenciamento do sistema. Um dos especialistas também sugeriu um repositório de modelos. Embora seja possível reutilizar modelos na abordagem copiando os arquivos, maneiras mais elaboradas de reutilização podem ser objeto de trabalho futuro, para que os modelos de plataformas possam ser reutilizados em uma comunidade. De maneira semelhante aos repositórios de códigos associados aos sistemas de gerenciamento de dependências, esse processo pode ajudar a reduzir o esforço inicial mencionado pelos especialistas ao usar a

abordagem.

Outros trabalhos pode ser a inclusão de interoperabilidade entre as plataformas. Além de gerar código para ser executado localmente pelos dispositivos, gerar código que faz os dispositivos "conversarem" entre si de forma automática, utilizando arquiteturas existentes de comunicação, como o REST/Json ou REST/XML. Também espera-se testar dispositivos proprietários. Fizemos o nosso próprio, e deu certo, mas seria interessante testar com dispositivos já existentes, para identificar possibilidades de melhoria. Dispositivos como os da Internet das Coisas podem ser utilizados para avaliar a capacidade de inclusão de plataformas na abordagem. Para esse caso, múltiplos dispositivos pequenos com funcionalidades semelhantes, especificados de de uma única forma através da GPL. Também é possível explorar a questão de equipes de especialistas dando manutenção e/ou criando novas funções para a inclusão nos modelos de plataformas. Isso pode contribuir para melhorar a experiência de desenvolvedores nos primeiros contato com a abordagem, permitindo um foco melhor nos conceitos do domínio e não nas especificidades de plataformas. Outro aspecto desse trabalho futuro é a possibilidade de um novo modelo de negócio, onde especialistas preparam códigos genéricos específicos de plataformas visando o aumento de produtividade.

Evoluir o suporte ferramental também é opção para trabalhos futuros. Nesta tese foi utilizado exclusivamente o IDE Eclipse para as implementações. A integração melhor da abordagem com os IDEs pode facilitar a ida-e-vinda de códigos entre as ferramentas. Possibilidades como links entre os códigos da GPL e o código dos IDEs podem ser opções para melhorar essa integração e agilizar as implementações nos modelos de plataformas.

A gramática também precisa de aperfeiçoamentos, pois ela ainda é um protótipo e pode evoluir para incluir outras construções. A linguagem funcional poderia servir de base para uma provável evolução da GPL. Isso porque, além de ser uma linguagem atualmente muito usada, o conceito de funções está bastante enraizado na abordagem.

10.4 Lições aprendidas

Considerando a quantidade e a frequência do surgimento de soluções para o desenvolvimento multiplataforma, a descoberta e a manutenção das possíveis contribuições para esta tese precisou ser realizada de forma iterativa com o contexto. Era evidente desde o inicio da pesquisa que as soluções existentes não estavam maduras e era comum a sua substituição com certa frequência pelos desenvolvedores. Tal comportamento foi uma das motivações para esta pesquisa e o uso da forma iterativa e incremental com o contexto foi uma decisão acertada. O acompanhamento contínuo do contexto foi essencial para encontrar e manter os aspectos que tornariam a abordagem única, além da necessidade de analisar de forma frequente as ameaças à validade da pesquisa.

Outra lição aprendida foi a importância da realização de avaliações com especialistas experientes e que estejam fora do grupo de pesquisa. As informações coletadas por essas avaliações mostraram aspectos (contribuições, limitações, dicas de melhorias, problemas encontrados) que seriam impossíveis visualizar pelos membros do próprio grupo de pesquisa. Especialistas experimentes na área foi outro ponto essencial para uma avaliação mais abrangente e comparativa com as ferramentas existentes atualmente. Muitos dos trabalhos futuros surgiram dos apontamentos dos especialistas, inclusive a terceira avaliação desta tese.

Por fim, entendemos que qualquer projeto é construído no tempo, e que a qualquer momento sua validade pode ser comprometida através de novos lançamentos da indústria ou academia. Dessa forma, encontrar contribuições válidas e ter tempo para implementá-las e testá-las é algo realmente difícil em um cenário globalizado. Por essa razão, acreditamos de fato que a abordagem desenvolvida tem contribuições válidas para a comunidade acadêmica e industrial e gostaríamos realmente de explorar o seu potencial através de mais pesquisas.

10.5 Produção acadêmica desta tese

Nesta seção são apresentadas as submissões do autor desta pesquisa durante o período de doutorado, explicando sua contribuição.

1) Elaboração de um auxílio regular FAPESP para o projeto tema desta tese (Uma abordagem holística para o desenvolvimento de software multiplataforma), processo: #2015 / 24429-1 e #2017 / 25343-9. O projeto foi aprovado e teve vigência entre out/2017 e set/2019. O projeto contou com o apoio de um aluno de Iniciação Científica, sob a orientação do autor desta tese, para desenvolver um dispositivo novo de realidade aumentada. Além disso, teve uma avaliação conduzida com usuários em ambiente simulado, que rendeu um artigo e um protótipo baseado na plataforma RaspBerry;

2) Elaboração do artigo intitulado: *OfflineManager: A Lightweight Approach for Managing Offline Status in Mobile Applications*. O artigo apresenta uma biblioteca para dispositivos móveis construída para ajudar os desenvolvedores a fornecer funcionalidade *offline* em seus aplicativos. O autor desta tese participou de uma das avaliações realizadas para comprovar as contribuições da pesquisa. No momento da escrita desta tese o artigo encontra-se em processo de avaliação;

3) Elaboração do artigo intitulado: *A Simple Library to Support the Development of Augmented Reality Software*. Este artigo descreve uma biblioteca para facilitar o desenvolvimento de software que utiliza imagens projetadas sobre objetos do mundo real para promover diferentes tipos de interatividade, explorando o conceito de realidade aumentada, além de estudos com usuários. No momento da escrita desta tese o artigo encontra-se

em processo de avaliação;

4) Elaboração do artigo intitulado: *A holistic approach for cross-platform software development*. Esse artigo resultou do projeto desta pesquisa e descreve a abordagem proposta, incluindo a GPL e as avaliações. No momento da escrita desta tese o artigo encontra-se em processo de avaliação; e

5) Elaboração do artigo intitulado *Cross-platform Development Challenges: A Survey from the Model-Driven Development Perspective*. O artigo apresenta uma revisão sistemática sobre as principais soluções para sistemas multiplataformas baseadas em DSDM. No momento da escrita desta tese o artigo encontra-se em processo de avaliação.

Agradecimentos

O trabalho de pesquisa relatado nesta tese recebeu apoio financeiro dos processos #2015 / 24429-1 e #2017 / 25343-9 - Fundação de Pesquisa de São Paulo (FAPESP) - Brasil.

Referências

ACCELEO. *Acceleo*. 2010. Disponível em: <http://www.eclipse.org/acceleo/>. Citado 3 vezes nas páginas 45, 57 e 79.

ACERBIS, R. et al. Model-driven development of cross-platform mobile applications with webratio and ifml. In: IEEE PRESS. *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*. [S.l.], 2015. p. 170–171. Citado na página 55.

AMATYA, S.; KURTI, A. Cross-platform mobile development: challenges and opportunities. In: *ICT Innovations 2013*. [S.l.]: Springer, 2014. p. 219–229. Citado na página 35.

ANNEKE KLEPPE, JOS WARMER, W. B. *The Model Driven Architecture : Practice and Promise*. [S.l.: s.n.], 2004. 192 p. ISBN 032119442X. Citado na página 76.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. In: *Encyclopedia of Software Engineering*. [S.l.]: Wiley, 1994. Citado na página 100.

BEHRENS, H. Mdsd for the iphone: developing a domain-specific language and ide tooling to produce real world applications for mobile devices. In: ACM. *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. [S.l.], 2010. p. 123–128. Citado na página 46.

BENOUDA, H. et al. Mda approach to automate code generation for mobile applications. In: *Mobile and Wireless Technologies 2016*. [S.l.]: Springer, 2016. p. 241–250. Citado na página 45.

BENOUDA, H. et al. Automatic code generation within mda approach for cross-platform mobiles apps. In: IEEE. *2017 First International Conference on Embedded & Distributed Systems (EDiS)*. [S.l.], 2017. p. 1–5. Citado na página 46.

BERNARDES, T. F.; MIYAKE, M. Y. Cross-platform mobile development approaches: A systematic review. *IEEE Latin America Transactions*, IEEE, v. 14, n. 4, p. 1892–1898, 2016. Citado na página 26.

BĒRZIŠA, S. et al. Capability driven development: an approach to designing digital enterprises. *Business & Information Systems Engineering*, Springer, v. 57, n. 1, p. 15–25, 2015. Citado na página 59.

BETTINI, L. *Implementing domain-specific languages with Xtext and Xtend*. [S.l.]: Packt Publishing Ltd, 2016. Citado na página 96.

BOTTURI, G. et al. Model-driven design for the development of multi-platform smartphone applications. In: IEEE. *Specification & Design Languages (FDL), 2013 Forum on*. [S.l.], 2013. p. 1–8. Citado 2 vezes nas páginas 47 e 49.

BRAMBILLA, M.; MAURI, A.; UMUHOZA, E. Extending the interaction flow modeling language (ifml) for model driven development of mobile applications front end. In:

SPRINGER. *International Conference on Mobile Web and Information Systems*. [S.l.], 2014. p. 176–191. Citado na página 51.

BUCANEK, J. Model-view-controller pattern. *Learn Objective-C for Java Developers*, Springer, p. 353–402, 2009. Citado na página 106.

BUCHHOLZ, S.; SCHILL, A. Adaptation-aware web caching: Caching in the future pervasive web. In: SPRINGER. *Kommunikation in Verteilten Systemen (KiVS)*. [S.l.], 2003. p. 55–66. Citado na página 36.

CHARLAND, A.; LEROUX, B. Mobile application development: web vs. native. *Communications of the ACM*, ACM, v. 54, n. 5, p. 49–53, 2011. Citado na página 25.

CHEN, S. et al. Automated cross-platform gui code generation for mobile apps. In: IEEE. *2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*. [S.l.], 2019. p. 13–16. Citado 5 vezes nas páginas 16, 46, 47, 179 e 188.

CIMINO, M. G.; MARCELLONI, F. An efficient model-based methodology for developing device-independent mobile applications. *Journal of Systems Architecture*, Elsevier, v. 58, n. 8, p. 286–304, 2012. Citado na página 45.

COOK, S. et al. *Domain-specific development with visual studio dsl tools*. [S.l.]: Pearson Education, 2007. Citado na página 79.

COSENTINO, V.; TISI, M.; IZQUIERDO, J. L. C. A model-driven approach to generate external dsls from object-oriented apis. In: SPRINGER. *International Conference on Current Trends in Theory and Practice of Informatics*. [S.l.], 2015. p. 423–435. Citado na página 60.

DAGEFÖRDE, J. C. et al. Generating app product lines in a model-driven cross-platform development approach. In: IEEE. *System Sciences (HICSS), 2016 49th Hawaii International Conference on*. [S.l.], 2016. p. 5803–5812. Citado 3 vezes nas páginas 47, 48 e 54.

DALMASSO, I. et al. Survey, comparison and evaluation of cross platform mobile application development tools. In: IEEE. *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. [S.l.], 2013. p. 323–328. Citado na página 26.

DANIEL LUCRÉDIO. *Uma Abordagem Orientada a Modelos para Reutilização de Software Uma Abordagem Orientada a Modelos para Reutilização de Software*. 1–273 p. Tese (Doutorado) — Universidade de São Paulo, , São Carlos, 2009. Citado 3 vezes nas páginas 7, 76 e 77.

DANYL, B. *Mobile Marketing Statistics 2019*. 2019. Disponível em: <http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/>. Citado 2 vezes nas páginas 23 e 27.

DAVID FRANKEL. *Getting Started with MOF , XMI , and JMI*. 2006. 1–4 p. Citado na página 78.

DEURSEN, A. V.; VISSER, E.; WARMER, J. Model-driven software evolution: A research agenda. *Technical Report Series TUD-SERG-2007-006*, Delft University of Technology, Software Engineering Research Group, 2007. Citado na página 76.

DIEP, C.-K.; TRAN, Q.-N.; TRAN, M.-T. Online model-driven ide to design guis for cross-platform mobile applications. In: ACM. *Proceedings of the Fourth Symposium on Information and Communication Technology*. [S.l.], 2013. p. 294–300. Citado na página 46.

ELOUALI, N. et al. Mimic: leveraging sensor-based interactions in multimodal mobile applications. In: ACM. *CHI'14 Extended Abstracts on Human Factors in Computing Systems*. [S.l.], 2014. p. 2323–2328. Citado na página 45.

FISCHER, P. Cross platform mobile tools. 2015. Disponível em: <http://www.infoq-.com/research/cross-platform-mobile-tools>. Citado na página 34.

FORCE, O. *Why do we need standards for the modernization of existing systems*. 2012. Citado na página 43.

FRANCE, R. et al. Model-driven Development of Complex Software : A Research Roadmap Model-driven Development of Complex Software : A Research Roadmap. p. 37–54, 2007. Citado 2 vezes nas páginas 75 e 85.

FRANCE, R. B. et al. Model-driven development using uml 2.0: promises and pitfalls. *Computer*, IEEE, v. 39, n. 2, p. 59–66, 2006. Citado na página 75.

FRANCESE, R. et al. Model-driven development for multi-platform mobile applications. In: SPRINGER. *International Conference on Product-Focused Software Process Improvement*. [S.l.], 2015. p. 61–67. Citado 2 vezes nas páginas 50 e 51.

FRANZAGO, M.; MUCCINI, H.; MALAVOLTA, I. Towards a collaborative framework for the design and development of data-intensive mobile applications. In: ACM. *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*. [S.l.], 2014. p. 58–61. Citado na página 51.

GERBER, A.; RAYMOND, K. MOF to EMF : There and Back Again. *03 Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, v. 3, p. 60–64, 2003. Citado na página 78.

GOAER, O. L.; WALTHAM, S. Yet another dsl for cross-platforms mobile development. In: ACM. *Proceedings of the First Workshop on the globalization of domain specific languages*. [S.l.], 2013. p. 28–33. Citado 2 vezes nas páginas 47 e 48.

GRACE, P.; PICKERING, B.; SURRIDGE, M. Model-driven interoperability: engineering heterogeneous iot systems. *Annals of Telecommunications*, Springer, v. 71, n. 3-4, p. 141–150, 2016. Citado na página 53.

GRAPHITI. *Graphiti*. 2008. Disponível em: <http://www.eclipse.org/graphiti-/documentation/gettingStarted.php>. Citado na página 78.

HEITKÖTTER, H.; HANSCHKE, S.; MAJCHRZAK, T. A. Comparing cross-platform development approaches for mobile applications. *Web Information Systems and Technologies*, v. 140, p. 120–138, 2013. Citado 5 vezes nas páginas 25, 26, 31, 33 e 34.

HEMEL, Z.; VISSER, E. *Declaratively programming the mobile web with Mobl*. [S.l.]: ACM, 2011. Citado na página 50.

HINKEL, G. et al. Experiences with model-driven engineering in neurorobotics. In: SPRINGER. *European Conference on Modelling Foundations and Applications*. [S.l.], 2016. p. 217–228. Citado na página 60.

HINKEL, G.; GOLDSCHMIDT, T. Tool support for model transformations: on solutions using internal languages. *Modellierung 2016*, Gesellschaft für Informatik eV, 2016. Citado 2 vezes nas páginas 59 e 61.

HOFFMAN, D. L.; NOVAK, T. P. Consumer and object experience in the internet of things: An assemblage theory approach. *Journal of Consumer Research*, Oxford University Press, v. 44, n. 6, p. 1178–1204, 2018. Citado na página 17.

HOFMANN, M.; BECK, A.; CONDRY, M. *Example services for network edge proxies*. [S.l.], 2000. Citado na página 37.

HÖPFNER, H. et al. Towards a target platform independent specification and generation of information system apps. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 36, n. 4, p. 1–5, 2011. Citado 2 vezes nas páginas 50 e 51.

HOYOS, J. R.; GARCÍA-MOLINA, J.; BOTÍA, J. A. A domain-specific language for context modeling in context-aware systems. *Journal of Systems and Software*, Elsevier, v. 86, n. 11, p. 2890–2905, 2013. Citado na página 75.

HU, H. et al. Studying the consistency of star ratings and reviews of popular free hybrid android and ios apps. *Empirical Software Engineering*, Springer, v. 24, n. 1, p. 7–32, 2019. Citado na página 27.

HUIZINGA, D.; KOLAWA, A. *Automated defect prevention: best practices in software management*. [S.l.]: John Wiley & Sons, 2007. Citado na página 180.

IDC. Smartphone os market share. 2020. Disponível em: <http://www.idc.com-/prodserv/smartphone-os-market-share.jsp>. Citado 2 vezes nas páginas 25 e 31.

INAYATULLAH, M.; AZAM, F.; ANWAR, M. W. Model-based scaffolding code generation for cross-platform applications. In: IEEE. *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. [S.l.], 2019. p. 1006–1012. Citado na página 55.

JONES, C.; JIA, X. The axiom model framework: transforming requirements to native code for cross-platform mobile applications. In: IEEE. *Evaluation of Novel Approaches to Software Engineering (ENASE), 2014 International Conference on*. [S.l.], 2014. p. 1–12. Citado 3 vezes nas páginas 41, 47 e 53.

KHAN, M. E.; KHAN, F. et al. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Computer Sciences and Applications*, v. 3, n. 6, p. 12–1, 2012. Citado na página 42.

KO, M. et al. Extending uml meta-model for android application. In: IEEE. *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*. [S.l.], 2012. p. 669–674. Citado 2 vezes nas páginas 45 e 52.

KOLOVOS, D. S. et al. Raising the level of abstraction in the development of GMF-based graphical model editors. *2009 ICSE Workshop on Modeling in Software Engineering*, Ieee, p. 13–19, maio 2009. Disponível em: <http://ieeexplore.ieee.org-/lpdocs/epic03/wrapper.htm?arnumber=5069891>. Citado 2 vezes nas páginas 75 e 78.

KONECKI, M. Mobile and responsive web applications. 2014. Disponível em: <http://hci.si/2014/12/10/mobile-responsive-web-applications/>. Citado na página 35.

KRAEMER, F. A. Engineering android applications based on uml activities. In: SPRINGER. *International Conference on Model Driven Engineering Languages and Systems*. [S.l.], 2011. p. 183–197. Citado na página 45.

LACHGAR, M.; ABDALI, A. Modeling and generating native code for cross-platform mobile applications using dsl. *Intelligent Automation & Soft Computing*, Taylor & Francis, v. 23, n. 3, p. 445–458, 2017. Citado 3 vezes nas páginas 47, 50 e 52.

LATIF, M. et al. Cross platform approach for mobile application development: A survey. In: IEEE. *2016 International Conference on Information Technology for Organizations Development (IT4OD)*. [S.l.], 2016. p. 1–5. Citado na página 79.

LISA, M. *What's The Most Popular Mobile Operating System In Your Country*. 2013. Disponível em: <http://www.ibtimes.com/android-vs-ios-whats-most-popular-mobile-operating-system-your-country-1464892>. Citado na página 24.

LITAYEM, N.; DHUPIA, B.; RUBAB, S. Review of cross-platforms for mobile learning application development. *IJACSA*, v. 6, n. 1, 2015. Citado na página 27.

MAJCHRZAK, T. A.; ERNSTING, J.; KUCHEN, H. Model-driven cross-platform apps: Towards business practicability. 2015. Citado na página 47.

MARCOTTE, E. *Responsive Web Design 2010*. 2010. Disponível em: <www-.alistapart.com/articles/responsive-web-design>. Citado 2 vezes nas páginas 27 e 35.

MARINHO, E. H.; RESENDE, R. F. Native and multiple targeted mobile applications. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2015. p. 544–558. Citado 3 vezes nas páginas 42, 44 e 189.

MARTINEZ, L.; PEREIRA, C.; FAVRE, L. Migrating c/c++ software to mobile platforms in the adm context. *International Journal of Interactive Multimedia & Artificial Intelligence*, v. 4, n. 3, 2017. Citado 3 vezes nas páginas 57, 60 e 72.

MEHTA, K.; JHA, J. Web cache technique responsive web design. 2014. Citado na página 36.

MIN, B.-K. et al. A uml metamodel for smart device application modeling based on windows phone 7 platform. In: IEEE. *TENCON 2011-2011 IEEE Region 10 Conference*. [S.l.], 2011. p. 201–205. Citado na página 46.

MIRAVET, P. et al. Framework for the declarative implementation of native mobile applications. *IET software*, IET, v. 8, n. 1, p. 19–32, 2014. Citado 3 vezes nas páginas 56, 81 e 84.

MORENO-VOZMEDIANO, R. et al. Cross-site virtual network in cloud and fog computing. *IEEE Cloud Computing*, IEEE, v. 4, n. 2, p. 46–53, 2017. Citado na página 37.

OMG. *MDA Guide Version 1.0.1.* 2003. Disponível em: <http://www.omg.org>. Citado 2 vezes nas páginas 77 e 78.

OMG. *MOF 2.0 Query / Views / Transformations Final Adopted Specification.* 2005. Disponível em: <http://www.omg.org>. Citado 2 vezes nas páginas 45 e 78.

PARADA, A. G.; BRISOLARA, L. B. de. A model driven approach for android applications development. In: IEEE. *Computing System Engineering (SBESC), 2012 Brazilian Symposium on.* [S.l.], 2012. p. 192–197. Citado na página 45.

PERCHAT, J.; DESERTOT, M.; LECOMTE, S. Component based framework to create mobile cross-platform applications. *Procedia Computer Science*, Elsevier, v. 19, p. 1004–1011, 2013. Citado 3 vezes nas páginas 42, 47 e 72.

PÉREZ-CASTILLO, R.; GUZMAN, I. G.-R. D.; PIATTINI, M. Knowledge discovery metamodel-iso/iec 19506: A standard to modernize legacy systems. *Computer Standards & Interfaces*, Elsevier, v. 33, n. 6, p. 519–532, 2011. Citado na página 61.

REYNOLDS, C. Ten of the Best Cross-Platform. 2017. Disponível em: <http://-/appindex.com/blog/ten-best-cross-platform-development-mobile-enterprises/>. Citado na página 35.

RIBEIRO, A.; SILVA, A. R. da. Xis-mobile: A dsl for mobile applications. In: ACM. *Proceedings of the 29th Annual ACM Symposium on Applied Computing.* [S.l.], 2014. p. 1316–1323. Citado na página 47.

RIEGER, C. et al. A model-driven approach to cross-platform development of accessible business apps. In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing.* [S.l.: s.n.], 2020. p. 984–993. Citado 2 vezes nas páginas 47 e 50.

ROUDAKI, A.; KONG, J.; YU, N. A classification of web browsing on mobile devices. *Journal of Visual Languages & Computing*, Elsevier, v. 26, p. 82–98, 2015. Citado na página 37.

SABRAOUI, A. et al. Mdd approach for mobile applications based on dsl. In: IEEE. *2019 International Conference of Computer Science and Renewable Energies (ICCSRE).* [S.l.], 2019. p. 1–6. Citado 2 vezes nas páginas 46 e 47.

SABRAOUI, A.; KOUTBI, M. E.; KHRISS, I. Gui code generation for android applications using a mda approach. In: IEEE. *Complex Systems (ICCS), 2012 International Conference on.* [S.l.], 2012. p. 1–6. Citado na página 46.

SANTANA, L. et al. Adaptação de páginas web para dispositivos móveis. *Simpósio Brasileiro de Sistemas Multimídia e Web*, p. 1–8, 2007. Citado na página 36.

SCHULER, A.; FRANZ, B. Rule-based generation of mobile user interfaces. In: IEEE. *Information Technology: New Generations (ITNG), 2013 Tenth International Conference on.* [S.l.], 2013. p. 267–272. Citado na página 46.

SHIOTSU, Y. *Web Development*. 2014. Disponível em: <https://www.upwork.com-/blog/2014/03/web-development-101-top-web-development-languages-2014/>. Citado 3 vezes nas páginas 24, 26 e 27.

SILVA, A. R. da. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, Elsevier, v. 43, p. 139–155, 2015. Citado na página 53.

SILVA, A. R. da et al. Preliminary experience using jetbrains mps to implement a requirements specification language. In: IEEE. *Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the*. [S.l.], 2014. p. 134–137. Citado na página 59.

SILVA, L. P. da; ABREU, F. B. e. Model-driven gui generation and navigation for android bis apps. In: IEEE. *Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on*. [S.l.], 2014. p. 400–407. Citado na página 45.

SNEED, H. M. Estimating the costs of a reengineering project. In: IEEE. *Reverse Engineering, 12th Working Conference on*. [S.l.], 2005. p. 9–pp. Citado na página 42.

SOLT, P. Swift vs. objective-c: 10 reasons the future favors swift. 2015. Disponível em: <http://www.infoworld.com/article/2920333/mobile-development/swift-vs-objective-c-10-reasons-the-future-favors-swift.html>. Citado na página 32.

SON, H. S.; KIM, W. Y.; KIM, R. Y. C. Mof based code generation method for android platform. *International Journal of Software Engineering and Its Applications*, v. 7, n. 3, p. 415–426, 2013. Citado 2 vezes nas páginas 45 e 61.

STAHL, T.; VOLTER, M. *Model-driven software development: technology, engineering, management*. [S.l.]: J. Wiley & Sons, 2006. Citado 2 vezes nas páginas 41 e 42.

STATISTA. Market share of BlackBerry OS. 2020. Disponível em: <https://www-.statista.com/statistics/271243/blackberry-os-market-share-in-the-united-kingdom-uk>. Citado na página 33.

STROULIA, E. et al. Wl++: code generation of multi-platform mobile clients to restful back-ends. In: IEEE. *Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on*. [S.l.], 2015. p. 136–137. Citado na página 50.

TAENTZER, G.; VAUPEL, S. Model-driven development of mobile applications: Towards context-aware apps of high quality. In: *PNSE@ Petri Nets*. [S.l.: s.n.], 2016. p. 17–29. Citado 2 vezes nas páginas 47 e 48.

UMUHOZA, E.; BRAMBILLA, M. Model driven development approaches for mobile applications: A survey. In: SPRINGER. *International Conference on Mobile Web and Information Systems*. [S.l.], 2016. p. 93–107. Citado 6 vezes nas páginas 16, 41, 42, 43, 62 e 66.

UMUHOZA, E. et al. Automatic code generation for cross-platform, multi-device mobile apps: Some reflections from an industrial experience. In: ACM. *Proceedings of the 3rd International Workshop on Mobile Development Lifecycle*. [S.l.], 2015. p. 37–44. Citado 2 vezes nas páginas 41 e 43.

USMAN, M.; IQBAL, M. Z.; KHAN, M. U. A model-driven approach to generate mobile applications for multiple platforms. In: IEEE. *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*. [S.l.], 2014. v. 1, p. 111–118. Citado na página 47.

USMAN, M.; IQBAL, M. Z.; KHAN, M. U. A product-line model-driven engineering approach for generating feature-based mobile applications. *Journal of Systems and Software*, Elsevier, v. 123, p. 1–32, 2017. Citado 3 vezes nas páginas 47, 48 e 72.

VAUPEL, S. et al. Model-driven development of mobile applications for android and ios supporting role-based app variability. *Software & Systems Modeling*, Springer, v. 17, n. 1, p. 35–63, 2018. Citado 2 vezes nas páginas 47 e 50.

VIANA, W.; ANDRADE, R. M. Xmobile: A mb-uid environment for semi-automatic generation of adaptive applications for mobile devices. *Journal of Systems and Software*, Elsevier, v. 81, n. 3, p. 382–394, 2008. Citado na página 16.

VOELTER, M. Best practices for dsls and model-driven development. *Journal of Object Technology*, v. 8, n. 6, p. 79–102, 2009. Citado na página 75.

WARREN, I. et al. Push notification mechanisms for pervasive smartphone applications. *IEEE Pervasive Computing*, IEEE, v. 13, n. 2, p. 61–71, 2014. Citado na página 28.

WASSERMAN, A. I. Software engineering issues for mobile application development. *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10*, ACM Press, New York, New York, USA, v. 978-1-4503, n. 6, p. 397–400, 2010. Disponível em: <http://portal.acm.org/citation.cfm?doid=1882362.1882443>. Citado na página 16.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: ACM. *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. [S.l.], 2014. p. 38. Citado na página 40.

XPAND. Xpand. 2008. Disponível em: <http://wiki.eclipse.org/Xpand>. Citado na página 79.

XTEXT. *XText*. 2006. Disponível em: <http://www.eclipse.org/Xtext>. Citado na página 78.

A Apêndice

Gramática da abordagem

```
grammar br.dslcross.crossplatform.CrossPlatform with org.eclipse.xtext.common.Terminals

generate crossPlatform "http://www.dslcross.br/crossplatform/CrossPlatform"

Domainmodel:
  (elements+=AbstractElement)*;

AbstractElement:
  PackageDeclaration | Type | Import;

PackageDeclaration:
  'package' name=QualifiedName '{'
  (elements+=AbstractElement)*
  '}';

QualifiedName:
  ID ('.' ID)*;

Import:
  'import' importedNamespace=QualifiedNameWithWildcard;

QualifiedNameWithWildcard:
  QualifiedName '.*'?;

Type:
  DataType | Entity | Deploy | PlatformDecl | GlobalOld | GlobalEntity | Global;

PlatformDecl:
  'platform' name=ID ':' Base=('Java' | 'Swift' | 'CPlusPlus' | 'CSharp') '{'
    cmdList+=CommandPlat*
  '}';

```

```
33 GlobalOld:
34   'globalOld' ('<' genericTypes+=DSLGenericType (',' genericTypes
     +=DSLGenericType)* '>')?
35   name=ID '(' (params+=Parameter (',' params+=Parameter)*)? ')'
     ':' type=DSLTypeRef ('<'genericTypesDecl=DSLGenericType'>')? (
     col1='[')?(array=INT)?(col2=']')?;
36
37 GlobalEntity:
38   'globalDetails' ('<' genericTypes+=DSLGenericType (','
     genericTypes+=DSLGenericType)* '>')? name=ID;
39
40 Global:
41   'global' ('<' genericTypes+=DSLGenericType (',' genericTypes+=
     DSLGenericType)* '>')?  name=ID '(' (params+=Parameter (','
     params+=Parameter)*)? ')' ':' type=DSLTypeRef ('<'
     genericTypesDecl=DSLGenericType'>')?;
42
43 Deploy:
44   'deploy' name=ID '{' (entityToDeploy+=Platform)* '}';
45
46 Platform:
47   ent=[Entity] ':' plat=[PlatformDecl] (':' Communication=('Input
     ' | 'Output' | 'InOut') (':'
48   EntityEspecification=EntityEsp)?)?;
49
50 EntityEsp:
51   EspEnt+=[Entity];
52
53 DataType:
54   'datatype' name=ID ('[')?(array=INT)?(']')? ('<' genericTypes+=
     DSLGenericType (',' genericTypes+=DSLGenericType)* '>')? ;
55
56 Entity:
57   'class' name=ID ('extends' superType=[Entity])? '{'
58   features += CommandFeature*
59   '}';
60
61 CommandFeature:
62   Attribute  | Operation | OperationGlobCallOld | EntityGlobCall
     | OperationGlobCall | StringTemplateFreeCod |
     CallAttributeObject;
63
```

```
64 CommandPlat:
65   Attribute | Operation  | OperationGlobalOld | EntityGlobal |
     OperationGlobal;
66
67 OperationGlobCallOld:
68   'usesGlobalOld'
69   ('<' concreteTypes+=[Type] (',' concreteTypes+=[Type])* '>')?
70   opGlobCall=[GlobalOld] '(' (params+=Parameter (',' params+=
     Parameter)*)? ')' ':' typeGlobal=DSLTypeRef ('<'
     genericTypesDecl=[Type]'>')? (col1='['?(array=INT)?(col2=']')
     ?;
71
72 EntityGlobCall:
73   'usesGlobalDetails' ('<' concreteTypes+=[Type] (','
     concreteTypes+=[Type])* '>')? opGlobCall=[GlobalEntity];
74
75 OperationGlobCall:
76   'usesGlobal' ('<' concreteTypes+=[Type] (',' concreteTypes+=[
     Type])* '>')? ('[' paramList+=ParamList (',' paramList+=
     ParamList)* ']')? opGlobCall=[Global]'(' (params+=Parameter
     (',' params+=Parameter)*)? ')' (':' typeGlobal=DSLTypeRef)?
     ('<'genericTypesDecl=[Type]'>')? (col1='['?(array=INT)?(col2
     =']')? ('{' cmdList+=CommandOperation* '}')?;
77
78 Operation:
79   'operation' (paramExtra=ExprArit)? name=ID '(' (params+=
     Parameter (',' params+=Parameter)*)? ')' (':' type=DSLTypeRef
     ('<' genericTypesDecl+=[Type] (',' genericTypesDecl+=[Type])*
     '>')? (col1='['?(array=INT)?(col2=']')?)? '{'
80   cmdList+=CommandOperation* '}';
81
82 OperationGlobalOld:
83   'implementsGlobalOld' ('<' genericTypes+=DSLGenericType (','
     genericTypes+=DSLGenericType)* '>')?
84   opGlobal=[GlobalOld]'(' (params+=Parameter (',' params+=
     Parameter)*)? ')' ':' type=DSLTypeRef ('<'genericTypesDecl=
     DSLGenericType'>')? (col1='['?(array=INT)?(col2=']')? '{'
85    cmdList+=CommandOperation* '}';
86
87 EntityGlobal:
88   'implementsGlobalDetails' ('<' genericTypes+=DSLGenericType
     (',' genericTypes+=DSLGenericType)* '>')? opGlobal=[
```

```
    GlobalEntity] '{'
89    cmdList+=CommandOperation* '}';
90
91 OperationGlobal:
92   'implementsGlobal' ('<' genericTypes+=DSLGenericType (','
     genericTypes+=DSLGenericType)* '>')? ('[' paramList+=ParamList
     (',' paramList+=ParamList)* ']')? opGlobal=[Global]'(' (params
     +=Parameter (',' params+=Parameter)*)? ')' ':' type=DSLTypeRef
     ('<'genericTypesDecl=DSLGenericType'>')? (col1='[')?(array=INT)
     ?(col2=']')? '{'
93    cmdList+=CommandOperation* '}';
94
95 Attribute:
96   (many?='many')? name=ID ':' type=[Type] ('[')?(arrayAllow=INT)
     ?(']')? ('<'concreteTypes+=[Type] (',' concreteTypes+=[Type])?
     '>')?;
97
98 ParamList:
99   name=ID;
100
101 DSLTypeRef:
102   DSLConcreteTypeRef |
103   DSLGenericTypeRef;
104
105 DSLConcreteTypeRef:
106   ref=[Type];
107
108 DSLGenericTypeRef:
109   '<' ref=[DSLGenericType] '>';
110
111 DSLGenericType:
112   name=ID
113 ;
114
115 Parameter:
116   name=ID ':' type=DSLTypeRef (col1='[')?(array=INT)?(col2=']')?
     ('<' ref=[Type] '>')?;
117
118 CommandOperation:
119   Attribute             |
120   StringTemplateFreeCod |
121   AbrevCmd              |
```

```
122   MethodCallCmd          |
123   MethodCallGlobal       |
124   OpLogicoWhile          |
125   OpLogicoDoWhile        |
126   OpLogicoIF             |
127   OpLogicoFor            |
128   EntityGlobCall         |
129   SpecialCommand
130   ;
131
132 SpecialCommand:
133   command = 'break' |
134   'return' attr=[Attribute] |
135    commandCont = 'continue'
136 ;
137
138 StringTemplateFreeCod:
139   value+=STRINGTEMPLATE;
140
141 AbrevCmd:
142   expr1=[Attribute] OpIn=('++' | '--') | expr2=[Attribute] OpIn
     =('+=' | '-=') expr3=ExprArit | AssignmentCmdFor;
143
144 MethodCallCmd:
145       args+=AttributeCall)*)? ')';
146     obj=ID ('.' call+=ID ('(' (args+=AttributeCall (',' args+=
    AttributeCall)*)? ')')*)+;
147
148 Vector: {Vector}
149   (col1='[')?(array=INT)?(col2=']')?;
150
151 CallAttributeObject:
152   obj=AttribID ('.' call+=AttribID )* ':=' e=ExprArit;
153  (args+=AttributeCall ('.' args+=AttributeCall)*)?;
154  (args+=AttributeCall ('.' args+=AttributeCall)*)?;
155
156 MethodCallGlobal:
157   call1=MethodCallGlob | call2=MethodCallOp
158 ;
159
160 OpLogicoWhile:
161   left=ComLog e=ExprLogica '{' cmdList+=CommandOperation* '}';
```

```
162
163 OpLogicoDoWhile:
164   'Do{' cmdList+=CommandOperation* '}While' e=ExprLogica;
165
166 OpLogicoIF:
167   'If' e=ExprLogica '{' cmdList+=CommandOperation* '}' (opElseIf
      +=OpElseIf)* (opElse+=OpElse)?;
168
169 OpElseIf:
170   'ElseIf' expLog=ExprLogica '{' cmdList+=CommandOperation* '}';
171
172 OpElse:
173   {OpElse} 'Else' '{' cmdList+=CommandOperation* '}';
174
175 ExprLogica:
176   termos+=TermoLogico (oplog+=('||') termos+=TermoLogico)*;
177
178 TermoLogico:
179   fatores+=FatorLogico (oplog+=('&&') fatores+=FatorLogico)*;
180
181 FatorLogico:
182   {FatorLogico}(
183   (n='!')? v=[Attribute]
                        |
184   (n='!')? op1='true'
                          |
185   (n='!')? op2='false'
                          |
186   '(' exprLog=ExprLogica ')'
                          |
187   (n='!')? expr1=ExprL OpRel=('>' | '>=' |  '<' | '<=' | '==' |
      '<>') expr2=ExprL);
188
189 ComLog:
190   {ComLog} (comLog=('While'));
191
192 ExprL: {ExprL}(
193  i=INT ('.')? (iD = INT)? | vAtr1=[Attribute]('.'vAtr2+=
      AttribList)+);
194   v=ID (col1='[')?(array=INT)?(col2=']')? | i=INT ('.')? (iD =
      INT)? | vAtr1=ID('.'vAtr2+=AttribID)+) | strVar=STRING | opLog
      = 'true' | opLog = 'false';
```

```
195
196 AttribID:
197   name=ID v=Vector;
198
199 OpLogicoFor:
200   'For' exprLogic=ExprLogicFor '{' cmdList+=CommandOperation*
      '}';
201
202 ExprLogicFor:
203   inicio=AssignmentCmdFor ',' condicao=ExprLogica ',' abrevCmd=
      CommandOperation | '(' inicio=AssignmentCmdFor ','
204   condicao=ExprLogica ',' abrevCmd=CommandOperation ')';
205
206 AssignmentCmdFor:
207   left=[Attribute] '=' e=ExprArit;
208
209 ExprArit:
210   termos+=Termo (op+=('+' | '-') termos+=Termo)*;
211
212 Termo:
213   fatores+=Fator (op+=('*' | '/') fatores+=Fator)*;
214
215 Fator:
216   {Fator} (
217     p=Parameter                                          |
218                                  |
219     v=ID (col1='[')?(array=INT)?(array2=ID)?(col2=']')?|
220     d=DECIMAL                                            |
221     strVar=STRING                                        |
222     mcOperation=MethodCallOp                             |
223     mcGlobal=MethodCallGlob                              |
224     '(' expr=ExprArit ')'                                |
225     vAtr1=AttribID ('.' vAtr2+=AttribID (col+='(' (value+=ExprL
      (',' value+=ExprL)*)? ')')? )+);
226
227 OpLogico:
228   op1='true'    |
229   op2='false';
230
231 MethodCallOp:
232    call=[Operation] '(' (args+=AttributeCall (',' args+=
      AttributeCall)*)? ')';  call=[Operation] '(' (params+=Parameter
```

```
    (',' params+=Parameter)*)? ')'
233
234 MethodCallGlob:
235    'global' (obj=[Attribute] '.')? calledOpGlobal=[Global] '(' (
       args+=AttributeCall (',' args+=AttributeCall)*)? ')'
236 ;
237
238 AttributeCall:
239      name=ID (col1='[')?(array=INT)?(col2=']')? ('.'vAtr2+=
       AttribID (col+='(' value+=ExprL ')')? )*;
240
241 Expr:
242   INT;
243
244 terminal STRINGTEMPLATE:
245   '\'\'\'' .* '\'\'\'';
246
247 DECIMAL:
248    INT ('.')? (INT)?;
```

Listagem A.1 – Gramática da abordagem.