**Uma abordagem dirigida por modelos para desenvolvimento
de aplicações multi-paas**

Elias Adriano Nogueira da Silva

Tese de Doutorado do Programa de Pós-Graduação em Ciências de
Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Elias Adriano Nogueira da Silva

Uma abordagem dirigida por modelos para desenvolvimento de aplicações multi-paas

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Profa. Dra. Renata Pontin de Mattos Fortes

USP – São Carlos
Outubro de 2017

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

N42a Nogueira da Silva, Elias Adriano
 Uma abordagem dirigida por modelos para
desenvolvimento de aplicações multi-paas / Elias
Adriano Nogueira da Silva; orientador Renata
Pontin de Mattos Fortes. -- São Carlos, 2017.
 175 p.

 Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional) --
Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2017.

 1. Computação em Nuvem. 2. Desenvolvimento
Dirigido por Modelos. 3. Plataforma-como-um-Serviço.
4. Multicloud. 5. MultiPaaS. I. de Mattos Fortes,
Renata Pontin, orient. II. Título.

Elias Adriano Nogueira da Silva

A model-driven approach to develop multi-paas applications

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Profa. Dra. Renata Pontin de Mattos Fortes

USP – São Carlos
October 2017

Primeiramente, dedico este trabalho à minha mãe Abma Nogueira Lima que desde quando eu era criança sempre me incentivou a me tornar professor. Este sonho é tão meu quanto dela.

Dedico a todos os jovens estudantes da minha querida cidade natal Brumado-Bahia. Mais especialmente a todos que passaram e passarão pela Residência dos Estudantes Universitários e Secundaristas de Brumado (REUSB).

Dedico à escola Municipal de Educação Agamenon Santana (CMEAS) e ao Colégio Estadual de Brumado (CEB). As sementes que formam as bases para um futuro melhor e para se alcançar mais este sonho foram plantadas nessas escolas.

Dedico a Universidade Estadual do Sudoeste da Bahia (UESB), a Universidade Federal de São Carlos (DC-UFSCar) e ao Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP).

Dedico a todos os brasileiros e brasileiras que defendem uma escola pública, gratuita e de qualidade. Sem a qual não seria possível eu alcançar esse objetivo.

*Por fim, dedico este trabalho a todas os brasileiros e brasileiras **da ralé brasileira** descrita na obra do genial sociólogo Jessé Souza. Essa conquista mostra mais um exemplo de que é possível para um de nós e que a educação é o meio mais efetivo para se alcançar uma sociedade mais justa e menos desigual.*

AGRADECIMENTOS

Para quem desde criança persegue um sonho, o doutorado não dura apenas período compreendido entre a matrícula no programa e a defesa da tese, mas todo o tempo que vai desde a tomada de decisão a respeito de se obter a formação até o seu fim. Portanto, costumo dizer que demorei 20 anos para terminar o doutorado. Nesse período inteiro, foi necessário superar incontáveis barreiras até a obtenção de todo o saber que envolve a formação que, consequentemente, possibilitou que eu alcançasse esse objetivo. No meu contexto em especial, as barreiras envolveram sair do sertão da Bahia tão bem descrito na obra Os sertões de Euclides da Cunha e caminhar em direção (ao litoral) a uma das universidades de maior prestígio e excelência no Brasil. Sendo assim, não poderia poupar palavras para agradecer a todos que direta ou indiretamente contribuíram para concretização deste grande sonho.

Como diria Euclides da Cunha: o sertanejo antes de tudo é um forte. Assim, meu primeiro agradecimento é a Deus por ter me dado essa força, além de coragem, inteligência e saúde necessárias para que eu pudesse alcançar mais este objetivo. A luta é árdua para chegar até aqui, os dias longe da família, a saudade, a solidão, as noites sem dormir, a preocupação, os choques culturais, os dias longe do meu amado Brasil e do sertão da Bahia, as dificuldades psicológicas e financeiras, todas as batalhas só foram possíveis de superar graças a fé.

Agradeço a minha querida esposa Jéssica. Muito obrigado por tudo meu amor, obrigado pelo apoio e amor incondicional em todos os momentos da minha trajetória. Obrigado por tudo suportar por mim nesses últimos 11 anos, a distância, minha ausência, as dificuldades, enfim, por tomar também como suas todas as minhas lutas. Esta conquista também é sua.

Gostaria também de agradecer a minha orientadora Renata por ter me apoiado sempre. Agradeço por ter ido além dos limites da orientação, por ter sido amiga e paciente, por suportar meu jeito diferente, por ter me compreendido e me ajudado a melhorar sempre. Durante minha trajetória, me ajudou a vencer as piores batalhas. Meu imenso muito obrigado Renata, por ter, como uma mãe, me apoiado em tudo, por perceber minhas dificuldades e limitações e por ter servido como uma luz que iluminou um caminho de escuridão. Levarei para sempre na memória as lições aprendidas e guardo com carinho todos os seus conselhos. Também agradeço imensamente o apoio da minha querida orientadora de estágio durante meu período na Universidade NOVA de Lisboa, a Ana Moreira. Muito obrigado por me acolher e pelo imenso aprendizado que tive durante meus dias na FCT. Agradeço ainda meu orientador Daniel Lucrédio por ter me inserido na vida científica e me apoiado em tudo. Sei o quanto é difícil a carreira docente, vocês para mim são um grande exemplo de dedicação, de amor ao próximo e

do verdadeira influência que os professores exercem em nossas vidas. Muito obrigado mais uma vez.

Meu imenso muito obrigado a minha família como um todo, por compreender a minha ausência e torcer muito por mim. Agradeço mais especialmente a minha mãe Abma pelo amor incondicional. Por ter dado a criação e os incentivos corretos e necessários, por ter colocado em mim desde criança o amor pela educação e a paixão pela carreira de docente.

Aos meus queridos amigos e colegas de laboratório os quais compartilhei as alegrias e dificuldades desse percurso. Meu muito obrigado mais especialmente ao Du Cirilo, André Oliveira, Marcos Prado, Kleber, Alfredo, Carlos R. S. Júnior, Rafael Dias, Paulo Papotti, Fernando Proença, Ricardo Scheicher, Rodrigo Barroso, Marquinho, George, Vitrola, Lucas Porto, André Abade, Fernando Chagas, Jãum, Amir, Faimisson, Adam, Mundim, Mantovas, Durelli, Thiago Gaspar, Valdemar Netão, Flávio Horita, Diego R. C. Dias, Ricardo Oliveira, Renato Costa, Alex Guido, Odair, Sandra Rodrigues, Humberto, André Salgado, Stevão, Maykon MK, Daniel Boy, Jadson, Valéria e Geraldo. Aos queridos amigos da FCT-UNL Fernando, Erick, Cristiano, João Araújo, Miguel e Vasco. Agradeço ainda a todos os queridos amigos de Brumado-Ba, mais especialmente ao amigo Valdirando Alves (por ter despertado em mim o amor pela informática ainda quando adolescente) e aos amigos Junior, Dailton e Binho da DJ Turismo.

Ao querido povo do estado de São Paulo que por meio da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) (processo Nr. 2012/24487-3) financiou parcialmente esta pesquisa, ao programa Brazil Erasmus Mundus - BeMundos (processo BM13DM0002) pelo financiamento do meu período de intercâmbio na Universidade NOVA de Lisboa e a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES). Agradeço ainda aos professores Antônio Francisco do Prado (DC-UFSCar), Ellen Barbosa (ICMC-USP) e Vinícius Garcia (Cin-UFPE) pelas importantes contribuições dadas a este trabalho quando membros da minha banca de qualificação de doutorado. Por fim, agradeço a todos os funcionários e docentes do Instituto Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP) e do Departamento de Computação da Universidade Federal de São Carlos (DC-UFSCar).

*“Mais vale errar se arrebitando do que poupar-se para nada”
(Darcy Ribeiro)*

RESUMO

SILVA, E. A. N. **Uma abordagem dirigida por modelos para desenvolvimento de aplicações multi-paas**. 2017. 175 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

No contexto da Engenharia de Software para a Computação em Nuvem as pesquisas relacionadas ao tema são cada vez mais crescentes e investiga-se como a Computação em Nuvem influenciará no desenvolvimento de sistemas de uma maneira geral. A atividade de construir sistemas para nuvem é uma tarefa complexa, criar aplicações de múltiplas nuvens, sobretudo, no contexto do modelo de serviço Plataforma-como-um-Serviço (PaaS), é ainda mais agravada devido especificidades de plataformas de nuvem que podem tornar a tarefa de desenvolvimento repetitiva, custosa e dependente de um provedor específico. As abordagens dirigidas por modelos (MDE) resolvem alguns desses problemas, elas propõem que a modelagem e mecanismos de transformação utilizados para gerar código a partir de modelos são uma melhor maneira de desenvolver sistemas de software, ao invés da codificação pura. Portanto, visando investigar como combinar os benefícios da Computação em Nuvem alinhados ao MDE, foi desenvolvida **uma abordagem dirigida por modelos para desenvolvimento de aplicações multi-PaaS**. Em direção a este objetivo foi realizado um Estudo de Caso em colaboração com uma empresa da indústria. Essa colaboração permitiu a criação de implementações de referencia que possibilitaram o desenvolvimento de uma Linguagem Específica de Domínio (DSL) e metaprogramas que compõem a abordagem. Para avaliar a abordagem desenvolvida foi realizado um Estudo de Caso. Os resultados mostram que MDE pode não só resolver o problema, mas trazer benefícios adicionais em relação a abordagens tradicionais de desenvolvimento de sistemas. Este trabalho explora esses benefícios, apresenta uma maneira de unir recursos heterogêneos de nuvem por meio de uma abordagem dirigida por modelos e aplicações orientadas a serviço.

Palavras-chave: Computação em Nuvem, Linguagens Específicas de Domínio, multi-cloud, multi-PaaS, Engenharia de Software, *Cross-Origin Resource Sharing*, *Platform-as-a-Service*.

ABSTRACT

SILVA, E. A. N. **A model-driven approach to develop multi-paas applications.** 2017. 175 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

Cloud Computing is a computational paradigm that has increasingly been used in various sectors of industry and academia. Researchers have been studying how cloud technologies can influence several areas of science and research. In the context of Software Engineering, the researches related to cloud are increasingly increasing. Researchers are studying how to develop better cloud services offerings and how to find a strategy for combining existing resources to build improved services and solve problems. Building cloud systems is a complex task, in the context of the Platform-as-a-Service (PaaS) cloud service model; this activity is further aggravated by cloud platform specificities that can make the task of development repetitive, costly, and platform-specific. Model-driven approaches (MDE) solve some of these issues; they propose that the modeling and transformation mechanisms used to generate code from models are a better way to develop software systems, rather than pure coding. Development with MDE is a comprehensive and relevant research area and needs to be better explored in a wide range of contexts. Therefore, in order to investigate how to combine the benefits of *multi-cloud* applications aligned to the MDE, we developed a model-driven approach to build multi-PaaS applications. Toward this objective, we performed a case study in collaboration with an industry company. This collaboration allowed the creation of reference implementations that enabled the development of a Domain Specific Language (DSL) and metaprograms that constitute the approach. To evaluate the approach, we performed a case study. The results show that MDE can not only solve the problem, but also bring additional benefits over traditional systems development approaches. This work explores these benefits, presents a way to combine heterogeneous cloud resources through a service-driven model and application-driven approach.

Keywords: Cloud Computing, Domain-Specific Languages, multi-cloud, multi-paas, Software Engineering, Cross-Origin Resource Sharing, Platform-as-a-Service.

LISTA DE ILUSTRAÇÕES

Figura 1 – Linha do tempo da história da Computação em Nuvem – principais acontecimentos relacionados à evolução da Computação em Nuvem	33
Figura 2 – Ontologia da Computação em Nuvem	37
Figura 3 – Arquitetura de Referência da Computação em Nuvem proposta pelo NIST	39
Figura 4 – Atores e modelos de serviços da Computação em Nuvem	43
Figura 5 – Tipos de interoperabilidade	55
Figura 6 – Classificação arquitetural do inter-cloud	58
Figura 7 – Arquitetura do Cloud4SOA	65
Figura 8 – Relação entre métodos e ofertas de PaaS no Cloud4SOA	66
Figura 9 – Taxonomia MD*	73
Figura 10 – Principais elementos de MDE	78
Figura 11 – Uso de elementos MDE	79
Figura 12 – Arquitetura de metamodelagem	81
Figura 13 – MDE para geração de aplicações portáteis	83
Figura 14 – MDE para geração de aplicações interoperáveis	84
Figura 15 – Arquitetura de referência	93
Figura 16 – Visão do <i>back-end</i> da aplicação de referência para a aplicação Heroku	112
Figura 17 – Aplicação desenvolvida	113
Figura 18 – Arquitetura <i>multi-cloud</i> derivada da arquitetura de referência com estratégia de consumo de recursos a partir do <i>front-end</i>	115
Figura 19 – Arquitetura multi-cloud derivada da arquitetura de referência com estratégia de consumo de recursos a partir de um serviço de <i>back-end</i>	116
Figura 20 – Metamodelo da DSL desenvolvida	126
Figura 21 – Funcionamento de um template MDE em JET	133
Figura 22 – Elementos da abordagem	137
Figura 23 – Página Web contendo a lista de <i>gadgets</i>	140
Figura 24 – Página Web que apresenta os dados de um determinado <i>gadget</i>	141
Figura 25 – Estrutura da aplicação phonecat	143
Figura 26 – Mapeamento entre aplicação gerada e elementos da gramática	147
Figura 27 – Aplicação gerada	148
Figura 28 – Interface da aplicação gerada	149
Figura 29 – Gerenciamento de multimídia	153
Figura 30 – Retorno em formato JSON de uma requisição enviada a um serviço REST	153

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – DAO genérico implementado em JPA na plataforma GAE	103
Código-fonte 2 – DAO genérico implementado em JDO na plataforma GAE	104
Código-fonte 3 – Generic DAO Objectify GAE	105
Código-fonte 4 – Controlador implementado utilizando Spring	105
Código-fonte 5 – Controlador implementado utilizando Vraptr	106
Código-fonte 6 – Controlador da camada <i>front-end</i> implementado utilizando Angular .	107
Código-fonte 7 – Configuração dos parâmetros do banco de dados mLab	110
Código-fonte 8 – Método de inserção no banco de dados mLab	111
Código-fonte 9 – Informando a URL do <i>back-end</i>	116
Código-fonte 10 – Enviando uma requisição ao <i>back-end</i> considerando o formato JSON	117
Código-fonte 11 – Gramática em xText da DSL desenvolvida	124
Código-fonte 12 – Trecho da gramática para especificação de entidades	127
Código-fonte 13 – Entidade especificada utilizando a DSL desenvolvida	128
Código-fonte 14 – Trecho da gramática para especificação de <i>cloudapplication</i>	128
Código-fonte 15 – Aplicação <i>single-cloud</i> especificada utilizando a DSL desenvolvida .	129
Código-fonte 16 – Trecho da gramática para especificação de recursos	129
Código-fonte 17 – Um recurso para a entidade Connectivity	129
Código-fonte 18 – Trecho da gramática para especificação de aplicações <i>single-cloud</i> . .	130
Código-fonte 19 – Uma aplicação <i>single-cloud</i>	130
Código-fonte 20 – Trecho da gramática para aplicações <i>multi-cloud</i>	131
Código-fonte 21 – Aplicação <i>multi-cloud</i>	132
Código-fonte 22 – Trecho de código em xTend	134
Código-fonte 23 – Dados originais de um gadget da aplicação phonecat em formato JSON	140
Código-fonte 24 – Dados para a listagem de gadgets cadastrados	142
Código-fonte 25 – Modelagem de domínio da aplicação phonecat	144
Código-fonte 26 – Exemplo de Controlador Angular gerado	148
Código-fonte 27 – Modelagem de domínio da aplicação phonecat <i>single-PaaS</i>	155

LISTA DE ABREVIATURAS E SIGLAS

APIs	<i>Application Programming Interfaces</i>
ASP	<i>Application Server Provider</i>
BaaS	<i>Business-as-a-Service</i>
CRUD	<i>Create-Retrieve-Update-Delete</i>
CSCC	<i>Cloud Standards Customer Council</i>
DaaS	<i>[Development, Database, Desktop]-as-a-Service</i>
DSML	<i>Domain-Specific Modeling Language</i>
EMF	<i>Eclipse Modeling Framework</i>
EMP	<i>Eclipse Modeling Project</i>
ERP	<i>Enterprise Resource Planning</i>
FaaS	<i>Framework-as-a-Service</i>
GAE	<i>Google App Engine</i>
GMF	<i>Graphical Modeling Framework</i>
HaaS	<i>Hardware-as-a Service</i>
IaaS	<i>Infrastructure-as-a-Service</i>
IDC	<i>International Data Corporation</i>
IDE	<i>Integrated Development Environment</i>
IFML	<i>Interaction Flow Modeling Language</i>
JMI	<i>Java Metadata Interface</i>
MBE	<i>Model-Based Engineerig</i>
MDA	<i>Model-Driven Architecture</i>
MDD	<i>Model-Driven Development</i>
MDE	<i>Model-Driven Engineering</i>
MOF	<i>Meta Object Facility</i>
NSA	<i>Agência Nacional de Segurança</i>
OMG	<i>Object Management Group</i>
OVF	<i>Open Virtualization Format</i>
PaaS	<i>Platform-as-a-Service</i>
REST	<i>Representational State Transfer</i>
RSL	<i>Revisão Sistemática de Literatura</i>
SaaS	<i>Software-as-a-Service</i>

SOA	<i>Service-Oriented Architecture</i>
TI	Tecnologias de Informação
WSDL	<i>Web Services Description Language</i>
XaaS	<i>Everything-as-a-Service</i>

SUMÁRIO

1	INTRODUÇÃO	23
1.1	Motivação	24
1.2	Objetivos	27
1.3	Material e métodos	28
1.4	Organização do Trabalho	29
2	COMPUTAÇÃO EM NUVEM	31
2.1	Conceitos de Computação em Nuvem	31
2.1.1	<i>Origem e Definição</i>	31
2.1.2	<i>Vantagens da Computação em Nuvem</i>	40
2.1.3	<i>Desvantagens da Computação em Nuvem</i>	41
2.2	Utilização de serviços de nuvem sob a perspectiva de Engenharia de Software	42
2.2.1	<i>SaaS vs SOA vs ASP</i>	43
2.2.2	<i>Utilização do Modelo SaaS</i>	44
2.2.3	<i>Utilização do Modelo IaaS</i>	45
2.2.4	<i>Utilização do Modelo PaaS</i>	46
2.3	Portabilidade na Computação em Nuvem	47
2.4	Interoperabilidade na Computação em Nuvem	50
2.5	Abordagens para Interoperabilidade na Computação em Nuvem	61
2.6	Considerações finais	69
3	DESENVOLVIMENTO DIRIGIDO POR MODELOS	71
3.1	Conceitos de MDE	71
3.2	Vantagens e desvantagens do MDE	75
3.3	Elementos de MDE	78
3.4	Metamodelagem	80
3.5	Abordagens da indústria para MDE	81
3.6	Portabilidade e interoperabilidade no contexto MDE	82
3.7	Considerações finais	85
4	DESENVOLVIMENTO E MIGRAÇÃO DE IMPLEMENTAÇÕES DE REFERÊNCIA PARA O MODELO PAAS	87
4.1	Contextualização do Estudo de Caso	87

4.2	Plano do Estudo de Caso	88
4.3	Condução do estudo	91
4.3.1	<i>Requisitos tecnológicos e arquiteturais da aplicação</i>	91
4.3.2	<i>PaaS selecionadas</i>	94
4.3.3	<i>Estudo das plataformas selecionadas</i>	95
4.3.4	<i>Desenvolvimento das bibliotecas de código e aplicações uniplataforma</i>	97
4.3.5	<i>Desenvolver aplicações e módulos distribuídos</i>	113
4.4	Considerações finais	119
5	UMA ABORDAGEM DIRIGIDA POR MODELOS PARA DESENVOLVIMENTO DE SISTEMAS MULTI-PAAS	121
5.1	Considerações Iniciais	121
5.2	O modelo independente de plataforma	123
5.3	Transformações para plataformas específicas	133
5.4	Considerações finais	138
6	DESENVOLVIMENTO DE APLICAÇÕES <i>MULTI-PAAS</i> COM MDE	139
6.1	Considerações Iniciais	139
6.2	A aplicação <i>phonecat</i>	140
6.3	Modelagem de domínio	144
6.4	Aplicação gerada	146
6.4.1	<i>Aplicação CORS front-end</i>	147
6.4.2	<i>Back-ends da aplicação gerada</i>	152
6.4.3	<i>Conclusões sobre a aplicação gerada</i>	153
6.5	Desenvolvimento de aplicações portáteis utilizando a abordagem desenvolvida	155
6.6	Considerações finais	157
7	CONCLUSÕES	159
7.1	Principais contribuições	160
7.2	Lições aprendidas	163
7.3	Limitações e trabalhos futuros	165
7.4	Trabalhos publicados e submetidos	165
7.5	Considerações finais	167
	REFERÊNCIAS	169

INTRODUÇÃO

Recentemente, a *IEEE Computer Society*¹ publicou um relatório técnico intitulado IEEE CS2022 (ALKHATIB *et al.*, 2014), que destaca as tecnologias que possuem potencial para mudar o cenário da Ciência da Computação, incluindo a visão da indústria, e definem caminhos de pesquisa relevantes até pelo menos o ano de 2022. Seu objetivo é prever os impactos e apresentar uma visão do futuro para cada uma das tecnologias ressaltadas. Entre os tópicos de pesquisa destacados, está a Computação em Nuvem.

A Computação em Nuvem é um modelo computacional que possibilita o acesso ubíquo, conveniente e sob demanda a um conjunto compartilhado de recursos (como redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados com o mínimo de esforço de gerenciamento ou interação com o provedor (MELL; GRANCE, 2010). Tendo ganhado maior notoriedade na indústria, a Computação em Nuvem tem recebido cada vez mais atenção por parte da comunidade acadêmica. O número de publicações científicas apresentando relatórios, propostas de métodos, processos e ferramentas relacionadas ao modelo de nuvem é crescente. Além disso, pesquisadores estão investigando como tecnologias relacionadas ao modelo de nuvem podem ser utilizadas para promover avanços em muitas áreas da ciência da computação e outras ciências (SILVA; LUCRÉDIO, 2012; ALKHATIB *et al.*, 2014). O relatório IEEE CS2022 (ALKHATIB *et al.*, 2014) ressalta ainda que construir sistemas para nuvem é uma tarefa complexa e que são necessários esforços em direção ao estudo de abordagens que facilitem o desenvolvimento desses sistemas.

Um dos principais problemas relacionados ao desenvolvimento de sistemas para nuvem é denominado *lock-in* (ARMBRUST *et al.*, 2009). O *lock-in* é o aprisionamento do consumidor de serviços de nuvem a um provedor de serviços, devido a falta de portabilidade entre os provedores (ARMBRUST *et al.*, 2009). Como consequência, as soluções são proprietárias e raramente interoperáveis. O *lock-in* dificulta a portabilidade e interoperabilidade e, portanto, esses três problemas

¹ <<http://www.computer.org/>>

(*lock-in*, portabilidade e interoperabilidade) normalmente caminham juntos. O relatório IEEE CS2022 propõe que sejam empregados esforços na construção de uma infraestrutura de nuvem que possibilitará à indústria cooperar na criação de módulos interoperáveis.

Uma das causas do *lock-in* é a natureza dinâmica das tecnologias de informação, que possibilita o surgimento de diversas ofertas de tecnologias de nuvem. Essa dinamicidade e variedade de tecnologias e de provedores de serviços levam a cenários em que diferentes aplicações ou conjunto de serviços precisam operar de forma conjunta (inter-operar), em uma cooperação heterogênea de recursos e de provedores de nuvem. Como resultado, é necessário um esforço para possibilitar a interoperabilidade entre esses elementos. O relatório IEEE CS2022 (ALKHATIB *et al.*, 2014) e outros trabalhos anteriores (SILVA; LUCRÉDIO, 2012; ARMBRUST *et al.*, 2009) destacam que o desenvolvimento de sistemas interoperáveis, que obtenham vantagens das características de diferentes provedores de serviços nuvem, é um grande desafio e, portanto, precisa ser melhor investigado.

Para Armbrust *et al.* (2009), o desenvolvimento de padrões pode facilitar o desenvolvimento de sistemas para nuvem. No entanto, na prática, a adoção de um determinado padrão pode levar ao *lock-in* de padrões. Ou seja, pode levar ao aprisionamento do usuário ao padrão. Os padrões são divergentes, muitas vezes escassos em determinados contextos, e não seguem uma abordagem metodológica. Faz-se necessário investigar um caminho alternativo. O *lock-in*, a portabilidade e a interoperabilidade são problemas que precisam ser melhor investigados. Uma análise de como esses problemas impactam a Computação em Nuvem e como podem ser tratados é uma contribuição relevante tanto para comunidade científica quanto para desenvolvedores de sistemas.

1.1 Motivação

Tratando-se de mercado, o *International Data Corporation* (IDC)² é uma das maiores consultorias do mundo sobre mercado e soluções relacionadas a Tecnologias de Informação (TI). O IDC estima que somente no ano de 2011, serviços de nuvem ajudaram organizações de todos os tamanhos, ao redor do mundo, a gerar mais do que 400 bilhões de dólares em receita e 1,5 milhões de novos empregos. Além disso, a previsão é de que o número de novos empregos gerados em decorrência do uso desse modelo computacional cresça constantemente (IDC, 2012). Ainda sob a perspectiva de mercado, o *Northbridge Survey 2013*³, considerado um dos maiores *surveys* indicadores de tendências e inovação na área de Computação em Nuvem, apresenta a interoperabilidade como um quesito chave para ampliar ainda mais a adoção do modelo de nuvem. O *survey*, divulgado no final do ano de 2013, já mostrava que o *lock-in* e a falta de interoperabilidade continuavam a atrapalhar a adoção das tecnologias relacionadas a Computação em Nuvem naquela época, uma vez que os consumidores são forçados a assumir todos os riscos

² IDC: <<http://www.idc.com/>>

³ <<http://www.northbridge.com>>

de suas decisões na escolha de um determinado serviço ou provedor (NORTHBRIDGE..., 2014). O *survey* de 2016 destaca que o *lock-in* ainda é um dos maiores inibidores de adoção de tecnologias relacionadas ao modelo de nuvem.

Aplicações interoperáveis possibilitarão a comparação e escolha entre diferentes ofertas com diferentes características, enquanto os consumidores de serviços de nuvem poderão, caso necessário, mudar de provedor sem colocar suas aplicações e dados em risco. Será ainda possível desenvolver aplicações compostas por diferentes serviços de nuvem. Armbrust *et al.* (2009) discutem os principais conceitos do modelo de nuvem e apresentam desafios que precisam ser contornados para ajudar a disseminação desse modelo computacional. Silva e Lucrédio (2012) apresentam uma visão da Computação em Nuvem sob a perspectiva de Engenharia de Software e levantam lacunas, desafios e oportunidades para pesquisadores em desenvolvimento de software para nuvem por meio de uma metodologia confiável e rigorosa, uma revisão sistemática de literatura (KITCHENHAM *et al.*, 2009).

Armbrust *et al.* (2009) destacam que o *lock-in* é um problema relevante e ressaltam a importância da portabilidade e interoperabilidade na Computação em Nuvem. Petcu (2011) destaca que a portabilidade e interoperabilidade na nuvem são importantes não somente para a proteção dos investimentos dos consumidores de serviços, mas para a computação como um todo. Soluções interoperáveis, de acordo com Petcu (2011), podem trazer diversos benefícios, entre eles:

- redução de custos,
- aumento de reuso de software,
- o consumidor de serviços pode reagir a mudanças nas ofertas de serviços por parte dos provedores,
- possibilitar que restrições de localização de dados e legislação sejam respeitadas,
- eliminar a dependência de um provedor
- garantir *backups*,
- possibilitar a replicação de aplicações em múltiplos provedores para garantir maior disponibilidade,
- consumir melhores serviços de diferentes provedores ao mesmo tempo.

As iniciativas mais consolidadas em direção à solução do problema da interoperabilidade são resultados de consórcio de grandes instituições/empresas no sentido de estabelecer um padrão⁴ (como OASIS, SNIA e DMTF) para desenvolvimento de aplicações para nuvem .

⁴ Uma lista contendo os diversos padrões que estão sendo desenvolvidos pode ser encontrada no endereço:<<http://cloud-standards.org/>>

Apesar da padronização ser importante, para que um padrão seja efetivamente estabelecido é necessário que uma grande parte da comunidade de desenvolvimento adote tal tecnologia. Além disso, cada provedor de nuvem normalmente quer oferecer serviços diferenciados visando atrair mais clientes, seguir um padrão pode não ser o mais adequado às suas necessidades. Padrões ainda estão em estágio inicial de desenvolvimento e podem demorar anos até que sejam completamente desenvolvidos e aceitos pela comunidade (PETCU, 2014). Provedores podem não concordar facilmente com os procedimentos e implementações propostos pelos padrões.

Como há diferentes modelos de serviço de nuvem (SaaS, PaaS e IaaS), é necessário que diferentes padrões sejam estabelecidos. Como consequência, há uma quantidade muito grande de padrões sendo propostos e desenvolvidos paralelamente, de modo que até a escolha por um padrão ou outro pode se tornar uma decisão complexa. Além disso, essa escolha pode levar ao *lock-in* também. O provedor/consumidor de serviços de nuvem pode mais uma vez ficar aprisionado/restrito a um determinado padrão. Adicionalmente, a não adoção de um padrão pode se dar porque os provedores de sistemas para nuvem podem querer utilizar tecnologias específicas para criar soluções que estejam aliadas aos seus requisitos de negócio e não optar por padronização. É importante que sejam pesquisadas abordagens alternativas, não apenas para explorar um caminho diferente, mas para contribuir para enriquecer o modelo de nuvem e consequentemente facilitar sua adoção.

Diversas propostas e abordagens relacionadas ao problema têm sido pesquisadas. No entanto, nenhuma das soluções encontradas cumpre com todos os requisitos de interoperabilidade no contexto da Computação em Nuvem (PETCU, 2014). As soluções encontradas na literatura ainda não podem ser consideradas como estabelecidas. Uma solução que cumpra todos os requisitos de interoperabilidade e que libere totalmente o consumidor de serviços de nuvem do *lock-in* não foi encontrada. Portanto, faz-se necessário que uma abordagem seja desenvolvida visando solucionar e explorar o problema da interoperabilidade. Em direção a esse objetivo, a tecnologia de *Model-Driven Engineering* (MDE)⁵ foi considerada neste trabalho.

Abordagens de MDE propõem que a modelagem e transformação de modelos em artefatos de implementação oferecem vantagens para desenvolvimento e manutenção de sistemas de software, quando comparados com codificação pura (FRANCE; RUMPE, 2007) e por isso foi considerada ao invés de abordagens tradicionais. Com MDE, os desenvolvedores de sistemas podem trabalhar em um nível mais alto de abstração, em modelos conceituais que melhor capturam a essência e a lógica da aplicação; livrando-se de tarefas repetitivas de codificação, que ficam a cargo de transformações automatizadas (LUCRÉDIO, 2009), e ficando assim protegidos das

⁵ Neste trabalho é adotada a sigla MDE para se referir ao “desenvolvimento dirigido por modelos”. Existem diversas taxonomias relacionadas a essa abordagem, mas em essência todas classificações referem-se ao mesmo conceito, cuja ideia principal é reconhecer a importância dos modelos no processos de software, utilizá-los não apenas como um “guia” para tarefas de desenvolvimento, mas como parte integrante do software assim como o código-fonte (BRAMBILLA; CABOT; WIMMER, 2012).

complexidades requeridas para implementação nos diferentes provedores de serviços de nuvem.

Ferry *et al.* (2013) provêm um catálogo de soluções e defendem a necessidade de técnicas relacionadas ao desenvolvimento dirigido por modelos e métodos que facilitam a especificação, provisionamento, implantação, monitoramento de aplicações interoperáveis. Essas duas tecnologias (MDE e Computação em Nuvem), aliadas, podem oferecer muitos benefícios aos desenvolvedores de sistemas. Em particular, MDE pode ajudar no problema da falta de interoperabilidade citado anteriormente, por possibilitar concentrar o esforço de desenvolvimento em um nível conceitual independente de plataforma (nesse caso, de provedor de serviços de nuvem) (SILVA; FORTES; LUCRÉDIO, 2013; BRUNELIÈRE; CABOT; JOUAULT, 2010). Assim, torna-se possível, por exemplo, portar um aplicativo de uma plataforma para outra, aplicando-se diferentes transformações a um mesmo conjunto de modelos conceituais (BRAMBILLA; CABOT; WIMMER, 2012).

Resolver os desafios de interoperabilidade no contexto da Computação em Nuvem numa única solução é muito complexo. Uma estratégia é resolvê-los para as diferentes camadas (IaaS-SaaS, IaaS-PaaS, PaaS-PaaS, etc.). Portanto, em primeiro momento, foi dado foco no modelo de serviço *Platform-as-a-Service* (PaaS). A oferta PaaS foi escolhida por oferecer um ambiente completo de desenvolvimento que facilita o trabalho do desenvolvedor. No PaaS os desenvolvedores não precisam se preocupar com o gerenciamento da infraestrutura, questões de escalabilidade e elasticidade são gerenciadas pela plataforma. O conceito de *multi-PaaS* será melhor explorado mais adiante, mas em essência, uma aplicação *multi-PaaS* consiste em uma aplicação de múltiplas nuvens (interoperável) que considera somente PaaS em sua composição. Inicialmente, os domínios de aplicações *Create-Retrieve-Update-Delete* (CRUD) sobre banco de dados não relacionais e criação de serviços foram considerados, mas outros domínios podem ser incluídos posteriormente.

1.2 Objetivos

Visando apresentar uma contribuição ao estado da arte relacionado ao desenvolvimento de sistemas interoperáveis no contexto da Computação em Nuvem, este trabalho teve objetivo geral desenvolver **uma abordagem dirigida por modelos para desenvolvimento de aplicações *multi-PaaS***. Em direção a esse objetivo, foram definidos quatro objetivos específicos conforme a seguir:

- Desenvolvimento de implementação de referência: implementações de referência são utilizadas no ciclo de vida de um processo MDE. Elas são abstraídas e servem de base para desenvolvimento das transformações que irão gerar o código final em conformidade com a implementação definida;
- Desenvolvimento de uma linguagem específica de domínio: a linguagem serve para

modelar a aplicação em conformidade com o domínio especificado. Os modelos de domínio especificados por meio da linguagem desenvolvida servem como entrada para um conjunto de transformações;

- Desenvolvimento de um conjunto de transformações: o conjunto de transformações são metaprogramas que geram o código alvo em conformidade com o modelo de domínio da aplicação e as implementações de referência;
- Avaliação da abordagem: estudos de caso foram desenvolvidos visando avaliar o funcionamento da abordagem.

Em essência, por meio de um processo MDE, foram geradas (desenvolvidas) aplicações que são executadas em diferentes provedores de serviços de nuvem ao mesmo tempo. Uma vez que a portabilidade entre plataformas, no contexto de MDE, já foi alcançada (SILVA; FORTES; LUCRÉDIO, 2013), neste trabalho foi empenhado um esforço em direção à interoperabilidade.

1.3 Material e métodos

Para levantamento do estado da arte relacionado à interoperabilidade no contexto da Computação em Nuvem (apresentados no Capítulo 2) e dos trabalhos relacionados a este, foi realizada uma Revisão Sistemática de Literatura (RSL) de acordo com o processo proposto por Kitchenham *et al.* (2007). A revisão conduzida teve por objetivo, por meio da sumarização de evidências, identificar como a interoperabilidade tem sido abordada no contexto da Computação em Nuvem e apresentar uma visão atualizada do tópico de pesquisa para possibilitar futuras atividades de pesquisa relacionadas ao tema (NOGUEIRA *et al.*, 2016b). Após a realização da RSL, de estudo aprofundado sobre as abordagens correlatas, da delimitação do escopo deste projeto e do tópico de pesquisa, de estudo das tecnologias relacionadas ao MDE, foram realizados estudos de caso e implementação de protótipos de prova de conceitos.

Para desenvolvimento das implementações de referência foi conduzido um Estudo de Caso em colaboração com a indústria. Foi desenvolvida uma aplicação *multi-PaaS*⁶ levando em consideração uma arquitetura de referência definida pela equipe de desenvolvimento. Essa implementação serviu como base para desenvolvimento de uma DSL e um conjunto de transformações que compõem a abordagem dirigida por modelos para desenvolvimento de aplicações *multi-PaaS*. Por fim, foi realizado um outro Estudo de Caso que teve por objetivo explorar o desenvolvimento de uma aplicação *multi-PaaS* utilizando os elementos desenvolvidos neste trabalho.

⁶ O conceito de *multi-PaaS* será melhor apresentado no Capítulo 2. Em síntese consiste em um *multi-cloud* formado somente pela combinação de PaaS. Um *multi-cloud* por sua vez consiste na combinação de serviços de nuvem onde não há consentimento por parte dos provedores a serem combinados.

1.4 Organização do Trabalho

O [Capítulo 2](#) apresenta os principais conceitos de Computação em Nuvem e os principais trabalhos relacionados a esta tese. O [Capítulo 3](#) apresenta os conceitos de desenvolvimento dirigido por modelos. O [Capítulo 4](#) apresenta o Estudo de Caso sobre o desenvolvimento de aplicações de referência *multi-PaaS*. O [Capítulo 5](#) apresenta o desenvolvimento dos elementos que compõem a abordagem MDE. O [Capítulo 6](#) apresenta o desenvolvimento de aplicações *multi-PaaS* utilizando a abordagem desenvolvida neste trabalho. Por fim, o [Capítulo 7](#) apresenta as conclusões e considerações finais relacionadas a esta tese.

COMPUTAÇÃO EM NUVEM

Este capítulo descreve os principais conceitos relacionados à Computação em Nuvem que fundamentam esta tese. Inicialmente, na [Seção 2.1](#) são apresentados os conceitos gerais de Computação em Nuvem e algumas vantagens e desvantagens da utilização do modelo de nuvem; logo após, na [Seção 2.2](#) é apresentado como os modelos de serviços de nuvem podem ser utilizados sob uma perspectiva de desenvolvimento de software; a seguir são apresentados os conceitos de portabilidade ([Seção 2.3](#)) e interoperabilidade ([Seção 2.4](#)) no contexto da Computação em Nuvem respectivamente; depois são apresentados os conceitos relacionados ao uso de múltiplas nuvens; por fim, na [Seção 2.5](#) são apresentadas abordagens da indústria para interoperabilidade na Computação em Nuvem.

2.1 Conceitos de Computação em Nuvem

A Computação em Nuvem tem se configurado como uma importante evolução tecnológica. Ainda em estágio de amadurecimento, o tema se tornou rapidamente um importante objeto de pesquisa em Engenharia de Software ([SILVA; LUCRÉDIO, 2012](#)). Esta seção apresenta os principais conceitos relacionados a essa tecnologia, bem como vantagens e desvantagens de sua utilização.

2.1.1 Origem e Definição

Conduzidos pelo conceito de *utility computing* (pagamento de recursos computacionais de acordo com o volume de utilização), pesquisadores da universidade de Stanford¹ desenvolveram, na década de sessenta, um projeto denominado *Time-Sharing*². O projeto *Time-Sharing* introduziu

¹ <http://www.stanford.edu/>

² <https://web.stanford.edu/~learnest/nets/timesharing.htm>

o conceito de multitarefa³ em hardware da época e permitia o compartilhamento de recursos computacionais de forma paralela, diferentemente das abordagens existentes até aquele momento (DURAO *et al.*, 2014). Embora o *Time-Sharing* tenha promovido o modelo de *utility computing* durante os anos 60, o pagamento de recursos computacionais de acordo com o uso já era explorado no processamento de cartões perfurados. Mas, o projeto, além do *utility computing*, promoveu também o acesso remoto a provedores de serviço. Isso possibilitou que uma grande parte dos usuários daquele tempo tivessem acesso a recursos até então inacessíveis. Em relação às empresas, foi feita a terceirização do setor de TI (DURAO *et al.*, 2014).

A *Utility computing* surgiu da ideia de tratar computação como “*utility*”, termo em inglês que se refere especificamente a serviços entregues sob demanda, tais como água encanada, gás encanado, energia elétrica, entre outros. O modelo de pagamento dos serviços citados se dá, normalmente, de acordo com o volume de utilização e seus consumidores não têm a necessidade de saber a origem dos recursos. Neste contexto, é preciso fazer uma distinção entre **serviço** e **recurso**. No contexto de *utility*, um **serviço** de água refere-se ao trabalho de provisionamento do **recurso** água. Na prática, a ideia por trás da Computação em Nuvem é a mesma, permitir que usuários adquiram recursos computacionais sob demanda e o pagamento desse serviço seja feito de acordo com o volume de utilização (*utility computing*) (ARMBRUST *et al.*, 2009). Neste contexto, um serviço de nuvem refere-se ao provisionamento de um recurso computacional, podendo ser software, hardware, plataforma, banco de dados, entre outros.

No que se refere às tecnologias utilizadas para implementação um serviço de nuvem é possível dizer que o modelo de nuvem não surgiu como um novo modelo de tecnologia, mas como a integração de tecnologias do passado (CHEN; LI; CHEN, 2011) que resultou em uma nova forma de usar e disponibilizar computação como um serviço, por meio da Internet. A Figura 1 apresenta uma linha do tempo com os principais acontecimentos relacionados à Computação em Nuvem.

Como é possível perceber na figura apresentada, a ideia de *utility computing* foi bastante utilizada desde o surgimento do conceito até a popularização dos dispositivos de computação pessoal (como Computadores pessoais – PCs, *smartphones*, *tablets*, *pages*, entre outros). Portanto, ao analisar a maneira como recursos computacionais vêm sendo utilizados desde a década de 60 até os dias atuais, é possível verificar que inicialmente havia a centralização de recursos, principalmente com os centros de processamentos de dados (CPDs). Com o advento dos computadores pessoais e demais dispositivos portáteis, o poder computacional foi distribuído. Com a ideia de Computação em Nuvem, a centralização de recursos ressurgiu como um “novo” paradigma. Todavia, como a ideia central por trás do conceito é relativamente antiga, alguns autores afirmam que a Computação em Nuvem seria mais um conceito de marketing para comercialização de serviços do que propriamente um novo modelo de computação. Wallis (2008) apresenta uma

³ Em computação, Multitarefa é a característica dos sistemas operacionais que permite repartir a utilização do processador entre várias tarefas aparentemente simultaneamente (TANENBAUM, 2007).

Figura 1 – Linha do tempo da história da Computação em Nuvem – principais acontecimentos relacionados à evolução da Computação em Nuvem

Fonte: adaptado de Durao *et al.* (2014)

perspectiva histórica das tecnologias relacionadas ao conceito de nuvem que foram sintetizadas na figura apresentada por Durao *et al.* (2014).

Ao analisar a evolução dos dispositivos de computação pessoal, é possível perceber que a distribuição do poder computacional que ocorreu com a popularização dos dispositivos como PCs e *smartphones* trouxe muitos benefícios aos usuários. Dentre eles, a possibilidade de um usuário ter em suas mãos um dispositivo para realizar suas atividades. Todavia, a análise isolada da centralização versus distribuição não necessariamente possibilita vislumbrar as vantagens reais do modelo de nuvem. Portanto, idealmente, uma estratégia de adoção de um desses modelos por uma empresa ou pessoa, não deve apenas considerar o uso isolado de centralização ou distribuição, mas a combinação da centralização com a distribuição para prover um serviço ou uma melhor experiência de uso que não seria possível alcançar utilizando apenas um dos modelos isoladamente. Isso significa que recursos computacionais centralizados podem ser combinados com dispositivos portáteis de baixo poder computacional para promover o avanço da onipresença da computação - computação ubíqua (WEISER, 1993). Essa é a ideia central que sintetiza todos

os benefícios do uso do modelo de nuvem.

Diversas definições da Computação em Nuvem podem ser encontradas na literatura. [Vaquero et al. \(2008\)](#) fizeram um rigoroso levantamento de várias propostas no sentido de alcançar uma definição mais completa e apresentam definições de nuvem sob diversas perspectivas. Neste trabalho, no entanto, será utilizada a definição proposta pelo Instituto Nacional de Padrões e Tecnologia dos Estados Unidos (NIST)⁴. Essa definição é bastante difundida na literatura, sendo citada em uma grande quantidade de estudos. Além disso, provê uma descrição abrangente e clara dos conceitos e tecnologias relacionadas ([HOGAN et al., 2011](#)):

“A computação em nuvem é um modelo para possibilitar acesso **prático, ubíquo e sob demanda** a um conjunto de **recursos computacionais compartilhados** (redes, servidores, armazenamento, aplicações e serviços) que podem ser **rapidamente provisionados** com o **mínimo de esforço de gerenciamento possível** ou interação com o provedor de serviço.”

Algumas palavras-chave encontradas na definição do NIST foram destacadas e são examinadas para melhor entendimento da definição, conforme segue:

“**prático, ubíquo e sob demanda**”: as palavras **ubíquo e prático** referem-se ao fato de que tecnologias que utilizam a nuvem devem, em tese, ser acessadas de qualquer dispositivo por meio da Internet de maneira simples, sem esforço adicional e transparente. A expressão “sob” demanda refere-se ao conceito de *utility computing*. Ou seja, os recursos computacionais são tratados como *utility*, o consumidor de serviços utiliza os recursos, paga pelo que utilizou e não precisa saber a origem dos mesmos. Além disso, não é necessário muito esforço por parte dos consumidores de serviços de nuvem para utilizar os recursos.

“**recursos computacionais compartilhados**”: recursos computacionais referem-se a tempo de processamento, armazenamento e demais recursos comuns da nuvem que podem ser utilizados. A definição cita o termo recursos de maneira genérica para destacar que muitos tipos de recursos computacionais podem ser utilizados. Os recursos podem ser compartilhados entre os consumidores. Isso significa que o provedor pode movê-los de um consumidor para outro para reduzir ociosidade e o investimento médio no provisionamento antecipado de recursos. Todavia, se um consumidor não quiser compartilhar recursos, é possível também que eles não sejam compartilhados; depende do tipo de contrato que está sendo feito com o provedor.

“**rapidamente provisionados**”: refere-se à característica de elasticidade. Ou seja, a habilidade de um provedor de nuvem entregar recursos computacionais rapidamente de acordo com a demanda do consumidor. Se um consumidor precisar de mais recursos, o provedor pode

⁴ <<http://www.nist.gov/>>

aumentar o conjunto de recursos compartilhados para ele. Se o consumidor não mais precisar dos recursos, o provedor pode diminuir o conjunto de recursos compartilhados. Isso pode ser feito em tempo real. A elasticidade somente é viável com o mínimo de esforço de gerenciamento possível por parte do consumidor.

“mínimo de esforço de gerenciamento possível”: significa que as tecnologias de nuvem devem tão automáticas quanto possível, de modo que a elasticidade pode ser de fato uma realidade. Em outras palavras, mínimo de gerenciamento pode tornar o provisionamento de recursos tão fácil que deve ser possível adquirir recursos somente com alguns cliques do *mouse*.

Há duas características de tecnologias de nuvem implícitas nessa definição que surgem da capacidade dos recursos serem rapidamente provisionados como serviço. A primeira é a **escalabilidade**, que refere-se a capacidade de fácil aquisição recursos computacionais de acordo com a necessidade de crescimento do consumidor. A segunda é a **elasticidade**, que refere-se a capacidade não só de aumentar o uso dos recursos, mas aumentar e diminuir rapidamente de acordo com a necessidade e com o mínimo esforço de gerenciamento possível por parte do consumidor de serviços. Esse mínimo esforço possível significa que tarefas de gerenciamento de centros de dados, atualizações de software e demais recursos computacionais são transferidas para o provedor de serviços de nuvem. Ou seja, o consumidor pode facilmente adquirir novos recursos computacionais de acordo com sua demanda. Neste contexto, surgem os conceitos de escalabilidade vertical e escalabilidade horizontal. Enquanto a escalabilidade vertical significa adicionar mais memória, disco ou poder de processamento a um computador virtual provido como serviço, a escalabilidade horizontal significa adição de mais servidores à nuvem computacional para suportar o aumento da demanda por serviço. Por outro lado, o termo *scale down* é utilizado para se referir a remoção de recursos (HOGAN *et al.*, 2011).

Em relação à oferta de serviços de nuvem, várias taxonomias⁵ da Computação em Nuvem foram sugeridas, com o objetivo de caracterizar quais recursos serão oferecidos sob o modelo pagamento por uso (configurando, portanto, um serviço). Muitas delas foram criadas sob uma perspectiva de negócios e certos autores afirmam que os elementos que compõem os serviços estão mais voltados para tecnologias que viabilizam a negociação de demanda computacional (portanto um modelo de negócio) do que tecnologias de infraestrutura como nos modelos de *grid* e *cluster* (RIMAL; CHOI; LUMB, 2009). As taxonomias propostas apresentam termos como *Hardware-as-a Service* (HaaS), *Software-as-a-Service* (SaaS), *Platform-as-a-Service* (PaaS), *[Development, Database, Desktop]-as-a-Service* (DaaS), *Infrastructure-as-a-Service* (IaaS), *Business-as-a-Service* (BaaS), *Framework-as-a-Service* (FaaS), ou mesmo o termo *Everything-as-a-Service* (XaaS), que engloba o conceito de serviço de forma genérica.

⁵ Taxonomia é a ciência da classificação. No contexto em que foi empregada a palavra se refere à classificação de serviços de nuvem de acordo com cada modelo de serviço.

Como o modelo de IaaS é um dos mais relevantes e adotados da Computação em Nuvem, é importante deixar claro a diferença entre IaaS e HaaS pois é possível que os mesmos sejam confundidos. Assim, IaaS refere-se a um conceito mais abrangente que HaaS, enquanto HaaS refere-se especificamente a Hardware comercializado como um serviço, IaaS refere-se à infraestrutura computacional que compreende Hardware, virtualização de servidores, software e diversos outros serviços. A [Figura 2](#) apresenta HaaS e IaaS em níveis diferentes. Na prática HaaS seria a aquisição sob demanda de hardware de um determinado provedor para que seja montada uma infraestrutura. As definições desses termos e serviços da Computação em Nuvem, de uma maneira geral, são bastante amplas e muitas vezes mal compreendidas ([ARMBRUST et al., 2009](#)). Observando-se a literatura, no entanto, é possível enquadrar as principais soluções nas três seguintes categorias:

- **Software como Serviço – SaaS:** refere-se a aplicações disponibilizadas como um serviço a consumidores através da Internet ou da Intranet ([ARMBRUST et al., 2009](#); [MELL; GRANCE, 2010](#)). As aplicações podem ser acessadas de vários dispositivos dos clientes, muitas vezes, por meio de navegadores *Web*. Os consumidores não gerenciam ou controlam a infraestrutura de nuvem, incluindo redes, servidores, sistemas operacionais, armazenamento, etc. Os consumidores, normalmente, só têm acesso a configurações específicas das aplicações que estão sendo utilizadas. A ideia principal do conceito de SaaS é que um sistema seja fornecido a um cliente e que o mesmo seja pago de acordo com o volume de utilização ([HOGAN et al., 2011](#)). Muitas vezes o termo serviço no contexto de SaaS pode ser confundido com *Service-Oriented Architecture* (SOA). A [Seção 2.2](#) apresenta uma discussão mais aprofundada sobre esses termos. Como exemplos de SaaS podem ser citados: Flickr⁶, Picasa⁷, Salesforce⁸, entre outros.
- **Plataforma como Serviço – PaaS:** é a ideia de prover uma plataforma de desenvolvimento de sistemas como um serviço, disponibilizando-a sobre uma infraestrutura de nuvem ([RIMAL; CHOI; LUMB, 2009](#); [MELL; GRANCE, 2010](#)). Inclui hospedagem, testes e entrega de sistemas. Ex.: Google App Engine⁹, Microsoft Azure¹⁰, entre outras.
- **Infraestrutura como Serviço – IaaS:** consiste na entrega de infraestrutura computacional como serviço ([RIMAL; CHOI; LUMB, 2009](#); [MELL; GRANCE, 2010](#)). Na prática, essa oferta trata-se basicamente do aluguel de um computador virtual na infraestrutura (processamento, armazenamento, rede, etc.) do provedor. O consumidor de recursos pode

⁶ Flickr: <<http://www.flickr.com/>>

⁷ Picasa: <<http://picasa.google.com/>>

⁸ <<http://www.salesforce.com/>>

⁹ Google App Engine: <<http://code.google.com/appengine>>

¹⁰ Windows Azure: <<http://www.windowsazure.com/>>

instalar sistemas operacionais e outras aplicações (HOGAN *et al.*, 2011). Como exemplos de IaaS podem ser citados: Amazon¹¹, GoGrid¹², IBM SmartCloud¹³, entre outras.

São muitos os requisitos tecnológicos necessários para que esses modelos funcionem de forma adequada. Entre eles estão tecnologias de virtualização, padrões e interfaces, que permitem o acesso compartilhado a servidores virtuais e *Application Programming Interfaces* (APIs). A escolha de um serviço de nuvem depende muito do tipo de aplicação que se pretende utilizar. Algumas ofertas têm como público alvo desenvolvedores; outras, usuários finais. Garg e Stiller (2015) apresentam fatores que afetam a adoção do modelo de nuvem e seus relacionamentos. A Seção 2.2 apresenta mais detalhes sobre a utilização desses serviços.

Youseff, Butrico e Silva (2008) propuseram uma ontologia¹⁴ (Figura 2) que ilustra a interdependência e composição entre os diferentes tipos de serviços oferecidos pelos provedores. As camadas mais inferiores são responsáveis por virtualizar servidores que serão utilizados pela infraestrutura, **são o que efetivamente podemos chamar de nuvem computacional** (ARM-BRUST *et al.*, 2009). O termo “serviço”, no contexto da definição dos conceitos, faz alusão ao modelo de entrega *Utility Computing*. Portanto, SaaS seria um serviço de software, IaaS um serviço de infraestrutura e PaaS um serviço de plataforma de desenvolvimento e hospedagem de aplicações.

Figura 2 – Ontologia da Computação em Nuvem

Fonte: adaptado de Youseff, Butrico e Silva (2008)

¹¹ Amazon: <<http://aws.amazon.com/pt/ec2/>>

¹² GoGrid: <<http://www.gogrid.com/>>

¹³ <<http://www.ibm.com/cloud-computing/>>

¹⁴ Ontologia: em Ciência da Computação, uma ontologia é um modelo de dados que representa um conjunto de conceitos dentro de um domínio e os relacionamentos entre estes (GUARINO, 1998)

Em relação aos *modelos de implantação* da Computação em Nuvem, quando os serviços são disponibilizados ao público em geral, normalmente por uma empresa especializada em vender serviços de nuvem, denomina-se **nuvem pública** (ARMBRUST *et al.*, 2009; MELL; GRANCE, 2010). Alguns exemplos atuais são as nuvens da Microsoft¹⁵, Google¹⁶ e Amazon¹⁷. O termo **nuvem privada** é utilizado para designar uma infraestrutura de nuvem operando exclusivamente para uma organização. Tal infraestrutura pode ser gerida pela organização ou por um terceiro (MELL; GRANCE, 2010). Na prática, é uma infraestrutura que funciona de forma semelhante a um ambiente de computação em nuvem pública, mas é dedicada a uma única empresa ou organização (RIMAL; CHOI; LUMB, 2009). Nesse modelo, capacidades de TI elásticas e escaláveis são oferecidas como serviços para usuários internos às organizações, isto é, destinadas a atender somente suas demandas e de suas possíveis filiais e não comercializadas a vários clientes como no modelo de nuvem pública. **Nuvem comunitária** refere-se a uma infraestrutura compartilhada por várias organizações que possuem interesses em comum (missão, requisitos de segurança, políticas internas, etc.) (LIU *et al.*, 2011; HOGAN *et al.*, 2011). O modelo de **nuvem híbrida** prevê uma utilização mista, porém integrada, dos demais paradigmas. Ou seja, é a combinação de serviços de nuvens públicas, comunitárias ou privadas (ARMBRUST *et al.*, 2009; MELL; GRANCE, 2010).

O NIST propôs uma arquitetura conceitual, genérica e em alto nível da Computação em Nuvem (LIU *et al.*, 2011). A taxonomia proposta não está vinculada a tecnologias, serviços, implementações e provedores específicos. A Figura 3 apresenta o relacionamento entre os diversos atores e demais elementos da proposta. A arquitetura contém um conjunto de visões e descrições que são a base para a discussão das características, usos e padrões para Computação em Nuvem. A proposta define cinco principais atores: *Consumidor de Serviços de Nuvem*, *Provedor de Serviços de Nuvem*, *Auditor de Serviços de Nuvem*, *Cloud Broker* e *Cloud Carrier* (HOGAN *et al.*, 2011). A seguir está a descrição de cada um desses atores.

- **Consumidor de serviços de nuvem:** é o indivíduo ou organização que mantém um relacionamento de negócios com um provedor e utiliza seus serviços. Um consumidor pode utilizar um catálogo de serviços disponibilizado pelo provedor de serviços, requisitar serviços, gerenciar contratos e utilizar os serviços contratados.
- **Provedor de Serviços de Nuvem:** é o indivíduo ou organização responsável por disponibilizar um serviço aos consumidores de serviços de nuvem. É um fornecedor de recursos, seguindo principalmente os modelos SaaS, PaaS e IaaS. Cada provedor dita a forma como os recursos são disponibilizados, ou seja, quais tecnologias, padrões e aplicações devem ser utilizadas pelos consumidores. As atividades do provedor de serviços de nuvem podem ser discutidas sob as perspectivas de Gerenciamento de Serviços de nuvem, Camadas de

¹⁵ Windows Azure: <<http://www.windowsazure.com/>>

¹⁶ Google App Engine: <<http://code.google.com/appengine>>

¹⁷ Amazon: <<http://aws.amazon.com/pt/ec2/>>

Figura 3 – Arquitetura de Referência da Computação em Nuvem proposta pelo NIST

Fonte: adaptado de [Liu et al. \(2011\)](#)

Serviço, Abstração e controle dos recursos, Camada de Recursos Físicos, Segurança e privacidade.

- **Auditor da Nuvem:** a pessoa que conduz avaliação independente de serviços, sistemas operacionais, performance e segurança e implementação da nuvem. Para auditoria de segurança é necessário verificar a conformidade do provedor com os requisitos de segurança do consumidor de serviços ([HOGAN et al., 2011](#)).
- **Cloud Broker:** uma entidade que gerencia o uso, performance e entrega de serviços de nuvem. Além disso, negocia relacionamentos entre provedores e consumidores. A integração de serviços de nuvem pode ser bastante complexa para ser gerenciada por consumidores de serviços e eles, por sua vez, podem requerer serviços de um *cloud broker*. Para um *broker* estabelecer a comunicação entre os tipos de serviços de nuvem é necessário definir a maneira como serão feitas essas interações. As funções do *broker* podem ser classificadas em três categorias ([HOGAN et al., 2011](#)):
 - **Intermediação do serviço:** um *broker* pode aprimorar um determinado serviço melhorando algum recurso do provedor e fornecer melhores serviços aos consumidores de serviços de nuvem.
 - **Agregação:** o *cloud broker* pode combinar e integrar múltiplos serviços em um ou mais novos serviços (os conceitos de orquestração e coreografia podem se relacionar

com essa função). Pode também prover a integração de dados e garantir a segurança da comunicação entre os consumidores de serviços e múltiplos provedores.

- **Arbitragem:** é similar ao serviço de agregação (portanto, os conceitos de orquestração e coreografia também podem se relacionar com essa função), exceto que os serviços sendo agregados não são fixos. Arbitragem significa que o *broker* pode, por exemplo, fazer um *ranking* de serviços com base em determinadas características dos serviços e selecionar os melhores classificados.
- **Cloud Carrier:** seria um intermediário que provê a conectividade e transporte de serviços de nuvem oferecidos pelos provedores e adquiridos pelos consumidores. Na prática, consiste de tecnologias e provedores de serviços de comunicação.

Cada ator é uma entidade (pessoa ou organização) que participa em transações ou processos e/ou realiza tarefas no modelo de nuvem. Os modelos de serviço apresentados na [Figura 2](#) estão na camada de serviços no conceito de Provedor de Nuvem.

2.1.2 Vantagens da Computação em Nuvem

Geralmente novas tecnologias trazem consigo grandes discussões. Com a Computação em Nuvem não seria diferente. Quando se fala nesse assunto, é possível perceber os benefícios evidentes que este novo modelo computacional oferece. Além das vantagens agregadas de sistemas Web, as aplicações da nuvem apresentam as seguintes vantagens:

- **Escalabilidade:** as empresas podem contratar recursos computacionais sob demanda. É possível crescer conforme a necessidade e a disponibilidade do provedor de ofertar mais recursos ([RIMAL; CHOI; LUMB, 2009](#)).
- **Independência de plataforma:** o ideal é que as pessoas possam usar os serviços de software, independentemente de sistema operacional e da plataforma de hardware. A Computação em Nuvem poderá suportar o uso de clientes magros¹⁸ e incorporar dispositivos móveis, além de hardware específico que porventura venha a surgir. Para isso, basta que eles estejam conectados à Internet ([VELTE; VELTE; ELSENPETER, 2009](#)).
- **Maior facilidade de gerenciamento:** não é necessário manter equipes especializadas para manter a infraestrutura de suporte aos aplicativos da empresa. Todas as tarefas de manutenção e backup são realizadas pelo provedor de serviços de nuvem ([ARMBRUST et al., 2009](#)), ou seja, na prática, a infraestrutura computacional é terceirizada e as empresas podem focar em suas atividades de negócio;

¹⁸ Cliente magro: é um computador cliente em uma rede cliente-servidor de duas camadas. O qual tem pouco ou nenhum aplicativo instalado, de modo que depende de um servidor central para o processamento de atividades ([TANENBAUM; Van Steen, 2002](#)).

- **Redução de Custos:** com o uso de *Utility Computing*, não é necessário adquirir antecipadamente sistemas ou infraestrutura computacional, o pagamento é feito de acordo com o volume de utilização. Além disso, não é necessário manter equipes de suporte no ambiente da empresa. Dessa maneira, a empresa economiza com energia, aquisição de infraestrutura e contratação de pessoal. No entanto, é importante ressaltar que talvez, a longo prazo, o serviço de nuvem fique caro e gere um custo demasiado para empresa. Por isso é importante avaliar com cuidado a viabilidade a longo prazo do uso da Computação em Nuvem antes da sua completa adoção (SILVA; LUCRÉDIO, 2012).

2.1.3 Desvantagens da Computação em Nuvem

Há muitos quesitos relacionados à Computação em Nuvem que ainda não estão bem estabelecidos ou ainda encontram-se em fase de pesquisa (SILVA; LUCRÉDIO, 2012; ARMBRUST *et al.*, 2010) e conseqüentemente acabam se tornando desvantagens, tais como:

Segurança: é possível que o sistema seja atacado por invasores e informações confidenciais caiam em mãos erradas. Todavia, é um engano pensar que provedores de nuvem tendem a ser menos seguros que os demais por centralizar informações. Muitos mantêm equipes de segurança especializadas, o que nem sempre ocorre nos provedores comuns ou em soluções próprias das pequenas e médias empresas. Além disso, o modelo de contratação é flexível e permite negociar condições adicionais com usuários que não se convencerem das políticas adotadas pelos provedores (KANDUKURI; PATURI; RAKSHIT, 2009). Todavia, ao analisar a segurança na nuvem, é possível perceber que a questão principal não é como o provedores vão proteger o sistema de invasores (visto que as soluções adotadas pelos provedores podem ser tão ou mais seguras que um provedor tradicional), mas de como eles vão proteger os usuários dos serviços de si mesmo. Há muitas pesquisas e considerações sendo feitas em torno desse tema. A *Cloud Security Alliance*¹⁹ reúne esforços para promover melhores práticas na garantia de segurança e oferecer treinamentos para ajudar a proteger todas as formas de computação. No futuro, é possível que as aplicações de Computação em Nuvem sejam mais seguras.

Desempenho: em alguns casos, é possível que determinadas aplicações fiquem lentas. Gargalos de transferência e tempo de resposta das requisições poderiam diminuir o desempenho, uma vez que o meio de comunicação é uma rede de computadores que está mais propícia a erros e lentidão do que um barramento físico de uma máquina local.

Tecnologia Proprietária: grande parte das plataformas de Computação em Nuvem são proprietárias e, portanto, fechadas. Há a necessidade de mais pesquisas em tecnologias *open source* (SILVA; LUCRÉDIO, 2012), isso possibilitaria a evolução das tecnologias com a contribuição de uma grande comunidade de desenvolvimento.

¹⁹ *Cloud Security Alliance*: <<https://cloudsecurityalliance.org/>>

Posse dos dados: aplicativos e informações confidenciais do cliente estarão em posse dos provedores de nuvem. Existem diversos questionamentos a respeito do que eles podem fazer com essas informações. Empresas privadas teriam acesso de forma privilegiada a perfis de usuários e suas preferências, podendo redirecionar propagandas, obter opiniões de grupo e obter lucro. Neste quesito, pode-se citar os aplicativos do Google²⁰ que utilizam informações dos usuários para redirecionar propagandas personalizadas. Além disso, questões relacionadas à legislação imposta pelo país em que o provedor se localiza devem ser consideradas. Aplicações de governo em especial devem levar em consideração esse quesito. Órgãos como a *Agência Nacional de Segurança* (NSA) americana utilizando de pressupostos de segurança podem tomar posse dos dados hospedados nos Estados Unidos para obter informações estratégicas sobre os governos.

2.2 Utilização de serviços de nuvem sob a perspectiva de Engenharia de Software

A escolha de um serviço de nuvem depende muito do tipo de aplicação que se pretende utilizar. A seguir será apresentado como os principais tipos de serviços podem ser utilizados. A perspectiva apresentada refere-se exclusivamente ao desenvolvimento de software. Tentou-se esclarecer alguns conceitos e apresentar algumas questões práticas relacionadas especificamente ao desenvolvimento de software para a nuvem. Uma melhor compreensão sobre o uso da nuvem permitirá que desenvolvedores elaborem mecanismos mais eficientes que, como consequência, podem facilitar a adoção deste modelo. Além disso, este trabalho trata do desenvolvimento de software para nuvem, portanto, é de fundamental importância a compreensão de como tais sistemas são desenvolvidos. A [Figura 4](#) apresenta uma síntese de como atores podem interagir com os modelos de serviço na Computação em Nuvem.

Entre os modelos de serviço, a Computação em Nuvem dá aos provedores de aplicação a escolha de negociar seus produtos como SaaS ([ARMBRUST et al., 2009](#)). O vendedor também conhecido como provedor disponibiliza seus serviços para venda. Os usuários finais e desenvolvedores normalmente consomem esses serviços e pagam sob demanda. Para o usuário final não é necessária, a aquisição de grande infraestrutura para utilizar os serviços. Até computadores com pequeno poder computacional podem ser utilizados como clientes.

Antes de abordar como cada uma das ofertas de nuvem podem ser adotadas, no entanto, discute-se a diferença entre *SaaS*, *SOA* e *Application Server Provider (ASP)*, para elucidar uma confusão recorrente referente a esses termos. Esses conceitos são fundamentais para compreensão do desenvolvimento de software para nuvem e estão diretamente relacionados com o contexto desta pesquisa.

²⁰ <<http://www.google.com.br/>>

Figura 4 – Atores e modelos de serviços da Computação em Nuvem

Fonte: adaptado de [Marinos e Briscoe \(2009\)](#)

2.2.1 SaaS vs SOA vs ASP

[Laplante, Zhang e Voas \(2008\)](#) fizeram um estudo aprofundado das características dos modelos SaaS e SOA e apontaram que há na literatura uma grande confusão entre os termos. Por exemplo, [Sharma e Sood \(2011\)](#) não distinguem os termos SOA e SaaS, eles propuseram uma abordagem para interoperabilidade entre aplicações de nuvem que utiliza MDE para gerar *Web Services Description Language* (WSDL)²¹ e se referiram à abordagem como desenvolvimento de SaaS, mas de fato estão focados no conceito de SOA. É importante destacar que a diferença fundamental é que SOA é um modelo de construção de Software e SaaS é um modelo de entrega ([LAPLANTE; ZHANG; VOAS, 2008](#)). Enquanto SOA se refere à criação de serviços para compor uma arquitetura mais flexível para uma aplicação, SaaS se refere à entrega da funcionalidade de software em forma de serviços, que são providos independentemente do estilo arquitetural do software.

Para esclarecer a diferença entre SaaS e SOA é possível citar como exemplo de SaaS, o Google Apps. O Google disponibiliza seu conjunto de aplicações, como Gmail, Google Calendar e Google Sites, para pequenas empresas que não desejam utilizar as versões públicas, como normalmente acontece com pessoas físicas. Para obterem tal serviço, as empresas devem inicialmente investir um determinado valor por mês para cada pessoa em sua conta²². Para o cliente pouco importam questões arquiteturais dos sistemas oferecidos, apenas o serviço.

Como exemplo de SOA, é possível citar um sistema que utiliza a composição de serviços para prover sua funcionalidade final. O site de compras Submarino²³, por exemplo, utiliza um

²¹ WSDL: é uma linguagem baseada em XML utilizada para descrever Web Services funcionando como um contrato do serviço. Além de descrevê-lo, especifica como acessar e quais as operações ou métodos disponíveis. <<http://www.w3.org/TR/wsdl>>

²² Google Apps: <<http://www.google.com/apps/intl/en/business/index.html>>

²³ Submarino: <<http://www.submarino.com.br>>

web service dos Correios²⁴ para efetuar cálculo do valor do frete. Para isso, o sistema faz uma chamada fornecendo alguns parâmetros como CEP de origem e CEP de destino, tipo de serviço (sedex ou comum) e peso/dimensões do produto. As informações devem ser passadas ao web service em formato XML, e ele enviará a resposta também em XML. Em SOA, a arquitetura interna do sistema deixa explícito o uso de serviços para proporcionar mais flexibilidade e permitir que módulos específicos sejam construídos (PAPAZOGLU *et al.*, 2007; PAPAZOGLU *et al.*, 2008).

Ainda com relação a esse assunto, cabe fazer uma comparação entre as definições de SaaS e o modelo de provedor de aplicações ASP, que é o modelo utilizado anteriormente ao surgimento dos conceitos de nuvem. No SaaS, o cliente não precisa ter a propriedade do software ou adquirir uma licença; simplesmente paga uma taxa, geralmente tendo por base a quantidade de acesso ao serviço, referente ao uso (ARMBRUST *et al.*, 2009). No modelo anterior (ASP), o cliente adquire uma licença e instala o sistema em um provedor, mas ainda é necessário se preocupar com questões de plataforma e execução da aplicação (SMITH; KUMAR, 2004). No exemplo anterior, do Google Apps, a empresa não paga pela licença de uso, e nem instala nenhum software, apenas utiliza o serviço (SaaS) pela Internet e paga sob demanda. Já sistemas como o famoso *Enterprise Resource Planning* (ERP) JD Edwards²⁵ da empresa Oracle, seguem o modelo ASP, ou seja, o cliente adquire uma licença de uso e instala o software em sua própria infraestrutura.

Feita a distinção entre SaaS e SOA, cabe ressaltar que a confusão provavelmente existe porque geralmente SaaS é implementado através de SOA, pois os dois modelos se complementam. SaaS ajuda a fornecer componentes para SOA, e SOA ajuda a realizar SaaS mais rapidamente (LAPLANTE; ZHANG; VOAS, 2008).

2.2.2 Utilização do Modelo SaaS

O modelo de entrega SaaS tem potencial para revolucionar a forma de como as empresas vão investir em sistemas de informação. Na perspectiva de adoção, esse modelo é ideal para empresas iniciantes que não têm capital para empregar em sistemas de grande porte (normalmente implantados sob a forma de licença de software), caso em que é necessário investir altas taxas em implantação e manutenção dos sistemas. Com o SaaS, a pequena empresa poderá contratar sistemas de informação robustos e consagrados no mercado pagando apenas uma fatura mensal referente ao uso. Muitas vezes, será possível a utilização do software de diversos dispositivos ou locais pagando-se apenas um único contrato. Na perspectiva de venda de serviços computacionais e software, empresas que possuem sistemas de alto custo poderão se inserir em um novo segmento de mercado na medida em que poderão vender seus serviços de acordo com o volume

²⁴ <<http://www.correios.com.br/>>

²⁵ <[JDEdwards:http://www.oracle.com/br/products/applications/jd-edwards-enterpriseone/index.html](http://www.oracle.com/br/products/applications/jd-edwards-enterpriseone/index.html)>

de utilização.

Do ponto de vista do desenvolvimento de sistemas, para adotar SaaS a equipe de desenvolvimento cria aplicações (fazendo o uso de PaaS ou de tecnologias compatíveis com a plataforma adotada) e as implanta na infraestrutura da nuvem. Em SaaS, os fornecedores de sistemas precisam se preocupar com questões referentes a cobrança dos serviços e um modelo de negócio orientado a serviços. Auditoria, relatórios de uso e contabilidade no uso de serviços também são funcionalidades que devem ser consideradas pelo engenheiro de software, que deve prover funções administrativas de apoio a esses tipos de tarefas, que não são necessárias no modelo de software tradicional. Os sistemas são disponibilizados ao usuário final através da rede (ARMBRUST *et al.*, 2009). O ideal é que não haja a preocupação com plataforma ou com que tipo de dispositivo será possível utilizá-los. Para tanto, é necessário que o sistema seja independente de plataforma ou possua várias versões de maneira a contemplar as diversas plataformas.

2.2.3 Utilização do Modelo IaaS

O modelo IaaS fornece recursos computacionais que estão na infraestrutura do provedor. O público alvo desse modelo geralmente são os arquitetos de infraestrutura (SILVA; LUCRÉDIO, 2012). O usuário não precisa se preocupar com demandas de computação, o provedor é quem se incumba de fornecer uma infraestrutura dinâmica, elástica e escalável através da Internet (ARMBRUST *et al.*, 2009). Manter um grande servidor na infraestrutura de uma empresa gera custos, tais como aquisição, manutenção, energia, mão de obra especializada, tempo, entre outros. A ideia é que a empresa possa focar em sua atividade principal e terceirizar a infraestrutura computacional, e que o pagamento desse serviço seja feito de acordo com o volume de utilização (ARMBRUST *et al.*, 2009). Além disso, pequenas empresas podem eliminar a necessidade de adquirir antecipadamente recursos computacionais, podendo investir mais capital em suas tarefas de negócio.

Com a contratação da infraestrutura sob demanda, a empresa não tem gastos desnecessários, ou seja, o objetivo é reduzir desperdícios. Com a redução de custo em relação à aquisição e manutenção de servidores, é possível tornar-se mais competitivo no mercado, saindo na frente das empresas que utilizam o modelo tradicional e que agregam os custos da manutenção de sua infraestrutura ao preço final de seu produto.

Outros usuários que podem ter grande benefício no uso desse modelo são as instituições ligadas ao governo. A redução de desperdícios e a simplicidade de gerenciamento estão diretamente alinhadas às suas necessidades. Todavia, há uma grande preocupação com a segurança das informações. Possivelmente, quando esse quesito estiver melhor resolvido, as demandas de serviços computacionais passarão a ser mais um item nas listas de licitações.

Na perspectiva de desenvolvimento de sistemas, para adotar esse modelo a equipe de

desenvolvimento cria aplicações como se estivesse desenvolvendo software para uma infraestrutura própria dedicada, diferindo assim do modelo PaaS, apresentado a seguir. Com exceção de alguns serviços adicionais, como por exemplo balanceamento de carga, escalabilidade automática, serviço de e-mail e de armazenamento de dados, a aplicação não apresenta nenhuma característica que exige conhecimento especial por parte do engenheiro de software acerca da infraestrutura do provedor de nuvem. Sua implantação na plataforma contratada, sob a forma de serviço, também acontece da forma tradicional, pois a equipe de desenvolvimento normalmente tem acesso irrestrito às máquinas virtuais, da mesma forma que teria em uma máquina real.

Em nuvens públicas há o diferencial de que a entrega ao usuário final sempre ocorre por meio da Internet, o que na prática já acontecia na maioria dos casos em que nuvens não eram utilizadas. Cabe lembrar que não necessariamente o sistema será entregue sob a modalidade SaaS e nem precisa ser Orientado a Serviço (SOA). Apenas está sendo feito aluguel da infraestrutura, o que requer que as tecnologias utilizadas no desenvolvimento do software estejam de acordo com os requisitos exigidos pelo provedor.

No modelo IaaS quem presta o serviço de Computação em Nuvem é o provedor de nuvem, ou seja, quem está sendo contratado para fornecer um centro de dados virtual sob a forma de serviço. IaaS são ideais para implantar sistemas Web legados (ESPARZA; ESCOÍ, 2011). É possível utilizar essa infraestrutura de várias maneiras, inclusive para hospedar um sistema que será comercializado sob a forma ASP ou somente para processamento de dados de origens diversas.

2.2.4 Utilização do Modelo PaaS

PaaS oculta as complexidades de desenvolvimento de sistemas para nuvem e provê uma plataforma consistente e de alto nível para desenvolver aplicações escaláveis (ARMBRUST *et al.*, 2009). É adequada para criação de novas aplicações, pois os aspectos de escalabilidade e detalhes de configuração e de plataforma são automaticamente gerenciados pelo provedor da plataforma (ESPARZA; ESCOÍ, 2011).

No modelo PaaS, a plataforma de desenvolvimento é provida como serviço. Seu público alvo são os desenvolvedores de sistemas, os quais adotam uma plataforma oferecida por alguma empresa e seguem suas tecnologias e padrões para criar seus sistemas. Nesse ambiente, comumente é disponibilizado um conjunto de bibliotecas que os desenvolvedores podem utilizar para desenvolver e hospedar suas aplicações. Inicialmente, adoção de PaaS pode exigir algum treinamento dos desenvolvedores que irão trabalhar com a plataforma. No entanto, em um segundo momento tais tecnologias podem reduzir o tempo de desenvolvimento por prover funcionalidades, facilitar o desenvolvimento e distribuição de aplicações para nuvem (SILVA; LUCRÉDIO, 2012). Por exemplo: o *Google App Engine* (GAE)²⁶ atualmente provê muitas

²⁶ <<https://appengine.google.com/>>

funcionalidades, dentre elas API para processamento de imagens, armazenamento de dados de maneira não relacional, entre outros.

Mais uma vez cabe lembrar que, assim como nos demais modelos, os sistemas desenvolvidos utilizando PaaS não necessariamente precisam seguir o estilo arquitetural SOA. Na perspectiva de desenvolvimento de sistemas, para adotar esse modelo, a equipe de desenvolvimento utiliza ferramentas e bibliotecas fornecidas pela plataforma para desenvolver suas aplicações. Os aplicativos são hospedados na infraestrutura do provedor e são disponibilizados aos usuários finais através da Internet. Nesse modelo quem presta o serviço de Computação em Nuvem é o provedor da plataforma, de maneira que a modalidade de entrega dos sistemas desenvolvidos utilizando PaaS fica sob responsabilidade da empresa que os desenvolveu, ou seja, não necessariamente será feito o uso de SaaS (conforme abordado na [Subseção 2.2.1](#)).

O problema em utilizar o PaaS é que o modelo praticamente impõe o uso de um conjunto de tecnologias (linguagens e bibliotecas) específicas. Comumente também se requer que um padrão de desenvolvimento seja seguido, forçando o desenvolvedor a organizar os componentes da aplicação de uma maneira particular e tornando muito mais complicada a migração entre plataformas levando ao surgimento do *lock-in* ([ESPARZA; ESCOÍ, 2011](#)). Todavia, esse modelo apresenta diversas vantagens porque libera o desenvolvedor de diversos quesitos relacionados a gerenciamento de infraestrutura.

2.3 Portabilidade na Computação em Nuvem

Conceitos de portabilidade e interoperabilidade na Computação em Nuvem estão diretamente relacionados e podem, muitas vezes, ser confundidos. As definições de interoperabilidade serão melhor exploradas mais adiante, mas em termos gerais interoperabilidade seria a habilidade de múltiplos provedores/ofertas de nuvem trabalharem juntos e portabilidade seria a facilidade na migração de provedor e/ou oferta de nuvem ([ZHANG; WU; CHEUNG, 2013](#); [NOGUEIRA et al., 2016b](#)).

Conforme abordado anteriormente, o problema denominado *lock-in* é o aprisionamento do usuário ao provedor ou serviço de nuvem causado pela dificuldade em migrar dados e programas de uma plataforma de nuvem para outra ([ARMBRUST et al., 2010](#)). A falta de portabilidade e interoperabilidade entre os provedores de nuvem causam o *lock-in*. O termo *lock-in* refere-se ao fato de que os desenvolvedores, muitas vezes, não podem mudar de provedor de nuvem sem um trabalho de reengenharia considerável, e portanto, acabam ficando “presos” a uma plataforma ou tecnologia.

Uma das causas do *lock-in* é a natureza dinâmica das tecnologias que levam a diferentes ofertas de serviços de nuvem. O problema pode ser mais facilmente visto no contexto de PaaS: para obter as vantagens do uso de PaaS, as aplicações precisam ser desenvolvidas levando-se em consideração o conjunto de tecnologias adotadas pela plataforma. PaaS impõe um estilo próprio

de programação e muitas vezes, possui uma maneira específica de armazenar dados. Assim, uma aplicação desenvolvida para um determinado PaaS não pode ser facilmente portada para outro provedor. Outra possibilidade de visualizar o *lock-in* seria no contexto de SaaS devido aos modelos de dados das aplicações SaaS e a possível complexidade e custo de uma migração.

Não é difícil ver porque o *lock-in* pode dificultar a adoção do modelo de nuvem. A possibilidade de ficar preso a um determinado provedor e, conseqüentemente, ser cobrado com taxas abusivas para sua utilização, coloca o usuário da nuvem em uma posição difícil para escolher quais ofertas são mais viáveis aos seus requisitos de negócio. Muitas empresas acabam por adotar o modelo de IaaS porque esse modelo é mais flexível em termos de portabilidade. Além disso, considerando os diversos modelos disponíveis e pelo fato de a Computação em Nuvem ser uma tecnologia em desenvolvimento, é possível que plataformas e bibliotecas fiquem rapidamente obsoletas, sendo necessário um grande trabalho de manutenção nos sistemas que as utilizam ou ainda um processo de reengenharia para adaptar os sistemas a novas demandas.

Devido a esse problema, usuários ficam vulneráveis ao aumento de preço, problemas de confiabilidade e dificuldade extra para abandonar o modelo, mudar de um provedor para outro ou ainda compor diversas ofertas para desenvolver um sistema que tire proveito de características específicas de cada provedor. Tudo isso ocorre devido à inexistência de uma tecnologia padrão amplamente estabelecida (ARMBRUST *et al.*, 2009), bem como de alternativas de portabilidade e interoperabilidade para desenvolvimento de aplicações para o modelo de nuvem.

Em relação à portabilidade, sua falta faz com que, muitas vezes, o trabalho empenhado em tarefas específicas de plataforma possa não ser reaproveitado em outras plataformas. A portabilidade entre provedores de serviços de nuvem é um elemento de qualidade de software e um tópico atual que tem sido investigado em diversos trabalhos na literatura (BOZMAN; CHEN, 2010; RANABAHU; SHETH, 2010; CUNHA; NEVES; SOUSA, 2012; SILVA; FORTES; LUCRÉDIO, 2013; PETCU, 2011; PETCU *et al.*, 2013). Existem diferentes tipos de portabilidade no contexto da nuvem, dentre os quais são destacados os seguintes:

- **Portabilidade de máquinas virtuais entre provedores de nuvem:** o modelo IaaS normalmente se utiliza da virtualização de servidores²⁷ para prover recursos computacionais aos clientes. Geralmente, as máquinas virtuais são gerenciadas pelos próprios clientes. Apenas migrar uma máquina virtual de um provedor para outro não causaria nenhum impacto nos sistemas que estão sendo virtualizados, apenas seria necessário fazer uma cópia do disco virtual de um provedor para o outro e em seguida a instanciação da máquina virtual. Neste sentido, o *Open Virtualization Format* (OVF)²⁸ consiste em uma alternativa para padronizar as máquinas virtuais e conseqüentemente permitir que as mesmas sejam executadas por diferentes provedores que suportam o formato.

²⁷ A Virtualização de Servidor é a técnica de execução de um ou mais servidores virtuais sobre um servidor físico (DANIELS, 2009).

²⁸ OVF: <<http://www.dmtf.org/standards/ovf>>

- **Portabilidade de aplicações no contexto de IaaS:** alguns provedores, ao invés de oferecer máquinas virtuais (VM), vendem planos de hospedagem que suportam apenas uma tecnologia específica. Muitos possuem planos para hospedagem Java, Ruby, PHP, entre outros. Neste caso, os usuários não têm controle sobre a VM, ou seja, o próprio provedor administra a implantação e execução das aplicações. Uma vez que as aplicações são implantadas em um determinado provedor, pode ser necessário um grande esforço para mudança, pois cada provedor oferece planos de acordo com seus objetivos de negócio. Por exemplo, a empresa Locaweb²⁹, que é uma popular prestadora desse tipo de serviço no Brasil atualmente, oferece planos específicos, sendo que o usuário tem a opção de comprar o pacote de acordo com as tecnologias que necessita. Caso opte por mudar de provedor, é necessário que o usuário verifique se o provedor de destino suporta as mesmas tecnologias utilizadas pelo provedor atual.
- **Portabilidade de aplicações no contexto de PaaS:** conforme abordado anteriormente, o usuário do modelo de serviço PaaS deve seguir estritamente o estilo de programação e fazer uso das bibliotecas e tecnologias disponibilizadas pelos provedores. Para migrar aplicações entre essas ofertas, muitas vezes é necessário um processo total de reengenharia. É necessário que o desenvolvedor conheça os detalhes e bibliotecas de cada plataforma, o que torna a migração difícil e custosa.
- **Portabilidade de dados entre provedores de nuvem:** de maneira semelhante ao item anterior, não há um padrão para uso das tecnologias de gerenciamento de banco de dados (SGBD) na nuvem. Portanto, a portabilidade de dados entre os provedores é mais um item a ser considerado. Em se tratando da oferta de serviço IaaS, existem basicamente dois cenários de uso: máquina virtual, na qual os sistemas instalados pelo consumidor podem ser diversos; ou tecnologia previamente estabelecida em que o provedor suporta apenas SGBDs específicos. Sendo assim, da mesma maneira que na portabilidade de aplicações, antes de fechar o contrato com o provedor, é importante verificar se o mesmo suporta o SGBD em questão. Shirazi, Kuan e Dolatabadi (2012) apresentam uma solução baseada em padrões de projeto para possibilitar portabilidade de dados entre alguns banco de dados de nuvem. No entanto, a solução ainda está em fase de pesquisa e mais estudos ainda são necessários.

Além de levantar a necessidade de portabilidade, a dinamicidade e variedade de tecnologias e provedores de serviços frequentemente leva a cenários nos quais diferentes aplicações ou conjunto de serviços precisam operar de forma conjunta, em uma cooperação heterogênea de recursos e de provedores. Como resultado, é também necessário um esforço para possibilitar a interoperabilidade entre esses elementos.

²⁹ Locaweb: <<http://www.locaweb.com.br>>

2.4 Interoperabilidade na Computação em Nuvem

Conforme apresentado anteriormente, a Computação em Nuvem é uma área de pesquisa em desenvolvimento e muitas pesquisas vêm sendo conduzidas para investigar como esse modelo computacional pode ser aplicado nos diversos campos do conhecimento (SILVA; LUCRÉDIO, 2012). Entre as oportunidades de pesquisa em Engenharia de Software para nuvem, interoperabilidade é um tópico relevante e precisa ser melhor explorado (ARMBRUST *et al.*, 2009; SILVA; LUCRÉDIO, 2012; NOGUEIRA *et al.*, 2016b). A medida em que as tecnologias relacionadas a nuvem vêm evoluindo, muitas propostas de termos, definições e conceitos têm sido apresentados pela comunidade científica (NOGUEIRA *et al.*, 2016b).

Vaquero *et al.* (2008) destacaram as diversas propostas de definições para a Computação em Nuvem e propuseram uma nova definição elaborada pelos autores. Com a interoperabilidade, especificamente no contexto do modelo de nuvem, tem acontecido algo semelhante, ainda resta muita discussão em relação a definições, perspectivas de soluções e caminhos de pesquisa.

Portanto, visando identificar como a interoperabilidade tem sido abordada na literatura e oferecer uma visão atualizada do tópico de pesquisa, foi realizada uma revisão sistemática de literatura conforme as diretrizes propostas por Kitchenham *et al.* (2007). Essa revisão de literatura (NOGUEIRA *et al.*, 2016b) teve por objetivo específico identificar quais são as definições, conceitos, abordagens, requisitos para soluções interoperáveis, características das soluções, modelos de avaliação e principais características relacionadas a uma solução interoperável especificamente no contexto da Computação em Nuvem.

Assim como Vaquero *et al.* (2008) fizeram a respeito da definição do termo Computação em Nuvem, Nogueira *et al.* (2016b) apresentam uma ampla discussão sobre a interoperabilidade especificamente no contexto de nuvem. Os autores apresentam uma nova definição para interoperabilidade, mas exclusivamente sob a perspectiva de Engenharia de Software. Para os autores, no contexto geral do modelo de Computação em Nuvem, a interoperabilidade vai além da simples comunicação entre sistemas e trata-se de um conceito que vai em direção a sistemas mais dinâmicos, heterogêneos e compostos que tiram vantagem dos melhores aspectos dos diversos modelos de serviço e implantação da Computação em Nuvem.

A heterogeneidade remete diretamente ao *lock-in* dada a necessidade da composição de serviços diversos para construção de uma aplicação e, portanto, soluções para o problema em diversos contextos do modelo de nuvem ainda são necessárias (NOGUEIRA *et al.*, 2016b). Uma discussão mais profunda sob interoperabilidade é apresentada por Nogueira *et al.* (2016b); esta seção apresenta os conceitos principais necessários como fundamentação teórica para este trabalho.

Definições

Para [Petcu \(2011\)](#), de uma maneira geral, as definições encontradas se referem a interoperabilidade na Computação em Nuvem como a habilidade de:

- Abstração das diferenças entre provedores de serviços de nuvem.
- Tradução entre abstrações suportadas por diferentes provedores de serviços de nuvem.
- Flexibilizar a execução de aplicações localmente, em nuvem ou ambos.
- Mover aplicações de um ambiente para outro ou executá-las em múltiplas nuvens.
- Mover serviços, processos, fluxos, trabalhos e dados entre nuvens.
- Utilizar a mesma ferramenta de gerenciamento, imagens de servidores e software em múltiplas nuvens.
- Comunicação entre provedores, portabilidade de aplicações e dados entre eles.
- Federação³⁰ de múltiplas nuvens para suportar uma aplicação única.

Na literatura é possível encontrar várias definições para o termo interoperabilidade no contexto da nuvem:

- i) O [IEEEGLOSSARY \(2013\)](#) define interoperabilidade como a capacidade de um sistema ou um produto trabalhar com outros sistemas ou produtos sem esforço especial por parte do cliente. E afirma que, normalmente, a interoperabilidade é possível graças a aplicação de padrões.
- ii) Para [Loutas, Kamateri e Tarabanis \(2011\)](#), interoperabilidade na Computação em Nuvem significa a habilidade de escrever código que funciona com um ou mais provedores de nuvem simultaneamente, independentemente das diferenças entre as plataformas.
- iii) [Rashidi, Sharifi e Jafari \(2013\)](#) definem interoperabilidade na Computação em Nuvem como a habilidade de um serviço interagir com outros serviços homogêneos ou heterogêneos, para melhorar seus serviços e podem ser implementados sob um ou mais domínios. Os autores argumentam que serviços de nuvem são implementados em diferentes camadas; serviços na mesma camada são classificados como homogêneos e serviços em diferentes camadas são heterogêneos.

³⁰ O conceito de federação será explorado mais adiante. Em resumo, consiste em um modelo de múltiplas nuvens, no qual se assume um contrato formal entre os provedores.

- iv) Para [Dowell et al. \(2011\)](#), interoperabilidade na Computação em Nuvem refere-se à facilidade na migração e integração de aplicações e dados entre provedores de nuvem diferentes.
- v) Segundo [Zhang, Wu e Cheung \(2013\)](#), interoperabilidade significa a habilidade de sistemas heterogêneos funcionarem e interagirem juntos.
- vi) Para [Petcu \(2011\)](#), interoperabilidade é uma propriedade que refere-se a habilidade de diversos sistemas ou organizações trabalharem juntos (inter-operar). No contexto da computação, muitas vezes, essa propriedade tem o significado da troca de informação e o uso da informação que foi trocada entre dois ou mais sistemas ou componentes. O principal objetivo da interoperabilidade é permitir tirar vantagens de propriedades da nuvem.
- vii) Para [Harsh et al. \(2012\)](#), interoperabilidade na computação em nuvem pode ser definida como a habilidade de um provedor entender formatos de aplicação, Templates de Service Level Agreement (SLA), formatos de *tokens* de autenticação e autorização e atributos de dados.
- viii) Para [Nogueira et al. \(2016b\)](#) interoperabilidade na Computação em Nuvem refere-se ao desenvolvimento de aplicações que combinam recursos providos por múltiplos provedores. Esses recursos podem inter-operar (trocar informações ou trabalhar juntos) para compor uma nova aplicação denominada **aplicação interoperável** formada pela combinação de serviços específicos de cada provedor envolvido na composição. A ideia é que serviços selecionados sejam compostos para formar uma nova aplicação.

É possível perceber que existem muitas definições e elas não necessariamente são compatíveis. A definição i) é bastante abrangente e, embora a mesma não tenha sido definida especificamente para nuvem, a mesma também pode ser aplicada. A definição ii) também é bastante abrangente, pois uma aplicação pode funcionar em mais de um provedor ao mesmo tempo e suas partes se comunicarem por meio de SOA. A definição iii) remete diretamente à definição i), mas determina uma nomenclatura para comunicação entre os diferentes tipos de serviços. A definição iv) remete mais à portabilidade do que propriamente interoperabilidade. As definições v) e vi) são bastante parecidas, mas a definição vi) ressalta que interoperabilidade na Computação em Nuvem vai além da simples comunicação entre dois ou mais sistemas. A definição vii) refere-se mais diretamente a formatos de dados.

Até o momento, não foi encontrada uma definição que fosse aceita como um padrão pela comunidade científica. Essa falta de definição consolidada aconteceu também com o conceito de Computação em Nuvem, até que a definição do NIST passou a ser a mais utilizada. Assim, a definição viii) é apresentada por [Nogueira et al. \(2016b\)](#) como uma proposta para visualização da interoperabilidade na Computação em Nuvem sob a perspectiva da Engenharia de Software.

Rezaei *et al.* (2014) afirmam que interoperabilidade é um conceito multidimensional e pode ser visualizado sob várias perspectivas e abordagens em diversas direções. Os autores identificaram quatro níveis de interoperabilidade: técnica, sintática, semântica e organizacional. A seguir são descritos esses níveis.

- **interoperabilidade técnica:** tipicamente associada a componentes de hardware/software, é alcançada na comunicação entre sistemas eletrônicas ou itens de equipamentos de comunicação eletrônicas quando serviços ou informações precisam ser trocadas entre eles e seus usuários.
- **interoperabilidade sintática:** é definida como a habilidade de trocar dados. É associado a formato de dados e as mensagens trocadas na comunicação devem possuir uma sintaxe bem definida.
- **interoperabilidade semântica:** normalmente relacionada à definição de conteúdo, e mais voltada para a interpretação de conteúdo, e não apenas seu formato.
- **interoperabilidade organizacional:** é a capacidade das organizações trocarem informações entre si independentemente do uso de diferentes sistemas de informação. O sucesso dos demais níveis de interoperabilidade (técnica, sintática e semântica) forma a base para interoperabilidade organizacional.

Em outra perspectiva, Rezaei *et al.* (2014) identificaram quatro níveis de problemas (*interoperability issues*) relacionados a interoperabilidade. A Figura 5 apresenta os domínios de problema e suas respectivas classificações em cada nível. Os níveis mais elevados podem incluir elementos dos níveis anteriores. Cada nível e seus respectivos tipos podem ser definidos como segue:

- **Nível 1:** é composto por interoperabilidade de dados, interoperabilidade de processo, interoperabilidade de regras, interoperabilidade de objetos, interoperabilidade de sistemas e interoperabilidade cultural (REZAEI *et al.*, 2014).
 - **Interoperabilidade de dados:** descreve a capacidade de dados (incluindo documentos, conteúdo multimídia e recursos digitais) serem universalmente acessíveis, reutilizáveis e compreensíveis por todas partes de transações (humano-máquina e máquina-máquina). Se duas aplicações possuem modelos de dados diferentes a interoperabilidade é comprometida.
 - **Interoperabilidade de processo:** a habilidade de alinhar um processo a diferentes entidades (empresas), para que essas empresas possam trocar dados ou conduzir negócios de forma transparente.

- **Interoperabilidade de regras:** é a habilidade de entidades (empresas) alinharem e combinarem seus negócios e regras para legitimar transações automatizadas que também são compatíveis com operações e negócios internos das outras.
- **Interoperabilidade de objetos:** refere-se às interconexões e cooperação de objetos do dia-a-dia das entidades. Esses objetos podem abranger aspectos que vão além de componentes de software, o conceito é consistente com o conceito de Internet das Coisas³¹.
- **Interoperabilidade de sistemas:** refere-se à habilidade de um sistema ou produto funcionar com outro sistema ou produto sem a necessidade de muitos esforços pelos interessados no sistema.
- **Interoperabilidade cultural:** é quando conhecimento e informação são apoiados por um modelo unificado entre as culturas. Sistemas que implementam a interoperabilidade cultural podem ser utilizados por diferentes pessoas em linguagens diferentes e culturas no mesmo domínio de interesse de uma maneira eficiente e econômica.
- **Nível 2:** é focado em interoperabilidade do conhecimento (que consiste de elementos de interoperabilidade de dados), interoperabilidade de serviços (que incorpora elementos de interoperabilidade de processo, dados, regras e sistemas), interoperabilidade de redes sociais (que incorpora elementos de interoperabilidade cultural) e interoperabilidade de identificação eletrônica (que está fortemente relacionado interoperabilidade de objetos, interoperabilidade de sistemas e interoperabilidade de regras) (REZAEI *et al.*, 2014).
 - **Interoperabilidade de conhecimento:** a habilidade de duas ou mais entidades compartilharem seus elementos intelectuais, tomar vantagem do conhecimento mútuo e utilizá-lo para estendê-lo através de cooperação.
 - **Interoperabilidade de serviços:** pode ser definido como a habilidade de uma empresa dinamicamente registrar, agregar e consumir serviços compostos de uma fonte externa, tais como parceiros de negócio ou provedor de serviços de maneira transparente.
 - **Interoperabilidade em redes sociais:** refere-se à habilidade de empresas se interconectarem e utilizarem redes sociais com o propósito de colaboração, alinhando sua estrutura interna a aspectos fundamentais das redes sociais.
 - **Interoperabilidade de identidade eletrônica:** refere-se à habilidade de diferentes sistemas de identidade eletrônica colaborarem de maneira a automaticamente autenticar e autorizar entidades a transmitir informações de segurança e permissões para o donos das identidades eletrônicas correspondentes.

³¹ A principal ideia por trás do conceito de Internet das coisas é a onipresença de uma variedade de coisas (objetos) como *Radio-Frequency Identification* (RFID), tags, sensores, atuadores, dispositivos móveis, etc. Esses objetos podem comunicar entre si para alcançar objetivos comuns (ATZORI; IERA; MORABITO, 2010).

- **Nível 3:** consiste da **interoperabilidade na Computação em Nuvem** (em especial, foco deste trabalho) que inclui elementos da interoperabilidade de serviços, conhecimento, identidade e tenta combinar com as características da nuvem. A interoperabilidade especificamente no contexto da Computação em Nuvem será explorada com mais detalhes em seguida (REZAEI *et al.*, 2014).
- **Nível 4:** inclui interoperabilidade de ecossistemas que pode ser definida como a capacidade de colaboração instantânea e transparente entre ecossistemas diferentes, bem como entidades independentes. Em essência, trata da colaboração entre as empresas e pode incluir interoperabilidade na Computação em Nuvem e interoperabilidade nas redes sociais (REZAEI *et al.*, 2014).

Figura 5 – Tipos de interoperabilidade

Fonte: adaptado de Rezaei *et al.* (2014)

É possível perceber que o nível de domínio do problema da interoperabilidade na Computação em Nuvem (Nível 3), é diferente da interoperabilidade de sistemas de uma maneira geral (mas pode incluí-la). Assim como a palavra nuvem, a interoperabilidade pode ter diferentes significados. Normalmente quando se fala sobre interoperabilidade, é natural pensar em conceitos de SOA.

Conforme apresentado na [Subseção 2.2.1](#), há diferenças entre SOA e os modelos de serviços de nuvem. Além disso, conforme apresentado anteriormente, existem diversos tipos e níveis de interoperabilidade. Portanto, é importante destacar que neste trabalho o termo interoperabilidade refere-se especificamente à interoperabilidade na Computação em Nuvem. Na perspectiva de [Petcu \(2011\)](#), vários critérios para classificação dos tipos de interoperabilidade especificamente na Computação em Nuvem podem ser identificados:

- Nível de contrato:
 - Interoperabilidade sintática: quando sistemas são capazes de se comunicar e trocar dados usando formatos de dados específicos e protocolos de comunicação.
 - Interoperabilidade semântica: uma interpretação automática de maneira significativa e com precisão da informação trocada, a fim de produzir resultados úteis (usando um modelo de referência comum a essa troca de informações).
- Nível de adoção
 - Por projeto: quando é feita uma documentação padrão para criar produtos.
 - Pós-fato: resultado de um domínio absoluto de um produto particular.
- Nível de implantação:
 - Horizontal: interoperabilidade dos serviços no mesmo nível de implantação. Ou seja, interoperabilidade IaaS-IaaS, PaaS-PaaS e SaaS-SaaS.
 - Vertical: interoperabilidade entre os diversos níveis de implantação em qualquer combinação. Ex. IaaS-PaaS, PaaS-SaaS, etc.
- Padrões de interação entre nuvens
 - Síncrono: chamadas diretas, adequadas para sistemas onde o tempo de resposta é fundamental.
 - Assíncrono: aplicações de baixo acoplamento em que um dos comunicantes não precisam esperar a resposta para continuar funcionando.
- Nível de solução
 - Centrada no Provedor (*Provider-Centric*): soluções baseadas em tecnologias para provedores de serviço (como iniciativas de padronização).
 - Centrada no usuário (*User-Centric*): soluções ou aplicações e serviços implementadas sobre infraestruturas de nuvens existentes.

Esses critérios referem-se a uma classificação da solução, ou seja, qual estratégia de interoperabilidade que está sendo implementada. É possível ainda, classificar as abordagens existentes conforme a tecnologia que está sendo utilizada na implementação da solução. Neste quesito, as abordagens podem utilizar de ou construir API's abertas, protocolos abertos, padrões, camadas de abstração, repositórios semânticos, *Domain-Specific Languages* (PETCU, 2011).

Identificados os níveis de interoperabilidade (técnica, sintática semântica e organizacional), os domínios de problema e seus respectivos níveis (Nível 1, Nível 2, Nível 3 e Nível 4), bem como as características de uma solução interoperável e os critérios de classificação; **fica claro que a interoperabilidade no contexto da Computação em Nuvem vai muito além da simples comunicação entre sistemas, camadas ou plataformas.**

Conceitos

Para Petcu (2011), as aplicações de múltiplas nuvens são a evolução da interoperabilidade na Computação em Nuvem. Isso não está explícito nas definições encontradas, é possível perceber que as definições são múltiplas, confusas e às vezes conflitantes entre si (NOGUEIRA *et al.*, 2016b). No entanto, um ponto ficou bastante claro: interoperabilidade no contexto da Computação em Nuvem não é somente SOA (ou tecnologias relacionadas). A tradicional interoperabilidade de sistemas é diferente da interoperabilidade no contexto da nuvem. Na nuvem, é evidente que muito mais conceitos estão envolvidos.

Enquanto diversas ontologias e arquiteturas estão disponíveis para o modelo de nuvem (como as apresentadas na Seção 2.1), modelos para o uso de múltiplas nuvens ainda não estão bem estabelecidos e há diferenças entre os termos que surgiram em decorrência dos cenários de uso de múltiplas nuvens. Existem muitas maneiras de se combinar nuvens para formar um novo serviço ou oferta. Entre essas maneiras, três principais conceitos se destacam por formar a base para o surgimento de diversos outros termos relacionados. São eles: *inter-cloud*, Federação e *multi-cloud*.

De acordo com a arquitetura proposta pelo NIST (LIU *et al.*, 2011), há um ator da Computação em Nuvem denominado *cloud broker* que gerencia o uso, performance e entrega de serviços de nuvem e negocia relacionamentos entre diversos provedores e consumidores das diferentes ofertas de serviços. Um *inter-cloud* é uma nuvem federada ou um *multi-cloud* que inclui ao menos um *cloud broker* e oferece serviços dinâmicos de provisionamento de recursos. A Figura 6 apresenta a classificação arquitetural do *inter-cloud*.

Com base na Figura 6, sob uma perspectiva do uso de múltiplas nuvens, o *inter-cloud* é classificado em dois modelos de entrega de serviços: nuvens federadas e *multi-cloud*. Os modelos podem ser diferenciados pelo grau de colaboração entre as nuvens envolvidas e a maneira como os usuários interagem com as nuvens.

No modelo de nuvens federadas há um consenso (entendimento) entre os diferentes

Figura 6 – Classificação arquitetural do inter-cloud

Fonte: adaptado de [Grozev e Buyya \(2014\)](#)

provedores de nuvem para compartilhar seus recursos (os termos *sky computing*, nuvens agregadas, nuvens *multi-tier*, ou *cross-cloud* são casos particulares de nuvens federadas), enquanto no modelo de *multi-cloud* não há esse consenso. Os modelos podem ainda ser classificados sob uma perspectiva do tipo de colaboração, voluntário no caso da federação e não voluntário no caso do *multi-cloud*. Na federação o usuário interage com a nuvem e não está ciente que os recursos ou serviços que são consumidos são de outra nuvem. O foco das nuvens federadas é principalmente IaaS ([PETCU, 2014](#)). No *multi-cloud*, o usuário está ciente das diferentes nuvens e é responsável (ou um terceiro) por lidar com o provisionamento dos serviços ou recursos, portanto, o termo pode denotar o uso de nuvens múltiplas e independentes por um cliente ou serviço ([GROZEV; BUYYA, 2014](#)).

No caso da federação de nuvens, o *cloud broker* é parte de uma entidade centralizada ou aparece para cada provedor. No caso do *multi-cloud* é parte de um serviço especial ou biblioteca. Na perspectiva arquitetural de [Grozev e Buyya \(2014\)](#), a federação pode ser classificada em centralizada ou par-a-par (*peer-to-peer*). Na arquitetura centralizada há uma entidade que realiza ou facilita a alocação de recursos. No modelo par-a-par os provedores comunicam diretamente entre si, sem um intermediário. O modelo *multi-cloud* pode também ser classificado em duas categorias: baseadas em serviços e baseadas em bibliotecas. As soluções baseadas em serviços consistem de um serviço especial que oferece as funções baseadas em *Software Level Agreements* (SLAs) do usuário que devem ser exercidas por um *cloud broker*. O serviço pode ser hospedado externamente ou nos próprios clientes. As bibliotecas facilitam o acesso a múltiplos serviços e recursos de maneira uniforme ([GROZEV; BUYYA, 2014](#)).

O conceito de *multi-cloud* é inerente ao contexto de interoperabilidade na Computação em Nuvem. *Multi-cloud* é um modelo de *inter-cloud* onde não há consenso para compartilhamento de informações entre os provedores e a colaboração entre eles não é voluntária. Portanto, uma solução para *multi-cloud* pode ser caracterizada pelo uso de múltiplas nuvens por um serviço ou biblioteca de maneira a facilitar o uso dos recursos sem que os próprios provedores tomem essa

iniciativa (uma vez que a decisão de compartilhar não é voluntária). Uma nuvem híbrida, por exemplo, seria um tipo de *multi-cloud* que conecta duas ou mais nuvens considerando diferentes modelos de implantação.

Requisitos

Vários cenários de uso que justificam a necessidade do uso de *multi-cloud* podem ser encontrados na literatura (HOGAN *et al.*, 2011; PETCU, 2013). Como exemplo deles podem ser citados:

- i** - Otimizar custos e melhorar a qualidade dos serviços;
- ii** - Reagir às mudanças de ofertas e tecnologias oferecidas pelos provedores;
- iii** - Se adaptar a regras e legislações vigentes nas diversas localidades (como por exemplo, empresas que não permitem que seus dados sejam hospedados em servidores de terceiros ou países que não permitem que dados governamentais sejam hospedados fora do seu território);
- iv** - Replicação de aplicações em diversos provedores ou consumir serviços de diferentes provedores para garantir maior disponibilidade na Web;
- v** - Evitar dependência de provedores específicos;
- vi** - Garantir cópias de segurança (backups);
- vii** - Agir como um intermediário de serviços de nuvem;
- viii** - Melhorar seus próprios recursos de nuvem e ofertas de serviços, de acordo com contratos com diversos provedores;
- ix** - Consumir diferentes serviços por suas particularidades não serem providas por outros.

Diferentes atores do modelo de nuvem podem estar interessados em diversos cenários de uso. Empresas podem se interessar pelos cenários i e ii e provedores pelos cenários i, vi e ix, por exemplo (PETCU, 2011). Esses cenários de uso reforçam a importância e a necessidade do desenvolvimento de soluções para *multi-cloud*. Os benefícios dessas soluções são vários e estão diretamente alinhados com os cenários.

Petcu (2013) realizou uma extensa varredura na literatura procurando por estratégias existentes, propostas e/ou projetos em fase de pesquisa e desenvolvimento relacionados a *multi-cloud*. A autora destaca diversos requisitos que aparecem nessas soluções e consequentemente precisam ser satisfeitos para construção dos ambientes *multi-cloud*:

- R1 - Requisitos do Grupo de desenvolvimento (refere-se à equipe de desenvolvimento de soluções e serviços) (PETCU, 2013):
 - R1.1 Oferecer um sistema (pode ser um portal, serviço ou interface) de gerenciamento de recursos ou serviços;
 - R1.2 Oferecer serviços que são independentes de plataforma de nuvem;
 - R1.3 Abstrair interfaces de controle de serviços de múltiplas nuvens;
 - R1.4 Oferecer uma interface para a descrição funcional e não funcional de requisitos dos clientes;
 - R1.5 Apoiar a portabilidade de aplicações entre nuvens conectadas;
 - R1.6 Oferecer um serviço de integração ou agregação de serviços para combinar recursos de nuvens diferentes (como *dashboards* ou *marshups*);
 - R1.7 Seguir os padrões e protocolos no gerenciamento de recursos de nuvem;
- R2 - Requisitos do Grupo de Implantação (refere-se ao grupo que implanta e gerencia serviços de nuvem, que pode incluir mas não se limita aos *cloud brokers* (PETCU, 2013):)
 - R2.1 Oferecer um serviço de seleção de recursos e serviços de nuvem;
 - R2.2 Oferecer apoio a implantação de componentes de aplicações em múltiplas nuvens;
 - R2.3 Preservar as particularidades de várias nuvens;
 - R2.4 Não impor restrições às nuvens conectadas;
 - R2.5 Oferecer uma junção transparente de um novo provedor sem modificar as políticas locais;
 - R2.6 Oferecer um serviço de *cloud broker* ou junção de nuvens;
 - R2.7 Implementar um mecanismo de busca baseado em uma taxonomia específica ou utilizando processamento semântico;
 - R2.8 Oferecer serviço de autenticação ou credenciais para repositório de nuvens;
 - R2.9 Apoiar a conexão com os provedores mais utilizados;
 - R2.10 Apoiar a realocação de aplicações entre nuvens;
 - R2.11 Permitir a implantação em nuvens privadas para possibilitar testes, *debug* e/ou privacidade;
 - R2.12 Implementar novas tecnologias de conectividade e rede para superar conectividades limitadas;
 - R2.13 Implementar mecanismos de agendamento, balanceamento de carga e escalonamento de serviços;

- R2.14 Utilizar procedimentos automáticos de implantação (como linguagens de descritores de implantação ou sistemas inteligentes de gerenciamento)
 - R2.15 Implementar um sistema de recomendação e/ou gerenciador de confiança ou reputação;
 - 2.16 Possibilitar um mercado justo para serviços e atualizações das ofertas das nuvens conectadas ao ambiente *multi-Cloud*.
- R3 - Requisitos do Grupo de Execução (refere-se a execução de soluções propriamente, inclui provedores de nuvem) (PETCU, 2013):
 - R3.1 Oferecer apoio a execução de componentes de software simultaneamente em múltiplas nuvens;
 - R3.2 Oferecer um serviço de monitoramento para as aplicações implantadas;
 - R3.3 Oferecer um serviço de monitoramento de consumo de recursos de nuvem;
 - R3.4 Possibilitar o controle de todo o ciclo de vida das aplicações implantadas;
 - R3.5 Possibilitar a medição do grau de cumprimento dos *SLAs*;
 - R3.6 Possibilitar a alocação dinâmica de recursos ou mecanismos de adaptação;
 - R3.7 Apresentar pouca sobrecarga em comparação a uma conexão direta com cada nuvem suportada no *multi-cloud*.

Os requisitos acima apresentados por Petcu (2013) constituem uma iniciativa no que se refere a conjuntos de requisitos para nuvem. Todavia, a autora não apresenta de onde os mesmos foram elicitados e como eles foram definidos. Mais estudos relacionados a essa área ainda são necessários (NOGUEIRA *et al.*, 2016b).

2.5 Abordagens para Interoperabilidade na Computação em Nuvem

Conforme apresentado anteriormente, interoperabilidade na Computação em Nuvem é um tópico de pesquisa bastante abrangente. Na literatura é possível encontrar várias iniciativas aplicadas a contextos diversos e em direção às várias perspectivas do problema. Muitas propostas e soluções têm surgido, tanto em termos de soluções acadêmicas quanto no que se refere às iniciativas de mercado. Os conceitos apresentados nas seções anteriores são resultados de uma vasta e rigorosa RSL (KITCHENHAM *et al.*, 2007). A quantidade de publicações científicas relacionadas à Computação em Nuvem como um todo é crescente (SILVA; LUCRÉDIO, 2012). Mas, na prática, as revisões sistemáticas de literatura são mais “acadêmicas”. Muitas empresas apresentam avanços extremamente relevantes sem a preocupação de publicar trabalhos científicos

(sobretudo no que se refere a Computação em Nuvem) e esses trabalhos somente são considerados na literatura muito tempo depois.

Além disso, apesar de todo o rigor envolvido na fase busca de uma RSL, nem sempre todos os estudos disponíveis e soluções existentes são encontrados. O mesmo acontece em relação às buscas *ad-hoc*. Nesta seção foram incluídas algumas das soluções mais referenciadas pelos estudos incluídos na revisão de literatura realizada (NOGUEIRA *et al.*, 2016b). Foram realizadas também buscas *ad-hoc* pela Internet para tentar encontrar iniciativas não identificadas na revisão sistemática (NOGUEIRA *et al.*, 2016b). Uma descrição sobre todas as soluções, seus cenários de aplicação, contextos de uso e eventuais problemas relacionados à adoção de uma solução está fora do escopo desta tese.

O *Cloud Standards Customer Council* (CSCC)³² é um grupo de usuários dedicado a acelerar a adoção do modelo de nuvem por meio da disseminação de seus padrões que visam tratar questões de segurança e interoperabilidade no que se refere à transição de modelos tradicionais de computação para o modelo de nuvem. O CSCC segue a linha do uso de padrões (*standards*) e busca estabelecer um conjunto de recomendações para garantir que usuários do modelo de nuvem (*adopters*) tenham (ao selecionar um serviço de nuvem) a mesma liberdade de escolha e flexibilidade que eles têm na tomada de decisão a respeito da adoção de tecnologias tradicionais.

O grupo CSCC publicou um documento denominado *interoperability and portability for cloud computing: a guide*³³. Nesse documento são apresentados cenários e recomendações para a interoperabilidade como problema. No que se refere a interoperabilidade no contexto do uso de múltiplos serviços (*multi-cloud* - cenário especial relacionado a esta tese), o conselho destaca que quando um serviço de nuvem em sua composição utiliza dois outros serviços (ou seja, dois serviços serão compostos para formar um terceiro a partir da combinação dos anteriores) e os componentes são equivalentes, seria melhor se os dois serviços envolvidos na composição utilizassem interfaces padronizadas. Se as funcionalidades forem diferentes, as interfaces não serão interoperáveis e, portanto, devem haver pelo menos alguns aspectos das duas interfaces que sejam baseados nas mesmas tecnologias.

Se a interoperabilidade acontecer no contexto de SaaS é necessário garantir que as interfaces do usuário, APIs e formato de dados sejam idênticos (ou tenham um mapeamento claro) para que funcionalidades equivalentes sejam executadas em provedores diferentes. No contexto de PaaS, é necessário garantir que os ambientes de desenvolvimento (*web servers*, servidores de banco de dados, entre outros) suportados pelos diferentes provedores envolvidos na combinação sejam compatíveis. Os demais cenários de combinação de serviços de nuvem podem envolver *web services* e comunicação/combinação de serviços em nível de rede.

Para a Microsoft³⁴, em muitos casos, as iniciativas relacionadas a padrões não se aplicam

³² <<http://www.cloud-council.org/>>

³³ em: <<http://www.cloud-council.org/deliverables/>>

³⁴ <<https://www.microsoft.com/cloud/interop/>>

perfeitamente. Padrões foram desenvolvidos para garantir a conectividade, mas que conexões de um determinado *datacenter* podem não funcionar bem considerando as nuvens públicas espalhadas por todos o mundo. A empresa argumenta ainda que a Computação em Nuvem acrescenta novos cenários de gerenciamento, monitoramento, acesso e identidade que são somente parcialmente abordados pelos padrões existentes. Para a empresa, novas plataformas para dar suporte à interoperabilidade devem ser construídas sobre *Web Services* e abordagens baseadas em *Representational State Transfer* (REST) (que consiste em um padrão para indústria) que podem ser facilmente descobertos e consumidos por outras plataformas (Microsoft, 2017).

O projeto mOSAIC³⁵ é robusto e envolve 6 (seis) organizações de 5 (cinco) países. Seu objetivo é contribuir com o avanço do estado da arte na Computação em Nuvem por meio da criação e promoção de uma API de código aberto e uma plataforma para desenvolvimento de aplicações. mOSAIC oferece como principais vantagens, o acesso simples e transparente a recursos heterogêneos de nuvem e um mecanismo para evitar o *lock-in*. A estratégia refere-se à interoperabilidade vertical e seu alvo é o modelo de nuvens híbridas (PETCU, 2011). Em uma perspectiva de mais alto nível, o mOSAIC é composto por uma API e um PaaS que permite implantar e gerenciar configuração de aplicações, oferece uma camada de abstração e aplica semântica no nível de interfaces e componentes (PETCU, 2014). A iniciativa propôs-se a projetar:

- uma plataforma de código aberto e portátil para utilização dos serviços de nuvem suportados por sua API e seus padrões;
- uma arquitetura para representar os vários atores interessados nos sistemas;
- um serviço centrado no usuário, uma ontologia e mecanismos para negociar recursos baseados em processamento de dados semânticos;
- aplicações de prova de conceito com ênfase em aplicações de uso intenso de dados.

No que se refere a API especificamente, os desenvolvedores do mOSAIC decidiram separar a aplicação em camada lógica e camada de nuvem. O desenvolvedor não se preocupa com a infraestrutura. Um descritor de aplicação é utilizado para selecionar a nuvem e o código para conectar a uma nuvem específica é gerado em tempo de execução, utilizando os parâmetros passados ao *framework*. As aplicações são decompostas em componentes executáveis sobre diferentes nuvens.

Em termos de interoperabilidade, garantem a interoperabilidade sintática, enquanto as APIs de alto nível garantem a interoperabilidade semântica. Foi adotado um padrão de comunicação assíncrona (PETCU, 2011). Em resumo, essa solução trata-se de um PaaS que possibilita o desenvolvimento, configuração e gerenciamento de aplicações que executam sobre IaaS (PETCU, 2014). O projeto está diretamente relacionado ao escopo desta proposta, mas

³⁵ <<http://www.mosaic-cloud.eu/>>

difere pelo fato de que a solução mOSAIC visa a combinação de IaaS enquanto que a abordagem apresentada neste trabalho tem foco específico em PaaS.

Cloud4SOA³⁶ apresenta uma arquitetura baseada em *cloud broker* na qual o principal objetivo é solucionar a interoperabilidade na camada de PaaS. A solução consiste em uma abordagem vertical (trata apenas do modelo de serviço PaaS), centrada no usuário (a solução é implementada levando em consideração soluções existentes e seu público alvo são desenvolvedores), semântica (interpreta automaticamente e com precisão a informação trocada entre provedores) e orientada a serviços (utiliza elementos de SOA).

O sistema do Cloud4SOA possibilita o gerenciamento, monitoramento e migração por meio da interconexão de diferentes PaaS entre provedores que compartilham a mesma tecnologia. A Figura 7 apresenta sua arquitetura. Os algoritmos semânticos implementados no Cloud4SOA permitem que dados e aplicações sejam gerenciados e migrados entre ofertas de PaaS que são baseadas nas mesmas tecnologias, mas possuem modelos de dados e APIs diferentes (ZEGINIS *et al.*, 2013; DANDRIA *et al.*, 2012).

A ideia por trás dessa abordagem é facilitar a tarefa dos desenvolvedores na busca, implantação e governança de aplicações em PaaS para melhor atender suas demandas. A arquitetura de referência foi projetada para possibilitar interoperabilidade e o desenvolvimento de aplicações inter-plataformas e consiste de cinco camadas (ZEGINIS *et al.*, 2013):

1. **Camada de front-end:** facilita o acesso do desenvolvedor e dos provedores a solução do Cloud4SOA.
2. **Camada Semântica:** é a parte central e se estende por toda a arquitetura, resolvendo conflitos de interoperabilidade, fornecendo uma base comum para a publicação e busca em diferentes ofertas de PaaS.
3. **Camada SOA:** é composta por um conjunto de ferramentas que pode ser acessada a partir da camada de serviço *front-end*. Implementa as principais funcionalidades oferecidas pelo sistema Cloud4SOA.
4. **Camada governança:** implementa a parte de negócio da solução Cloud4SOA, na qual provedores PaaS e desenvolvedores de aplicações podem estabelecer relacionamento de negócios por meio de SLAs e gerenciar aplicações de nuvem.
5. **Camada de Repositório e camada API harmonizada:** age como um intermediário entre a plataforma e as demais ofertas de PaaS provendo uma API que possibilita o gerenciamento de aplicações entre diferentes PaaS.

³⁶ <<http://www.cloud4soa.eu/>>

Figura 7 – Arquitetura do Cloud4SOA

Fonte: adaptado de [Zeginis et al. \(2013\)](#)

A solução Cloud4SOA apresenta quatro principais funcionalidades: junção semântica, gerenciamento de aplicações multiplataforma, migração de aplicação e monitoramento ([ZEGINIS et al., 2013](#); [DANDRIA et al., 2012](#)):

1. **Junção semântica:** permite a busca por ofertas PaaS existentes que melhor se adequam às necessidades do desenvolvedor. Provê uma interface na qual é possível especificar todos os recursos requeridos pela aplicação. Essa especificação é utilizada por um algoritmo para fazer uma ligação (*match*) entre os requisitos e as características das plataformas de modo a retornar quais plataformas estão mais alinhadas aos requisitos da aplicação.
2. **Gerenciamento de aplicações:** apoia a implantação e governança de aplicações. Os desenvolvedores podem gerenciar o ciclo de vida de suas aplicações independentemente da plataforma em que a mesma foi implantada.

3. **Migração de aplicação:** oferece a funcionalidade de migrar aplicações já implantadas de um PaaS para outro. Mover aplicações entre diferentes PaaS no Cloud4SOA consiste de duas principais etapas: i) mover os dados da aplicação ii) mover e reimplantar a aplicação na nova oferta de PaaS.
4. **Monitoramento:** oferece um mecanismo independente de plataforma para monitorar a performance de aplicações críticas hospedadas em ambientes multi-cloud de maneira a garantir que sua performance é consistente com as expectativas estabelecidas nos SLAs.

Há diferenças entre a maioria das características das ofertas de PaaS, portanto, é evidente a necessidade de uma API que seja genérica o suficiente para incluir uma variedade ampla de plataformas. No Cloud4SOA, adaptadores são responsáveis por preencher a lógica de negócios entre a APIs origem e as APIs destino das ofertas de PaaS, traduzindo as funções de um para o outro e vice-versa. Os adaptadores permitem implantação, migração, governança e monitoramento de uma aplicação. A [Figura 8](#) apresenta a visualização do sistema que relaciona métodos das APIs e as ofertas de PaaS.

Figura 8 – Relação entre métodos e ofertas de PaaS no Cloud4SOA

Características Cloud4SOA	AWS Beanstalk	CloudBees RUN@cloud	CloudControl	Heroku	CloudFoundry	OpenShift
Mecanismo						
Informações sobre o PaaS no mecanismo de busca Cloud4SOA.	✓	✓	✓	✓	✓	✓
Governança						
Implantação	✓	✓	✗	✗	✓	✗
Implantação via GIT	✗	✗	✓	✓	✗	✗
Iniciar	✓	✓	✓	✓	✓	✓
Parar	✓	✓	✓	✓	✓	✓
Apagar	✓	✓	✓	✓	✓	✓
Criar banco de dados	✓	✓	✓	✓	✓	✓
Apagar banco de dados	✓	✓	✓	✓	✓	✓
Importar dados do banco*	✓	✓	✓	✓	✓	✓
Exportar dados do banco*	✓	✓	✓	✓	✓	✓
Monitoramento						
verificar Status	✓	✓	✓	✓	✓	✓
tempo de resposta Http	✓	✓	✓	✓	✓	✓
tempo de resposta ICMP ping	✓	✓	✓	✓	✓	✓
Migração						
Migrar**	↔ (1)	↔ (1)	↔ (2)	↔ (2)	↔ (1)	✗
*	Importar e exportar dados para um banco é feito utilizando parâmetros de configuração do banco de dados e é permitido a um conjunto limitado de banco de dados.					
**	Migração de aplicação e seus dados entre diferentes plataformas é permitido, mas somente é possível entre plataformas específicas que são mencionadas com o mesmo número em uma tabela.					

Fonte: adaptado de [Zeginis et al. \(2013\)](#)

O projeto do Cloud4SOA é aberta e disponível em [github³⁷](#), robusta e pode resolver muitos problemas diretamente relacionados não só ao problema do *lock-in*, mas à governança e ao monitoramento de soluções. Recentemente, foi lançado um gerenciador denominado *Cloud*

³⁷ github é plataforma de hospedagem de código com controle de versão e colaboração - <https://github.com/Cloud4SOA/Cloud4SOA>

Pier³⁸ que tem por base a solução aberta do Cloud4SOA. Zeginis *et al.* (2013) afirmam que interoperabilidade no contexto do Cloud4SOA é definida como a habilidade de aplicações e seus dados serem facilmente implantados e/ou migrados (portanto portabilidade) entre diferentes ofertas de PaaS que estão utilizando a mesma base tecnológica, mas informações e APIs diferentes.

Como foi apresentado no Capítulo 2, interoperabilidade, no contexto desta pesquisa, se refere à habilidade de múltiplos provedores/ofertas de nuvem trabalharem juntos e portabilidade seria a facilidade na migração de provedor e/ou oferta de nuvem. Portanto, sob essa perspectiva, a solução Cloud4SOA está voltada para portabilidade. A portabilidade entre múltiplas plataformas é a base para se alcançar a interoperabilidade, mais especificamente no que se refere a *multi-cloud*, uma vez que a solução proposta não necessita de consentimento por parte do provedor.

São muitas as propostas e protótipos para aplicações de múltiplas nuvens que podem ser encontrados na literatura. A seguir, são apresentadas abordagens agrupadas de acordo com a tecnologia a qual cada uma se baseia e conforme a classificação proposta por Petcu (2011).

Em termos de tecnologia, as abordagens existentes para portabilidade e interoperabilidade podem utilizar de ou construir: (1) APIs abertas, (2) protocolos abertos, (3) padrões, (4) camadas de abstração, (5) repositórios semânticos, (6) DSLs. Assim, essas abordagens se apresentam conforme descritas a seguir.

(1) APIs Abertas: conjuntos de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web. No caso de Computação em Nuvem, pode-se destacar as seguintes APIs:

- **Jclouds**³⁹: provê um *framework* para Java ou Clojure⁴⁰ interagirem com armazenamento (*Blob Storage and Queue Storage*) e diversos recursos de uma grande variedade de plataformas. É uma biblioteca *open-source* e promete ao usuário a liberdade para utilizar abstrações ou tecnologias específicas de nuvem. Possui suporte para mais de 30 provedores e pilhas de software incluindo Amazon⁴¹, GoGrid⁴², OpenStack⁴³ e vCloud⁴⁴.
- **Simple Cloud**⁴⁵: é uma API PHP para interagir com serviços de armazenamento blob, *table* e *queue* de vários provedores.

38 <www.opencloudpier.org>

39 <<http://jclouds.apache.org/>>

40 <<http://clojure.org/>>

41 <<http://aws.amazon.com/ec2/>>

42 <<http://www.gogrid.com/>>

43 <<https://www.openstack.org/>>

44 <<http://vcloud.vmware.com/>>

45 <<http://simplecloud.org/>>

- **CloudLoop**⁴⁶: provê uma interface de sistema de arquivos para serviços de blob storage (não inclui armazenamento *Table* e *Queue*).

(2) **Protocolos Abertos**: exemplos de protocolos abertos são OCCI⁴⁷ (HTTP) e DeltaCloud⁴⁸ e protocolos proprietários seriam Amazon EC2⁴⁹ ou VMware vCloud⁵⁰.

- **OCCI**: é um protocolo para todos os tipos de gerenciamento. Foi originalmente desenvolvido como um solução para o gerenciamento remoto de IaaS.
- **DeltaCloud**: abstrai as diferenças entre diversos provedores e é mais focado em recursos computacionais.

(3) **Padrões**: Armbrust et al. (2009) sugeriram **padronização** como uma possível solução. As principais iniciativas da indústria têm seguido esse caminho e muitos padrões têm sido propostos nos últimos anos (ZHANG; WU; CHEUNG, 2013). Entre as iniciativas mais ativas é possível citar CloudAudit⁵¹, CSA⁵², DMTF⁵³, ETSI⁵⁴, TC CLOUD⁵⁵, OMG⁵⁶, SNIA⁵⁷ e IEEE⁵⁸.

(4) **Camadas de Abstração**: são utilizadas em projetos como RESERVOIR, SLA@SOI e CSAL.

- **RESERVOIR**: desenvolveu tecnologias com foco em camadas de abstração para IaaS. As funções de gerenciamento providas pelo projeto são divididas em diferentes camadas, cada uma tratando de diferentes aspectos da nuvem.
- **SLA@SOI**: um *framework* que trata de *Services Level Agreements* para serviços de nuvem.
- **CSAL**: camada de abstração que provê funcionalidades para *blob*, *queue* e *table storage* em entre múltiplos provedores.

(5) **Repositórios semânticos**: a semântica pode ser aplicada nos níveis de interface, componente e dados por meio da utilização de um modelo genérico de dados semânticos como nas soluções Cloud4SOA e mOSAIC. Além disso, pode ser usada para fazer anotação de

⁴⁶ <<https://java.net/projects/cloudloop>>

⁴⁷ <<http://occi-wg.org/>>

⁴⁸ <<https://deltacloud.apache.org/>>

⁴⁹ <<http://aws.amazon.com/ec2/>>

⁵⁰ <<http://www.vmware.com/products/vcloud-suite>>

⁵¹ <<http://cloudataudit.org/CloudAudit/Home.html>>

⁵² <<https://cloudsecurityalliance.org/>>

⁵³ <<http://dmf.org/standards/cloud>>

⁵⁴ <<http://csc.etsi.org/website/home.aspx>>

⁵⁵ <<http://cloudtc.com/>>

⁵⁶ <<http://www.omg.org/>>

⁵⁷ <<http://www.snia.org/cloud>>

⁵⁸ <<https://standards.ieee.org/develop/project/2301.html>>

serviços e aplicações implantadas pelo usuário na nuvem (PETCU, 2011; AMATO *et al.*, 2013; RANABAHU; SHETH, 2010; LOUTAS; KAMATERI; TARABANIS, 2011).

(6) Linguagens específicas de domínio: permitem criar aplicações independentes de plataforma de nuvem. Silva *et al.* (2013) apresentam uma abordagem baseada em DSL para desenvolvimento de sistemas para nuvem e resolução do *Lock-in*. A ideia principal desse trabalho é que aplicações sejam desenvolvidas independente de plataforma e, por meio de transformações, o código seja traduzido para plataformas específicas. Basicamente, trata-se da implementação da ideia apresentada na Figura 13. O foco da abordagem é o modelo de PaaS. Ranabahu e Sheth (2010) apresentam uma abordagem top-down baseado no particionamento semântico da aplicação. O objetivo da proposta, assim como no trabalho de Silva, Fortes e Lucrédio (2013), é possibilitar o desenvolvimento de sistemas em alto nível de abstração e possibilitar portabilidade. A ideia é permitir que as aplicações sejam modeladas de acordo com a separação em camadas adotando-se na arquitetura *Model-View-Controller* (MVC). Todavia, ambas abordagens são focadas em portabilidade de aplicações e não no conceito de interoperabilidade conforme definido por (NOGUEIRA *et al.*, 2016b).

2.6 Considerações finais

Este capítulo apresentou conceitos relacionados à Computação em Nuvem. Foram destacados, entre outros conceitos, como os modelos de serviços de nuvem podem ser utilizados no desenvolvimento de sistemas, portabilidade, interoperabilidade e os requisitos de uma solução interoperável.

O principal ponto de destaque em relação a este trabalho é a questão da interoperabilidade na Computação em Nuvem. Como pôde ser observado, não foi encontrada uma definição padrão para interoperabilidade no contexto da nuvem a partir de um ponto de vista único. Existem diversas definições que podem até ser conflitantes entre si e uma definição sob a perspectiva de Engenharia de Software foi apresentada. Em uma perspectiva de desenvolvimento de sistemas, as soluções de múltiplas nuvens existem e podem trazer diversos benefícios em diversos cenários de aplicação. Foram apresentados os principais conceitos relacionados a aplicações de múltiplas nuvens e os requisitos para soluções correlatas. Petcu (2014) destaca que algumas das soluções para *multi-cloud* cumprem parte dos requisitos. Todavia, nenhuma delas satisfaz a todos. Uma análise sobre quais soluções cumprem cada requisito está fora do escopo deste trabalho. Diretrizes para pesquisas futuras relacionadas ao tema são apresentadas em (NOGUEIRA *et al.*, 2016b).

Conforme mencionado anteriormente, o foco deste trabalho é no modelo de PaaS, não foi encontrada uma abordagem que considera aplicações existentes e vai em sentido *bottom-up* na direção de uma abordagem de alto nível que abstrai as especificidades das plataformas e trata a combinação das mesmas em nível de modelos. Uma solução nesse sentido configura-se uma

contribuição importante para a comunidade e consiste em uma estratégia não tem por base o uso de padrões. Em direção a esse objetivo, ou seja, a uma abordagem dirigida por modelos para desenvolvimento de aplicações *multi-PaaS*, portanto, faz-se necessário o desenvolvimento de aplicações de referência considerando cada plataforma envolvida na combinação, a posterior combinação de serviços para formar uma aplicação global que agrega recursos e tira vantagens de cada provedor. Logo após, deve ser feita a abstração utilizando elementos de MDE; construção de DSLs, modelos e geradores.

Este trabalho considera que a padronização é um caminho possível e muitas iniciativas nesse sentido devem ser estudadas e melhor exploradas. Todavia, no momento, a padronização pode não ser uma boa ideia principalmente para empresas que ainda estão adotando o modelo de nuvem ou que não exploraram padrões. Os padrões ainda estão se desenvolvendo e mesmo a seleção de um padrão pode levar ao problema do *lock-in* de padrões. Portanto, como um caminho que não envolve a padronização de serviços se faz necessário, as colocações da empresa Microsoft apresentadas na seção anterior devem ser fortemente consideradas. Para a empresa, novas plataformas para dar suporte à interoperabilidade devem ser construídas sobre *Web Services* e abordagens baseadas em *REST* (que consistem propriamente em padrões para indústria) que podem ser facilmente descobertos e consumidos por outras plataformas (Microsoft, 2017). No capítulo seguinte serão apresentados os conceitos de desenvolvimento dirigido por modelos.

DESENVOLVIMENTO DIRIGIDO POR MODELOS

O [Capítulo 2](#) apresentou os conceitos relacionados à Computação em Nuvem. Conforme mencionado anteriormente, abordagens MDE propõem que a modelagem e transformação de modelos em artefatos de implementação ofereçam vantagens para o desenvolvimento de software. Com MDE, os desenvolvedores podem trabalhar em alto nível de abstração em modelos conceituais que melhor captam a essência da aplicação, livrando-se assim de tarefas repetitivas de codificação e das especificidades de plataforma. Portanto, as abordagens MDE são focadas em modelos de domínio a partir dos quais aplicações são total ou parcialmente geradas. Para construir esses modelos e os metaprogramas relacionados a abordagem desenvolvida, faz-se necessário construir linguagens, desenvolver transformações e demais artefatos MDE.

Assim, este capítulo apresenta os principais conceitos sobre o desenvolvimento dirigido por modelos (MDE). A [Seção 3.1](#) apresenta os conceitos fundamentais; a [Seção 3.2](#) apresenta vantagens e desvantagens do uso do MDE; a [Seção 3.4](#) apresenta a arquitetura de metamodelagem; a [Seção 3.5](#) apresenta abordagens MDE para indústria e, por fim, a [Seção 3.6](#) apresenta os conceitos de portabilidade e interoperabilidade especificamente no contexto de MDE.

3.1 Conceitos de MDE

O conceito de abstração consiste na capacidade de encontrar algo comum a partir da observação de algo e gerar uma representação mental da realidade. Essa representação pode, dentre outras coisas, generalizar características específicas de objetos reais, classificar objetos em conjuntos coerentes e concisos e agregar/compor objetos básicos em outros mais complexos ([BRAMBILLA; CABOT; WIMMER, 2012](#)). A abstração é aplicada em diversos contextos tanto da ciência quanto da indústria, sendo, normalmente referida como modelagem. Na prática, a abstração se concretiza por meio de maquetes nas áreas de Engenharia Civil e Arquitetura,

protótipos de peças industriais moldados em madeira nas áreas relacionadas à Engenharia Mecânica, modelos matemáticos utilizados nas engenharias e em estatística, entre outros. Na Física e Química, por exemplo, diversos modelos como o modelo atômico de Thompson, Rutherford e Bohr foram utilizados para facilitar o estudo e compreensão da estrutura da matéria (BRAMBILLA; CABOT; WIMMER, 2012).

No contexto da Engenharia de Software, a abstração e modelagem são conceitos fundamentais, tendo sido aplicados nos mais diversos contextos. Em um contexto mais geral da Engenharia de Software, normalmente, a modelagem e os modelos gerados são utilizados como referência para o desenvolvimento de software. As abordagens dirigidas por modelos propõem que a modelagem e mecanismos de transformação utilizados para gerar código a partir de modelos são uma melhor maneira de desenvolver sistemas de software, ao invés da codificação pura (BRAMBILLA; CABOT; WIMMER, 2012). Assim, os modelos ao invés de serem utilizados apenas como uma referência para a construção do software, passam a ser o centro do processo (por isso o termo *model-driven*). Isso se dá porque apesar da evolução das técnicas e ferramentas de desenvolvimento de sistemas, preocupações com compreensão do problema, reúso, produtividade, manutenção, documentação, validação, otimização, portabilidade e interoperabilidade ainda são recorrentes no desenvolvimento de software. O conceito de Desenvolvimento Dirigido por Modelos surgiu com o objetivo de ajudar a resolver esses problemas (KLEPPE; JOS; WIM, 2003).

A ideia central é que o desenvolvedor não precise interagir manualmente com todo o código-fonte. Pode concentrar-se em modelos, o que o protege das complexidades inerentes às diferentes plataformas (FRANCE; RUMPE, 2007) e das tecnologias abstraídas, utilizando-se do ferramental relacionado ao Desenvolvimento Dirigido por Modelos. Metaprogramas são empregados para gerar automaticamente código e outros artefatos de implementação a partir dos modelos, o que reduz os esforços dos desenvolvedores (BITTAR *et al.*, 2009; KLEPPE; JOS; WIM, 2003) quando assumem o papel de usuários dos elementos de um processo de desenvolvimento de software que considera o desenvolvimento dirigido por modelos (CIRILO, 2011; BRAMBILLA; CABOT; WIMMER, 2012).

Taxonomias foram propostas para o desenvolvimento dirigido por modelos, termos como MDA (*Model-Driven Architecture*, *Model-Driven Development* (MDD), *Model-Driven Engineering* (MDE), *Model-Based Engineerig* (MBE), MD*, entre outros, podem ser encontrados na literatura. A Figura 9 apresenta como esses termos estão relacionados segundo Brambilla, Cabot e Wimmer (2012).

- MBE (*Model-Based Engineering*) se refere a um processo no qual os modelos de software apresentam importantes papéis, embora eles não são necessariamente artefatos-chave do desenvolvimento.
- MDE (*Model-Driven Engineering*) é um superconjunto de MDD, pois como o “E” em

Figura 9 – Taxonomia MD*

Fonte: [Brambilla, Cabot e Wimmer \(2012\)](#)

MDE sugere, MDE vai além de atividades de desenvolvimento isoladas e engloba outras tarefas baseadas em modelos de um completo processo de Engenharia de Software.

- MDD (*Model-Driven Development*) é um paradigma de desenvolvimento que utiliza modelos como principal artefato no processo de desenvolvimento. Em MDD, geralmente, a implementação é (semi)automática, ou seja, gerada a partir dos modelos. O foco, portanto, em MDD é dado nas atividades de implementação, sobretudo, na redução de tarefas repetitivas de programação.
- MDA (*Model-Driven Architecture*) é uma visão particular do MDD, proposta pelo *Object Management Group* (OMG), e assim dependente de alguns padrões criados pelo grupo. Portanto, MDA é um subconjunto do MDD, no qual a modelagem e linguagens de transformações são padronizadas pelo OMG ([ASADI; RAVAKHAH; RAMSIN, 2008](#); [OMG, 2003](#)).

Os conceitos de MBE e MDD, da maneira como foram definidos pelos autores, podem dar uma impressão de que são equivalentes. No entanto, no contexto em que os termos estão inseridos é necessário destacar a diferença entre “*based*” em modelos e “*driven*” por modelos. O MBE utiliza modelos como um dos elementos que compõem a abordagem no que se refere à modelagem de domínio, nesse caso os modelos não são os principais elementos da estratégia. Por outro lado, no MDD a estratégia é dirigida por modelos, o que significa que modelos são utilizados como artefatos principais do processo de desenvolvimento. Por isso, considerando o nível de abrangência dos elementos de modelagem, um conceito é subconjunto do outro.

No entanto, em sua essência, todas essas classificações referem-se ao mesmo conceito/-modelo de desenvolvimento de software, cuja ideia principal é reconhecer a importância dos modelos no processos de software, utilizá-los não apenas como um “guia” para tarefas de desenvolvimento, mas como parte integrante do software assim como o código-fonte (LUCRÉDIO, 2009). Neste trabalho, para representar o desenvolvimento dirigido por modelos será utilizada a sigla MDE.

No contexto de MDE, um modelo é uma descrição ou especificação abstrata do sistema, sendo geralmente representado como uma combinação de elementos gráficos e textuais (OMG, 2003). Na prática, os elementos da abordagem no que se refere aos elementos textuais compõem uma DSL¹ e os elementos da abordagem relacionados aos modelos textuais compõem uma *Domain-Specific Modeling Language* (DSML) (BRAMBILLA; CABOT; WIMMER, 2012). No desenvolvimento de sistemas de maneira tradicional, os modelos, normalmente, são originados nas fases de projeto. Geralmente, nas fases iniciais do ciclo de vida do software, eles servem para facilitar a análise de problemas.

Todavia, à medida que o desenvolvimento avança, os modelos acabam ficando inconsistentes e perdem seu valor; devido ao fato de que, muitas vezes, mudanças são feitas diretamente no código-fonte, considerando geralmente as restrições de tempo dos desenvolvedores. Além disso, criar e manter a documentação atualizada são normalmente tarefas manuais não muito apreciadas por desenvolvedores porque essas tarefas de são vistas como repetitivas, demoradas, burocráticas e pouco criativas. Dessa maneira, os modelos rapidamente tornam-se incapazes de representar a realidade do sistema, o que faz com que tempo e esforços na construção desses artefatos não sejam diretamente aproveitados na produção do software (BITTAR *et al.*, 2009; LUCRÉDIO, 2009).

O foco nos modelos visa simplificar o processo de desenvolvimento, visto que o uso apenas de tecnologias centradas em código-fonte requer um esforço maior para construir o software (FRANCE; RUMPE, 2007; KLEPPE; JOS; WIM, 2003). A linguagem do código-fonte é demasiadamente densa e codificada, de forma que às vezes é difícil reutilizar o conhecimento expresso nos algoritmos. Além disso, esse conhecimento está impregnado por trechos altamente associados a detalhes de plataforma, de modo que a reutilização de uma determinada lógica de negócio em uma plataforma ou linguagem diferente requer um trabalho cuidadoso de reengenharia. Os modelos, no entanto, são formas mais intuitivas de representação do conhecimento, menos dependentes do código-fonte, de forma que podem reutilizados facilmente em projetos semelhantes (LUCRÉDIO, 2009).

¹ DSL é uma linguagem projetada para ser útil para um conjunto específico de tarefas em um domínio específico (DEURSEN; KLINT; VISSER, 2000)

3.2 Vantagens e desvantagens do MDE

As principais vantagens da abordagem MDE estão relacionadas aos problemas recorrentes no desenvolvimento de software que as mesmas visam resolver. A seguir são elencadas algumas dessas vantagens (LUCRÉDIO, 2009):

- **Produtividade:** supostamente, o MDE provê maior produtividade devido a automatização da geração de código a partir de modelos por meio do uso de ferramentas de transformação. Tarefas repetitivas de codificação são implementadas nas transformações, poupando tempo e esforço que podem ser despendidos em tarefas mais importantes.
- **Portabilidade:** a partir de um mesmo modelo pode-se gerar código para diferentes plataformas.
- **Interoperabilidade:** cada parte do modelo pode ser transformado em código para uma plataforma diferente, o que resulta em um software que pode ser executado em um ambiente heterogêneo, mas que mantém sua funcionalidade global.
- **Manutenção e documentação:** alterações são realizadas diretamente nos modelos, mantendo-os consistentes com o código-fonte, o qual é gerado automaticamente a partir de transformações aplicadas nesses modelos.
- **Comunicação:** uma vez que os modelos são mais abstratos que o código-fonte, o que não exige conhecimento técnico associado à plataforma de implementação para sua compreensão, os especialistas do domínio podem utilizar diretamente os modelos para identificar mais facilmente as questões associadas ao negócio; Os especialistas em tecnologia da informação podem identificar os elementos técnicos usando os mesmos modelos.
- **Reúso:** o reúso é feito em nível de modelos ao invés de em nível de código-fonte. O código pode ser automaticamente regenerado para novos contextos através de ferramentas de transformação apropriadas. A documentação permanece atualizada, o que facilita as tarefas de manutenção.
- **Verificações e Otimizações:** os modelos facilitam a análise por verificadores semânticos, conforme a sintaxe de seu metamodelo, e a execução de otimizações automáticas. É possível minimizar a ocorrência de erros semânticos, o que fornece implementações mais eficientes.
- **Correção:** ferramentas de transformação evitam a introdução de erros acidentais, tais como erros de digitação e de sintaxe. Erros conceituais podem ser identificados em um nível mais alto de abstração.
- **Alto nível de abstração quando necessário:** as DSLs e DSMLs são focadas em um domínio particular e consistem de elementos que possibilitam a modelagem do problema

em alto nível de abstração. Assim, o problema pode ser modelado de maneira mais simples com uma linguagem específica para seu domínio.

Todavia, a MDE não oferece somente benefícios. A seguir são apresentadas algumas desvantagens dessa abordagem (LUCRÉDIO, 2009):

- **Rigidez:** o MDE pode ser adotado de acordo com diferentes estratégias. Em uma estratégia híbrida (ou seja uso de código e modelos em conjunto) por exemplo, grande parte do código gerado fica longe do alcance do desenvolvedor. A identificação do que é gerado e do que não é pode ser dificultada, portanto, dificultando modificações e tornando o código gerado mais rígido
- **Complexidade:** os artefatos necessários para uma abordagem baseada em modelos, como por exemplo, ferramentas de modelagem, transformações e geradores de código, introduzem complexidade adicional ao processo, pois tratam-se de artefatos inerentemente mais difíceis de construir e manter
- **Desempenho:** apesar de algumas otimizações poderem ser realizadas em nível mais alto de abstração, a regra geral é que geradores acabam incluindo muito código desnecessário e, portanto, o resultado pode não apresentar desempenho ótimo, quando comparado com código escrito à mão
- **Curva de aprendizado:** o desenvolvimento dos artefatos específicos do MDE exige profissionais com habilidades na construção de linguagens, ferramentas de modelagem, transformações e geradores de código. O aprendizado dessas técnicas, apesar de não ser extremamente difícil, requer um treinamento dedicado; MDE exige profissionais bastante habilidosos nessa abordagem. No entanto, uma vez desenvolvidos os elementos a complexidade diminui
- **Alto investimento inicial:** assim como a reutilização de software, a MDE depende de maior investimento inicial, uma vez que a construção de uma infraestrutura de reutilização dirigida por modelos requer mais tempo e esforço
- **Complexidade de testes de transformações:** conforme mencionado anteriormente, os metaprogramas implementados em uma abordagem MDE são inerentemente mais complexos. Consequentemente, o teste e depuração desses artefatos acabam se tornando tarefas mais complexas também
- **Modelo de programação:** O modelo de programação dirigido por modelos, para muitos desenvolvedores, constitui uma desvantagem apenas por ser focado em modelos. Isso acontece porque muitos desenvolvedores preferem o código a modelos

Steimann e Kühne (2005) têm uma posição bastante crítica a respeito de MDE:

“Os modelos têm seu grande valor para pessoas que não sabem programar. Todavia, isso acontece, não porque modelos são mais expressivos que programas (no sentido em que eles são capazes de expressar coisas complexas de maneira simples), mas porque eles simplificam demais. Para criar um programa usável (ou seja, um programa que efetivamente realiza o que o usuário precisa) a partir dos modelos, o modelo deve ser tão complexo que a pessoa que não programa não poderá entendê-lo e uma pessoa que programa preferiria escrever o software em código do que desenhar o modelo correspondente. Muito embora a simplificação por meio dos modelos possa, muitas vezes, ser útil, a mesma certamente não será suficiente para criar um programa final satisfatório.”

Uma situação recorrente relacionada ao que foi dito por [Steimann e Kühne \(2005\)](#) acontece com projetistas de banco de dados. As tecnologias que permitem uma modelagem completa de um sistema de banco de dados já estão evoluídas o suficiente e possibilitam que o desenvolvedor trabalhe somente em nível de modelos para projetar um banco de dados. No entanto, muitos desenvolvedores preferem trabalhar em mais baixo nível, com *data definition language* (DDL).

Para [Brambilla, Cabot e Wimmer \(2012\)](#), no entanto, a simplificação provida pelos modelos consiste em uma vantagem, não uma desvantagem. Os autores argumentam que as técnicas de MDE atuais já são abrangentes e desenvolvidas o suficiente para prover um grande conjunto de ferramentas eficientes que podem ser utilizadas desde a concepção do software até sua entrega. Um bom exemplo disso seria aplicações implementadas utilizando o padrão *Interaction Flow Modeling Language* (IFML)². O IFML, recentemente adotado pela OMG como um padrão, é projetado para expressar o conteúdo, interação com o usuário e controle do comportamento do *front-end* das aplicações.

Uma outra limitação das abordagens MDE refere-se à necessidade de adaptações e/ou modificações no código gerado. Se for necessária uma mudança diretamente no código gerado, o ciclo de vida da abordagem MDE pode ser quebrado. Uma eventual mudança precisa ser refletida tanto no código gerado quanto nas transformações e nos modelos originários de maneira automática. Isso não é uma tarefa trivial. Normalmente, recomenda-se deixar o código gerado sem modificar e todas as alterações necessárias devem ser realizadas em nível de modelos.

Para visualizar esse problema mais facilmente basta fazer um paralelo com os conceitos de derivada e integral da matemática. Assim sendo, o código gerado seria uma derivada dos modelos e as modificações diretamente no código gerado seriam refletidas nos modelos por meio de uma integral. Esse processo (de integral) constitui uma área de pesquisa ainda em aberto da área de MDE. As pesquisas relacionadas ao tema são normalmente referenciadas como *round-trip engineering* ([ANTKIEWICZ; CZARNECKI, 2006](#); [HETTEL; LAWLEY; RAYMOND, 2008](#)).

[Possatto e Lucrédio \(2015\)](#) apresentam uma estratégia para migração automática de

² <http://www.ifml.org/>

código no contexto de *round-trip engineering* para manter transformações e implementações de referência atualizados. Todavia, trata-se ainda de um protótipo acadêmico ainda não pronto para utilização na indústria ou considerado por ferramentas ligadas ao *Eclipse Modeling Project* (EMP) por exemplo.

3.3 Elementos de MDE

Para que abordagem MDE ocorra de fato, são necessários determinados elementos. A [Figura 10](#) apresenta os principais elementos do desenvolvimento dirigido por modelos e como eles estão presentes no processo de desenvolvimento.

Figura 10 – Principais elementos de MDE

Fonte: [Lucrédio \(2009\)](#)

- **Ferramenta de Modelagem:** utilizada para produzir modelos que descrevem conceitos do domínio. Os modelos criados por essa ferramenta precisam seguir regras de semântica, uma vez que serão interpretados pelo computador. Normalmente é utilizada uma DSL ([LUCRÉDIO, 2009](#)).
- **Ferramenta para definir transformações:** são utilizadas para gerar outros modelos e código-fonte a partir de modelos recebidos como entrada. Através delas são construídas as regras de mapeamento, ou seja, como os modelos serão transformados em outros tipos de modelo ou código ([LUCRÉDIO, 2009](#)).
- **Modelos:** servem de entrada para transformações e representam o conhecimento do domínio da aplicação ([LUCRÉDIO, 2009](#)).
- **Transformações:** são metaprogramas que utilizam os modelos de domínio como entrada e são utilizadas para efetivamente gerar o código alvo.

- **Mecanismos para executar transformações:** executa as transformações seguindo as regras de mapeamento de modelo para modelo ou de modelo para código. Mantém informações de rastreabilidade, possibilitando muitas vezes saber a origem de cada elemento gerado, seja modelo ou código-fonte (LUCRÉDIO, 2009).
- **Outros Modelos e código-fonte:** são resultantes do processo de transformação (LUCRÉDIO, 2009).

Os elementos apresentados, na prática, são utilizados pelos desenvolvedores para criar DSLs, DSMLs, transformações, validadores, entre outros elementos necessários. A Figura 11 mostra um detalhamento ainda maior do que o observado na Figura 10 apresentando os elementos MDE aplicado ao modelo de nuvem (SILVA; FORTES; LUCRÉDIO, 2013). O conhecimento extraído do domínio é utilizado para criar um metamodelo de domínio (ponto 2); o metamodelo possibilita a construção das DSLs e DSMLs (ponto 3) que servirão para a criação dos modelos de domínio (ponto 4). Os modelos de domínio por sua vez servem como entrada para os geradores (ponto 5) que irão utilizar os metaprogramas/templates para gerar o código final e/ou outros modelos/artefatos (pontos 6, 7, 8).

Figura 11 – Uso de elementos MDE

Fonte: Silva *et al.* (2013)

Um ponto de destaque em relação à construção de uma abordagem MDE conforme

apresentado na [Figura 11](#) é como as linguagens podem ser criadas. Duas abordagens podem ser consideradas:

- (i) **bottom-up** - consiste na construção dos elementos da abordagem considerando primeiramente os elementos relacionados ao que será gerado para em seguida criar os demais elementos da abordagem. No contexto de uma plataforma, por exemplo, esse caminho parte dos elementos específicos de plataforma até serem construídos as DSLs (caminho 1, 6-7-8, 5, 4-3-2 na [Figura 11](#)).
- (ii) **top-down** - consiste na construção dos elementos da abordagem considerando primeiramente o domínio para que logo em seguida sejam construídos os elementos da abordagem (caminho 2-3-4, 5, 6-7-8, 1 na [Figura 11](#)).

3.4 Metamodelagem

As principais abordagens MDE fundamentam-se no conceito de metamodelagem. A metamodelagem suporta diferentes linguagens de programação e ajuda a garantir que os modelos construídos estejam completos e semanticamente corretos. Além disso, possibilita a definição e execução das transformações ([LUCRÉDIO, 2009](#)). Um metamodelo consiste em um modelo que fornece elementos e regras para se construir outro modelo. A diferença entre modelagem e metamodelagem é subjetiva. Apesar de ambos serem modelos, um é especificado utilizando-se o outro, ou seja, um modelo deve estar em conformidade com o metamodelo ([GRONBACK, 2009](#)) que consequentemente deve estar em conformidade com um metametamodelo. A [Figura 12](#) apresenta a arquitetura de metamodelagem comumente utilizada pela comunidade de desenvolvimento.

- **Nível M3:** a camada metametamodelo é utilizada para definir linguagens de modelagem (como a UML). Normalmente, o meta-metamodelo consiste em uma instância de si próprio. Existem alguns meta-metamodelos utilizados na indústria, adotados por diferentes ferramentas de modelagem e transformação. Na [Figura 12](#), é citado o MOF, um padrão estabelecido pelo *OMG (Object Management Group)* ([OMG, 2003](#)).
- **Nível M2:** a camada metamodelo (modelo para definir modelos) é utilizada para construção de modelos. A especificação da UML é um exemplo de metamodelo que define os elementos para criação de seus diagramas.
- **Nível M1:** a camada de metadados (dados sobre dados) ou modelo são os modelos propriamente ditos. É nessa camada que são descritos os conceitos do domínio.
- **Nível M0:** corresponde aos dados que instanciam os conceitos do domínio.

Figura 12 – Arquitetura de metamodelagem

Fonte: adaptado de Brambilla, Cabot e Wimmer (2012)

3.5 Abordagens da indústria para MDE

Há na indústria diversas abordagens para o desenvolvimento dirigido por modelos, entre elas está a *Model-Driven Architecture* (MDA). A MDA é iniciativa de MDE do OMG (OMG, 2003). O metamodelo utilizado nesta abordagem é *Meta Object Facility* (MOF).

Outra abordagem importante é o conjunto de ferramentas do *Eclipse Modeling Project*, destacando-se o *Eclipse Modeling Framework* (EMF). O EMF permite a manipulação de modelos segundo seu correspondente metamodelo, seguindo um meta-metamodelo denominado Ecore, ao invés do MOF. O Ecore surgiu como uma implementação do MOF, mas evoluiu para uma forma mais eficiente, a partir da experiência obtida após alguns anos na construção de ferramentas. O EMF não segue fielmente o padrão *Java Metadata Interface* (JMI)³, mas utiliza um subconjunto de mapeamentos de metamodelo para Java otimizados para manipulação em memória, o que o torna mais eficiente e mais simples de ser utilizado. Todavia, ele é menos abrangente do que o conjunto mais completo formado pela combinação JMI/MOF (MOORE et al., 2004).

Um projeto relacionado ao EMF que merece bastante atenção é o *Graphical Modeling Framework* (GMF)⁴. Esse projeto permite a definição completa de um ambiente de modelagem

³ <<https://jcp.org/en/jsr/detail?id=40>>

⁴ <<http://www.eclipse.org/modeling/gmp/>>

para uma linguagem visual específica para um determinado domínio (DSML). A partir da definição do metamodelo da linguagem, da aparência gráfica dos elementos visuais dessa linguagem (notação), e das ferramentas necessárias para a criação dos elementos, é gerado um ambiente completo, que permite que o engenheiro de software construa modelos segundo a linguagem e notação pré-definidos. Há também os demais projetos do *Eclipse Modeling Project*, incluindo algumas ferramentas que apóiam o MDE e o desenvolvimento de DSLs, que podem ser integradas ao ambiente de desenvolvimento Eclipse. Um exemplo é Xtext⁵, que é uma ferramenta para criação de DSLs textuais e ambientes de desenvolvimento baseados na ferramenta Eclipse. Pode-se também citar as ferramentas M2T (*model-to-text* - modelo-para texto) para transformação e geração de código, das quais se destacam o Acceleo⁶ e Xtend⁷.

O EMP é um projeto robusto, muitas ferramentas estão disponíveis e podem ser utilizadas de maneira gratuita para o desenvolvimento de DSLs e DSMLs. Essas ferramentas são facilmente integradas à *Integrated Development Environment (IDE) Eclipse⁸ modeling*. Além disso, os *plugins* para diversas plataformas de Computação em Nuvem já estão evoluídos para essa IDE, portanto, a adoção dessa IDE quando se utiliza o EMP é natural.

3.6 Portabilidade e interoperabilidade no contexto MDE

A indústria de software é extremamente dinâmica, novas tecnologias e plataformas surgem frequentemente, oferecendo vantagens que forçam as organizações a se adaptarem rapidamente para não ficar desatualizadas em relação aos principais concorrentes. A possibilidade de ficar aprisionado a um determinado provedor e consequentemente ser cobrado com taxas abusivas para sua utilização coloca o usuário da nuvem em uma posição difícil para escolher quais ofertas são mais viáveis aos seus requisitos de negócio. Além disso, pelo fato da Computação em Nuvem ser uma tecnologia em desenvolvimento, é possível que plataformas e bibliotecas fiquem rapidamente obsoletas, sendo necessário um grande trabalho de manutenção nos sistemas que as utilizam ou ainda um processo de reengenharia para adaptar os sistemas a novas demandas.

A portabilidade, no contexto de MDE, refere-se à geração de código (utilizando-se elementos MDE) para diferentes plataformas a partir de um mesmo modelo (KLEPPE; JOS; WIM, 2003). A Figura 13 ilustra uma possível utilização da MDE, em sua proposta original, para o desenvolvimento de aplicações portáteis entre plataformas de Computação em Nuvem.

Por meio de um processo MDE são geradas aplicações para diferentes plataformas de Computação em Nuvem. Silva, Fortes e Lucrédio (2013) apresentam o protótipo de uma DSL que implementa a abordagem apresentada na Figura 13. Como há várias ofertas de serviço de nuvem, os autores construíram diversos mecanismos que os exploram. Isto é, a partir de

⁵ <<http://www.eclipse.org/Xtext/>>

⁶ <<http://www.eclipse.org/acceleo/>>

⁷ <<http://www.eclipse.org/xtend/>>

⁸ <www.eclipse.org>

Figura 13 – MDE para geração de aplicações portáveis

Fonte: autoria própria

um mesmo modelo genérico foi possível gerar sistemas (ou parte deles) específicos para um ou mais PaaS. O que garante a portabilidade entre plataformas é fato de gerar uma aplicação para uma plataforma A e caso seja necessária uma mudança de plataforma ou tipo de oferta de serviço, a partir do mesmo modelo utilizado no primeiro momento gera-se uma aplicação para uma plataforma B. É importante ressaltar que os detalhes específicos de plataforma podem ser especificados direto nas transformações/processo ou como modelos extras.

Propostas para os problemas de portabilidade são apresentadas na literatura ([SILVA; FORTES; LUCRÉDIO, 2013](#); [SILVA et al., 2015](#); [NOGUEIRA et al., 2016a](#); [DELGADO, 2014](#); [LOUTAS; KAMATERI; TARABANIS, 2011](#); [AMATO et al., 2013](#); [PETCU, 2011](#)), o tópico de pesquisa ainda precisa ser melhor investigado. As abordagens encontradas são protótipos acadêmicos, o problema ainda não pode ser considerado como resolvido.

Como extensão do trabalho apresentado por [Silva, Fortes e Lucrédio \(2013\)](#), [Silva et al. \(2015\)](#), [Nogueira et al. \(2016a\)](#) apresentam uma aplicação do protótipo para conciliar diferentes modelos de dados no contexto da Computação em Nuvem. A abordagem consiste em uso da modelagem de domínio para especificar o modelo de dados que consequentemente será traduzido pelo modelo aceito por cada um dos provedores envolvidos.

A vantagem fundamental da abordagem dirigida por modelos para portabilidade consiste no encapsulamento (nas transformações) dos detalhes específicos de cada plataforma. Isso faz com que o desenvolvedor não gaste esforços de trabalho repetitivo de programação e seja obrigado a ter conhecimento sobre cada plataforma. A portabilidade pode ser alcançada também

por meio da transformação de modelos conforme sugerido por [Brambilla, Cabot e Wimmer \(2012\)](#). Todavia, a transformação de modelos, na prática, pode ser ainda mais complexa do que os processos modelo-para-plataforma normalmente utilizados nas abordagens MDE conforme proposto por [Silva, Fortes e Lucrédio \(2013\)](#).

No contexto de MDE, interoperabilidade consiste em fazer com que uma mesma aplicação execute sobre uma combinação heterogênea de plataformas e provedores, mantendo uma funcionalidade global consistente ([FRANCE; RUMPE, 2007](#)). Sob a perspectiva de modelagem, a ideia é que a modelagem da aplicação seja realizada considerando apenas seu domínio e que cada parte dessa modelagem possa ser transformada em código para uma plataforma diferente. O software resultante deverá executar em um ambiente heterogêneo, mas manter sua funcionalidade global ([KLEPPE; JOS; WIM, 2003](#)). A [Figura 14](#) ilustra um dos cenários de aplicação e sua relação entre o MDE e o desenvolvimento de aplicações interoperáveis entre plataformas de Computação em Nuvem.

Assim como na abordagem para portabilidade, os detalhes de plataforma podem ficar encapsulados nas transformações ou serem representados através de outros modelos de alto nível e podem servir de entrada para os transformadores. Deve haver também uma preocupação com a comunicação, ou seja, as unidades de software que executam em diferentes plataformas devem ser capazes de estabelecer uma comunicação e colaborar entre si.

Figura 14 – MDE para geração de aplicações interoperáveis

Fonte: autoria própria

Apesar de serem conceitos distintos, portabilidade e interoperabilidade compartilham da mesma preocupação com a independência de plataforma. Nesse sentido, pode-se argumentar que uma abordagem para portabilidade de aplicações é um passo inicial para resolução do problema de interoperabilidade, já que a mesma permite separar os conceitos independentes de plataforma daqueles dependentes de plataforma. A diferença principal é que na portabilidade considera-se a

aplicação completa, e na interoperabilidade deve-se considerar cada componente (ou módulos) de forma independente.

Com relação a esses dois conceitos (portabilidade e interoperabilidade), [Armbrust et al. \(2009\)](#) sugeriram que fosse criado um padrão para desenvolvimento de sistemas para Computação em Nuvem de modo que fosse mais fácil a migração de dados e aplicativos entre as diversas plataformas. Esse é um caminho diferente, que não se baseia em MDE, mas que pode levar a uma solução para os problemas citados.

3.7 Considerações finais

Este capítulo apresentou conceitos relacionados ao desenvolvimento dirigido por modelos. No que se refere especificamente a esta tese, o que se destaca é o conceito de interoperabilidade no contexto de desenvolvimento dirigido por modelos que é definido por [France e Rumpe \(2007\)](#) por uma mesma aplicação que é executada sobre uma combinação heterogênea de plataformas e provedores, mantendo uma funcionalidade global consistente ([FRANCE; RUMPE, 2007](#)).

A ideia de aplicar MDE ao problema da interoperabilidade na Computação em Nuvem é relevante não só para apresentar um caminho diferente para construção de aplicações, mas também para contribuir para enriquecer o modelo de nuvem e propagar ainda as tecnologias relacionadas. No contexto exclusivo de PaaS, devido às especificidades de plataforma, faz mais sentido uma abordagem *bottom-up*. Uma abordagem *top-down*, embora seja possível, pode apresentar custo adicional a depender das plataformas selecionadas. Em direção a uma abordagem *bottom-up*, portanto, faz-se necessário desenvolver implementações de referência específicas de plataforma, desenvolver estratégia para combinação de serviços, desenvolver uma DSL e metaprogramas para geração de código. Os próximos capítulos apresentam o desenvolvimento desses elementos.

DESENVOLVIMENTO E MIGRAÇÃO DE IMPLEMENTAÇÕES DE REFERÊNCIA PARA O MODELO PAAS

Este capítulo apresenta um Estudo de Caso conduzido em conjunto com uma empresa da indústria visando criar implementações de referência para compor uma abordagem MDE para desenvolvimento de aplicações *multi-cloud*. Em ciclo de vida MDE as implementações de referência servem como base para desenvolvimento das transformações e consistem em um modelo de código a ser gerado. A [Seção 4.1](#) apresenta uma introdução ao Estudo de Caso e o perfil da empresa parceira; a [Seção 4.2](#) apresenta o planejamento do estudo e; por fim, a [Seção 4.3](#) apresenta a condução do mesmo.

4.1 Contextualização do Estudo de Caso

Um Estudo de Caso é utilizado para examinar, analisar e avaliar projetos, atividades e atribuições (WOHLIN *et al.*, 2000). É a técnica na qual os fatores chave que podem afetar os resultados de uma abordagem proposta são identificados e documentados (YIN, 2008). Os Estudos de Caso visam observar um atributo específico e estabelecer o relacionamento entre atributos diferentes (TRAVASSOS; GUROV; AMARAL, 2002). Uma das vantagens dessa metodologia é que é de fácil planejamento e execução, é um método utilizado em estudos empíricos em várias ciências. Na Engenharia de Software pode, por exemplo, ser utilizado para determinar qual o melhor entre dois métodos (YIN, 2008) ou ainda para pesquisa exploratória sobre um determinado tópico. Para Yin (2008), um Estudo de Caso deve ser aplicado quando existe a necessidade de se responder as questões “por que?” ou “como?”. O estudo apresentado neste capítulo foi conduzido com o objetivo de investigar como desenvolver uma abordagem para solução do problema especificado.

O foco deste trabalho é no modelo de PaaS, não foram encontradas soluções que considerassem aplicações existentes e fosse em sentido *bottom-up* na direção de uma abordagem MDE que abstraísse as especificidades das plataformas visando a interoperabilidade. Uma solução conforme mencionado pode ser classificada como *multi-cloud* e como somente o modelo PaaS foi selecionado, a solução interoperável pode ser classificada como *multi-PaaS*.

A necessidade e importância do desenvolvimento de uma solução desse tipo já foi explorada nos capítulos anteriores. Construir uma abordagem dirigida por modelos para desenvolvimento de aplicações *multi-PaaS* envolve desenvolver aplicações de referência considerando cada plataforma envolvida na combinação; combinar recursos providos pelas plataformas e, o por fim, abstrair os detalhes específicos de plataforma utilizando elementos de MDE. Na prática, a criação da solução consiste especificamente no desenvolvimento de implementações de referência, construção de DSLs, especificação de modelos e desenvolvimento de metaprogramas.

Embora o desenvolvimento de aplicações para nuvem sob perspectiva de Engenharia de Software (inclusive PaaS) já tivesse sido explorado por meio de uma revisão sistemática de literatura conforme apresentado por Silva e Lucrédio (2012), foi necessário validar os resultados apresentados na revisão sistemática e investigar na prática os conceitos relacionados às plataformas de nuvem. Em direção a esse objetivo foi necessário: (i) verificar os conceitos do modelo de nuvem e os problemas encontrados na adoção e desenvolvimento de aplicações para o modelo de serviço PaaS. (ii) criar aplicações para o modelo PaaS visando desenvolver sistemas específicos plataforma. Para isso, foi realizada uma parceria com uma empresa da indústria para condução do um Estudo de Caso apresentado neste capítulo.

A empresa selecionada consiste em uma companhia especializada no desenvolvimento de soluções sob demanda. Possui mais 2.400 colaboradores de 17 nacionalidades diferentes. Tendo atuado em áreas de Energia, Serviços Financeiros, Governo, Telecomunicações e Transportes. A empresa disponibilizou uma equipe de desenvolvimento para participação na condução do estudo. Após diversas reuniões, foi definido um plano que norteou a condução do estudo de caso, conforme descrito a seguir.

4.2 Plano do Estudo de Caso

O objetivo principal desse Estudo de Caso foi **o desenvolvimento de aplicações de referência para o modelo de PaaS da Computação em Nuvem**. Inicialmente, antes de desenvolver as aplicações, três restrições foram definidas para delimitar o escopo da pesquisa:

- **Não adoção de padrões:** embora muitas iniciativas relacionadas a padronização existam¹, padrões ainda estão ganhando maturidade e pode demorar anos até que eles estejam com-

¹ Uma lista que inclui uma boa parte dos diversos padrões sendo desenvolvidos pode ser encontrada no endereço: <<http://cloud-standards.org>>.

pletamente estabelecidos e amplamente adotados pela comunidade de desenvolvimento. Além disso, os provedores de nuvem podem querer implementar sua própria solução sem uso de padrões para evitar riscos e promover a dominação tecnológica de suas soluções na corrida pelo mercado. Padrões podem ainda não estar alinhados com as necessidades estratégicas dos provedores e eles precisam trabalhar com interoperabilidade sobre suas próprias soluções. Portanto, a primeira restrição consiste na não adoção de padrões, visando assim encontrar um caminho alternativo à padronização.

- **Desenvolver uma solução alinhada aos interesses estratégicos da empresa considerando seus requisitos tecnológicos:** uma empresa com um grande número de colaboradores, muitas vezes, hesita fazer mudanças drásticas em suas tecnologias devido a necessidade de treinamento de equipe e demais problemas relacionados à adoção de novas tecnologias. Assim, a aplicação a ser desenvolvida nesse Estudo de Caso deve considerar tecnologias já adotadas pela empresa. No entanto, evita-se mudanças drásticas, sobretudo, no que se refere às linguagens. Além disso, evita-se uma solução puramente teórica e explora-se um problema real da empresa.

Primeiramente é necessário entrevistar os desenvolvedores da empresa para compreender a real necessidade, levantar quais tecnologias utilizadas e identificar quais provedores poderão ser utilizados no futuro. A ideia é que protótipos desenvolvidos possam servir como base para criação de aplicações em situações futuras.

- **A aplicação deve ser projetada para reúso:** como o objetivo final do projeto é a abstração dos detalhes de plataforma visando uma estratégia de interoperabilidade utilizando MDE, a aplicação desenvolvida deve ser projetada para reúso. Devido a heterogeneidade dos serviços de nuvem, é importante que a arquitetura da aplicação seja flexível e técnicas de modularização sejam utilizadas visando facilitar o reúso.

Objetivos específicos do Estudo de Caso

Uma das ideias centrais deste desta tese foi apresentar uma solução para facilitar o desenvolvimento de aplicações *multi-cloud*. Assim sendo, o objetivo inicial foi desenvolver aplicações que tirassem vantagens de vários provedores de nuvem simultaneamente, propondo uma arquitetura ou uma biblioteca que facilitasse o desenvolvimento dessas soluções. Essa aplicação servirá para definição de uma linguagem específica de domínio e de transformações. O Odrive² é um exemplo de aplicação que combina diferentes ofertas de armazenamento de dados numa nova que tira vantagens de diversos outros utilizados como base. Neste contexto, como pretendia-se combinar recursos de diferentes PaaS em uma aplicação única, sugeriu-se o desenvolvimento aplicações (*toy problems*) que utilizassem pelo menos dois PaaS em simultâneo.

² <<http://new.odrive.com/linkallyourstuff>>

Uma das aplicações sugeridas foi um sistema de microblog, na qual os requisitos de multimídia poderiam, por exemplo, ser implementados utilizando a oferta PaaS da Microsoft Azure e demais requisitos fossem implementados utilizando, por exemplo, o Google App Engine, ou qualquer outra PaaS que a empresa parceira considerasse interessante³.

Outra sugestão foi desenvolver uma rede social para desenvolvedores de sistemas, visando aproximar pessoas com diferentes conhecimentos. Na prática, muitas dúvidas sobre desenvolvimento de sistemas ou questões específicas sobre uma determinada tecnologia são publicadas em fóruns. Seria interessante desenvolver uma pequena rede social para desenvolvedores, na qual os mesmos pudessem publicar ofertas de empregos, responder a perguntas, colocar perguntas, etc. Esta rede social, deveria ser implementada em múltiplas-nuvens como no caso anterior. Por fim, de acordo com interesse manifestado pela empresa, outra sugestão seria um sistema de universidade que incluísse cadastro de alunos para efetivação de uma matrícula.

Atividades

As seguintes objetivos e atividades foram definidos:

- **Definição dos requisitos tecnológicos e arquiteturais da aplicação:** selecionar tecnologias em uso pela empresa para que fossem utilizadas no Estudo de Caso e definição da arquitetura de referência da aplicação.
- **Selecionar 3 PaaS e definir tecnologias para que fossem utilizadas no projeto:** selecionar 3 PaaS que fossem alinhadas com o interesse da empresa. Como a ideia era investigar aplicações *multi-cloud*, pelo menos duas plataformas precisavam ser selecionadas. Por exemplo, o *Google App Engine* poderia ser uma delas. A depender dos serviços que possui, além do conhecimento sobre as novas plataformas, esta atividade poderia levar a empresa a economizar na hospedagem e *deployment* de novas/existentes aplicações. As plataformas deveriam ser selecionadas de acordo com os requisitos tecnológicos da empresa.
- **Explorar as plataformas selecionadas:** estudar e compreender as plataformas selecionadas, reconhecendo as características, serviços e modo de funcionamento de cada uma delas, identificando o que é semelhante e o que é variável ou específico. Isso possibilitaria a criação de uma biblioteca de código para cada plataforma (atividade seguinte).
- **Desenvolver biblioteca de código para cada plataforma:** o objetivo foi desenvolver código reutilizável, ou seja, código base a partir do qual os demais membros de uma equipe pudessem desenvolver sistemas. Cada plataforma permite e disponibiliza um conjunto próprio de tecnologias, é necessário conhecê-las e criar um ambiente básico (conjunto de códigos e arquivos de configuração ricamente comentados) de desenvolvimento.

³ (em <http://www.paasify.it/vendors>) haviam várias alternativas possíveis)

- **Desenvolver aplicações “uniplataforma”:** desenvolver aplicações propostas como Estudo de Caso deste trabalho considerando os elementos específicos de cada plataforma. Como cada PaaS tem suas especificidades, deveria ser criada uma aplicação considerando cada PaaS adotada.
- **Desenvolver aplicações distribuídas:** distribuir as aplicações definidas. Planejar a distribuição da aplicação entre as plataformas de acordo com as funcionalidades de cada aplicação, de maneira a diminuir o custo final da fatura de serviços e tirar vantagens das plataformas.
- **Desenvolver módulos distribuídos:** desenvolver estratégia de comunicação entre os módulos, propondo a composição dos vários módulos.
- **Analisar o custo-benefício:** analisar as aplicações uniplataforma e as *multi-cloud*. Definir métricas de desempenho.
- **Definir método de avaliação de custo:** definir um plano de avaliação que métricas e elementos de comparação para as aplicações desenvolvidas.

Antes da efetiva condução do estudo, o planejamento foi revisado pela equipe de desenvolvimento e passou por uma alteração de escopo. A equipe de desenvolvimento decidiu desenvolver uma única aplicação de referência. Além disso, as atividades de avaliação de custo-benefício e definição de método de custo foram removidas por serem difíceis de estimar.

4.3 Condução do estudo

O Estudo de Caso foi realizado seguindo o plano apresentado anteriormente. As seções a seguir apresentam os detalhes da condução do estudo.

4.3.1 Requisitos tecnológicos e arquiteturais da aplicação

Para considerar PaaS em um projeto há basicamente dois caminhos:

- **Da plataforma para aplicação:** (i) selecionar a plataforma de acordo com o conjunto de requisitos da aplicação ou por dominação de mercado e desenvolver a aplicação de acordo com as tecnologias e especificidade de plataforma
- **Da aplicação para plataforma:** (ii) definir o conjunto de tecnologias e requisitos tecnológicos da aplicação, desenvolvê-la e após selecionar a plataforma mais alinhada com a aplicação criada

Os caminhos (i) e (ii) para adoção de PaaS em um projeto têm seus respectivos *trade-offs*; o caminho (i) é mais natural em termos de adaptabilidade da aplicação à plataforma. Ou seja, a aplicação é desenvolvida especificamente para aquela plataforma. Por outro lado, as aplicações desenvolvidas considerando esse caminho geralmente apresentam pouco ou nenhum potencial para reúso (visto que é específica de plataforma), o que provavelmente levará ao *lock-in* (SILVA; LUCRÉDIO, 2012). Ao se considerar o caminho (ii), ou seja, uma aplicação deve ser desenvolvida utilizando todos requisitos tecnológicos previamente definidos e após o desenvolvimento da aplicação deve ser escolhido o PaaS, a aplicação pode não se adaptar a qualquer provedor. Uma análise cuidadosa do provedor o qual será implantada a aplicação se faz necessária. Em muitos casos, o provedor não suportará a aplicação e caso a empresa escolha um provedor por dominação de mercado (ou seja, a empresa escolhe o provedor porque a comunidade está utilizando bastante), a aplicação precisará passar por um processo de reengenharia para adaptar a aplicação ao provedor (o que levará ao *lock-in*). Assim, o custo final da aplicação pode aumentar (SILVA; LUCRÉDIO, 2012).

Como a empresa parceira já adota um conjunto de tecnologias em seus projetos, o caminho (ii) foi escolhido, visto que, a princípio, a adoção de uma tecnologia nova pode impactar as tarefas dos seus colaboradores e exigir treinamento específico para cada plataforma. Em uma reunião de *brainstorming* foi decidido que, naquele primeiro momento, seria desenvolvida uma aplicação básica considerando os requisitos tecnológicos definidos pela empresa e, logo após, seriam escolhidas plataformas que suportaria as tecnologias utilizadas na implementação. Assim, para o *front-end* da aplicação, além das tecnologias tradicionais como *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e *JavaScript*, foram selecionadas as tecnologias *AngularJS*⁴ e *Twitter Bootstrap*⁵; e para o *back-end* *Spring MVC*⁶, *Java Persistence API* (JPA), *Web Services REST* que aceitam somente os dados em formato *JavaScript Object Notation* (JSON) e, por fim, o sistema de banco de dados não relacional *MongoDB*⁷. Também foram consideradas as tecnologias de apoio *NodeJS*⁸, *Bower*⁹, *Github*¹⁰ e *Maven*¹¹.

Considerando que uma aplicação *multi-cloud* é composta pela combinação de recursos heterogêneos das plataformas selecionadas, a arquitetura da aplicação precisa ser projetada levando em consideração a modularização. Portanto, uma arquitetura para esse tipo de aplicação deve possibilitar que serviços específicos de plataforma comuniquem com as outras partes da aplicação em um formato de dados comum. Considerando esse requisito, após uma reunião de *brainstorming* com a empresa foi definida uma arquitetura em seis camadas conforme ilustrada

⁴ <<https://angularjs.org/>>

⁵ <<http://getbootstrap.com/2.3.2/>>

⁶ <<https://spring.io/>>

⁷ <<https://www.mongodb.com/>>

⁸ <<https://nodejs.org>>

⁹ <<https://bower.io/>>

¹⁰ <<https://github.com/>>

¹¹ <<https://maven.apache.org/>>

na Figura 15. A Figura 15 apresenta ainda um mapeamento do nível de reúso dos elementos da aplicação a serem implementados em cada camada.

Figura 15 – Arquitetura de referência

Fonte: autoria própria

A arquitetura definida é dividida em duas grandes camadas: *front-end* e *back-end*. Essas camadas, por sua vez, são subdivididas em três outras camadas cada uma. A camada de *front-end* possui as subcamadas *View* que refere-se a todos os elementos de visão da aplicação, *Controller* que refere-se a parte de controladores Angular e *Services* que refere-se aos elementos de comunicação com as camadas de *back-end*. Essas três camadas de *front-end* utilizam conceito do *framework* AngularJs de *two-way data bind*¹². A camada de *front-end* é independente de plataforma por ser desenvolvida utilizando apenas em tecnologias de *front-end* como JavaScript, Angular, HTML e CSS possui alto potencial para reúso. O formato de dados escolhido para

¹² <https://www.w3schools.com/angular/angular_databinding.asp>

troca de informações e interoperabilidade, ao nível de comunicação, considerando as requisições ao *back-end* foi o *JavaScript Object Notation (JSON)*. O formato foi selecionado devido a simplicidade, facilidade de uso e amplo ferramental tecnológico que suporta o JSON. A camada de *front-end*, por meio da camada *Services*, faz requisições à camada de *back-end*.

A camada de *back-end* consiste em três outras subcamadas, a subcamada *Router Layer* recebe todas as requisições do *front-end* e utiliza o padrão de projeto *Data Transfer Object Pattern - DTO* para transferir dados e chamar o método correto da camada *Services*. A camada *Services* do *back-end* consiste na camada de serviços de fato, recebe os dados do *front-end*, trata a requisição, implementa regras de negócio do *back-end* se necessário, faz operações com camada de persistência *persistence* e envia resposta de requisições para a camada de *front-end*.

A arquitetura apresentada na [Figura 15](#) é recorrente e comumente utilizada na indústria. Porém, considerando sistemas tradicionais (não PaaS) o mapeamento de nível de reuso conforme cada camada, evidentemente, pode variar. Em direção ao objetivo a que se destina este Estudo de Caso, foi realizada uma nova seção de *brainstorming* e foi definido que seria desenvolvido um sistema *toy* considerando domínio de universidade.

Como o foco do trabalho inicialmente é definir implementações que consideram o domínio de *Create-Read-Update-Delete (CRUD)* que servirão como referência para o desenvolvimento de mais aplicações, foi definido que seria desenvolvida uma aplicação que incluísse funcionalidades CRUD para gerenciamento de Estudantes, Disciplinas e Matrícula (estudantes matriculados). Portanto, a empresa desenvolveu uma aplicação conforme requisitos tecnológicos a seguir:

- **Front-end:** HTML, CSS, JavaScript, AngularJS, JSON e Twitter Bootstrap
- **Back-end:** Spring MVC, JPA, Java, JSON e REST

4.3.2 PaaS selecionadas

Após o desenvolvimento da aplicação, a empresa selecionou três plataformas para serem consideradas para dar continuidade ao estudo: *Google App Engine (GAE)*, *IBM Bluemix* e *Heroku*. A plataforma GAE foi selecionada por critérios de dominação mercadológica. A plataforma IBM Bluemix foi selecionada após sugestão de um especialista da equipe de desenvolvimento a partir do argumento de que a empresa teria interesse prévio em investigar essa plataforma. Por fim, a plataforma Heroku foi selecionada devido a critérios de flexibilidade e dominação mercadológica. A plataforma *Windows Azure* não foi selecionada para este estudo pois a equipe de desenvolvimento alegou que a empresa já possuía conhecimentos sobre essa plataforma e que gostaria de considerar novas plataformas ainda não adotadas pela empresa, como o GAE, IBM Bluemix e Heroku.

4.3.3 Estudo das plataformas selecionadas

Esta atividade consiste em efetivamente estudar e desenvolver aplicações considerando as plataformas selecionadas. Foi necessário verificar como cada plataforma funciona: suas especificidades, mecanismos de *deploy*, exigências de cada plataforma e limites de utilização.

O GAE é uma PaaS que permite que sejam executados serviços na infraestrutura do Google. O GAE possui ambiente de desenvolvimento para as linguagens Java, Python e Go. Para este estudo foi selecionado seu próprio ambiente Java¹³. O ambiente Java do GAE fornece uma máquina virtual Java, uma interface de Java *Servlets*, e serviço de armazenamento de dados.

O GAE executa aplicativos Java usando a máquina virtual Java (JVM) 6. Seu SDK (*Software Development Kit* ou Kit de desenvolvimento de software) suporta o Java 5 e mais recentes, e a JVM 6 pode usar classes compiladas com qualquer versão do compilador Java até a versão 6. O ambiente usa o padrão Java *Servlets* para aplicativos da web. O desenvolvedor fornece as classes de *Servlet* do aplicativo ou ainda páginas do tipo *Java Server Pages* (JSPs), arquivos estáticos e arquivos de dados, além do descritor de implantação (o arquivo *web.xml*) e outros arquivos de configuração, em uma estrutura de diretórios WAR (Java *Web Archive*) padrão. O GAE atende às solicitações chamando os *servlets* de acordo com o descritor de implantação. Porém, caso o desenvolvedor não queira trabalhar diretamente com *Servlets* é possível integrar *frameworks* tais como VRaptor¹⁴, Struts¹⁵, entre outros.

Os aplicativos podem usar os serviços de armazenamento de dados do GAE, em sua oferta NoSQL está o *Datastore*. Há suporte para duas interfaces Java nativas: *Java Data Objects* (JDO)¹⁶ 2.3 e *Java Persistence API*¹⁷ (JPA). Essas interfaces são implementadas usando a Plataforma de acesso *DataNucleus*¹⁸, uma implementação de código aberto desses padrões. A plataforma suporta também o framework *objectify*¹⁹, uma API projetada especificamente para o GAE que provê uma camada de abstração visando facilitar a manipulação de dados na plataforma.

Nenhum investimento inicial é necessário para começar a desenvolver aplicações utilizando o GAE. Todos os aplicativos podem usar gratuitamente até 500 MB de armazenamento e largura de banda suficiente para suportar uma aplicação que serve cerca de 5 milhões de visualizações de página por mês. É necessário pagamento somente para utilizar recursos acima dos níveis gratuitos. O GAE ainda disponibiliza gratuitamente um domínio de rede do tipo: “<<http://<nomedaaplicacao>.appspot.com>>”. Como auxílio ao desenvolvedor, existe um suporte

¹³ Para maiores informações a respeito das demais linguagens suportadas pelo GAE acesse: <<https://developers.google.com/appengine/>>

¹⁴ VRaptor: <<http://vraptor.caelum.com.br/>>

¹⁵ Struts: <<http://struts.apache.org/>>

¹⁶ JDO: <<http://www.oracle.com/technetwork/java/index-jsp-135919.html>>

¹⁷ JPA: <<http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html>>

¹⁸ <<http://www.datanucleus.org/>>

¹⁹ <<https://github.com/objectify/objectify/wiki>>

personalizado na forma de *plugins* para a IDE Eclipse, de modo a facilitar o desenvolvimento de aplicações para o GAE.

A plataforma IBM Bluemix²⁰ é a PaaS da IBM. Bluemix tem por base a *open PaaS Cloud Foundry*²¹. Assim, permite a implantação de aplicações desenvolvidas em diversas linguagens, dentre elas Java, Python, NodeJs, PHP, Ruby and .Net. Possui integração com as *Integrated Development Environment* (IDEs) de desenvolvimento Eclipse e Visual Studio. A plataforma possibilita o teste por trinta dias. A equipe de desenvolvimento descartou o uso dessa plataforma considerando que os limites gratuitos não eram suficientes para execução do Estudo de Caso e que suas especificidades privilegiam soluções relacionadas à tecnologias IBM. Assim, optou-se por priorizar plataformas que possibilitam testes gratuitos além do permitido pela Bluemix.

A Heroku é uma PaaS que utiliza o sistema de *containers* leves. A plataforma oferece um ambiente completo que possibilita adição de *add-ons* diretamente integrados à aplicação implantada. Os desenvolvedores podem implantar aplicações escritas em Node, Ruby, Java, PHP, Python, Go, Scala ou Clojure. A ideia é que o desenvolvedor trabalhe em uma abordagem *app-centric*, ou seja, com foco no desenvolvimento das funcionalidades da aplicação sem a necessidade de se preocupar com detalhes relacionados ao gerenciamento de infraestrutura. A plataforma oferece uma *dashboard* completa e permite a implantação de aplicações por meio de Github, Dropbox ou Heroku dev - uma ferramenta específica para realizar operações específicas da plataforma como *deploy*, exibição de logs, entre outros. O sistema de *add-ons* da Heroku é bastante intuitivo e possibilita que serviços de nuvem terceiros sejam facilmente adicionados à aplicação.

Adotar a Heroku, como previsto, exige uma curva de aprendizado. As aplicações precisam estar configuradas e utilizar o conjunto de tecnologias exatamente como esperado pela plataforma. Esse sistema de *add-ons* da Heroku, embora seja bastante simples de utilizar pode influenciar no desempenho das aplicações. Muitos desses *add-ons* não são nativos da plataforma e, na prática, consistem um serviço de nuvem de terceiros trabalhando em conjunto com os recursos nativos providos. O sistema de *add-ons* implementado na Heroku consiste em um exemplo ideal de uma aplicação em conformidade com a definição de interoperabilidade apresentada por Nogueira *et al.* (2016b) que argumenta que os diversos serviços de nuvem devem inter-operar.

Assim, na prática, uma aplicação Heroku consiste em um software global, formado pela composição de diversos serviços. Esse software tira melhor proveito de cada uma das tecnologias adicionadas e forma uma aplicação heterogênea melhorada pela composição de serviços adicionados de acordo com a funcionalidade da aplicação. Por outro lado, é possível verificar gargalos de comunicação em nível de rede e um possível espalhamento de dados por diversos países que precisa ser levado em consideração no contexto de aplicações de governo. Em uma aplicação Heroku Java, o conjunto de tecnologias, gerenciamento de referências, arquivos

²⁰ <<https://www.ibm.com/blogs/bluemix/>>

²¹ <<https://www.cloudfoundry.org/>>

de configuração precisam estar em conformidade com o que é esperado pela plataforma. Não é qualquer aplicação que pode ser implantada nessa plataforma.

A partir da análise das plataformas selecionadas foi possível perceber que o *lock-in* ainda é uma grande preocupação. Como pode ser visto, as plataformas possuem especificidades que impedem que aplicações Web tradicionais sejam implantadas nessas plataformas. As aplicações, portanto, devem ser específicas de plataforma. Outro detalhe que pode ser destacado é que uma aplicação escrita em Java, implementada considerando o conjunto de requisitos tecnológicos apresentados anteriormente, conforme previsto, precisariam passar por um processo de reengenharia antes de serem implantadas nas plataformas conforme apresentado na revisão de literatura (SILVA; LUCRÉDIO, 2012). Assim, o caminho (ii) (da aplicação para a plataforma) previamente definido como caminho a ser seguido neste trabalho mostrou-se não ser um caminho possível no contexto deste estudo.

Após a análise realizada, foram selecionadas as plataformas GAE e Heroku. A plataforma GAE foi incluída por critérios de dominação mercadológica, a empresa queria explorar essa plataforma para obtenção de *know-how* visando cenários futuros de aplicação. A plataforma Heroku foi selecionada devido a flexibilidade e alinhamento com os requisitos tecnológicos definidos para aplicação. Para dar continuidade ao Estudo de Caso foi necessário, portanto, fazer a reengenharia da aplicação desenvolvida para as plataformas selecionadas.

4.3.4 ***Desenvolvimento das bibliotecas de código e aplicações uniplataforma***

As atividades de **desenvolvimento de uma biblioteca de código para cada plataforma e desenvolvimento de aplicações uniplataforma**, na prática, se complementam. No caso específico deste Estudo de Caso, a empresa parceira desenvolveu uma aplicação Web tradicional em conformidade com os requisitos tecnológicos elencados pela mesma. A aplicação precisa ser migrada para as plataformas selecionadas, faz-se necessário a reengenharia da aplicação considerando as especificidades das plataformas de destino. Assim, a aplicação base precisou passar por dois processos de reengenharia, um para a plataforma do Google, outro para a Heroku. Ao fim, obteve-se duas novas aplicações com funcionalidades idênticas, mas específicas de plataforma e que, consequentemente, agregam as vantagens e desvantagens da plataforma selecionada.

Em direção a esse objetivo foi necessário fazer uma auditoria rigorosa da aplicação implementada pela equipe da empresa parceira. Essa auditoria, em uma visão mais geral, consiste na análise da arquitetura utilizada e do código fonte, tanto de *back-end* quanto de *front-end* da aplicação, para verificação de conformidade com os requisitos definidos para o estudo.

Auditoria da aplicação implementada

Para alcançar os objetivos a que se propõe a auditoria, os seguintes objetivos específicos foram definidos:

- (i) Verificação da estrutura do projeto
- (ii) Verificação dos elementos da camada *View* do *front-end* e interface com *controller*
- (iii) Verificação dos elementos da camada *controller* do *front-end* e interface com *front-end Service*
- (iv) Verificação dos elementos da camada *front-end Service*
- (v) Verificação dos elementos de requisição para *back-end* e tratamento de resposta
- (vi) Verificação dos elementos da camada de *router*
- (vii) Verificação dos elementos da camada de serviços com *Spring* no *back-end* e integração com *Spring Security*
- (viii) Verificação arquitetural dos elementos de *back-end*
- (ix) Verificação da estratégia utilizada no *Spring services*
- (x) Verificação da integração com o banco de dados

A seguir, são descritos com detalhes como foram desenvolvidos cada um desses objetivos específicos.

i) Verificação da estrutura do projeto

Para este objetivo específico, foi necessário analisar toda a estrutura da aplicação desenvolvida, quais seus elementos, artefatos, pastas, estrutura, códigos, arquivos, etc. Na verificação da estrutura do *back-end*, foram analisados como estavam estruturados e organizados todos os elementos relacionados a essa camada e como cada um dos elementos são instanciados. A aplicação utiliza padrão de projeto DTO, elementos de *Spring Security*²², *Data Access Objects* (DAOs) com JPA e *Spring services*.

Nessa análise podem ser destacados classes de configuração de contexto, comunicação com banco NoSQL e uso do *Spring Security*. A aplicação utiliza Angular para validação de formulário. O uso de *angular-form-validations* é feito diretamente no campo dos formulários. Todavia, uma re-validação é feita a nível de controladores Angular. As respostas das requisições

²² <<https://projects.spring.io/spring-security/>>

REST não são validadas. O formato de resposta não é padronizado e o desenvolvedor necessita utilizar depuração Angular para identificar os formatos retornados função a função.

Em relação aos formatos retornados e requisições enviadas há um fluxo direto em conformidade com a arquitetura projetada. No entanto, não há padronização no formato de envio das requisições, bem como dos seus tipos e cabeçalhos de requisições REST. Isso viola o elemento central de comunicação entre camadas conforme definido que seria *JSON*. O envio de requisições do tipo *form-submission* é efetivo e de fato funciona, mas não possibilita por exemplo chamadas explícitas a domínios cruzados (um dos objetivos do desenvolvimento deste projeto). Esta primeira fase da avaliação foi mais estrutural, a etapa seguinte é caixa branca e envolve análise direta do código implementado.

Após a análise realizada, concluiu-se que a aplicação não foi projetada para reuso e não há padronização de código. Faz-se necessário uma refatoração para preparar a aplicação para abstração com tecnologias MDE.

ii) Verificação dos elementos da camada View do front-end e interface com Controller

Nessa fase foi realizada uma verificação caixa branca dos elementos implementados. Foram analisados, por exemplo, o escopo e sub-escopos dos controladores Angular, forma das chamadas ao *back-end*, formato de requisições, formatos de respostas, padrões de código e de projeto utilizados. Após a análise realizada, concluiu-se que era necessário uma refatoração para preparação da aplicação para abstração e pra adaptar as estruturas da aplicação para ficar em conformidade com a arquitetura de referência.

iii) Verificação dos elementos da camada Controller do front-end e interface com front-end Services

Uma das estruturas fundamentais do projeto é a definição do funcionamento dos *forms*. O uso do Angular em *forms* tem uma lógica específica. É necessário definir o escopo e ligação dos elementos de visualização com os controladores da camada de *front-end*. Além dessa lógica, uma sintaxe relacionada ao JSON precisa ser seguida. Os elementos de *two-way data bind* envolvidos precisam considerar esse formato.

Uma vez que os elementos da camada View, Controller e Services do *front-end* foram verificados, foi necessário analisar a comunicação com o *back-end*. Na aplicação implementada, isso pode ser feito basicamente de duas maneiras: *form submission* e requisições *Asynchronous JavaScript And XML* (AJAX). As requisições AJAX por sua vez podem ser feitas de diversas maneiras. Essa forma de comunicação não estava implementada na aplicação desenvolvida. Concluiu-se, portanto, que era necessário modernizar a aplicação para que fique totalmente em conformidade com os requisitos arquiteturais e tecnológicos definidos.

iv) Verificação dos elementos da camada *front-end Services*

A sub-camada *front-end Services* faz a comunicação efetiva com o *back-end*. Por meio de um controlador Angular é feita a injeção de dependências dos serviços necessários para a comunicação e é feita uma requisição utilizando Angular AJAX para o *back-end*. Primeiramente, é feita a definição de um controlador Angular e a injeção de dependências em nível de camada *Controller*, em seguida é apresentado um método denominado “createUser” que faz uma validação na camada do *front-end controller*, envia e trata o retorno de uma requisição AJAX em formato JSON. Assim, a camada *front-end services* na prática consiste nas requisições de fato.

Ao fim dessa verificação, concluiu-se que os formatos de requisição não estavam padronizados. Havia *form-submissions* e AJAX. Concluiu-se, portanto, que era necessário refatorar a aplicação para ficar em conformidade com a arquitetura de referência. Além disso, era necessário definir serviços que possibilitam a identificação de domínios cruzados para desenvolvimento das aplicações *multi-cloud*.

v) Verificação dos elementos de requisição para *back-end* e tratamento de resposta

Em termos de requisições e formatos de comunicação com *back-end*, conforme mencionado anteriormente, várias formas e chamadas diferentes foram encontradas. Isso viola a arquitetura de referência definida para a aplicação. Foram encontrados submissões tanto a nível de *form submission* quanto AJAX JSON. Para que aplicação esteja em conformidade com a arquitetura é necessário refatorar todas as chamadas *url encoded* para AJAX JSON, bem como utilizar um formato de retorno específico para o *back-end*. A definição de um formato único de comunicação pode aumentar o reuso da aplicação e facilitar ainda mais a manutenção.

Concluiu-se, portanto, que era necessário refatorar todas as requisições para assíncronas e com cabeçalho de requisição em formato JSON. Além disso, visto que o formato da requisição *front-end* deveria modificada, o serviço de *back-end* também precisaria ser refatorado para estar em conformidade com a arquitetura e esperando o formato especificado.

vi) Verificação dos elementos da camada *Router*

A aplicação implementada não utiliza Angular routers, faz tratamento tradicional do fluxo da aplicação. Em termos de *back-end*, a camada Router apenas recebe requisições e as trata gerando uma resposta conforme cada requisição. No *front-end*, para encapsulamento da lógica inteira da aplicação a nível de Angular routers, bem como injeção de dependência seria necessária uma grande refatoração na aplicação para colocá-la em conformidade com as melhores práticas de Angular. Isso permitiria também um melhor encapsulamento e modularização da aplicação e facilitaria o reuso. No entanto, para manter conformidade com a estrutura e uso de Angular adotado pela empresa foi mantido o padrão de utilização.

vii) Verificação dos elementos da camada de serviços com Spring no back-end e integração com Spring Security

Foi realizada a verificação do uso do *Spring MVC* em nível de *back-end*. Em termos de camadas arquiteturais o *back-end* estava em conformidade com a arquitetura projetada. Todavia, considerando as especificidades de plataforma todo o *back-end* precisa passar por um processo de reengenharia para se adaptar às especificidades de plataforma.

viii) Verificação arquitetural dos elementos de back-end

Foi realizada a verificação do uso das tecnologias de *back-end*, mais especificamente, como são criados os serviços e como as camadas se comunicam com os elementos de persistência. Concluiu-se que o *back-end* estava em conformidade com a arquitetura de referência. No entanto, seria necessário uma refatoração para adaptar a aplicação às especificidades de plataforma.

ix) Verificação da estratégia utilizada no Spring services

Foi realizada uma verificação do uso do Spring e de seus arquivos de configuração. Concluiu-se que o *back-end* estava em conformidade com a arquitetura de referência. No entanto, seria necessário uma refatoração para adaptar a aplicação às especificidades de plataforma.

x) Verificação da integração com o banco de dados

Foi realizada uma verificação da integração com o banco de dados. A aplicação foi desenvolvida utilizando MongoDB. Concluiu-se que o *back-end* estava em conformidade com a arquitetura de referência. No entanto, seria necessário uma refatoração para adaptar a aplicação às especificidades das plataformas selecionadas.

Refatoração de aplicação em conformidade com as tecnologias e arquitetura obtidas como resultado do Estudo de Caso

Conforme mencionado anteriormente, foi realizada uma auditoria na aplicação desenvolvida pela empresa parceira. Essencialmente, a auditoria visou analisar todo o código produzido para verificar a conformidade arquitetural da aplicação com a arquitetura projetada. Como pôde ser observado, foram encontrados pontos de melhoria e de necessidade refatoração e adequação aos requisitos definidos.

Um estudo detalhado sobre as vantagens e desvantagens na adoção de cada um dos modelos de serviço de nuvem é necessário antes da adoção de qualquer oferta relacionada ao modelo. Além disso, faz-se necessário a consideração de eventuais problemas pós adoção, tais como mudança de plataforma, preço a longo prazo, entre outros. Como pôde ser observado, a empresa não desenvolveu a aplicação com uma filosofia de desenvolvimento relacionada ao

modelo de PaaS. Inicialmente, tentou-se seguir o caminho ii) (da aplicação para a plataforma) para adoção de PaaS, mas devido as plataformas selecionadas constatou-se que não é possível.

A equipe de desenvolvimento concluiu que, em se tratando de PaaS no contexto deste trabalho, é necessário iniciar a implementação da aplicação considerando a plataforma (*bottom-up*), não o oposto. Isso significa que para desenvolver sistemas para as PaaS selecionadas, primeiramente faz-se necessário estudar a plataforma para posteriormente desenvolver o sistema. O caminho oposto, desenvolver o sistema e só depois implantar na plataforma embora possível, em outros contextos, pode não dar certo, dependendo das plataformas escolhidas. Para que a aplicação implementada pela empresa funcionasse no modelo de PaaS, foi necessária uma refatoração completa. Essa refatoração precisou estar alinhada com as tecnologias permitidas por cada PaaS selecionada no estudo.

Refatoração da aplicação de referência para a GAE

Conforme mencionado anteriormente, para migrar o sistema implementado para as plataformas selecionadas neste estudo foram necessários dois processos de refatoração, um para cada plataforma escolhida. Dessa forma, ao fim do processo, obtém-se todo um conjunto de artefatos de software que serão denominados neste trabalho como bibliotecas de código e implementações de referência.

Após uma reunião de *brainstorming*, foi definido que as aplicações seriam modernizadas, refatoradas para ficar em conformidade com a arquitetura de referência definida. Em relação aos requisitos tecnológicos, como o Estudo de Caso tem caráter exploratório, foi definido que as tecnologias de *back-end* deveriam ser modernizadas utilizando *frameworks* que visam facilitar o trabalho sobre as tecnologias básicas definidas. Ou seja, já existem *frameworks* modernos que possibilitam o uso de controle de MVC com *Spring*, criação de serviços e injeção de dependências no *back-end* (o Angular faz injeção de dependências em nível de *front-end*) em alto nível que deveriam ser investigados visando facilitar o desenvolvimento dos sistemas. Isso também se faz necessário para verificar a adaptabilidade e possibilidade de utilização de *frameworks* com a plataforma, bem como para preparar a aplicação para reúso e abstração utilizando-se mecanismos de MDE.

Assim, após sugestão de um especialista da equipe foram considerados os *frameworks* VRaptor²³ para MVC, criação de serviços REST, tratamento de requisição e resposta e injeção de dependências em nível de código. O *framework* foi selecionado por estar diretamente alinhado às necessidades do projeto e por supostamente facilitar a criação de serviços e demais elementos de *back-end*. Para serialização das requisições e respostas considerando o formato JSON foi utilizado o *framework* Ext.Js²⁴ para evitar problemas de incompatibilidade na manipulação do

²³ <<http://vraptor3.vraptor.org/en/>>

²⁴ <<http://docs.sencha.com/extjs/>>

Document Object Model (DOM) nas camadas de front-end.

Conforme mencionado na seção anterior, o GAE permite três mecanismos de comunicação com o banco de dados não relacional. É possível utilizar o *datastore* com JPA ou JDO implementados sobre *datanucleus*²⁵ e também o *framework* objectify²⁶ que possui um sistema próprio de anotações. Para migrar para e/ou desenvolver um sistema considerando o GAE, a equipe de desenvolvimento deve escolher uma das opções. Tecnologias como JPA e JDO já são bastante utilizadas na indústria e o mecanismo de mapeamento do objectify segue a mesma filosofia de Mapeamento Objeto Relacional (MOR) dessas tecnologias. As APIs JPA e JDO são comumente utilizadas em banco de dados relacionais e o GAE implementa um subconjunto das operações permitidas por essas APIs para utilização do seu banco não relacional. Isso configura mais um item que deve ser levado em consideração no desenvolvimento de sistemas para essa plataforma.

O desenvolvedor ou equipe de desenvolvimento deve aprender todas as restrições impostas pela plataforma e seu mecanismo de armazenamento. Para Armbrust *et al.* (2009) essa especificidade do mecanismo de armazenamento de dados é um dos principais problemas relacionados ao *lock-in*, denominado *Data lock-in* (*lock-in* de dados). O formato de dados e o mecanismo de armazenamento aprisiona o usuário à plataforma não permitindo que o mesmo troque facilmente de plataforma. Em outras palavras, a JPA e a JDO do GAE são apenas dialetos e os mapeamentos feitos para sistemas desenvolvidos para essa plataforma não necessariamente podem ser reutilizadas em outras plataformas que permitem o uso de JPA ou JDO. Além disso, o Objectify é uma tecnologia específica de plataforma e funciona em mais baixo nível que JPA e JDO.

Foram criados *Data Access Object* (DAOs) genéricos para operações CRUD e consultas por meta-dados para os três mecanismos do GAE. O [Código-fonte 1](#) apresenta o trecho de código referente à operação *CREATE* do CRUD considerando os mecanismos do JPA.

Código-fonte 1 – DAO genérico implementado em JPA na plataforma GAE

```
1: public void create(Object bean) {
2:     em = EMF.get().getEntityManager();
3:     try {
4:         em.persist(bean);
5:     }
6:     catch(Exception ex){
7:         ...
8:     }
9:     finally {
10:         em.close();
```

²⁵ <http://www.datanucleus.org/>

²⁶ <https://github.com/objectify/objectify/wiki>

```
11:     }
12: }
13: ... outros metodos
```

A linha 1 do [Código-fonte 1](#) apresenta a instanciação de um objeto do tipo `EntityManagerFactory` (EMF) da API JPA para manipulação de dados. A Linha 4 (do [Código-fonte 1](#)) apresenta o uso o objeto para persistir uma entidade.

A chamada “`EMF.get().getEntityManager()`” utiliza o padrão de projeto *factory* para retornar objetos do tipo `EntityManagerFactory` a partir dos quais todas as manipulações de dados utilizando o padrão JPA são possíveis.

Como pode ser visto no [Código-fonte 2](#), o padrão JDO utiliza um mecanismo bastante similar, mudando apenas a sintaxe e os mecanismos internos de implementação. No código apresentado foram inseridos tratamentos de exceção e controle de transação.

Código-fonte 2 – DAO genérico implementado em JDO na plataforma GAE

```
1:  public boolean save(Object bean) {
2:      pm = PMF.get().getPersistenceManager();
3:      Transaction tx = pm.currentTransaction();
4:      try {
5:          tx.begin();
6:          pm.makePersistent(bean);
7:          tx.commit();
8:          return true;
9:      } catch (DatastoreTimeoutException ex) {
10:         message = "Timeout: An error has occurred: Was not possible to
        persist the"
11:             + bean.getClass().getName();
12:         System.out.println(message);
13:         ex.printStackTrace();
14:         return false;
15:     } catch (Exception ex) {
16:         message = "An error has occurred: Was not possible to persist the"
17:             + bean.getClass().getName();
18:         System.out.println(message);
19:         ex.printStackTrace();
20:         return false;
21:     }
22:     finally {
23:         if (tx.isActive()) {
24:             tx.rollback();
```

```
25:     }
26:     pm.close();
27: }
28: }
29: ... outros metodos
```

No [Código-fonte 2](#), a linha 5 apresenta o código referente ao início de uma transação; a linha 6 apresenta a utilização do *persistence manager* para efetivamente armazenar o objeto genérico. A seguir, o [Código-fonte 3](#) apresenta o mesmo método, mas considerando o *framework* Objectify.

Código-fonte 3 – Generic DAO Objectify GAE

```
1: public void create(T object) {
2:     try {
3:         ofy().save().entity(object).now();
4:     } catch (Exception ex) {
5:         errors.add("An error has occurred in method insertEntity(T object): "
6:             + ex.getCause());
7:         ex.printStackTrace();
8:     }
9: }
10: ... outros metodos
```

No [Código-fonte 3](#), a linha 3 apresenta a utilização da API do objectify para persistir um objeto genérico. A filosofia da API de baixo nível objectify é semelhante, utilizar MOR para fazer o mapeamento e utilização da filosofia de orientação a objetos para tratar os dados. Uma avaliação de desempenho considerando as diferenças entre os três mecanismos está fora do escopo deste trabalho, mas a própria documentação do GAE²⁷ informa que JPA foi projetado para bancos de dados relacionais tradicionais e que a maioria dos desenvolvedores terão uma melhor experiência utilizando objectify. Por esse motivo, todos os exemplos implementados neste trabalho utilizam essa API.

Dando continuidade à refatoração e à modernização da aplicação, os serviços referentes à camada *service* antes implementados utilizando o *framework* Spring nativo conforme apresentado no [Código-fonte 4](#) forma modernizado para utilização do *VRaptor*.

Código-fonte 4 – Controlador implementado utilizando Spring

```
1: @Controller
2: @RequestMapping("/discipline")
```

²⁷ <<https://cloud.google.com/appengine/docs/standard/java/datastore/jpa/overview>>

```
3: public class DisciplineController {
4:     private static final Logger LOGGER = Logger.getLogger(
        DisciplineController.class);
5:     @Autowired
6:     DisciplineService disciplineService;
7:     @ResponseStatus(HttpStatus.OK)
8:     @RequestMapping(method = RequestMethod.POST)
9:     public void createDiscipline(@RequestBody DisciplineDTO discipline) {
10:         disciplineService.createDiscipline(discipline.getDisciplinename(),
            discipline.getTimeschedule(), discipline.getHours());
11:     }
12:     @ExceptionHandler(Exception.class)
13:     public ResponseEntity<String> errorHandler(Exception exc) {
14:         LOGGER.error(exc.getMessage(), exc);
15:         return new ResponseEntity<>(exc.getMessage(), HttpStatus.BAD_REQUEST)
            ;
16:     }
17: }
```

Na Listagem do [Código-fonte 4](#), as linhas 4, 12, 14 e 15 apresentam código referente ao mecanismo de *tracer* ou *logger* da aplicação. As linhas 1, 2, 5, 7 e 8 apresentam anotações do *framework* Spring para implementação do serviço que irá receber e tratar as requisições. A linha 9 apresenta a assinatura do serviço que utiliza o padrão de projeto DTO para transferência e serialização dos dados a serem passados pela camada de controle. A Listagem do [Código-fonte 5](#) apresenta o mesmo controlador, mas utilizando o *framework* VRaptor.

Código-fonte 5 – Controlador implementado utilizando VRaptor

```
1: ...imports...
2: @Resource
3: public class DisciplineController {
4:     private GenericDAOObjectify<Discipline> genericDaoObjectify;
5:     private Result result;
6:     ...
7:     @Post
8:     @Consumes("application/json")
9:     public void create(Discipline discipline) {
10:         AjaxReturn ajaxReturn = new AjaxReturn();
11:         try{
12:             genericDaoObjectify.insertEntity(discipline);
13:             ajaxReturn.setSucess(errors.isEmpty());
14:         }catch(Exception ex){
```

```

15:     ajaxReturn.setSucess(false);
16:     ajaxReturn.addErrors(genericDaoObjectify.getErrors());
17: }
18:     result.use(ExtJSJson.class).from(ajaxReturn).serialize();
19: }
20: }

```

Na Listagem do [Código-fonte 5](#), a linha 2 apresenta uma anotação *VRaptor* que transforma a classe *DisciplineController* em um serviço REST. Cada método dessa classe automaticamente será convertido em serviço que pode ser invocado por meio de uma chamada REST pelas tecnologias de *front-end*. Por exemplo, o método “create” (linha 9) recebe uma requisição com dados em formato JSON e como parâmetro um objeto do tipo “Discipline”. Esse objeto é tratado utilizando o DAO genérico e uma resposta para requisição em formato “ExtJson” é realizada na linha 18. No código apresentado não é possível visualizar o mecanismo de *tracer* ou *logger* porque esse mecanismo é um interesse transversal e foi implementando utilizando Orientação a Aspectos²⁸.

Após essa refatoração, o *back-end* da aplicação passa a cumprir todos os requisitos arquiteturais conforme a arquitetura de referência. Assim, toda a aplicação inicial implementada passou pelo processo de refatoração para o GAE. A Camada *model* da aplicação inicialmente desenvolvida considerando o MongoDB foi reestruturada para utilização do DataStore do Google. A camada *controller* da aplicação foi inteiramente refatorada em conformidade com a arquitetura de referência e as tecnologias apresentadas na Listagem do [Código-fonte 5](#). Toda a camada de *front-end* da aplicação de referência da empresa também precisou passar por uma refatoração. Conforme identificado na auditoria realizada, o *front-end* não estava em conformidade com a arquitetura de referência. A Listagem do [Código-fonte 6](#) apresenta um controlador Angular que envia uma requisição AJAX ao *back-end*.

Código-fonte 6 – Controlador da camada *front-end* implementado utilizando Angular

```

1 angular.module('disciplineApp',[ 'common', 'spring-security-csrf-token-
    interceptor','editableTableWidgets' ])
2 .controller('DisciplineController',['$scope','$http',function($scope,
    $http) {
3     ... outros metodos ...
4     $scope.create = function() {
5         var data = {
6             discipline : {
7                 "id" : $scope.vm.id,
8                 "name" : $scope.vm.name != undefined ? $scope.vm.name:'',

```

²⁸ <http://www.eclipse.org/aspectj/>

```
9         "schedule" : schedule = $scope.vm.schedule != undefined ?
    $scope.vm.schedule : '',
10         "hours" : $scope.vm.hours != undefined ? $scope.vm.hours : ''
11     }
12 }
13 $http({
14     method : 'POST',
15     url : '/discipline/create',
16     data : data,
17     headers : {
18         "Content-Type" : "application/json"
19     }
20 })
21 .then(function(response){
22     if(response.status == 200) {
23         if (response.data.data.success == true) {
24             $scope.message = "Discipline registered successfully!";
25             $scope.vm.name = "";
26             $scope.vm.schedule = "";
27             $scope.vm.hours = "";
28         }
29     }else{
30         $scope.vm.errorvmessages = [];
31         $scope.vm.errorvmessages.push({description : response.data.
errors});
32         console.log("failed registration creation: " + response.data.
errors);
33     }
34 });
35 } ... end controller...
```

Na Listagem do [Código-fonte 6](#), a linha 1 apresenta a criação do módulo Angular. A linha 2 apresenta criação do controlador e a injeção de dependências a nível de *front-end*. A linha 4 apresenta o método “create” do controlador. Esse método capta os dados da camada de visão (linhas 7 a 10) prepara para enviar a requisição para a URL “discipline/create” (linhas 13 a 20) e, por fim, trata a resposta e utiliza *two-way data bind* para reapresentá-la a camada de *view* (24 a 27 ou 30 a 32). Da maneira que foi reestruturado todo o *front-end* da aplicação se torna independente de plataforma e possui alto potencial para reúso, embora seja dependente de formato de dados JSON.

Refatoração da aplicação de referência para a Heroku

Conforme apresentado anteriormente, a plataforma Heroku possui suas especificidades. Essa plataforma suporta tecnologias diferentes daquelas suportadas pelo GAE e os seus mecanismos de implantação e gerenciamento de aplicações são diferentes. Portanto, a aplicação PaaS desenvolvida para o GAE não funciona nessa plataforma. Mesmo utilizando a mesma linguagem Java, a Heroku “espera” que a aplicação tenha toda uma estrutura conforme suporta a plataforma²⁹. Como pode ser observado na arquitetura de referência, as camadas mais inferiores da aplicação possuem menor reuso devido as especificidades da plataforma.

Com a Heroku, não é diferente, os componentes tecnológicos da subcamada *Controller* do *back-end* podem ser reutilizados. Porém, os elementos da camada *Persistence* precisam passar por mudanças para se adaptar às especificidades de plataforma. Além disso, toda a estrutura da aplicação em termos de projeto e gerenciamento de bibliotecas precisaram ser modificados visando a implantação na plataforma.

A Heroku utiliza a ferramenta Maven para fazer o gerenciamento de bibliotecas e a estrutura da aplicação é própria. Todos os elementos de configuração da aplicação precisam estar em conformidade com as configurações exigidas pela Heroku para que a mesma funcione. A plataforma ainda exige que sejam criados arquivos de configuração para execução da aplicação em seu ambiente próprio.

Heroku funciona com sistema de *add-ons*, ou seja, serviços de nuvem externos podem ser adicionados à aplicação. Um desses serviços, por exemplo, é o sistema de armazenamento de dados. Como a aplicação básica da empresa utiliza o sistema MongoDB, o *add-on* mLab foi adicionado à aplicação. O mLab é um serviço de armazenamento de dados em nuvem baseado em MongoDB. Na prática, o sistema de *add-ons*³⁰ da plataforma Heroku funciona como uma estratégia de *multi-cloud* porque cada um desses serviços é um externo a plataforma. Em outras palavras, uma aplicação Heroku é um sistema em nuvem que utiliza de vários outros serviços em sua composição, configurando nativamente uma aplicação *multi-cloud*.

O serviço mLab foi selecionado porque é um dos *add-ons* relacionados ao MongoDB suportados pela plataforma e também pela possibilidade de uso gratuito de uma cota gratuita de 500MB de armazenamento. Todavia, qualquer outra oferta relacionada pode ser selecionada. O mLab armazena dados nos Estados Unidos e Europa, portanto, se o usuário do serviço estiver fora dessas regiões, questões relacionadas a *Services Level Agreements* (SLAs) devem ser consideradas. Outra preocupação está relacionada à adição dos serviços externos. Quesitos como privacidade, gargalos de performance e segurança e *lock-in* devem também ser considerados.

Assim, foi desenvolvido um DAO genérico considerando o uso de mLab como *add-on*

²⁹ Instruções para *deploy* de aplicações Java podem ser encontradas no endereço: <<https://devcenter.heroku.com/articles/deploying-java>>

³⁰ A lista completa de *add-ons* suportados pela plataforma podem ser encontrada no endereço: <<https://elements.heroku.com/addons>>

do Heroku. Em termos de implementação foi necessário criar uma classe para configurar os parâmetros de uso do serviço. O [Código-fonte 7](#) apresenta a classe necessária para configuração do mLab.

Código-fonte 7 – Configuração dos parâmetros do banco de dados mLab

```
1: public class MongoDBDatabaseSettings {
2:     private MongoClientURI uri;
3:     private MongoClient client;
4:     private MongoDBDatabase database;
5:     private MongoCollection<Document> collection;
6:
7:     public MongoDBDatabaseSettings() {
8:         super();
9:         uri = new MongoClientURI(
10:             "mongodb://mongodatabaseaccount:mongopassword:39195/databaseaccount
11:             ");
12:         client = new MongoClient(uri);
13:         database = client.getDatabase(uri.getDatabase());
14:         collection = database.getCollection("default");
15:     }
16:     ...
17:     // constructors, getters and setters
18: }
```

No [Código-fonte 7](#), as linhas 10 e 11 apresentam a URL do banco de dados na nuvem do mLab. Essa URL é provida pelo serviço e uma senha de acesso também é gerada. Neste ponto é interessante destacar uma característica importante e uma vantagem interessante dos sistemas em nuvem. A classe apresentada no [Código-fonte 7](#) “aponta” para a URL dos serviços do mLab na nuvem. O mLab possibilita o uso da cota de 500MB de armazenamento por conta cadastrada. Uma aplicação dinâmica que possua N classes como essa pode multiplicar por N a cota grátis, apenas replicando as contas e implementando mecanismos de consumo e sincronização de dados na camada de controle. Esse pequeno exemplo mostra uma vantagem das aplicações *multi-cloud*, a redução de custo. Por outro lado apresenta a complexidade da sincronização de dados, combinação de serviços e gargalos de comunicação via rede.

O [Código-fonte 8](#) apresenta um trecho de código que contém um método de inserção no banco de dados pertencente ao DAO desenvolvido para o mLab usado pela aplicação implantada na Heroku³¹.

³¹ A documentação completa do mongodb pode ser encontrada no endereço: [<https://docs.mongodb.com/>](https://docs.mongodb.com/)

Código-fonte 8 – Método de inserção no banco de dados mLab

```
1: public static boolean saveOrUpdate(final MongoCollection<Document>
    collection, final Object object,
2:     String id) {
3:     boolean status = true;
4:     ObjectMapper objectMapper = new ObjectMapper();
5:     String jsonString;
6:     try {
7:         jsonString = objectMapper.writeValueAsString(object);
8:         BasicDBObject query = new BasicDBObject();
9:         query.append("_id", id);
10:        BasicDBObject doc = BasicDBObject.parse(jsonString);
11:        Bson newDocument = new Document("\$set", doc);
12:        UpdateResult result = collection.updateOne(query, newDocument, (new
            UpdateOptions()).upsert(true));
13:    } catch (IOException e) {
14:        status = false;
15:    }
16:    return status;
17: }
```

No [Código-fonte 8](#), a linha 7 apresenta a utilização de um objeto da classe “ObjectMapper” para “serializar” o objeto como *string*. Esse objeto é então convertido para um “BasicDBObject” que posteriormente é escrito no banco de dados mongo em formato JSON (linha 12). Nas demais camadas da aplicação, acima da subcamada de persistência foram utilizadas mesmas tecnologias e conceitos e utilizados na aplicação GAE. Os elementos dessas camadas superiores por serem mais reutilizáveis, conforme apresentado arquitetura de referência, foram mantidos no sem modificação porque possuem alto potencial de reúso e não são específicos de plataforma. Assim, a aplicação passou por uma refatoração completa apenas em nível de persistência. Os demais elementos das camadas acima foram reutilizados da aplicação GAE, visto que já estavam prontos. A [Figura 16](#) apresenta os artefatos implementados e seus relacionamentos.

Como pode-se observar na [Figura 16](#), o mLab é um serviço terceiro. Assim, o fluxo indicado pelo ponto 1 na [Figura 16](#) representa um ponto de gargalo de comunicação da aplicação, podendo tornar as aplicações mais lentas do que se o serviço fosse nativo do provedor da aplicação implantada.

Ao final dos processos de refatoração da aplicação *Web* tradicional implementada pela empresa, tem-se duas novas aplicações equivalentes, mas que são específicas para as plataformas GAE e Heroku. Essas aplicações nos níveis acima das subcamadas de persistência possuem médio e alto potencial de reúso. Não foi realizada uma comparação experimental a respeito do

Figura 16 – Visão do *back-end* da aplicação de referência para a aplicação Heroku

Fonte: autoria própria

reúso dessas camadas, o potencial de reúso se dá devido a semelhança do código quando usado em novas aplicações; considera-se, portanto, reúso *copy e paste*. A maioria das especificidades de plataforma são relacionadas a modelos de dados e componentes tecnológicos que podem ser suportados ou não por cada plataforma escolhida.

Adicionalmente, este Estudo de Caso possibilitou também que muitos conceitos até então puramente teóricos apresentados na literatura (SILVA; LUCRÉDIO, 2012) se confirmassem na prática. Verificou-se que desenvolver uma aplicação para PaaS possui uma curva de aprendizado elevada, sobretudo, devido a necessidade que o desenvolvedor precisar conhecer os detalhes específicos de cada plataforma. As empresas precisam fazer uma análise cuidadosa das vantagens e desvantagens envolvidas na seleção dessa oferta antes de sua efetiva adoção. Para empresas que possuem grandes aplicações *Web* tradicionais implementadas um processo de reengenharia como apresentado neste estudo pode não ser viável e a oferta de IaaS pode ser mais adequada às suas necessidades.

O código produzido pela empresa não estava em conformidade com as melhores práticas relacionadas ao desenvolvimento de aplicações com Angular e não faz uso de elementos como *Angular Router*. Além disso, também não estava conforme o modelo de PaaS. Trata-se de uma aplicação *Web* que pode ser implantada em um modelo de IaaS, mas que não considera nenhuma PaaS. Em outras palavras, a empresa parceira não desenvolveu a aplicação com uma filosofia de desenvolvimento relacionada a Computação em Nuvem, sobretudo, no que se refere ao desenvolvimento utilizando o modelo PaaS. Idealmente, em termos de adoção de PaaS, a

aplicação tem que partir da plataforma, não o oposto. Caso o caminho oposto seja tomado (como tentou-se fazer neste Estudo de Caso) a refatoração necessita ser alinhada com as tecnologias permitidas por cada PaaS selecionada conforme as necessidades estratégicas de cada empresa.

Ao final do processo de reengenharia das aplicações, obteve-se bibliotecas de código e implementações de referência para desenvolvimento de aplicações para os PaaS GAE e Heroku. As demais atividades que visam a distribuição das funcionalidades da aplicação entre os provedores são apresentadas nas seções seguintes. A [Figura 17](#) apresenta uma das telas da aplicação desenvolvida. O layout apresentado foi implementado em *bootstrap* e é específico da empresa parceira. Para cenários futuros e para que um maior número de desenvolvedores possam utilizar os protótipos implementados neste trabalho, aplicações de referência implementadas nas transformações considerarão o tema *bootstrap* SBAdmin³².

Figura 17 – Aplicação desenvolvida

Name	Address	Telephone	Class	Delete
John Smith	First Street, number 1, New York, US	99 9 9999-9999	Advanced Programming	

Fonte: autoria própria

4.3.5 Desenvolver aplicações e módulos distribuídos

Conforme apresentado planejamento, o objetivo desta atividade é distribuir as aplicações desenvolvidas nas primeiras atividades deste Estudo de Caso. Ou seja, realizar a distribuição da aplicação entre as plataformas de acordo com suas funcionalidades de maneira a diminuir o custo final da fatura de serviços e tirar vantagens das plataformas. Essa característica de redução de custo foi apresentada na refatoração da aplicação para a Heroku em se tratando de banco de dados. Como pode ser observado, ao se criar uma aplicação que “aponta” para diversos serviços de banco de dados, por exemplo, da plataforma mLab é possível reduzir custo e utilizar serviços vindo de diversos destinos. Porém, a proposta a que se destina este trabalho

³² <https://startbootstrap.com/template-overviews/sb-admin-2/>

consiste no estudo de distribuição das camadas e funcionalidades da aplicação como um todo, não somente a distribuição considerando uma plataforma única. Essa distribuição é necessária para explorar cada uma das plataformas. No futuro, a ideia é que as funcionalidades distribuídas sejam compostas para construir aplicação que podem **inter-operar**.

Distribuir as funcionalidades da aplicação e/ou construir uma aplicação única que combina funcionalidades implantadas em diferentes provedores na prática consiste propriamente na construção de uma aplicação interoperável *multi-cloud*. Um caso que ilustraria perfeitamente uma aplicação *multi-cloud* considerando as duas plataformas envolvidas neste estudo seria uma aplicação que armazena dados e meta-dados em um provedor e, por exemplo, multimídia em um outro. Assim, seria possível utilizar serviços de um provedor ou de outro conforme as necessidades da aplicação. Adicionalmente, uma aplicação desse tipo seria exatamente o tipo de aplicação relacionada à definição de interoperabilidade apresentada no [Capítulo 2](#) e que é adotada neste trabalho. A arquitetura de referência em seis camadas utilizada como referência é bastante flexível. A distribuição em nível de persistência é apenas uma das possibilidades identificada ao longo deste estudo.

Uma conclusão importante tirada a partir da execução deste estudo é que, na prática, a ideia fundamental de obter vantagens na combinação de serviços providos por diferentes provedores de nuvem está relacionada a dois quesitos que precisam ser fortemente levados em consideração no desenvolvimento das aplicações *multi-cloud*:

- i) um provedor é na prática um vendedor de algo, como vendedor, é natural que esse provedor queira vender mais que seus concorrentes e obter dominação de mercado. Sendo assim, vender serviços proprietários com características específicas que podem trazer diferencial de mercado é uma ideia muito mais interessante para os provedores. Assim, eles podem não querer seguir padrões e inclusive desenvolver suas próprias tecnologias. Assim, a ideia de combinar serviços proprietários de um provedor com serviços de um outro é natural para o clientes, no entanto, não é trivial conforme apresentado neste trabalho
- ii) as tecnologias proprietárias podem ser exatamente o que o cliente precisa. Muitos trabalhos e iniciativas de mercado têm proposto estratégias para evitar o *lock-in*, porém, em muitos contextos o *lock-in* não é somente necessário, mas pode ser também desejado. Portanto, faz muito sentido o conceito de *lock-in* planejado. Ou seja, a empresa determina se de fato está disposta a adotar uma tecnologia e arcar com as consequências, malefícios e benefícios de sua decisão. Isso é absolutamente normal no desenvolvimento de sistemas.

Uma aplicação *multi-cloud*, portanto, deve não só prever o *lock-in* planejado, mas ter uma estratégia para combinação de serviços de acordo com as necessidades da aplicação e dos desenvolvedores. Idealmente, cada parte da aplicação deverá ser desenvolvida alinhando as características dos serviços às funcionalidades da aplicação. Uma das possibilidades de

combinação de recursos seria desenvolver uma aplicação que agrega os serviços diretamente sem intermediários como apresentado na [Figura 18](#).

Figura 18 – Arquitetura *multi-cloud* derivada da arquitetura de referência com estratégia de consumo de recursos a partir do *front-end*

Fonte: autoria própria

Como pode ser observado, a figura apresenta uma aplicação que pode utilizar três serviços de nuvem distintos, onde *front-end* está um provedor e “consome” serviços de dois *back-ends* implantados em provedores distintos. Esses *back-end* são específicos de plataforma. Outra possibilidade seria desenvolver uma aplicação servidor que “agrega” os serviços e disponibiliza para um “consumidor” de recursos conforme apresentado na [Figura 19](#).

Em ambas arquiteturas apresentadas, é possível perceber pontos de gargalo de comunicação na aplicação. Isso pode deixar a aplicação final lenta se comparada com aplicações tradicionais. As relações custo-benefício da construção de aplicações desse tipo devem ser consideradas na tomada de decisão sobre qual estratégia utilizar. Diversas variações das arquiteturas apresentadas são possíveis, tudo depende da aplicação que está sendo desenvolvida. Dependendo da variação arquitetural conforme a necessidade da aplicação, mecanismos de sincronização/composição/orquestração de serviços devem ser implementados em diferentes níveis/camadas da aplicação.

Em direção ao objetivo de se criar uma abordagem dirigida por modelos para desenvolvimento de aplicações multi-PaaS, a arquitetura apresentada na [Figura 18](#) foi selecionada. Foi desenvolvida uma prova de conceito e exemplo de uma aplicação *multi-cloud* que futuramente pode ser abstraída com tecnologias relacionadas ao MDE. Outra estratégia distinta poderia ser

Figura 19 – Arquitetura multi-cloud derivada da arquitetura de referência com estratégia de consumo de recursos a partir de um serviço de *back-end*

Fonte: autoria própria

seguida conforme a necessidade da aplicação, essa estratégia foi escolhida porque apresenta poucos pontos de gargalo. Assim, as funcionalidades da aplicação apresentada na seção anterior foram divididas para que a aplicação armazenasse multimídia no GAE e demais dados na plataforma Heroku. A aplicação *front-end* consumidora de recursos é independente de plataforma e pode ser implementada em qualquer uma das plataformas selecionadas.

A construção da aplicação *multi-cloud* seguindo a arquitetura apresentada na [Figura 18](#), envolve mecanismos de identificação/localização/consumo/combinação de serviços que foram implementados em nível de *front-end*. Mais precisamente, na camada *Services* do *front-end* foram implementados módulos Angular que armazenam a localização dos recursos. Esses recursos são utilizados nos controladores Angular. O [Código-fonte 9](#) apresenta um trecho de código que se refere à construção de um módulo Angular para identificação das localização dos *back-end* da aplicação.

Código-fonte 9 – Informando a URL do *back-end*

```

1: var backendLocation = angular.module('BackendLocation', []);
2:
3: backendLocation.factory('backendLocation', function() {

```

```
4:   return {
5:     getLocation: function() {
6:       return "https://herokuapp.herokuapp.com";
7:     },
8:     getLocation2: function() {
9:       return "https://gaeapp.appspot.com";
10:    }
11:  };
12: });
```

No [Código-fonte 9](#), a linha 1 apresenta a criação do módulo denominado “BackendLocation”. A linha 3 efetivamente cria a função de retorno da localização dos serviços. O método “getLocation” informa ao *front-end* onde estão os serviços de *back-end*. Todas as localizações de recursos de *back-end* devem ser informadas nesse módulo. O módulo posteriormente será utilizado nos controladores Angular por meio de injeção de dependências. Na camada de controle é que as operações, requisições e tratamento de respostas ocorrerão de fato. Todas as requisições ao *back-end* devem ser feitas de acordo com os requisitos da aplicação de maneira assíncrona e considerando o formato JSON. O [Código-fonte 10](#) apresenta uma requisição AJAX sendo enviada para o *back-end*.

Código-fonte 10 – Enviando uma requisição ao *back-end* considerando o formato JSON

```
1: $scope.create = function () {
2:   ... // logs and tracing operations
3:   var data = {
4:     student : {
5:       "id": $scope.vm.id,
6:       "name" : $scope.vm.name != undefined ? $scope.vm.name : '',
7:       "address" : address = $scope.vm.address != undefined ? $scope.
vm.address : '',
8:       "telephone" : $scope.vm.telephone != undefined ? $scope.vm.
telephone : '',
9:       "schoolClass": {
10:        "id": $scope.schoolclass.id != undefined ? $scope.schoolclass.
id : ''
11:      }
12:    }
13:  }
14:   $http({
15:     method: 'POST',
16:     url: backendLocation.getLocation() + '/student/create',
17:     data: data,
```

```
18:         headers: {
19:             "Content-Type" : "application/json"
20:         }
21:     })
22:     .then(function (response){
23:         // return handling
24:     });
25: }
```

Inicialmente é instanciado um objeto em formato JSON (linhas 4 a 13) logo após um método Angular é invocado para realizar a requisição (linhas 14 a 24). A linha 16 apresenta a URL aonde o recurso REST está localizado. Ou seja, qual método será chamado no *back-end*. A linha 19 apresenta a especificação dos cabeçalhos da requisição. A linha 23, por fim, permite o tratamento de resposta. O objeto “response” presente no escopo da linha 23 contém a resposta dada pelo servidor *back-end*. Depuração Angular pode facilitar a manipulação da resposta e *two-way data bind* facilita a apresentação da resposta se necessário. A função apresentada está dentro do escopo de um controlador Angular, é possível colocar neste escopo requisições a diversos servidores diferentes e assim montar uma aplicação que, por exemplo, consome dados/recursos de diversas plataformas ao mesmo tempo utilizando-se de *Cross-Origin Resource Sharing* (CORS). CORS ou Compartilhamento de Recursos de Origem Cruzada é uma tecnologia embutida nos navegadores para possibilitar que recursos de domínios diversos sejam acessados por uma página Web³³. Essa tecnologia possibilita que uma página consuma e acesse serviços de diversos servidores e diversos provedores de nuvem.

Se a estratégia apresentada na Figura 19 for seguida não é necessário CORS, mas por outro lado um mecanismo de consumo/distribuição deve ser implementado em nível de *back-end*. Aplicações do tipo SPAs (Single-Page Web Application) (OH *et al.*, 2012) podem obter muitas vantagens da combinação PaaS + Serviços + Angular + CORS + multi-cloud devido sua necessidade natural de trabalhar com requisições assíncronas.

Vantagens do uso de CORS incluem, por exemplo, a possibilidade de desenvolvimento de uma aplicação com foco em *front-end* que consome recurso diretamente de diversos serviços de nuvem sendo eles implementados sobre PaaS ou não. Desvantagens incluem o fato de ainda ser uma tecnologia em desenvolvimento que possui seu risco. Uma auditoria de segurança da aplicação se faz necessária, sobretudo, devido o fato de muitos mecanismos de manipulação de dados e regras de negócios serem implementados em JavaScript.

³³ Mais informações no endereço: <<https://www.w3.org/TR/cors/>>

4.4 Considerações finais

Ao final deste Estudo de Caso, foi desenvolvida uma aplicação de referência com funcionalidades CRUD considerando armazenamento NoSQL tendo sido implementadas utilizando-se PaaS, CORS, REST, JSON e a arquitetura de referência apresentada na [Figura 19](#). Os elementos de implementação são apresentados neste Estudo de Caso e uma estratégia de identificação de recursos de *back-end* implementada por meio de serviços Angular e utilizada por meio de chamada de funções e injeção de dependências a nível de controladores Angular também foi apresentada.

Após os processos de refatoração e modernização da aplicação de referência implementada pela empresa parceira, tem-se dois sistemas com funcionalidades idênticas, completamente refatorados para duas PaaS selecionadas. Esse sistema implementa uma estratégia de composição de recursos em nível de *front-end* utilizando Angular Services, requisições CORS e serviços REST.

Foram desenvolvidas bibliotecas de código que permitem o armazenamento de multimídia em ambas plataformas selecionadas. Os recursos são requisitados por meio de AJAX e combinados ao nível de controladores Angular. Outro ponto de destaque é que a aplicação segue restrições definidas conforme apresentado na [Seção 4.2](#). Ou seja, não foram adotados padrões relacionados ao modelo de Computação em Nuvem, a solução considera arquitetura e elementos de implementação alinhados ao interesse da empresa parceira e foi projetada para reúso e abstração com tecnologias MDE. Isso significa que todo o código, estilo de programação, interfaces de comunicação foram refatorados. Técnicas de modularização, separação de interesses transversais e formatos de dados JSON foram utilizados.

Desenvolver uma aplicação conforme apresentado neste estudo não é trivial, há muitos pontos de tomada de decisão que precisam estar diretamente alinhados com a aplicação que está desenvolvida e com as plataformas selecionadas. A aplicação desenvolvida está relacionada ao domínio de CRUD e sistemas *Web* que fazem utilização de modelo de dados voltado aos bancos NoSQL. As aplicações implementadas neste estudo também são fortemente baseadas em comunicação via rede, principalmente por meio de requisições REST. A contribuição central no que se refere a condução deste Estudo de Caso consiste em um caminho alternativo em relação à padronização para desenvolvimento de aplicações que utilizam múltiplos provedores de nuvem. Além disso, dentre as conclusões alcançadas, é possível destacar que o desempenho final da aplicação pode estar comprometido devido a comunicação via rede. A comunicação pela rede é mais lenta do que barramentos físicos de um servidor tradicional. No entanto, as tecnologias estão evoluindo e no futuro é possível alcançar um ponto satisfatório, em que o desempenho das aplicações não seja tão prejudicado. No momento atual, as questões relacionadas a desempenho precisam ser consideradas na adoção do modelo de serviço PaaS.

No planejamento do Estudo de Caso estava previsto uma análise de custo de implantação e

uso *as-a-service* da aplicação. Como pôde ser observado, utilizando-se das técnicas apresentadas neste estudo, é possível que aplicação seja sem custo (no que se refere a custos de implantação) pela combinação/agregação de recursos gratuitos conforme apresentado. Isso inclusive aconteceu com o protótipo desenvolvido. Portanto, neste contexto, não faz sentido desenvolver um plano de análise de custos.

Este estudo apresentou um caminho para desenvolvimento de aplicações *multi-PaaS*. Foi apresentada uma estratégia possível para combinação de recursos de nuvem, por meio de serviços e CORS, que não utiliza padrões e re-utiliza sistemas e conceitos já utilizados pela empresa e uma arquitetura flexível que permite o desenvolvimento de sistema *multi-cloud*. Variações dessa arquitetura são possíveis e recorrentemente apresentadas na literatura, bem como adotadas na indústria. O ponto de destaque em relação ao presente trabalho é apresentar a combinação de recursos e os benefícios e malefícios dos sistemas quando se considera exclusivamente o modelo PaaS. Outros modelos de serviço da Computação em Nuvem podem também ser inseridos no contexto das aplicações desenvolvidas a depender da necessidade do projeto.

A aplicação apresentada aqui, no entanto, é suficiente para ser utilizada como implementação de referência em direção a uma abordagem MDE *bottom-up* para desenvolvimento de sistemas *multi-PaaS*. Outro ponto de destaque é a questão da composição de serviços e possibilidade de desenvolvimento de aplicações que multiplicam por N a quantidade de recursos gratuitos de plataformas por meio da utilização de CORS e de recursos de nuvem que podem inter-operar. Isso destaca uma característica fundamental da nuvem e uma de suas vantagens essenciais, a questão da centralização-distribuição.

Conforme apresentado no [Capítulo 2](#), centralização versus distribuição não devem ser analisadas isoladamente quando se trata das vantagens da nuvem, mas a combinação desses dois conceitos. Isso se configurou na prática porque as aplicações desenvolvidas combinam recursos descentralizados por meio de CORS, recursos esses que tiram vantagens e desvantagens do modelo PaaS (centralizado). O próximo capítulo apresenta uma abordagem dirigida por modelos para desenvolvimento de aplicações *multi-PaaS* que utiliza como implementação de referência a aplicação implementada neste Estudo de Caso.

UMA ABORDAGEM DIRIGIDA POR MODELOS PARA DESENVOLVIMENTO DE SISTEMAS MULTI-PAAS

Este capítulo apresenta o desenvolvimento de uma abordagem dirigida por modelos para desenvolvimento de sistemas *multi-PaaS*. Essa abordagem consiste em uma linguagem específica de domínio (DSL) e um conjunto de transformações que possibilitam a geração das aplicações. Modelos de domínio são especificados utilizando a linguagem desenvolvida neste trabalho, esses modelos servem como entrada para o conjunto de transformações desenvolvidas. As transformações consistem em metaprogramas que geram a aplicação final. A [Seção 5.2](#) apresenta o desenvolvimento da DSL utilizando a ferramenta xText. A [Seção 5.3](#) apresenta o desenvolvimento dos metaprogramas utilizando a ferramenta xTend. A DSL desenvolvida possibilita a especificação de modelos de domínio. Os modelos de domínio servem como entrada para metaprogramas xTend que efetivamente geram o código específico de plataforma.

5.1 Considerações Iniciais

Conforme apontado no capítulo introdutório desta tese, a Computação em Nuvem tem um grande potencial para revolucionar a maneira como sistemas são desenvolvidos e comercializados. Muitas tecnologias relacionadas ao modelo de nuvem ainda estão se desenvolvendo e ainda existem diversos problemas que precisam ser contornados. Dentre tais problemas, está a interoperabilidade na Computação em Nuvem, que é definida como a capacidade de serviços de nuvem trabalharem em conjunto (inter-operar) para compor aplicações que tiram proveito de características dos provedores envolvidos na composição ([NOGUEIRA et al., 2016b](#)).

Para solucionar o problema da interoperabilidade e apresentar uma estratégia para desenvolvimento de sistemas interoperáveis *multi-PaaS*, nesta tese, foi desenvolvida uma abordagem

dirigida por modelos conforme apresentado neste capítulo. Os requisitos funcionais centrais estudados neste trabalho incluem funcionalidades CRUD que representam operações típicas de acesso a bases de dados (*Create, Retrieve, Update, Delete*), e que constituem parte importante da maioria das aplicações. Considerou-se o modelo de armazenamento *Not Only SQL* (NoSQL) tipicamente utilizado em aplicações da Computação em Nuvem e definido como requisito pela empresa parceira. Além disso, foi feita a metamodelagem de domínio da criação de serviços REST e de aplicações *multi-cloud* compostas a partir de aplicações (*single-cloud*) de nuvem específicas.

A abordagem segue um caminho *bottom-up*, desconsiderando padrões, e os elementos MDE foram desenvolvidos considerando aplicações específicas de plataforma. Desse modo, a abordagem proposta é apresentada como uma estratégia alternativa para o desenvolvimento de sistemas *multi-cloud* e para realização de *lock-in* planejado. Considerando os domínios da aplicação e as implementações de referência apresentadas no capítulo anterior, foi feito o projeto do metamodelo de domínio. Junto com especificações de sintaxe textual concreta, esse metamodelo deu origem a uma linguagem específica de domínio (DSL) que descreve os conceitos do domínio em alto nível de abstração.

Uma DSL pode ser considerada uma linguagem de programação ou uma linguagem de especificação que considera um determinado domínio. Ou seja, é uma linguagem projetada para ser útil para um conjunto específico de tarefas em um domínio específico. Em outras palavras é uma linguagem pequena, normalmente declarativa, que é focada em um domínio particular (DEURSEN; KLINT; VISSER, 2000). Existem várias DSLs, alguns exemplos de DSLs bastante utilizadas são: Structured Query Language (SQL), HyperText Markup Language (HTML) e Cascading Style Sheets (CSS). Apesar de custos como aumento da complexidade e redução da manutenibilidade da abordagem serem adicionados ao projeto, DSLs aumentam o reuso e a produtividade no desenvolvimento de sistemas (LUCRÉDIO; ALMEIDA; FORTES, 2012).

A DSL desenvolvida neste trabalho possibilita a construção de modelos independentes de plataforma de nuvem. A ideia central da abordagem desenvolvida nesta tese é que aplicações *multi-PaaS* sejam geradas a partir dos modelos especificados. Em outras palavras, a DSL possibilita a criação de modelos que servem de entrada para um conjunto de metaprogramas que irão efetivamente gerar a aplicação final. Esses metaprogramas também denominados geradores incluem conhecimento específico de plataforma e da arquitetura de referência da aplicação a ser gerada. Assim, para desenvolver uma abordagem *bottom-up* é preciso primeiramente fazer a definição das implementações de referência para em seguida a abstração utilizando os conceitos e tecnologias relacionados ao MDE. As implementações de referência foram apresentadas no capítulo anterior.

Portanto, a partir do conhecimento adquirido no Estudo de Caso apresentado no [Capítulo 4](#), foram definidas e implementadas transformações que são utilizadas para gerar código específico de plataforma e aplicações distribuídas *multi-PaaS* em conformidade tanto com a

arquitetura apresentada na [Figura 18](#) quanto com a estratégia de implementação apresentada no [Capítulo 4](#). Considerou-se que era necessária e suficiente a construção de geradores para pelo menos duas plataformas de nuvem e foram escolhidas as plataformas GAE e Heroku.

Para implementação das aplicações foi selecionada a linguagem Java¹. No [Capítulo 4](#) foi descrito um Estudo de Caso que teve por finalidade desenvolver implementações de referência a serem abstraídas e utilizadas como base para criação dos elementos MDE. As seções a seguir apresentam o detalhamento dos conceitos que foram utilizados e o desenvolvimento da DSL. As implementações apresentadas no [Capítulo 4](#) são aplicações base a partir das quais foram projetadas e construídas transformações que levam em conta suas características e as funcionalidades relacionadas as operações CRUD, serviços REST e aplicações CORS. Portanto, o conhecimento embutido nas transformações desenvolvidas, além dos detalhes de plataforma, levam em consideração o modelo arquitetural das implementações de referência, o domínio selecionado, o estilo de programação previamente adotado pela empresa parceira e as tecnologias utilizadas no desenvolvimento da implementação de referência.

5.2 O modelo independente de plataforma

A definição da linguagem normalmente requer um metamodelo que seja capaz de representar os pontos comuns e variáveis do domínio. Um metamodelo é uma estrutura similar a um diagrama de classes, e possui elementos como classes, atributos, associações e agregações. Seja qual for a representação visual final (diagrama ou representação textual) da DSL, a existência de um metamodelo como esquema conceitual, em geral, indica que uma maior quantidade de instâncias de um modelo pode ser expressa.

Como parte do desenvolvimento da abordagem proposta neste trabalho, uma DSL foi implementada. Essa DSL possibilita a especificação de entidades, serviços, aplicações específicas para as plataformas envolvidas no estudo e aplicações *multi-cloud* criadas a partir de implementações de referência específicas de plataforma. É possível estender a abordagem para tratar outros domínios e plataformas não tratados neste trabalho.

As operações CRUD nas plataformas estudadas são realizadas seguindo um modelo orientado a objetos de entidades persistentes. Portanto, a DSL obtida tem também esse foco. Para desenvolvê-la, foi utilizada a ferramenta xText² e sua representação é textual. Tal representação foi utilizada devido a sua simplicidade, facilidade de aprendizado e flexibilidade das ferramentas utilizadas em seu desenvolvimento. Todavia, é possível projetar mecanismos que especificam o domínio de maneira equivalente por meio de diagramas.

¹ Entre as diversas outras linguagens suportadas pelas plataformas, foi escolhida a linguagem Java para desenvolvimento deste trabalho. Portanto, deve-se levar em consideração que os exemplos de código e demais conceitos referem-se a testes feitos utilizando a linguagem citada. Todavia, qualquer outra linguagem suportada pelas plataformas é possível de ser adotada.

² <https://eclipse.org/Xtext/>

A gramática desenvolvida foi implementada utilizando sintaxe concreta e abstrata próprias do xText. Internamente, o xText utiliza *ANother Tool for Language Recognition (ANTLR)* - uma ferramenta para geração de *parsers* LL(*)³. Essencialmente, a gramática é um conjunto de regras de produção de cadeias em uma linguagem formal. Uma gramática permite especificar uma linguagem, suas regras descrevem como formar cadeias a partir do alfabeto da linguagem. O [Código-fonte 11](#) apresenta a gramática da linguagem desenvolvida.

Código-fonte 11 – Gramática em xText da DSL desenvolvida

```

1: grammar br.com.xtext.cloud.multipaas.dsl.MultiPaaS with org.eclipse.xtext.
    common.Terminals
2:
3: import "http://www.eclipse.org/xtext/common/JavaVMTypes" as jvmTypes
4: generate multiPaaS "http://www.com.br/xtext/cloud/multipaas/dsl/MultiPaaS"
5:
6: DomainModel:
7:   domainModel+=Elements*;
8:
9: Elements:
10:   Type | Application | Service | Multicloud;
11:
12: Entity:
13:   'entity' name=ID ('extends' superType=[Entity])?
14:   '{'
15:   properties+=Property*
16:   '}' ;
17:
18: Property:
19:   'property' name=ID ':' type=[Type] (many?='[]')?;
20:
21: Media:
22:   'media' name=ID '{'
23:   'type' type=STRING
24:   '}' ;
25:
26: Type:
27:   Entity | SimpleType | Media;
28:
29: SimpleType:
30:   'type' name=ID ('package' javaType=[jvmTypes::JvmType|QualifiedName])?;
31:

```

³ <http://www.antlr.org/>

```
32: QualifiedName:
33:   ID ('.' ID)*;
34:
35: Application:
36:   'cloudapplication' name=ID '{'
37:   ('type' ':' applicationType=ApplicationType)
38:   ('provider' ':' applicationProvider=ID)
39:   ('url' ':' url=STRING)
40:   // resources+=Resource*
41:   '}'
42:
43: Resource:
44:   'resource' name=ID 'from' entity=[Entity] '{'
45:   ('method' ':' method+=(ResourceType))*
46:   '}'
47:
48: enum ResourceType:
49:   create | update | delete | retrieve;
50:
51: enum ApplicationType:
52:   front | back;
53:
54: enum BoolType:
55:   true | false;
56:
57: Multicloud:
58:   'multicloud' name=ID '{'
59:   ('type' ':' applicationType=ApplicationType)
60:   ('provider' ':' applicationProvider=ID)
61:   cloudservices+=MulticloudServices*
62:   '}'
63:
64: MulticloudServices:
65:   'cloudapp' name=[Application] '{'
66:   appresources+=MulticloudService*
67:   '}'
68:
69: MulticloudService:
70:   'service' ':' name=[Service];
```

A linha 1 do [Código-fonte 11](#) refere-se à definição de pacote e nome da DSL. A linha 3

importa “jvmTypes” para utilização de tipos Java na sintaxe da linguagem desenvolvida. Assim, a linguagem será um dialeto de Java. As linhas de 12 a 20 apresentam o trecho da gramática que possibilita a especificação de entidades. As linhas 21 a 24 apresentam o trecho da gramática que especifica o tipo multimídia “media” a ser utilizada na aplicação. As linhas 35 a 41 apresentam o trecho que possibilita a especificação de aplicações *single-cloud*, mais especificamente refere-se aos atributos que possibilitarão a identificação do provedor em que a aplicação será implantada e quais recursos REST serão criados para a aplicação. As linhas 57 a 70 referem-se ao trecho da gramática que possibilita a especificação de *multi-cloud* pela composição de recursos de aplicações de uma nuvem única. Na linha 66 é possível perceber que um *multi-cloud* é composto por uma ou mais aplicações *single-cloud*. A seguir é detalhado os principais trechos de código de gramática e seu funcionamento.

A partir da gramática apresentada no [Código-fonte 11](#) é possível utilizar as ferramentas do Eclipse modeling⁴ para gerar o *ecore-diagram*. Um *ecore-diagram* uma representação gráfica para o metamodelo desenvolvido. A [Figura 20](#) apresenta o metamodelo da DSL criada.

Figura 20 – Metamodelo da DSL desenvolvida

Fonte: autoria própria

No metamodelo apresentado na [Figura 20](#), um modelo de domínio possibilita a criação de elementos de implementação. Esses elementos, por sua vez, podem ser entidades (região 1 da figura), uma aplicação de nuvem (região 2 da figura), uma aplicação (informações sobre implantação) *multi-cloud* (região 3) ou a definição de funções de *front-end*. Uma entidade (*Entity*)

⁴ <https://eclipse.org/modeling/>

representa uma classe persistente e pode possuir propriedades (*properties*) que representam seus atributos persistentes. A modelagem de *resource* (região 3) consiste em informar quais serão os controladores do *back-end* que deverão ser gerados considerando as entidades modeladas e quais métodos CRUD esses controladores devem implementar. Um *multi-cloud services* especifica quais controladores de *front-end* deverão ser gerados.

Em termos mais abstratos, na gramática da DSL desenvolvida, dentre principais elementos de domínio está o elemento “Entity” que refere-se ao modelo de entidades e classes persistentes do sistema. Um “Entity” é composto por “properties”, um conjunto de elementos do tipo “Property” que se refere às propriedades da entidade. Os valores para um “property” incluem tipos básicos do Java, um “media” ou uma entidade previamente declarada. O [Código-fonte 12](#) apresenta o trecho da gramática que se refere a modelagem de entidades e suas propriedades.

Código-fonte 12 – Trecho da gramática para especificação de entidades

```
1: Entity:
2:   'entity' name=ID ('extends' superType=[Entity])?
3:   '{'
4:   properties+=Property*
5:   '}' ;
6:
7: Property:
8:   'property' name=ID ':' type=[Type] (many?='[]')?;
9:
10: Media:
11:   'media' name=ID '{'
12:   'type' type=STRING
13:   '}' ;
14:
15: Type:
16:   Entity | SimpleType | Media;
17:
18: SimpleType:
19:   'type' name=ID ('package' javaType=[jvmTypes::JvmType|QualifiedName])?;
20:
21: QualifiedName:
22:   ID ('.' ID)*;
```

Na linha 2, a propriedade “name” pode receber uma *String* em formato de identificador; o atributo “superType” aponta para uma eventual entidade em que será definida a relação de herança. Na linha 4 é apresentado o atributo “properties” que refere-se às propriedades da entidade. O elemento “Media” refere-se a uma declaração de uma entidade a partir da qual é

possível gerar todo o código de manipulação de multimídia. Basicamente, é necessário informar o nome da media e em seguida seu formato (.mp3, mp4, etc.). O elemento “Type” refere-se ao tipo que uma entidade poderá ser. O elemento “SimpleType” faz referência aos tipos Java que podem ser incluídos na modelagem.

Assim, um modelo de domínio referente à uma entidade que especifica propriedades de uma “Conectividade” especificado utilizando a gramática apresentada no [Código-fonte 11](#) seria na forma:

Código-fonte 13 – Entidade especificada utilizando a DSL desenvolvida

```
1: entity Connectivity {  
2:   property bluetooth : String  
3:   property cell : String  
4:   property gps : boolean  
5:   property infrared : boolean  
6:   property wifi : String  
7: }
```

Conforme apresentado no [Código-fonte 13](#), uma entidade é semelhante a uma classe da programação orientada a objetos. A entidade é especificada por meio dos elementos “Entity” e “Property”. A linha 1 do [Código-fonte 13](#) apresenta a declaração da entidade, as linhas de 2 a 6 apresentam as propriedades.

O elemento “Application” se refere à especificação de uma aplicação *single-provider* e seus respectivos atributos. Uma aplicação *multi-cloud* consiste na composição de duas ou mais aplicações específicas de plataforma *single-cloud*. O elemento “Application” permite que sejam especificadas os atributos das aplicações *single-cloud*. O [Código-fonte 14](#) apresenta os elementos da gramática para especificação de “Application”.

Código-fonte 14 – Trecho da gramática para especificação de *cloudapplication*

```
1: Application:  
2:   'cloudapplication' name=ID '{'  
3:   ('type' ':' applicationType=ApplicationType)  
4:   ('provider' ':' applicationProvider=ID)  
5:   ('url' ':' url=STRING)  
6:   // resources+=Resource*  
7:   '}' ;
```

No [Código-fonte 14](#), “name” receberá uma *String* em formato de identificador (ID); “type” indica o tipo de aplicação que pode ser *front-end* (utilizando o valor “front”) ou *back-end* (utilizando o valor “back”); “provider” refere-se ao nome do provedor a que se destina essa

aplicação e consiste em uma *String* em formato de identificador (o nome do provedor será utilizado em tempo de execução do metaprograma para a seleção da transformação específica para o provedor especificado); “url” refere-se a uma URL para especificação da localização da aplicação após sua implantação e consiste em uma *String* que especifica uma url no formato “http://localizacao.do.serviço:porta”. O [Código-fonte 15](#) apresenta um modelo de domínio que, por exemplo, especifica uma aplicação *back-end* para a plataforma GAE.

Código-fonte 15 – Aplicação *single-cloud* especificada utilizando a DSL desenvolvida

```
1: cloudapplication gaeApplicationBackendGenerated {  
2:   type : back  
3:   provider : google  
4:   url: "1-dot-appexemplo.appspot.com"  
5:   //url : "localhost:8888"  
6: }
```

No [Código-fonte 15](#), a linha 1 apresenta a declaração da “cloudapplication”; a linha 2 apresenta a especificação do tipo da aplicação; a linha 3 apresenta o nome do provedor a que a essa aplicação se destina e, por fim, a linha 4 apresenta a URL em que a aplicação será implantada.

Por meio do elemento “Resource” é possível especificar os recursos da camada *back-end*. Um recurso é um serviço REST criado com uma finalidade específica. O recurso recebe requisições da camada *front-end* e emite respostas. Portanto, considerando as entidades do sistema é possível gerar recursos alinhados às operações CRUD. O [Código-fonte 16](#) apresenta um trecho da gramática que refere-se à especificação de recursos.

Código-fonte 16 – Trecho da gramática para especificação de recursos

```
1: Resource:  
2:   'resource' name=ID 'from' entity=[Entity] '{'  
3:     ('method' ':' method+=(ResourceType))*  
4:   '}' ;
```

No [Código-fonte 14](#), “name” poderá receber uma *String* em formato identificador; “entity” receberá o nome de uma entidade previamente cadastrada e, por fim, “method” receberá os valores referentes às operações *create*, *update*, *retrieve* e *delete*. Consequentemente, um modelo de domínio que especifica recursos para uma entidade considerando a DSL desenvolvida será:

Código-fonte 17 – Um recurso para a entidade Connectivity

```
1: resource connectivity from Connectivity {  
2:   method : create
```

```
3:  method : update
4:  method : retrieve
5:  method : delete
6: }
```

No [Código-fonte 17](#), a linha 1 refere-se à declaração do recurso; as linhas 2 a 5 apresentam a declaração dos métodos a serem gerados. Os valores possíveis para a variável *method* são: *create* | *update* | *delete* | *retrieve* representadas pelos elementos “ResourceType”. Na implementação de referência, a camada “Service” inclui os métodos da camada *front-end* de uma aplicação que farão a comunicação com o *back-end*. Um “Service”, portanto, se comunica com um “Resource” por meio de chamadas REST. Na gramática, um “Service” é representado por meio do elemento “MulticloudResource”.

Conforme apresentado anteriormente, uma aplicação *multi-cloud* é composta de duas ou mais aplicações *single-cloud*. É necessário, portanto, modelar a aplicação *single-cloud* e especificar quais recursos essa aplicação terá. Logo após, é necessário especificar a aplicação *multi-cloud* considerando as aplicações *single-cloud*. Um “Multicloud” se refere à aplicação *front-end* CORS que fará a composição de recursos do *back-end*. Esses recursos são especificados a partir do elemento “MulticloudServices”; “MulticloudServices” refere-se à especificação da ligação entre os métodos de *front-end* e seus respectivos *back-ends*. O [Código-fonte 18](#) apresenta um trecho da gramática para a especificação de um “cloudapp” *single-cloud*.

Código-fonte 18 – Trecho da gramática para especificação de aplicações *single-cloud*

```
1: MulticloudServices:
2:   'cloudapp' name=[Application] '{'
3:   appresources+=MulticloudService*
4:   '}';
```

No [Código-fonte 18](#), “name” apontará para uma aplicação *single-cloud* previamente declarada e informará por meio de “appresources” quais serão os “Services” que deverão ser gerados considerando o *front-end* da aplicação CORS. O [Código-fonte 19](#) apresenta um modelo de domínio para uma aplicação *single-cloud* considerando a DSL desenvolvida.

Código-fonte 19 – Uma aplicação *single-cloud*

```
1: cloudapp  gaeApplicationBackendGenerated {
2:   service : android
3:   service : battery
4:   service : camera
5:   service : gadget
6:   service : discipline
```

```
7: }
```

No [Código-fonte 19](#), a linha 1 apresenta a declaração do elemento de domínio. As linhas de 2 a 6 apresentam a declaração de “services”. Conforme mencionado anteriormente, considerando a arquitetura de referência, um “service” comunica com a camada de *back-end* da aplicação. Ou seja, o “service” faz chamadas a recursos especificados por meio de elementos “Resources”. Portanto, nas linhas 2 a 6 especifica-se quais elementos de código em AngularJS serão gerados visando a comunicação com o *backend*.

É possível perceber que os métodos de *front-end* são referenciados como “services” e os métodos de *back-end* são referenciados como “resources”. Em outras palavras, um “resource” é um recurso/método VRaptor REST de *back-end* e um “services” é um método JavaScript que utiliza CORS e AJAX para utilização dos recursos e chamadas aos métodos de *back-end*. Essas abstrações foram definidas em função dos principais conceitos de domínio que possibilitam que aplicações inter-operem. A linguagem implementada é relacionada ao domínio da aplicação, pois ela refere-se diretamente ao tipo, funcionalidades e aos módulos/aplicações distribuídas que serão geradas a partir do modelo de domínio que a linguagem possibilita especificar.

Um elemento “Multicloud” refere-se efetivamente à composição de aplicações *single-cloud*. Como a estratégia da implementação de referência para composição de recursos tem por base CORS e chamadas a recursos REST de aplicações *single-cloud* de *back-end*, é necessário, portanto, modelar a composição dessas aplicações. Assim, a ideia central no que se refere às aplicações *multi-cloud* é que recursos de aplicações *single-cloud* sejam compostos visando alcançar uma nova aplicação formada pela composição de outras. O [Código-fonte 20](#) apresenta um trecho da gramática que possibilita a especificação de aplicações *multi-cloud*.

Código-fonte 20 – Trecho da gramática para aplicações *multi-cloud*

```
1: Multicloud:
2:   'multicloud' name=ID '{'
3:   ('type' ':' applicationType=ApplicationType)
4:   ('provider' ':' applicationProvider=ID)
5:   cloudservices+=MulticloudServices*
6:   '}';
```

No [Código-fonte 20](#), “multicloud” pode receber uma *String* em formato identificador; “type” especifica se a estratégia de composição de *single-cloud* será realizada no *front-end* (com CORS, chamadas Ajax e Angular como na Figura 18) ou *back-end* (por meio da composição de aplicações em conformidade com a arquitetura apresentada na Figura 19); “provider” especifica para qual provedor será gerada a aplicação e, por fim, “cloudservices” especifica quais serão as “cloudapp” que serão compostas para formar a aplicação *multi-cloud*. Um exemplo de um

modelo de domínio que realiza a especificação de aplicações *multi-cloud* e, conseqüentemente, a composição de aplicações *single-cloud* é apresentado no [Figura 27](#).

Código-fonte 21 – Aplicação *multi-cloud*

```
1: multicloud phonecatCORS {
2:   type : front
3:   provider : google
4:   cloudapp gaeApplicationBackendGenerated {
5:     service : android
6:     service : battery
7:     service : camera
8:     service : gadget
9:     service : discipline
10:  }
11:
12:  cloudapp phonecatHeroku{
13:    service : connectivity
14:    service : display
15:    service : hardware
16:    service : storage
17:  }
18: }
```

No [Código-fonte 21](#), a linha 1 refere-se à declaração da “multicloud”. As linhas de 4 a 10 referem-se à aplicação “gaeApplicationBackendGenerated” que será composta na *multi-cloud* junto com a aplicação “phonecatHeroku” (linhas 12 a 17). É possível perceber, portanto, que o *multi-cloud* “phonecatCORS” será uma aplicação de *front-end* gerada para o GAE e composta pelas aplicações “gaeApplicationBackendGenerated” e “phonecatHeroku”.

A gramática apresentada no [Código-fonte 11](#) pode ser estendida para incluir outros elementos de implementação e/ou domínios de aplicação. Elementos específicos de plataforma não são considerados na meta-modelagem apresentada. Assim, uma modelagem que utiliza a linguagem desenvolvida é independente de plataforma. Elementos específicos de plataforma são embutidos nas transformações. A aplicação *multi-cloud* gerada é uma aplicação global composta por “sub-aplicações” específicas de plataforma. Uma extensão possível para a linguagem desenvolvida seria, por exemplo, uma meta-modelagem visando implantação automática da aplicação conforme as especificidades de cada provedor. Seria possível ainda integrar a abordagem apresentada neste trabalho com outras, como a linguagem desenvolvida por [Thiery et al. \(2014\)](#).

5.3 Transformações para plataformas específicas

Uma transformação consiste em um meta-programa, um *template* que possibilita a geração de código. Uma transformação é criada considerando-se uma implementação de referência e um domínio, contém dados e metadados relacionados ao código final aplicação. Dados vindos do modelo de domínio servem como entrada para o *template* que irá gerar o código final da aplicação. O Capítulo 3 apresenta todos os conceitos de MDE. A Figura 21 apresenta, em síntese, o exemplo do funcionamento de um *template* implementado em *Java Emmitter Templates* (JET).

Figura 21 – Funcionamento de um template MDE em JET

Fonte: (POSSATTO; LUCRÉDIO, 2015)

O exemplo da Figura 21 apresenta um modelo de domínio “i1.xml” que serve de entrada para a transformação “t1.jet” e quando combinados possibilita a geração do código final “f1.txt”. Como é possível observar, o *template* possui o código que corresponde a um meta-programa (programa que possibilita gerar programa). As linhas 3, 6, 7 e 9 do *template* apresentam códigos parecidos com *tags Java Server Pages* (JSP). Essas *tags* são código JET. As linhas 1, 2, 4, 5 e 8 referem-se a código estático que será gerado considerando a lógica implementada nas *tags* do *template*. O arquivo “f1.txt” apresenta o código gerado após a execução do *template*.

A ideia por trás de *templates* implementados em Xtend⁵ ou Acceleo⁶ é mesma do JET, as principais mudanças ao nível de *template* referem-se à sintaxe da implementação do mesmo. Os geradores JET, Acceleo e Xtend utilizam o metamodelo ecore do *Eclipse Modeling Project*⁷, para desenvolvimento de DSLs considerando tecnologias Microsoft é possível utilizar o DSL Tools⁸. Neste trabalho foi escolhida a ferramenta xTend porque a mesma está diretamente alinhada a aplicações em linguagem Java. O xTend é um dos *frameworks* do *Eclipse Modelling Project*, possui uma linguagem com sintaxe própria muito próxima da linguagem Java e de relativamente

⁵ <<http://www.eclipse.org/xtend/>>

⁶ <<https://www.eclipse.org/acceleo/>>

⁷ <<https://eclipse.org/modeling/>>

⁸ <https://msdn.microsoft.com/en-us/library/bb126259.aspx>

baixa curva de aprendizado. O [Código-fonte 22](#) apresenta um trecho de código em xTend. Trata-se do trecho do gerador de um controlador Angular, pertencente à camada *controller* do *front-end* da aplicação *multi-cloud*.

Código-fonte 22 – Trecho de código em xTend

```

1: class GAECORSFrontendTransformationsController implements IGenerator {
2:     override void doGenerate(Resource resource, IFileSystemAccess fsa) {
3:         println("== gae.cors.frontend.transformations.controller==")
4:         for (s : resource.allContents.toIterable.filter(Multicloud)) {
5:             s.compile(fsa)
6:         }
7:
8:         fsa.generateFile("common.js", fsa.generateCommon)
9:     }
10:    def compileJSON(Entity entity) {
11:        var jsonResult = ""
12:        jsonResult += entity.name.toFirstLower + ":" + "{"
13:        jsonResult += "\"id\" : $scope.id"
14:        for (property : entity.properties) {
15:            if (property.type.eClass.name.equals("Entity")) {
16:                jsonResult += "," + "\"" + property.name.toFirstLower + "\"" + ":"
17:                + "JSON.parse(" + "$scope." + property.name.toFirstLower + ")"
18:            } else {
19:                if (property.type.eClass.name.equals("SimpleType")) {
20:                    jsonResult += ",\"" + property.name + "\"" + ":" + "$scope." +
21:                    property.name + "!= undefined ? $scope." +
22:                    property.name + ":" + " \'\\' "
23:                }
24:            }
25:        }
26:        jsonResult += "}"
27:        return jsonResult
28:    } // end compile
29:
30:    def controller(MulticloudResource service, MulticloudServices services,
31:        Multicloud multicloud)'''
32:        ... cod xTend para o template aqui...
33:    }'''

```

Como é possível observar, no [Código-fonte 22](#), a sintaxe do código é bastante parecida com a da linguagem de programação Java. Esse código é traduzido para Java pelo xTend

para código Java que constituirá o programa que possibilitará a geração de código de fato. A linha 1 apresenta a declaração da classe “GAECORSFrontendTransformationController”; essa classe possibilita a geração de um controlador Angular completo. A linha 3 é equivalente ao “System.out.println()” do Java. A linha 4 apresenta uma busca nos modelos de domínio por meio do método “resource.allContents.toIterable.filter(Multicloud)”. Ao encontrar um elemento do tipo “Multicloud” o xTend chama o método “compile” passando o parâmetro “fsa” do tipo ‘`IFileSystemAccess`’ esse objeto permite acessar o sistema de arquivos para geração. As linhas entre 10 e 26 apresentam um serializador de JSON. Ou seja, recebe uma entidade e escreve no controlador Angular sua representação em formato JSON. A linha 28 apresenta o método “controller”, como é possível perceber esse método não se inicia com uma chave “{” mas com acentos agudos. Os métodos que se iniciam dessa maneira, diferente do método “compileJSON” por exemplo, devem ser escritos em outra linguagem também pertencente ao xTend, mas que é totalmente diferente da apresentada no método “compileJSON”. Essa linguagem é o que se pode chamar de linguagem de *template* do xTend e mistura código final com *tags* xTend⁹ como o exemplo JET apresentado na Figura 21.

Assim, utilizando xTend, a linguagem específica de domínio apresentada na Seção 5.2 e a implementação de referência *multi-cloud* apresentada no Estudo de Caso apresentado na Capítulo 4, foram desenvolvidos *templates* que possibilitam a geração das aplicações *multi-PaaS* com CORS. Esses elementos constituem **uma abordagem dirigida por modelos para desenvolvimento de aplicações *multi-PaaS***.

Os geradores desenvolvidos consideram as plataformas selecionadas neste estudo. Foi necessário desenvolver transformações que considerem as especificidades de cada plataforma. Todo o conhecimento específico de plataforma é embutido nessas transformações. Também é embutido todo conhecimento relacionado aos componentes tecnológicos envolvidos na construção da aplicação. Dessa maneira, os desenvolvedores, ao trabalhar somente com os modelos de domínio, não necessariamente necessitam ter conhecimento a respeito dos elementos de implementação da abordagem, apenas da linguagem de domínio.

Em direção a implementação da abordagem foi necessário desenvolver as seguintes 12 transformações:

1. **GAE *Back-end* DAO:** para geração de DAO considerando o objectify;
2. **GAE *Back-end* Controller:** para geração de controladores em nível de *back-end* para a plataforma GAE;
3. **GAE *Back-end* Entities:** para geração das classes de modelo e MOR da aplicação *back-end* GAE;

⁹ A documentação completa da linguagem de *templates* do JET pode ser encontrada no endereço: http://www.eclipse.org/xtend/documentation/203_xtend_expressions.html

4. **GAE CORS *front-end* page:** para geração das páginas HTML da aplicação *front-end* considerando a implantação no GAE;
5. **GAE CORS *front-end* controller:** para geração dos controladores Angular da aplicação de *Front-end* considerando a implantação no GAE;
6. **GAE CORS *front-end* service:** para geração da camada services da aplicação considerando implantação na GAE;
7. **Heroku CORS *front-end* page:** para geração das páginas HTML da aplicação *front-end* considerando implantação na Heroku;
8. **Heroku CORS *front-end* controller:** para geração dos controladores Angular da aplicação *front-end* considerando implantação na Heroku;
9. **Heroku CORS *front-end* service:** para geração da camada services da aplicação considerando a implantação na Heroku;
10. **Heroku *Back-end* DAO:** para geração de DAO considerando o MongoDB;
11. **Heroku *Back-end* Controller:** para geração de controladores a nível de *back-end* para a plataforma Heroku;
12. **Heroku *Back-end* Entities:** para geração das classes de modelo e MOR da aplicação Heroku.

Pode-se observar que foram desenvolvidas transformações que possibilitam o desenvolvimento de aplicações *back-end* para as plataformas GAE e Heroku e aplicações CORS *front-end*, que também podem ser implantadas no GAE ou Heroku. As aplicações *front-end* são independentes de plataforma. As especificidades de plataforma, arquivos de configuração e formato de projetos são embutidas nas transformações.

Seguindo a mesma estratégia, é possível adicionar qualquer outra plataforma, basta desenvolver as implementações de referência e abstrair utilizando os conceitos e ferramentas apresentados neste trabalho. Em síntese, para a inclusão de uma nova plataforma, é necessário:

1. Estudar as especificidades de plataforma;
2. Desenvolver uma aplicação de exemplo (implementação de referência) para a plataforma desejada;
3. Identificar elementos de abstração da implementação;
4. Desenvolver um conjunto de transformações para gerar código considerando a implementação de referência;

5. Desenvolver um conjunto de transformações que considera a estratégia de interoperabilidade adotada.

A Figura 22 apresenta uma visão da abordagem desenvolvida e os passos necessários para criação de cada um dos elementos que a compõem.

Figura 22 – Elementos da abordagem

Fonte: autoria própria.

Na legenda da Figura 22 estão descritas as duas perspectivas de elementos centrais para a criação de uma abordagem MDE: elementos criados por pessoas (implementado pelo engenheiro) e os criados por geração automática de código (geradores). Os elementos 1, 3, 4 e 5 são criados por desenvolvedores. Em um ciclo MDE os componentes 1 e 5 são criados pelo chamados engenheiros de domínio, enquanto os modelos de domínio são utilizados pelos engenheiros de aplicação. Idealmente, os engenheiros de aplicação não precisam ter conhecimento dos elementos internos dos geradores de código (ou seja, não necessitam de conhecimento sobre como o código é gerado efetivamente), apenas da linguagem de domínio, que consiste de elementos de mais alto nível de abstração.

5.4 Considerações finais

Em síntese, este capítulo apresentou o desenvolvimento de uma abordagem dirigida por modelos para desenvolvimento de aplicações *multi-PaaS*. Essa abordagem consiste na construção de uma linguagem específica de domínio que possibilita a abstração de detalhes específicos de plataforma e geração de uma aplicação em conformidade com uma implementação de referência que faz a combinação de recursos em nível da camada *services* de *front-end*.

A modelagem implementada remete à definição de interoperabilidade seguida neste trabalho, possibilitar que sistemas *inter-operem* (definição apresentada por [Nogueira et al. \(2016b\)](#)). Alternativamente, podem ser utilizadas técnicas como linhas de produto e outras. O problema da interoperabilidade na perspectiva de comunicação entre módulos separados e combinação de recursos heterogêneos distribuídos remete à composição de interesses transversais apresentada por [Whittle, Araújo e Moreira \(2006\)](#) e que é uma das soluções possíveis que também pode ser abstraída utilizando-se de MDE, mas considerar aspectos não é o foco deste trabalho. Para a MDE, embora seja possível utilizar geração parcial, a ideia é que os modelos passem a ser o código. Portanto, considerando o ciclo de vida de um processo MDE genérico, os modelos em si consistem em uma aplicação interoperável. O capítulo seguinte apresenta um Estudo de Caso em que a DSL desenvolvida neste projeto é utilizada para desenvolver um sistema.

DESENVOLVIMENTO DE APLICAÇÕES *MULTI-PAAS* COM MDE

Este capítulo apresenta dois Estudos de Caso conduzidos com o objetivo de explorar o desenvolvimento de uma aplicação *multi-cloud* utilizando a abordagem dirigida por modelos, desenvolvida nesta tese, conforme descrita no [Capítulo 5](#). A [Seção 6.2](#) apresenta a aplicação utilizada como referência para condução dos estudos; a [Seção 6.3](#) apresenta um Estudo de Caso referente à modelagem de domínio utilizando a DSL desenvolvida; a [Seção 6.4](#) apresenta a aplicação gerada pela abordagem; por fim, a [Seção 6.5](#) apresenta um segundo Estudo de Caso sobre como o problema da portabilidade pode também ser tratado utilizando a abordagem desenvolvida.

6.1 Considerações Iniciais

Conforme apresentado anteriormente, este trabalho teve por objetivo o desenvolvimento de uma abordagem dirigida por modelos para desenvolvimento de aplicações *multi-PaaS*. Em direção a esse objetivo, foram desenvolvidas aplicações de referência para as plataformas GAE e Heroku e uma estratégia de interoperabilidade com foco em CORS e serviços REST. Após o desenvolvimento das aplicações de referência foi desenvolvida uma DSL que permite fazer a modelagem dos sistemas em alto nível de abstração. Transformações em xTend foram utilizadas para gerar o código final das aplicações a partir de modelos de domínio especificados com a DSL.

Foi realizado um Estudo de Caso visando responder a seguinte questão: “como um sistema *multi-PaaS* pode ser desenvolvido utilizando a abordagem desenvolvida?”. Portanto, para atingir esse objetivo foi necessário desenvolver um sistema utilizando todos os elementos implementados. A ideia central do estudo foi a mesma seguida na abordagem desenvolvida nesta tese, desenvolver um sistema *multi-cloud* a partir de uma aplicação existente considerando todos

os requisitos da aplicação. As seções a seguir apresentam o Estudo de Caso conduzido.

6.2 A aplicação phonecat

A phonecat¹ consiste em uma aplicação implementada em AngularJS². A aplicação é baseada no “Google Phone Gallery”, uma antiga loja virtual de *gadgets* do google³. A aplicação foi selecionada por ser o *case* utilizado pela equipe do Google no tutorial oficial do *framework* AngularJS⁴ e também por ser exatamente o tipo de aplicação a que este projeto se destina. A phonecat foi implementada originalmente pela equipe do Google é uma aplicação Web em três camadas conforme a lógica do AngularJS. Por meio de chamada REST a aplicação faz chamadas a arquivos armazenados no servidor. Esses arquivos contém os dados da aplicação que são enviados aos controladores Angular pela requisição e à interface utilizando o *two-way data bind*. A aplicação possui duas páginas, uma para listagem de todos os *gadgets* (Figura 23) armazenados e outra para apresentação de todos os dados relacionados a um *gadget* (Figura 24).

Figura 23 – Página Web contendo a lista de *gadgets*

Fonte: autoria própria.

O Código-fonte 23 apresenta o conteúdo de um dos arquivos contendo dados relacionados a um *gadget* como apresentado na Figura 24.

Código-fonte 23 – Dados originais de um gadget da aplicação phonecat em formato JSON

¹ <<https://github.com/angular/angular-phonecat>>

² <<https://docs.angularjs.org/tutorial>>

³ <<http://web.archive.org/web/20131215082038/http://www.android.com/devices>>

⁴ <<https://docs.angularjs.org/tutorial>>

Figura 24 – Página Web que apresenta os dados de um determinado *gadget*

MOTOROLA XOOM™

MOTOROLA XOOM has a super-powerful dual-core processor and Android™ 3.0 (Honeycomb) — the Android platform designed specifically for tablets. With its 10.1-inch HD widescreen display, you'll enjoy HD video in a thin, light, powerful and upgradeable tablet.

Availability and Networks Availability Verizon	Battery Type Other (3250 mAh) Talk Time 24 hours Standby time (max) 336 hours	Storage and Memory RAM 1000MB Internal Storage 32000MB	Connectivity Network Support CDMA 800 /1900 LTE 700, Rx diversity in all bands WiFi 802.11 a/b/g/n Bluetooth Bluetooth 2.1 Infrared ✗ GPS ✓	Android OS Version Android 3.0 UI Android	Size and Weight Dimensions 249.0 mm (w) 168.0 mm (h) 12.7 mm (d) Weight 726.0 grams	Display Screen size 10.1 inches Screen resolution WXGA (1200 x 800) Touch screen ✓	Hardware CPU 1 GHz Dual Core Tegra 2 USB USB 2.0 Audio / headphone jack 3.5mm FM Radio ✗ Accelerometer ✓
--	---	---	---	--	--	--	--

Camera Primary 5.0 megapixels Features Flash, Video	Additional Features Front-facing camera. Sensors: proximity, ambient light, barometer, gyroscope.
--	---

Fonte: autoria própria.

```

1: {
2:   "additionalFeatures": "Front Facing 1.3MP Camera",
3:   "android": {
4:     "os": "Android 2.2",
5:     "ui": "Dell Stage"
6:   },
7:   "availability": [
8:     "T-Mobile"
9:   ],
10:  "battery": {
11:    "standbyTime": "",
12:    "talkTime": "",
13:    "type": "Lithium Ion (Li-Ion) (2780 mAh)"
14:  },
15:  ... código ...
16:  // a seguir, imagens originalmente não armazenadas em um banco de dados,
    mas em arquivos
17:  "id": "dell-streak-7",
18:  "images": [
19:    "img/phones/dell-streak-7.0.jpg",
20:    "img/phones/dell-streak-7.1.jpg",
21:    "img/phones/dell-streak-7.2.jpg",
22:    "img/phones/dell-streak-7.3.jpg",
23:    "img/phones/dell-streak-7.4.jpg"
24:  ],

```

```
25:     ... restante do código...
26: }
```

É possível perceber que, embora haja uma estrutura lógica nos dados, não existe uma estratégia mais sofisticada de armazenamento de dados na aplicação. Ou seja, não foi utilizando nenhum sistema de gerenciamento de banco de dados (SGBD), apenas arquivos. Até as imagens utilizadas são passadas por URL (linhas 44 a 50). Outro ponto em destaque é que a aplicação possui dados redundantes, há um arquivo que armazena os dados da listagem de todos os *gadgets*, a lista não é gerada a partir de dados previamente cadastrados. O [Código-fonte 24](#) apresenta um trecho de código do arquivo.

Código-fonte 24 – Dados para a listagem de gadgets cadastrados

```
1: [
2:   {
3:     "age": 0,
4:     "id": "motorola-xoom-with-wi-fi",
5:     "imageUrl": "img/phones/motorola-xoom-with-wi-fi.0.jpg",
6:     "name": "Motorola XOOM\u2122 with Wi-Fi",
7:     "snippet": "The Next, Next Generation\r\n\r\nExperience the future
with Motorola XOOM with Wi-Fi, the world's first tablet powered by
Android 3.0 (Honeycomb).\"
8:   },
9:   {
10:    "age": 1,
11:    "id": "motorola-xoom",
12:    "imageUrl": "img/phones/motorola-xoom.0.jpg",
13:    "name": "MOTOROLA XOOM\u2122",
14:    "snippet": "The Next, Next Generation\r\n\r\nExperience the future with
MOTOROLA XOOM, the world's first tablet powered by Android 3.0 (
Honeycomb).\"
15:   },
16:   ... gadgets ...
17: ]
```

As linhas 2 a 8 apresentam os dados de um *gadget* que aparecerá na listagem. A listagem completa deve ser inserida manualmente considerando todos os *gadgets* cadastrados. A [Figura 25](#) apresenta a estrutura da aplicação. Na pasta “img” são armazenadas as imagens da aplicação em formato “.jpg”. A pasta “js” armazena todo o código JavaScript da aplicação. A pasta “partials” armazena as páginas HTML. Por fim, a pasta “phones” consiste no banco de dados de informações da aplicação. Cada arquivo dessa pasta refere-se a um *gadget* da aplicação.

Figura 25 – Estrutura da aplicação phonecat

Fonte: autoria própria.

A migração manual dessa aplicação, para o GAE, por exemplo, seria bastante complexa, pois seria necessário desenvolver todos os elementos específicos de plataforma, fazer o MOR manualmente para cada classe da aplicação, armazenamento de mídia considerando as especificidades da plataforma e apresentação de mídia no *front-end* considerando que não mais seria passada a URL da imagem, mas dados binários vindos de um banco de dados. Seria necessário também o desenvolvimento de serviços e readaptação do formato de dados e requisições do *front-end* ao *back-end*.

Pensar em uma aplicação *multi-cloud* que utiliza recursos de mais de uma plataforma seria ainda mais complexo. Considerando, por exemplo, que a aplicação armazenaria dados de

multimídia na plataforma Heroku, seria necessário desenvolver todos os componentes específicos de plataforma, fazer MOR para o MongoDB (ou outra estratégia como Redis ou outro serviço), implementar serviços, *front-end* entre outros elementos. Além disso, a aplicação não possui um sistema de administração para cadastro de novos *gadgets*, seria, portanto necessário desenvolvê-lo.

6.3 Modelagem de domínio

Utilizando a DSL desenvolvida neste projeto, por outro lado, toda a aplicação seria gerada a partir da modelagem de domínio. Para isso, seria necessário apenas modelar as entidades considerando o modelo de dados da aplicação. Decidir quais serviços seriam implementados e como seria a interoperabilidade da aplicação e como seriam distribuídas as funcionalidades. Seguindo o exemplo de aplicação *multi-cloud* citado no [Capítulo 4](#), os serviços de multimídia deveriam ser desenvolvidos considerando a plataforma GAE enquanto parte das funcionalidades deverá ser implantada na Heroku. Todavia, esse é apenas um dos cenários possíveis. Seria possível fazer o oposto também, ou seja, serviços de multimídia implantados na Heroku e demais funcionalidades no GAE. O [Código-fonte 25](#) apresenta um trecho do código da modelagem de domínio da aplicação *phonecat* utilizando a DSL desenvolvida.

Código-fonte 25 – Modelagem de domínio da aplicação *phonecat*

```
1: /* application's modeling */
2: cloudapplication gaeApplicationBackendGenerated {
3:   type : back
4:   provider : google
5:   url: "1-dot-appexemplo.appspot.com"
6: }
7:
8: cloudapplication herokuApplicationBackendGenerated{
9:   type : back
10:  provider : heroku
11:  url: "heroku-phonecat-backend.herokuapp.com"
12: }
13:
14: /***** media's modeling *****/
15: media Image {
16:   type "jpeg"
17: }
18:
19: /***** entities' modeling *****/
20: entity Android {
```

```
21:    property os : String
22:    property ui : String
23: }
24:
25: ... other entities...
26:
27: entity Gadget {
28:    property additionalFeatures : String
29:    property android : Android
30:    property availability : String
31:    property battery : Battery
32:    property camera : Camera
33:    property connectivity : Connectivity
34:    property description : String
35:    property display : Display
36:    property hardware : Hardware
37:    property images : Image[]
38:    property name : String
39:    property height : String
40:    property width : String
41:    property lenght : String
42:    property weight : String
43:    property storage : Storage
44:    property snippet : String
45: }
46:
47: /***** resources modeling *****/
48:
49: resource android from Android {
50:    method : create
51:    method : update
52:    method : retrieve
53:    method : delete
54: }
55:
56: ...other resources...
57:
58:
59: /** **/
60:
61: /***** modeling multicloud *****/
62:
```

```
63: multicloud phonecatCORS {
64:   type : front
65:   provider : google
66:   cloudapp gaeApplicationBackendGenerated {
67:     service : android
68:     service : battery
69:     service : camera
70:     service : gadget
71:     service : discipline
72:   }
73:
74:   cloudapp phonecatHeroku{
75:     service : connectivity
76:     service : display
77:     service : hardware
78:     service : storage
79:   }
80: }
```

Um elemento “cloudapplication” (linhas 2 e 9) declara uma aplicação a ser implantada em um provedor. Para isso são especificadas o tipo da implementação de referência, se é um *back-end* ou um *front-end*, o provedor a ser implantado e a URL da aplicação. Um elemento “media” informa para aplicação que tipo de media será utilizada. Os elementos “entity” referem-se ao modelo de dados da aplicação. Os elementos “resource” (linha 52) informa o nome da URL do recurso REST, para qual entidade esse recurso REST será gerado e quais os métodos do CRUD serão gerados para o recurso. Um elemento “multicloud” refere-se a modelagem da aplicação CORS. Essa aplicação é composta por elementos “cloudapp” que informam quais aplicações *back-end* serão consideradas e quais os serviços “service” serão gerados e ligados ao *back-end*. Um elemento “service” especifica para quais entidades deverão ser geradas funcionalidades de *front-end* da aplicação.

6.4 Aplicação gerada

Como mencionado anteriormente, a modelagem de domínio apresentada serve de entrada para um conjunto de transformações que geram a aplicação *multi-PaaS*. Assim, foram gerados os seguintes artefatos conforme a modelagem:

- Uma aplicação *back-end* em conformidade com a aplicação apresentada no [Capítulo 4](#) denominada “gaeApplicationBackendGenerated”. Essa aplicação é específica para a plataforma GAE. A URL dessa aplicação após implantação na plataforma é “1-dot-

appexemplo.appspot.com”, para efeitos de testes locais é possível colocar uma URL local. Para essa aplicação serão gerados recursos modelados de acordo com os services especificados no no “cloudapp”. Também serão geradas páginas Web e todo um *front-end* que possibilitará a realização de operações CRUD sobre o datastore.

- Uma aplicação **back-end** denominada “herokuApplicationBackendGenerated” com características semelhantes a aplicação anterior. A aplicação é implantada e acessada por meio da URL “heroku-phonecat-backend-herokuapp.com”.
- Uma aplicação **front-end CORS** denominada “phonecatCORS” que compõe funcionalidades de duas “cloudapp”.

A Figura 26 apresenta um mapeamento entre os elementos da gramática da DSL desenvolvida e a aplicação gerada.

Figura 26 – Mapeamento entre aplicação gerada e elementos da gramática

Fonte: autoria própria.

Um elemento “multicloud” consiste na aplicação final, formada pela composição de duas ou mais “cloudapplication”. Uma “cloudapplication” por sua vez é composta por duas grandes camadas, uma *front-end* e outra *back-end*. Os serviços a serem gerados para camada de *front-end* são especificados por meio do elemento “service”. Esse serviço por sua vez se comunica com um elemento “resource” do *back-end*. Um “resource” é gerado considerando um elemento “entity” que por sua vez consiste na especificação das entidades do sistema.

6.4.1 Aplicação CORS front-end

A camada “services” do *front-end* informa para a aplicação CORS onde estão localizados os serviços de *back-end*. A Figura 27 apresenta a estrutura da aplicação gerada. Na prática a aplicação final consiste de três aplicações que funcionam em conjunto. A aplicação CORS

Figura 27 – Aplicação gerada

Fonte: autoria própria.

de *front-end* é independente de plataforma e *cross-browser*⁵. A abordagem está gerando o projeto conforme exigido pela plataforma GAE, mas qualquer tecnologia poderia ser utilizada alternativamente. A aplicação *front-end* se comunica com os servidores de *back-end* por meio de chamadas AJAX aos recursos especificados e utilizando o formato JSON.

A aplicação gerada está em conformidade com a arquitetura de referência apresentada na Figura 3. A Figura 28 apresenta a interface da aplicação gerada. Foi utilizado o twitter bootstrap com SBAAdmin, mas qualquer outro modelo poderia ter sido utilizado conforma necessidade da empresa.

Para o usuário final, não é possível perceber que a aplicação é *multi-PaaS*, apenas os desenvolvedores por meio da análise do código da aplicação. A entidade “gadget” é composta por várias outras e utiliza recursos das duas plataformas combinados. O Código-fonte 26 apresenta o controlador gerado para a entidade.

Código-fonte 26 – Exemplo de Controlador Angular gerado

⁵ Uma aplicação *cross-browser* é uma aplicação web que funciona independentemente do navegador utilizado.

Figura 28 – Interface da aplicação gerada

phonecatCORS Admin

Search... Q

- Dashboard
- Android
- Battery
- Camera
- Gadget
- GadgetMedia
- Discipline
- Connectivity
- Display
- Hardware
- Storage

Android Management

Manage Android

Os

Enter Os

UI

Enter UI

Submit Button

Search: Search value Sort by: ▾

Existing Android

Os	UI	
Android 2.2	Dell Stage	
Android 2.2	Dell Stage	

Fonte: autoria própria.

```

1: var application = angular.module('application', [ 'BackendLocation' ]);
2: application
3:   .controller(
4:     'GadgetController',
5:     [
6:       '$scope',
7:       '$http',
8:       'backendLocation',
9:       function($scope, $http, backendLocation) {
10:         // ----- get all Android -----
11:         $http(
12:           {
13:             method : 'GET',
14:             url : backendLocation
15:               .getGaeApplicationBackendGeneratedLocation()
16:               + '/android/list'
17:           })
18:         .success(
19:           function(response) {
20:             if (response.success == true) {
21:               $scope.rowCollectionAndroid = response.data;
22:             } else {

```

```

23:             $scope.message = "an error has occurred";
24:         }
25:     });
26:     // ----- get all Hardware -----
27:     $http(
28:         {
29:             method : 'GET',
30:             url : backendLocation
31:                 .getPhonecatHerokuLocation()
32:                 + '/hardware/list'
33:         })
34:         .success(
35:             function(response) {
36:                 if (response.success == true) {
37:                     $scope.rowCollectionHardware = response.data;
38:                 } else {
39:                     $scope.message = "an error has occurred";
40:                 }
41:             });
42:     //... requests for all other entities .... -----
43:
44:     // ----- generating: CRUD methods -----
45:
46:     $scope.create = function() {
47:         var data = {
48:             gadget : {
49:                 "id" : $scope.id,
50:                 "additionalFeatures" : $scope.additionalFeatures !=
undefined ? $scope.additionalFeatures
51:                     : '',
52:                 "android" : JSON.parse($scope.android),
53:                 "availability" : $scope.availability != undefined ?
$scope.availability
54:                     : '',
55:                 "battery" : JSON.parse($scope.battery),
56:                 "camera" : JSON.parse($scope.camera),
57:                 "connectivity" : JSON
58:                     .parse($scope.connectivity),
59:                 "description" : $scope.description != undefined ?
$scope.description
60:                     : '',
61:                 "display" : JSON.parse($scope.display),

```



```

62:             "hardware" : JSON
63:                 .parse($scope.hardware),
64:             "name" : $scope.name != undefined ? $scope.name
65:                 : '',
66:             "height" : $scope.height != undefined ? $scope.height
67:                 : '',
68:             "width" : $scope.width != undefined ? $scope.width
69:                 : '',
70:             "lenght" : $scope.lenght != undefined ? $scope.lenght
71:                 : '',
72:             "weight" : $scope.weight != undefined ? $scope.weight
73:                 : '',
74:             "storage" : JSON.parse($scope.storage),
75:             "snippet" : $scope.snippet != undefined ? $scope.
        snippet
76:                 : ''
77:         }
78:     }
79:
80:     $http(
81:         {
82:             method : 'POST',
83:             url : backendLocation
84:                 .getGaeApplicationBackendGeneratedLocation()
85:                 + '/gadget/create',
86:             data : data,
87:             headers : {
88:                 "Content-Type" : "application/json"
89:             }
90:         })
91:     .then(
92:         function(response) {
93:             if (response.status == 200) {
94:                 if (response.data.data.success == true) {
95:                     $scope.message = "Entity registered
        successfully!";
96:                     $scope.id = undefined;
97:                     $scope.list();
98:                 }
99:             } else {
100:                 $scope.message = "Requisition was sent, but was
        not possible to persist the entity, please try again!"

```

```
101:                }  
102:                });  
103:            } // endcreate  
104:        } ] );
```

As linhas 11 a 24 apresentam uma requisição AJAX para o recurso “android’/list’ que recupera todos android cadastrados. É possível observar que a requisição é feita à aplicação de *back-end* hospedada no GAE (linha 15). As linhas 26 a 40 apresentam uma requisição AJAX para o recurso “hardware/list” que recupera todos hardware cadastrados. Essa requisição, por outro lado, é feita à aplicação de *back-end* hospedada na Heroku. As linhas 45 a 102 apresentam como um “gadget” é armazenado. Inicialmente é feita a instanciação de um objeto em formato JSON. Esse objeto por sua vez é composto por dados de outros objetos.

A [Figura 29](#) apresenta as funcionalidades de gerenciamento de multimídia da aplicação. Essa funcionalidade é gerada a partir do elemento “media” presente na modelagem de domínio da aplicação. O tipo mídia é um atributo denominado “images” na entidade gadget. Assim, é gerado um conjunto de funcionalidades que permite que imagens sejam atribuídas a um gadget.

A questão da multimídia é tratada tanto a nível de *back-end* quanto de *front-end*. No *back-end* do GAE a imagem/video é armazenada no datastore como array de bytes por meio do método “media.setMediaData(new Blob(ByteStreams.toByteArray(file.getFile())));”. No *back-end* da Heroku a imagem/video é armazenada em formato *base 64* por meio do método “media.setMediaData(new String(Base64.encodeBase64(ByteStreams.toByteArray(file.getFile()))));”. Para ser apresentada no *front-end* a imagem é requisitada pelo controlador Angular ao local em que está armazenada por uma chamada AJAX. O controlador recupera a imagem do destino informado no services e apresenta na interface utilizando *two-way-data-bind* e decodificação base64 como na tag ”.

6.4.2 Back-ends da aplicação gerada

Conforme a especificação, foram gerados duas aplicações de *back-end*, uma contendo as funcionalidades especificadas para a plataforma GAE e outra contendo as funcionalidades para a plataforma Heroku. Essas aplicações são independentes e funcionam uma sem a necessidade da outra, mas isso poderia ser diferente a depender a necessidade da aplicação. Uma chamada direta, por exemplo, ao recurso “android/list” da aplicação diretamente pelo *browser* resultaria no retorno apresentado na [Figura 30](#).

A [Figura 30](#) apresenta o retorno de uma requisição ao recurso de *back-end* contendo elementos previamente cadastrados. O retorno consiste em dados em formato JSON, esses dados podem ser tratados por qualquer serviço *front-end* consumidor dos serviços.

Figura 29 – Gerenciamento de multimídia

Fonte: autoria própria.

Figura 30 – Retorno em formato JSON de uma requisição enviada a um serviço REST

Fonte: autoria própria.

6.4.3 Conclusões sobre a aplicação gerada

A aplicação gerada a partir do modelo de domínio consiste em uma aplicação única (em termos de funcionalidade) formada pela composição de três aplicações de diferentes nuvens, por isso é uma aplicação multi-cloud. Como o único modelo de serviço envolvido na combinação é PaaS é possível dizer que a aplicação é *multi-PaaS*. Em termos arquiteturais é possível perceber que as diversas partes da aplicação se comunicam via rede, isso inevitavelmente insere pontos de gargalo e pode reduzir o desempenho da aplicação. Uma aplicação desenvolvida para um provedor exclusivo pode apresentar melhor desempenho. Todavia, não possibilita a combinação de serviços de diversas plataformas. No futuro, aplicações *multi-Cloud* serão a regra, não a

exceção.

No cenário atual, conforme apresentado por indicadores de mercado como o Northbridge survey ([NORTHBRIDGE... , 2014](#)) as empresas já estão adotando mais de um serviço de nuvem. No survey do ano 2016 o survey apurou, por exemplo, que 56% das empresas questionadas estão usando mais que um SaaS. Todavia, a utilização da combinação de múltiplos serviços de nuvem para compor uma aplicação única conforme proposto neste trabalho não foi mencionado. O *lock-in* ainda é um forte inibidor da adoção das tecnologias relacionadas ao modelo, perdendo apenas para o quesito segurança.

Em relação ao *lock-in*, este estudo apresenta uma contribuição fortemente relacionada a esse problema. Foi explorada uma estratégia não baseada em padrões para a combinação de recursos específicos de plataforma que geram o *lock-in*. A estratégia explorada nos protótipos desenvolvidos nesta tese poderá ser utilizada para o desenvolvimento de aplicações *multi-cloud* em geral. Nesta tese, foi utilizada para desenvolvimento de aplicações *multi-PaaS*. Os protótipos desenvolvidos mostram que é possível combinar recursos e alcançar uma aplicação interoperável por meio de serviços e formato de dados comum sem a utilização de padrões de nuvem. Outro quesito importante que foi levantado neste trabalho é questão do *lock-in* planejado. Ou seja, em muitos casos o *lock-in* não somente é necessário, como desejado. A combinação de recursos por meio de serviços e CORS favorece o *lock-in* planejado porque possibilita a combinação de serviços específicos de plataforma e que as soluções tirem melhor proveito de determinadas características proprietárias das plataformas.

No momento, os estudos de caso apresentados neste trabalho mostram a construção da abordagem e cumprem sua função de explorar como as empresas podem desenvolver sua própria solução dirigida por modelos para desenvolvimento de aplicações *multi-cloud*. Proposta inicial deste trabalho. A solução CORS apresentada aqui não se limita apenas a combinação de serviços PaaS. Por meio da estratégia apresentada neste trabalho é possível expandir as soluções e domínios explorados para incluir contextos diversos. A aplicação de MDE, neste contexto, agrega inerentemente vantagens e desvantagens relacionadas a essa tecnologia. Para [Kleppe, Jos e Wim \(2003\)](#), uma aplicação interoperável pode ser alcançada por meio de um modelo global e um processo MDE que faz com que cada parte deste modelo possa ser transformada em código para uma plataforma diferente. O software resultante deverá executar em um ambiente heterogêneo, mas manter sua funcionalidade global. Esse conceito foi demonstrado nesta tese por meio dos protótipos desenvolvidos. A portabilidade seria uma característica inerente a uma aplicação portátil no contexto de MDE.

6.5 Desenvolvimento de aplicações portáteis utilizando a abordagem desenvolvida

A portabilidade, no contexto de MDE, refere-se à geração de código (utilizando-se elementos MDE) para diferentes plataformas a partir de um mesmo modelo (KLEPPE; JOS; WIM, 2003). A abordagem dirigida por modelos para interoperabilidade contém uma abordagem dirigida por modelos para portabilidade conforme apresentado por Silva, Fortes e Lucrédio (2013). Assim, para desenvolver aplicações portáteis a nível de MDE basta considerar a modelagem da aplicação considerando apenas um provedor por vez. No momento, os geradores construídos permitem a portabilidade de aplicações entre a GAE e a Heroku. Todavia, no futuro, é possível que sejam construídos geradores para outras plataformas a partir da mesma linguagem.

Para demonstrar a portabilidade utilizando o mesmo Estudo de Caso apresentado neste capítulo seria necessário apenas modificar a modelagem de domínio para considerar o desenvolvimento de uma aplicação que utiliza apenas uma plataforma. O Código-fonte 27 apresenta a modelagem de domínio considerando a plataforma GAE.

Código-fonte 27 – Modelagem de domínio da aplicação *phonecat single-PaaS*

```
1: /* application's modeling */
2: cloudapplication gaeApplicationBackendGenerated {
3:   type : back
4:   provider : google
5:   url: "1-dot-appexemplo.appspot.com"
6: }
7:
8: /***** media's modeling *****/
9: media Image {
10:   type "jpeg"
11: }
12:
13: /***** entities' modeling *****/
14: entity Android {
15:   property os : String
16:   property ui : String
17: }
18:
19: ... other entities...
20:
21: entity Gadget {
22:   property additionalFeatures : String
23:   property android : Android
```

```
24:  property availability : String
25:  property battery : Battery
26:  property camera : Camera
27:  property connectivity : Connectivity
28:  property description : String
29:  property display : Display
30:  property hardware : Hardware
31:  property images : Image[]
32:  property name : String
33:  property height : String
34:  property width : String
35:  property lenght : String
36:  property weight : String
37:  property storage : Storage
38:  property snippet : String
39: }
40:
41: /***** resources modeling *****/
42:
43: resource android from Android {
44:   method : create
45:   method : update
46:   method : retrieve
47:   method : delete
48: }
49:
50: ...other resources...
51:
52: /***** modeling multicloud *****/
53:
54: multicloud phonecatCORS {
55:   type : front
56:   provider : google
57:   cloudapp gaeApplicationBackendGenerated {
58:     service : android
59:     service : battery
60:     service : camera
61:     service : gadget
62:     service : discipline
63:     service : connectivity
64:     service : display
65:     service : hardware
```

```
66:     service : storage
67: }
68: }
```

A partir do modelo de domínio apresentado a abordagem irá gerar uma aplicação *phonocat* considerando apenas o GAE. O que garante a portabilidade entre plataformas seria o fato de gerar uma aplicação para uma plataforma A e caso seja necessária uma mudança de plataforma ou tipo de oferta de serviço, a partir do mesmo modelo utilizado no primeiro momento gera-se uma aplicação para uma plataforma B conforme também apresentado por [Silva, Fortes e Lucrédio \(2013\)](#). Conclui-se, portanto, que a abordagem para portabilidade é parte constituinte da abordagem para interoperabilidade no contexto apresentado neste trabalho, conforme previsto por [France e Rumpe \(2007\)](#).

6.6 Considerações finais

Este capítulo apresentou dois Estudos de Caso relacionados a uma abordagem dirigida por modelos para desenvolvimento de uma aplicação *multi-PaaS*. A abordagem foi desenvolvida a partir de uma aplicação de referência implementada em conjunto com uma empresa da indústria. As teses centrais que nortearam o desenvolvimento deste trabalho foram:

- (i) é possível desenvolver aplicações interoperáveis sem a adoção de padrão;
- (ii) o MDE pode não só facilitar o desenvolvimento das aplicações, mas agregar vantagens inerentes a essa tecnologia quando comparadas com o desenvolvimento tradicional de software.
- (iii) as empresas podem utilizar suas próprias tecnologias para alcançar uma solução mais alinhada às seus interesses estratégicos

O primeiro Estudo de Caso apresentado neste capítulo mostra exatamente isso. Ou seja, como uma abordagem dirigida por modelos pode ser desenvolvida pelo próprio time de desenvolvimento da empresa e como ela pode ser utilizada para desenvolvimento de aplicações; a vantagens e desvantagens do protótipo desenvolvido são agregadas inerente as tecnologias utilizadas. O segundo Estudo de Caso apresenta como utilizar a abordagem dirigida por modelos desenvolvida para resolver também o problema da portabilidade de aplicações.

Uma primeira e mais evidente ameaça à validade desse estudo é a sua realização pelo próprio pesquisador autor do trabalho. Embora desvantagens do uso do MDE como complexidade, alto custo inicial e dificuldade de inserção em ambiente real de desenvolvimento tenham sido destacadas, o conhecimento privilegiado do autor pode ter distorcido positivamente as observações, sendo que com outros desenvolvedores os resultados poderiam favorecer menos

o uso de MDE nesse contexto. No entanto, uma vez que a infraestrutura esteja pronta, não há motivos para que outros desenvolvedores não possam aprender a utilizá-la e, depois de algum treinamento, chegar ao mesmo nível de eficiência no seu uso. Adicionalmente, é possível citar:

- **Condução do estudo:** o estudo foi conduzido pelo próprio autor e desenvolvedor da abordagem.
- **Domínio e casos de teste simplificados:** embora seja suficiente para demonstrar a tese a que se propõe este trabalho, a aplicação utilizada no Estudo de Caso foi relativamente pequena e apresenta um domínio de problema simplificado. Tais elementos podem facilitar a análise e é possível que pode não ter se testado tudo.
- **Tratamento estatístico e abordagem experimental:** não foi realizada avaliação experimental dos protótipos conforme proposto por [Wohlin et al. \(2000\)](#), consequentemente, nenhum tipo de tratamento estatístico se aplica ao Estudo de Caso realizado.

O capítulo seguinte apresenta as principais conclusões alcançadas com o desenvolvimento deste trabalho.

CONCLUSÕES

Desenvolver aplicações ótimas com custo mínimo e de preferência que utilizem as melhores tecnologias e serviços de acordo com a necessidade dos clientes é o objetivo dos desenvolvedores de sistemas. A Computação em Nuvem, dentre outras vantagens, possibilita que seus usuários tenham acesso a serviços complexos pagando somente a fatura conforme o volume de utilização. A centralização combinada com a distribuição de recursos computacionais pode trazer imensas vantagens a computação como um todo e também a outros domínios. Um dos principais problemas relacionados ao modelo de nuvem é o *lock-in* que inibe a adoção de serviços e dificulta o desenvolvimento de aplicações. Neste trabalho, no entanto, foi apresentado que embora o *lock-in* seja de fato um problema relevante, em muitos contextos, o mesmo não somente é necessário mas desejado devido as próprias especificidades da plataforma. Assim, a ideia de *lock-in* planejado considera a necessidade dos provedores tentarem a dominação mercadológica com suas soluções e propõe o uso sistemático de serviços diversos em direção a uma aplicação *multi-cloud*.

Adicionalmente, [Alkhatib et al. \(2014\)](#) destacam que a Computação em Nuvem possui potencial para mudar o cenário da computação. Esta tese explorou esse potencial por meio da interseção de várias áreas de pesquisa como MDE, Interoperabilidade, Sistemas Web, REST, CORS, Arquiteturas e Computação em Nuvem. Foi apresentada uma maneira de unir recursos heterogêneos por meio de uma abordagem dirigida por modelos e aplicações CORS com serviços REST. Os protótipos desenvolvidos neste trabalho mostram que é possível, por exemplo, utilizar as plataformas e CORS para multiplicar por N recursos gratuitos das plataformas. CORS possibilita o consumo de recursos de diversos domínios, assim, é possível que uma aplicação *front-end* com foco em múltiplas requisições conforme suportado pela tecnologia.

Isso configura uma grande vantagem das aplicações *multi-cloud*. Por outro lado, a utilização da rede, distribuição da aplicação, questões de privacidade e localização de dados ainda configura-se um problema.

O desenvolvimento dessas aplicações é complexo devido as especificidades de plataforma e heterogeneidade das mesmas. A abordagem proposta nesta tese visa agregar as vantagens do desenvolvimento dirigido por modelos com as vantagens das aplicações interoperáveis e Web, reduzindo assim a complexidade no desenvolvimento das aplicações *multi-cloud* e aumentando o reuso de software. Foi desenvolvido um protótipo de uma linguagem específica de domínio (DSL) que permite o desenvolvimento de aplicações interoperáveis. As aplicações exploradas neste trabalho têm por base o modelo de serviço PaaS. Este modelo foi selecionado porque provê um alto nível de abstração em relação ao modelo IaaS, sendo que os elementos de plataforma são gerenciados pelo provedores. Assim, as aplicações desenvolvidas podem ser denominadas *multi-PaaS*.

Este capítulo apresenta uma discussão sobre os resultados desta pesquisa e as principais conclusões alcançadas. [Seção 7.1](#) apresenta as principais contribuições desta pesquisa. A [Seção 7.2](#) apresenta as lições aprendidas. Por fim, a [Seção 7.3](#) apresenta as principais limitações da abordagem desenvolvida neste trabalho e os trabalhos futuros.

7.1 Principais contribuições

A contribuição relacionada a este trabalho, em uma perspectiva mais holística, consiste na interseção de vários “domínios”, MDE, CORS, PaaS, REST, NoSQL, JSON, Angular em uma abordagem dirigida por modelos para desenvolver aplicações heterogêneas que combinam recursos providos por diversas plataformas. Essa estratégia, que percorre o caminho da não padronização, visa possibilitar o máximo possível o uso de tecnologias já adotadas pelos desenvolvedores, bem como prover maior reuso dos artefatos de desenvolvimento.

Neste trabalho foi explorado como a arquitetura de referência para Computação em Nuvem, apresentada na [Figura 3](#), pode ser utilizada para flexibilizar o desenvolvimento das aplicações e considerar as especificidades de plataformas. Essa flexibilização mostrou-se uma estratégia para *lock-in* planejado. A ideia de *lock-in* planejado é outra demanda que se apresentou durante o desenvolvimento desta abordagem. Muitos trabalhos propostos na literatura visam evitar o *lock-in*, este trabalho, por outro lado, considera que os provedores, visando obter dominação de mercado, podem não querer seguir padrões. Conclui-se que embora o *lock-in* possa ser um grande inibidor para adoção das tecnologias de nuvem, abordagens como a apresentada neste trabalho, quando estabelecidas como tecnologias de mercado, no futuro, podem tirar proveito das especificidades que geram o *lock-in*. Assim sendo, esse problema possivelmente passará a não constar mais como um dos principais inibidores.

A combinação de serviços REST com o formato de dados JSON e CORS se mostrou um caminho possível para o desenvolvimento de aplicações interoperáveis, a maioria das iniciativas para se resolver o problema da falta de interoperabilidade giram em torno da padronização de APIs ([NOGUEIRA et al., 2016b](#)). A abordagem apresentada neste trabalho, diferentemente das

demais, segue um caminho alternativo. O uso de MDE no desenvolvimento de software oferece benefícios e malefícios (inerentes ao MDE) adicionais em relação a abordagens tradicionais. Com MDE, é possível automatizar diversas tarefas, aumentar o reuso da modelagem, entre outros. A automatização possibilita a redução de trabalho repetitivo de programação.

No que se refere à comunicação via rede, uma característica que se destaca é a questão do desempenho das aplicações. Nos dias atuais esse quesito pode influenciar muito nas tomadas de decisão a respeito do desenvolvimento das aplicações e da adoção das tecnologias de nuvem. Na comunicação via rede, quando comparada à comunicação via barramento, as aplicações tendem a ser muito mais lentas. No futuro, na medida em que as tecnologias de redes sejam melhoradas essa questão poderá se tornar menos preocupante. No cenário atual, considerar o modelo de implantação de nuvem híbrida é mais natural para tentar atenuar os impactos negativos da comunicação via rede.

Estudos de caso em colaboração com a indústria foram apresentados. O conteúdo dos estudos apresentados neste trabalho podem auxiliar as empresas no entendimento e adoção do modelo nuvem. Testes de carga e stress utilizando a ferramenta JMeter¹ foram executados sobre os serviços de *back-end* gerados. Esses testes não foram apresentados pois considera-se que a apresentação de uma estratégia para avaliar os provedores está fora do escopo deste trabalho. A ferramenta JMeter pode auxiliar na tomada de decisão sobre quais serviços adotar a depender do contexto em que se pretende utilizar a aplicação.

A partir do estudo conduzido, é possível concluir que uma abordagem MDE não necessariamente precisa gerar toda a aplicação, técnicas MDE podem ser adotadas em partes, para reduzir o esforço repetitivo de programação. O que é comum pode ser gerado e especificidades podem ser tratadas diretamente em código fonte tradicional. A linguagem desenvolvida permite a especificação de três domínios: CRUD, serviços e composição de aplicações que *inter-operam*, ou seja, faz a combinação de serviços de nuvem para alcançar uma aplicação que aproveita recursos de uma plataforma ou de outra. Um nível de abstração maior ou menor pode ser aplicado.

A abordagem implementada neste trabalho, por meio das implementações de referência, considera tecnologias já utilizadas pelas empresas. A ideia central é que cada empresa possa desenvolver sua própria estratégia de interoperabilidade sem necessariamente seguir um padrão de nuvem. A aplicação de MDE e o uso de tecnologias de nuvem em um projeto traz benefícios e malefícios inerentes a essas tecnologias. As vantagens e desvantagens da nuvem e do MDE foram apresentadas nos capítulos anteriores. A abordagem desenvolvida neste trabalho possui diferenças fundamentais em relação aos trabalhos encontrados na literatura. A ideia essencial por trás deste trabalho é que as empresas desenvolvam sua própria estratégia de interoperabilidade sem uso de um padrão e considerando as tecnologias já adotadas pela mesma. Na literatura, no entanto, as abordagens seguem o caminho da padronização. A estratégia apresentada aqui é apenas um

¹ <http://jmeter.apache.org/>

dos modelos possíveis. A questão central do uso do MDE aplicado a este contexto consiste na tese de que uma abordagem MDE pode não somente resolver o problema da complexidade em desenvolver sistemas que inter-operam (*multi-cloud*), mas levar a benefícios adicionais inerentes a essa tecnologia, sobretudo, a um maior reúso de software (conforme apresentado por [Lucrédio \(2009\)](#)). O problema das aplicações inter-operarem foi resolvido ao nível de serviços com um modelo comum de dados em JSON, serviços REST e CORS; o MDE foi originalmente proposto como uma tecnologia que eleva o nível de abstração na construção dessas aplicações e, portanto, resolve o problema do desenvolvimento dos sistemas, não propriamente o problema da interoperabilidade no que se refere a comunicação entre sistemas.

Como contribuição adicional do projeto que resultou nesta tese foi apresentada uma revisão sistemática de literatura que fornece uma visão atualizadas do tópico de pesquisa de interoperabilidade no contexto da Computação em Nuvem. Uma revisão sistemática visa a exploração do estado da arte do tópico de pesquisa de uma maneira mais rigorosa. As contribuições são apresentadas no trabalho publicado por [Nogueira et al. \(2016b\)](#). Esse trabalho além dos conceitos relacionados ao problema da interoperabilidade apresenta dez lacunas de pesquisa em aberto e possibilidades de trabalhos futuros relacionados ao tema. A mesma pode ser utilizada como ferramenta e arcabouço teórico para pesquisadores e desenvolvedores da área.

Constatou-se também na prática que, no contexto de desenvolvimento dirigido por modelos, a abordagem dirigida por modelos para portabilidade é inerente a uma abordagem para interoperabilidade. Aplicações passarão a ser um “mosaico” de recursos, desenvolvidos em alto nível e com menos trabalho repetitivo. As empresas poderão desenvolver sua própria estratégia, incluindo nos metaprogramas tecnologias que já adotam, inclusive eventuais padrões que a empresa venha adotar ou que eventualmente venham a surgir no futuro.

Soluções interoperáveis podem trazer diversos benefícios; um mapeamento entre os benefícios destacados por [Petcu \(2014\)](#) e a abordagem desenvolvida neste trabalho pode ser descrita a partir das seguintes vertentes:

- **Redução de custos:** o estudo mostrou que por meio de CORS é possível utilizar recursos muitas vezes de maneira gratuitas e combiná-los para formar uma aplicação global;
- **Aumento de reúso de software:** a aplicação de MDE, na perspectiva do Engenheiro de Domínio, reduz o esforço repetitivo de programação levando o desenvolvimento do sistema para um nível mais elevado de abstração podendo aumentando o reúso do software;
- **Reagir a mudanças nas ofertas de serviços por parte dos provedores:** a possibilidade de portabilidade de aplicações deixa o desenvolvedor em uma posição mais confortável para tomada de decisão;
- **Possibilitar que restrições de localização de dados e legislação sejam respeitadas:** a

combinação de recursos possibilita esse elemento. Provedores locais podem ser combinados com externos visando proteger os dados de eventuais questões de privacidade;

- **Eliminar a dependência de um provedor:** característica inerente a uma solução portátil. Como foi visto, a abordagem para interoperabilidade no contexto de modelos contempla portabilidade;
- **Garantir *backups*:** característica inerente a uma solução em nuvem;
- **Possibilitar a replicação de aplicações em múltiplos provedores para garantir maior disponibilidade:** característica inerente a uma solução interoperável;
- **Consumir melhores serviços de diferentes provedores ao mesmo tempo:** característica inerente e desejável de uma aplicação interoperável.

7.2 Lições aprendidas

As contribuições citadas são de caráter científico, apresentando uma estratégia para desenvolvimento de aplicações e explorando um tópico de pesquisa pouco considerado na literatura. Nesta subseção são apresentadas observações empíricas e lições aprendidas durante o desenvolvimento deste trabalho.

Durante esta pesquisa, muitas tecnologias e conceitos foram explorados. Foi possível perceber diversos avanços no estado da arte relacionado a Computação em Nuvem desde 2011, ano em que esta pesquisa se iniciou, até o ano atual. Desde então, a Computação em Nuvem que na época era uma tecnologia pouco explorada e ainda em fase de adoção se estabeleceu. Surgiram inúmeros provedores e muitas empresas passaram a adotar tecnologias correlatas resultantes da evolução da nuvem. No que se refere-se ao modelo de serviço PaaS, foco deste trabalho, hoje em dia já é possível que o desenvolvedor crie seu próprio PaaS utilizando de tecnologias como CloudFoundry² e Docker³. A adoção do modelo de nuvem é cada vez mais crescente e muitos conceitos que até então se mostravam complexos e desconhecidos da comunidade se estabeleceram.

O *lock-in* ainda é um grande problema a ser considerado (NORTHBRIDGE..., 2016), abordagens como as apresentadas nesta tese visam reduzir seus impactos, mas desenvolver aplicações considerando exclusivamente os modelos de serviço da nuvem ainda é uma tarefa complexa e, muitas vezes, repetitiva. O modelo IaaS é mais flexível e a facilidade de uso de máquinas virtuais se mostrou excelente para redução de esforço e aumento da facilidade de adoção das tecnologias de nuvem. No início das pesquisas relacionadas ao modelo, um dos desafios principais era facilitar a adoção das tecnologias de nuvem, atualmente a nuvem já é

² <http://www.cloudfoundry.com/>

³ <https://www.docker.com/>

uma realidade embora muitos problemas de pesquisa e questões relacionadas ao uso do modelo ainda estejam em aberto. Ao longo do desenvolvimento deste projeto o autor passou analisar o *lock-in* em outra perspectiva. Ao invés de tentar a todo custo evitá-lo acredita-se que em muitos casos ele não somente se faz necessário, mas é desejado. Uma boa tomada de decisão precisa ser considerada para adoção de serviços a curto, médio e longo prazo.

As empresas que estão em fase de adoção das tecnologias ainda possuem problemas relacionados a adoção das tecnologias de nuvem. Para empresas que já adotaram o modelo e consideram adotar mais de uma oferta de nuvem ao mesmo tempo, o cenário ainda se mostra complexo. Tecnologias *multi-cloud* ainda estão se desenvolvendo. Neste momento, as aplicações interoperáveis ainda estão se estabelecendo, mas futuro é possível que a utilização de *multi-cloud* seja uma regra, não uma exceção.

Aplicações com foco bem forte em tecnologias de *front-end* estão se disseminando como o caso da *Rich Internet Applications* (RIAs), CORS possibilitará um enriquecimento ainda maior das aplicações. Desvantagens do desenvolvimento de aplicações utilizando JavaScript vem sendo superadas com ferramentas como Angular e o JQuery. A depuração dessas aplicações também foi facilitada com as ferramentas e extensões adicionadas aos navegadores. A questão da segurança na implementação de elementos de negócio em JavaScript ainda é um problema, visto a necessidade dos navegadores baixarem os arquivos de código fonte para máquinas clientes.

Em termos mais pragmáticos, para um time de desenvolvimento criar aplicações como as desenvolvidas neste trabalho é necessário que o desenvolvedores tenham conhecimento de muitas tecnologias. Para as empresas encontrar profissionais previamente capacitados pode ser uma tarefa bastante difícil. O desenvolvimento de software de maneira tradicional em si já é uma tarefa complexa, o MDE facilita na perspectiva do usuário da DSL desenvolvida. Todavia ainda é necessário alguém para desenvolver metaprogramas, linguagens de mais elementos do MDE, isso não é uma tarefa trivial.

O risco envolvido na adoção de qualquer das tecnologias envolvidas neste projeto precisa ser gerenciado. Muitas tecnologias ainda estão em fase de desenvolvimento e estabelecimento na indústria, pode ser que fiquem obsoletas rapidamente. Um exemplo disso, é entre o início do desenvolvimento da abordagem e seu fim foram lançadas novas versões das tecnologias VRaptor e Angular. O VRaptor ganhou a versão VRaptor 4 com mudanças significativas em relação ao VRaptor 3 e o Google lançou o Angular 2⁴. Uma mudança qualquer, por exemplo, no GAE significaria a necessidade de manutenção em muitos dos elementos MDE desenvolvidos. Portanto, até a questão do uso ou não de MDE na prática em atividades da indústria precisa ter seu risco considerado.

⁴ <https://angular.io/>

7.3 Limitações e trabalhos futuros

As principais limitações da abordagem desenvolvida estão diretamente relacionadas aos componentes tecnológicos utilizados e ao domínio de aplicação selecionado. Neste estudo foram utilizadas aplicações que consideram banco de dados não relacionais para armazenamento de dados, portanto, a linguagem desenvolvida é centrada em dados e possui esse foco - modelar aplicações considerando modelos de dados não relacionais. Além disso, todos os riscos relacionados a adoção de PaaS, MDE, Angular, CORS e demais tecnologias envolvidas devem ser considerados antes a efetiva utilização da abordagem em um cenário da indústria. Os estudos apresentados se limitam a um protótipo com objetivo de prova de conceito.

Como trabalhos futuros, espera-se que sejam realizadas mais avaliações da abordagem e comparações com abordagens correlatas que eventualmente venham a surgir. Tais estudos visarão verificar a viabilidade da construção dos artefatos, bem como avaliar processos de desenvolvimento utilizando a DSL desenvolvida neste trabalho. Os artefatos desenvolvidos nesta tese passarão por melhorias e serão aproveitados no desenvolvimento de novos protótipos.

Abordagens correlatas a esta que possivelmente venham a surgir deverão ser comparados. Outro ponto em destaque é questão da segurança das aplicações. Nos protótipos desenvolvidos muitos elementos foram implementados utilizando JavaScript. O JavaScript é uma linguagem *client-side*, o código fica exposto e pode ser facilmente copiado e analisado por eventuais invasores.

Inclusão de novos provedores de serviços de nuvem nas transformações desenvolvidas e estudo de como a abordagem pode contribuir também para facilitar o desenvolvimento de software para esses provedores. A abordagem desenvolvida considerou os provedores GAE e Heroku. No futuro, faz-se necessário que novos provedores sejam incluídos.

Outro quesito que se faz necessário é verificar a viabilidade da construção, teste e manutenção dos elementos do MDE. Assim como a Computação em Nuvem, o desenvolvimento dirigido por modelos ainda está se desenvolvendo e precisa ser estudado em diversos contextos de aplicação.

7.4 Trabalhos publicados e submetidos

Os seguintes trabalhos foram publicados e submetidos no decorrer da execução deste projeto.

Publicados:

- (SILVA *et al.*, 2013) - Silva, E. A. N.; Silva, V. G. ; Fortes, R. P. M. ; Lucrédio, D. . Towards a model-driven approach for cloud PaaS portability. In: *Proceedings of XXXIX*

- Latin American Computing Conference (CLEI 2013)*, 2013, Caracas. Latin American Symposium of Software Engineering, 2013.
- (SILVA; FORTES; LUCRÉDIO, 2013) - Silva, E. A. N.; Fortes, R. P. M. ; Lucrédio, D. . A model-driven approach for promoting cloud PaaS portability. In: *Proceedings of Annual International Conference on Computer Science and Software Engineering(CASCON)*, 2013, Toronto. Annual International Conference on Computer Science and Software Engineering(CASCON 2013), 2013.
 - (SILVA; NETO, 2014) - Silva, E. A. N.; Graciano Neto, V. V. . MODEFS: Towards a Lightweight Model-Driven Environment for Formal Software Engineering. In: *Anais da Escola Regional de Informática - SBC Mato Grosso*, 2014, Barra do Garças.
 - (SILVA *et al.*, 2015) - Silva, E. A. N.; Graciano Neto, V. V. . An Architectural Proposal for a Lightweight Model-Driven Environment to Support Formal Software Engineering. *Revista RETEC*, v. 1, p. 1, 2014.
 - (ANTONELLI; SILVA; FORTES, 2015) Antonelli, H. L. ; Silva, E. A. N. ; Fortes, R. P. M. . A model-driven development approach for creating accessible web menus. In: *Proceedings of 6th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*, 2015, Sankt Augustin, Alemanha. DSAI 2015, 2015.
 - (SILVA *et al.*, 2015) - Silva, E. A. N.; Lucrédio, D. ; Moreira, A. ; Fortes, R. P. M. . Supporting multiple persistence models for PaaS applications using MDE: issues on cloud portability. In: *Proceedings of 5th International Conference on Cloud Computing and Services Science*, 2015, Lisboa. 2015.
 - (NOGUEIRA *et al.*, 2016a) - Nogueira, Elias; Lucrédio, Daniel ; Moreira, Ana ; Fortes, Renata . Using Model-Driven Development to Support Portable PaaS Applications. In: Springer CCIS. (Org.). *Communications in Computer and Information Science*. 1ed.: Springer International Publishing, 2016, v. 581, p. 115-134.
 - (NOGUEIRA *et al.*, 2016b) - Nogueira, Elias; Moreira, Ana ; Lucrédio, Daniel ; Garcia, Vinícius ; Fortes, Renata . Issues on developing interoperable cloud applications: definitions, concepts, approaches, requirements, characteristics and evaluation models. *Journal of Software Engineering Research and Development*, v. 4, p. 7, 2016.
 - (FORTES *et al.*, 2017) - Fortes, R. P. M; Salgado, A. L; Santos, F. S; Amaral, L. A; Nogueira, E. Game accessibility evaluation models: a literature survey. International Conference on Human-Computer Interaction (HCII) 2017.

Prêmios e títulos:

- *Best Paper Award* - MODEFS: Towards a Lightweight Model-Driven Environment for Formal Software Engineering, Escola Regional de Informática - SBC Mato Grosso - ERI-MT.

7.5 Considerações finais

Esta tese apresentou o desenvolvimento de uma abordagem dirigida por modelos para desenvolvimento de aplicações *multi-PaaS*. A Computação em Nuvem é relativamente recente e ainda está se desenvolvendo. O desenvolvimento das aplicações ainda é complexo e precisa ser melhor explorado. Este trabalho traz uma contribuição que une diversos domínios da computação, como MDE, Interoperabilidade e Computação em Nuvem, mostrando que as aplicações podem utilizar serviços provenientes de diferentes provedores ao mesmo tempo. Foi desenvolvida uma abordagem para criação de aplicações multi-PaaS centrada em dados. Ou seja, que uma abordagem possibilita a modelagem de aplicações com foco em modelo de dados não relacional. A modelagem de domínio não relacional pode ser feita utilizando a DSL desenvolvida e os modelos especificados servem de entrada para um conjunto de transformações que geram uma aplicação multi-PaaS considerando os provedores de nuvem incluídos no estudo.

Os protótipos desenvolvidos neste trabalho mostram uma prévia de que no futuro as aplicações serão *multi-Cloud*, desenvolvidas, muitas vezes, combinando diversos recursos gratuitos. A combinação de serviços mostra que as aplicações podem ficar mais baratas (em termos de custos de serviços), mas mais complexas em termos de desenvolvimento. O uso de MDE mostra que é possível reduzir o esforço do desenvolvimento, porém tecnologias MDE ainda estão se desenvolvendo. Isso tudo será somado aos avanços futuros na área de redes de computadores que visam reduzir o *delay* de comunicação de aplicações como as apresentadas neste estudo. Assim, as aplicações serão *multi-Cloud* e desenvolvidas em alto nível de abstração que possibilitará que aplicações em nuvem sejam mais baratas, sejam mais fáceis de desenvolver e tenham um desempenho ainda melhor.

REFERÊNCIAS

ALKHATIB, H.; FARABOSCHI, P.; FRACHTENBERG, E.; KASAHARA, H.; LANGE, D.; LAPLANTE, P.; MERCHANT, A.; MILOJICIC, D.; SCHWAN, K. **IEEE CS 2022 Report**. [S.l.], 2014. Citado nas páginas 23, 24 e 159.

AMATO, F.; MAZZEO, A.; MOSCATO, V.; PICARIELLO, A. A Framework for Semantic Interoperability over the Cloud. In: **Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on**. [S.l.: s.n.], 2013. p. 1259–1264. Citado nas páginas 69 e 83.

ANTKIEWICZ, M.; CZARNECKI, K. Framework-specific modeling languages with round-trip engineering. **Model Driven Engineering Languages and Systems**, Springer, p. 692–706, 2006. Citado na página 77.

ANTONELLI, H. L.; SILVA, E. A. N. da; FORTES, R. P. M. A model-driven development for creating accessible web menus. **Procedia Computer Science**, v. 67, p. 95 – 104, 2015. ISSN 1877-0509. Proceedings of the 6th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion. Citado na página 166.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. Above the clouds: A Berkeley view of cloud computing. **Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS**, v. 28, 2009. Disponível em: <<http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf>>. Citado nas páginas 23, 24, 25, 32, 36, 37, 38, 40, 42, 44, 45, 46, 48, 50, 68, 85 e 103.

_____. A view of cloud computing. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, 2010. ISSN 0001-0782. Citado nas páginas 41 e 47.

ASADI, M.; RAVAKHAH, M.; RAMSIN, R. An MDA-Based System Development Lifecycle. In: **Modeling Simulation, 2008. AICMS 08. Second Asia International Conference on**. [S.l.: s.n.], 2008. p. 836–842. Citado na página 73.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer networks**, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. Citado na página 54.

BITTAR, T. J.; FORTES, R. P. M.; LOBATO, L. L.; WATANABE, W. M. Web communication and interaction modeling using model-driven development. In: **SIGDOC'09 - Proceedings of the 27th ACM International Conference on Design of Communication**. [S.l.: s.n.], 2009. p. 193–197. Citado nas páginas 72 e 74.

BOZMAN, J.; CHEN, G. **Cloud Computing: The Need for Portability and Interoperability**. [S.l.], 2010. Disponível em: <http://www.ciosummits.com/media/pdf/solution{_}spotlight/redhat{_}idc-insights-cloud-computi>. Citado na página 48.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. Model-driven software engineering in practice. **Synthesis Lectures on Software Engineering**, Morgan & Claypool Publishers, v. 1, n. 1, p. 1–182, 2012. Citado nas páginas 26, 27, 71, 72, 73, 74, 77, 81 e 84.

BRUNELIÈRE, H.; CABOT, J.; JOUAULT, F. Combining Model-Driven Engineering and Cloud Computing. 2010. Disponível em: <http://hal.archives-ouvertes.fr/docs/00/53/91/68/PDF/CombiningMDE-CloudComputing{_}2010.> Citado na página 27.

CHEN, Y.; LI, X.; CHEN, F. Overview and analysis of cloud computing research and application. In: **E -Business and E -Government (ICEE), 2011 International Conference on**. [S.l.: s.n.], 2011. p. 1–4. Citado na página 32.

CIRILO, C. E. **Model Driven RICHUB - Processo Dirigido a modelos para a construção de interfaces ricas de aplicações ubíquas sensíveis ao contexto**. Tese (Doutorado) — Universidade Federal de São Carlos - UFSCar, 2011. Citado na página 72.

CUNHA, D.; NEVES, P.; SOUSA, P. Interoperability and portability of cloud service enablers in a PaaS environment. 2012. Citado na página 48.

DANDRIA, F.; BOCCONI, S.; CRUZ, J. G.; AHTES, J.; ZEGINIS, D. Cloud4soa: multi-cloud application management across paas offerings. In: IEEE. **Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on**. [S.l.], 2012. p. 407–414. Citado nas páginas 64 e 65.

DANIELS, J. Server virtualization architecture and implementation. **Crossroads**, ACM, New York, NY, USA, v. 16, n. 1, p. 8–12, sep 2009. ISSN 15284972. Citado na página 48.

DELGADO, J. C. M. A Framework for Cloud Interoperability Based on Compliance and Conformance. In: MAHMOOD, Z. (Ed.). **Continued Rise of the Cloud**. [S.l.]: Springer London, 2014, (Computer Communications and Networks). p. 195–232. ISBN 978-1-4471-6451-7. Citado na página 83.

DEURSEN, A. V.; KLINT, P.; VISSER, J. Domain-specific languages: An annotated bibliography. **ACM SIGPLAN NOTICES**, v. 35, p. 26–36, 2000. Citado nas páginas 74 e 122.

DOWELL, S.; BARRETO, A.; MICHAEL, J. B.; SHING, M.-T. Cloud to cloud interoperability. In: IEEE. **System of Systems Engineering (SoSE), 2011 6th International Conference on**. [S.l.], 2011. p. 258–263. Citado na página 52.

DURAO, F.; CARVALHO, J. F. S.; FONSEKA, A.; GARCIA, V. C. A systematic review on cloud computing. **The Journal of Supercomputing**, Springer, p. 1–26, 2014. Citado nas páginas 32 e 33.

ESPARZA, J.; ESCOÍ, M. Towards the next generation of model driven cloud platforms. In: **CLOSER 2011 - Proceedings of the 1st International Conference on Cloud Computing and Services Science**. [S.l.: s.n.], 2011. p. 494–500. Citado nas páginas 46 e 47.

FERRY, N.; ROSSINI, A.; CHAUVEL, F.; MORIN, B.; SOLBERG, A. Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In: **CLOUD 2013: IEEE 6th International Conference on Cloud Computing**. [S.l.: s.n.], 2013. p. 887–894. Citado na página 27.

FORTES, R. P. M.; SALGADO, A. de L.; SANTOS, F. de S.; AMARAL, L. Agostini do; SILVA, E. A. Nogueira da. Game accessibility evaluation methods: A literature survey. In: _____. **Universal Access in Human–Computer Interaction. Design and Development Approaches and Methods: 11th International Conference, UAHCI 2017, Held as Part of HCI International 2017, Vancouver, BC, Canada, July 9–14, 2017, Proceedings, Part I**. Cham: Springer International Publishing, 2017. p. 182–192. Disponível em: <https://doi.org/10.1007/978-3-319-58706-6_15>. Citado na página 166.

FRANCE, R.; RUMPE, B. Model-driven Development of Complex Software: A Research Roadmap. In: **2007 Future of Software Engineering**. Washington, DC, USA: IEEE Computer Society, 2007. (FOSE '07), p. 37–54. ISBN 0-7695-2829-5. Citado nas páginas 26, 72, 74, 84, 85 e 157.

GARG, R.; STILLER, B. Factors affecting cloud adoption and their interrelations. In: **CLOSER**. [S.l.: s.n.], 2015. p. 87–94. Citado na página 37.

GRONBACK, R. C. **Eclipse modeling project: a domain-specific language toolkit**. [S.l.]: Addison-Wesley Professional, 2009. Citado na página 80.

GROZEV, N.; BUYYA, R. Inter-Cloud architectures and application brokering: taxonomy and survey. **Software: Practice and Experience**, Wiley Online Library, v. 44, n. 3, p. 369–390, mar 2014. ISSN 00380644. Citado na página 58.

GUARINO, N. **Formal ontology in information systems: proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy**. [S.l.]: Ios PressInc, 1998. v. 46. Citado na página 37.

HARSH, P.; DUDOUET, F.; CASCELLA, R. G.; JEGOU, Y.; MORIN, C. Using open standards for interoperability issues, solutions, and challenges facing cloud computing. In: **Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualization management (svm)**. [S.l.: s.n.], 2012. p. 435–440. Citado na página 52.

HETTEL, T.; LAWLEY, M.; RAYMOND, K. Model synchronisation: Definitions for round-trip engineering. **Theory and Practice of Model Transformations**, Springer, p. 31–45, 2008. Citado na página 77.

HOGAN, M.; LIU, F.; SOKOL, A.; TONG, J. Nist cloud computing standards roadmap. **NIST Special Publication 500-291, Version 2**, 2011. Citado nas páginas 34, 35, 36, 37, 38, 39 e 59.

IDC, I. D. C. Cloud Computing's Role in Job Creation. 2012. Citado na página 24.

IEEEGLOSSARY. 2013. Disponível em: <http://www.ieee.org/education/{_}careers/education/standards/standards/{_}glossary>. Citado na página 51.

KANDUKURI, B. R.; PATURI, V. R.; RAKSHIT, A. Cloud Security Issues. In: **Services Computing, 2009. SCC '09. IEEE International Conference on**. [S.l.: s.n.], 2009. p. 517–520. Citado na página 41.

KITCHENHAM, B.; BRERETON, O. P.; BUDGEN, D.; TURNER, M.; BAILEY, J.; LINK-MAN, S. Systematic literature reviews in software engineering: A systematic literature review. **Information and Software Technology**, v. 51, n. 1, p. 7–15, 2009. ISSN 0950-5849. Citado na página 25.

- KITCHENHAM, B.; CHARTERS, S.; DUBGEN, D.; PEARL, B.; TURNER, M.; LINKMAN, S.; JORGENSEN, M.; MENDES, E.; VISAGGIO, G. **Guidelines for performing systematic literature reviews in software engineering**. [S.l.], 2007. Citado nas páginas 28, 50 e 61.
- KLEPPE, A.; JOS, W.; WIM, B. MDA Explained, The Model-Driven Architecture: Practice and Promise. In: . [S.l.]: Addison-Wesley, 2003. Citado nas páginas 72, 74, 82, 84, 154 e 155.
- LAPLANTE, P. A.; ZHANG, J.; VOAS, J. What's in a Name? Distinguishing between SaaS and SOA. **IT Professional**, v. 10, n. 3, p. 46–50, may 2008. ISSN 1520-9202. Citado nas páginas 43 e 44.
- LIU, F.; TONG, J.; MAO, J.; BOHN, R.; MESSINA, J.; BADGER, L.; LEAF, D. NIST cloud computing reference architecture. **NIST Special Publication**, v. 500, p. 292, 2011. Citado nas páginas 38, 39 e 57.
- LOUTAS, N.; KAMATERI, E.; TARABANIS, K. A Semantic Interoperability Framework for Cloud Platform as a Service. In: **Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on**. [S.l.: s.n.], 2011. p. 280–287. Citado nas páginas 51, 69 e 83.
- LUCRÉDIO, D. Uma Abordagem Orientada a Modelos para Reutilização de Software. In: . [S.l.]: Tese de Doutorado, 2009. Citado nas páginas 26, 74, 75, 76, 78, 79, 80 e 162.
- LUCRÉDIO, D.; ALMEIDA, E. S.; FORTES, R. P. M. An investigation on the impact of MDE on software reuse. In: IEEE. **Software Components Architectures and Reuse (SBCARS), 2012 Sixth Brazilian Symposium on**. [S.l.], 2012. p. 101–110. Citado na página 122.
- MARINOS, A.; BRISCOE, G. Community cloud computing. In: **Cloud Computing**. [S.l.]: Springer, 2009. p. 472–484. Citado na página 43.
- MELL, P.; GRANCE, T. The NIST definition of cloud computing. **Communications of the ACM**, v. 53, n. 6, p. 50, 2010. Citado nas páginas 23, 36 e 38.
- Microsoft. 2017. Disponível em: <<https://www.microsoft.com/cloud/interop/>>. Citado nas páginas 63 e 70.
- MOORE, B.; WARSCHOFSKY, R.; THOMAS, I.; WILLEMS, C.; MEINEL, C. **Eclipse Development using the Graphical Editing Framework and the EclipseModeling Framework**. [S.l.]: ibm.com/redbooks. [S.l.: s.n.], 2004. Citado na página 81.
- NOGUEIRA, E.; LUCRÉDIO, D.; MOREIRA, A.; FORTES, R. Using model-driven development to support portable paas applications. In: _____. **Cloud Computing and Services Science: 5th International Conference, CLOSER 2015, Lisbon, Portugal, May 20-22, 2015, Revised Selected Papers**. Cham: Springer International Publishing, 2016. p. 115–134. ISBN 978-3-319-29582-4. Disponível em: <http://dx.doi.org/10.1007/978-3-319-29582-4_7>. Citado nas páginas 83 e 166.
- NOGUEIRA, E.; MOREIRA, A.; LUCRÉDIO, D.; GARCIA, V.; FORTES, R. Issues on developing interoperable cloud applications: definitions, concepts, approaches, requirements, characteristics and evaluation models. **Journal of Software Engineering Research and Development**, v. 4, n. 1, p. 7, 2016. Disponível em: <<http://dx.doi.org/10.1186/s40411-016-0033-6>>. Citado nas páginas 28, 47, 50, 52, 57, 61, 62, 69, 96, 121, 138, 160, 162 e 166.

- NORTHBRIDGE Survey. 2014. Disponível em: <http://www.northbridge.com/2013-cloud-computing-survey>. Citado nas páginas 25 e 154.
- NORTHBRIDGE Survey. 2016. Disponível em: <http://www.northbridge.com/2016-future-cloud-computing-survey>. Citado na página 163.
- OH, J.; CHOI, H. C.; LIM, S. H.; AHN, W. H. Reengineering template-based web applications to single page ajax applications. **KIPS Transactions on Software and Data Engineering**, Korea Information Processing Society, v. 1, n. 1, p. 1–6, 2012. Citado na página 118.
- OMG. MDA Guide Version 1.0. 1. 2003. Citado nas páginas 73, 74, 80 e 81.
- PAPAZOGLU, M. P.; TRAVERSO, P.; DUSTDAR, S.; LEYMAN, F. Service-Oriented Computing: State of the Art and Research Challenges. **Computer**, v. 40, n. 11, p. 38–45, 2007. ISSN 0018-9162. Citado na página 44.
- _____. Service-oriented computing: A research roadmap. **International Journal of Cooperative Information Systems**, v. 17, n. 2, p. 223–255, 2008. Citado na página 44.
- PETCU, D. Portability and interoperability between clouds: challenges and case study. In: **Towards a Service-Based Internet**. [S.l.]: Springer, 2011. p. 62–74. Citado nas páginas 25, 48, 51, 52, 56, 57, 59, 63, 67, 69 e 83.
- _____. Multi-Cloud: expectations and current approaches. In: **ACM. Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds**. [S.l.], 2013. p. 1–6. Citado nas páginas 59, 60 e 61.
- _____. Consuming Resources and Services from Multiple Clouds. **Journal of Grid Computing**, v. 12, n. 2, p. 321–345, 2014. ISSN 1570-7873. Citado nas páginas 26, 58, 63, 69 e 162.
- PETCU, D.; MACARIU, G.; PANICA, S.; CRĂCIUN, C. Portable Cloud applications—From theory to practice. **Future Generation Computer Systems**, v. 29, n. 6, p. 1417–1430, aug 2013. ISSN 0167739X. Citado na página 48.
- POSSATTO, M. A.; LUCRÉDIO, D. Automatically propagating changes from reference implementations to code generation templates. **Information and Software Technology**, v. 67, p. 65–78, 2015. ISSN 0950-5849. Citado nas páginas 77 e 133.
- RANABAHU, A.; SHETH, A. Semantics Centric Solutions for Application and Data Portability in Cloud Computing. In: **Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on**. [S.l.: s.n.], 2010. p. 234–241. Citado nas páginas 48 e 69.
- RASHIDI, B.; SHARIFI, M.; JAFARI, T. A Survey on Interoperability in the Cloud Computing Environments. **International Journal of Modern Education and Computer Science**, v. 5, n. 6, p. 17–23, jul 2013. ISSN 20750161. Citado na página 51.
- REZAEI, R.; CHIEW, T. K.; LEE, S. P.; ALIEE, Z. S. Interoperability evaluation models: A systematic review. **Computers in Industry**, v. 65, n. 1, p. 1–23, 2014. ISSN 0166-3615. Citado nas páginas 53, 54 e 55.
- RIMAL, B. P.; CHOI, E.; LUMB, I. A Taxonomy and Survey of Cloud Computing Systems. In: **INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on**. [S.l.: s.n.], 2009. p. 44–51. Citado nas páginas 35, 36, 38 e 40.

- SHARMA, R.; SOOD, M. A Model Driven Approach to Cloud SaaS Interoperability. **International Journal of Computer Applications**, v. 30, n. 8, p. 1–8, 2011. Citado na página 43.
- SHIRAZI, M. N.; KUAN, H. C.; DOLATABADI, H. Design Patterns to Enable Data Portability between Clouds' Databases. In: **Computational Science and Its Applications (ICCSA), 2012 12th International Conference on**. [S.l.: s.n.], 2012. p. 117–120. Citado na página 49.
- SILVA, E. A. da; SILVA, V. da; LUCRÉDIO, D.; de Mattos Fortes, R. Towards a model-driven approach for promoting cloud PaaS portability. In: **Computing Conference (CLEI), 2013 XXXIX Latin American**. [S.l.: s.n.], 2013. p. 1–9. Citado nas páginas 79 e 165.
- SILVA, E. A. N. da; FORTES, R. P. M.; LUCRÉDIO, D. A Model-driven Approach for Promoting Cloud PaaS Portability. In: **Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research**. Riverton, NJ, USA: IBM Corp., 2013. (CASCON '13), p. 92–105. Citado nas páginas 27, 28, 48, 69, 79, 82, 83, 84, 155, 157 e 166.
- SILVA, E. A. N. da; LUCRÉDIO, D. Software Engineering for the Cloud: A Research Roadmap. **2012 26th Brazilian Symposium on Software Engineering**, IEEE, Brazil, UFSCar, p. 71–80, sep 2012. Citado nas páginas 23, 24, 25, 31, 41, 45, 46, 50, 61, 88, 92, 97 e 112.
- SILVA, E. A. N. da; MOREIRA, A.; LUCRÉDIO, D.; FORTES, R. P. M. Supporting multiple persistence models for PaaS applications using model-driven engineering. In: **Proceedings of the 5th International Conference on Cloud Computing and Services Science, CLOSER 2015**. Lisbon, Portugal: INSTICC, 2015. (CLOSER'15). Citado nas páginas 83 e 166.
- SILVA, E. A. N. da; NETO, V. V. G. Towards a Lightweight Model-Driven Environment for Formal Software Engineering. In: **Proceedings of Escola Regional de Informática - SBC Mato Grosso**. Barra do Garças, Brasil: SBC, 2014. (ERIMT'14). Citado na página 166.
- SILVA, E. A. N. da; NETO, V. V. G.; PALMEIRAS, A.; QUADRA, D.; SAMAMBAIA, C. An Architectural Proposal for a Lightweight Model-Driven Environment to Support Formal Software Engineering. In: **Proceedings of Escola Regional de Informática - SBC Mato Grosso**. Barra do Garças, Brasil: RETEC, 2015. (RETEC'15, v. 1). Citado na página 166.
- SMITH, M. A.; KUMAR, R. L. A theory of application service provider (ASP) use from a client perspective. **Information & Management**, v. 41, n. 8, p. 977–1002, 2004. ISSN 0378-7206. Citado na página 44.
- STEIMANN, F.; KÜHNE, T. Coding for the code. **Queue**, ACM, New York, NY, USA, v. 3, n. 10, p. 40:44–40:51, dez. 2005. ISSN 1542-7730. Disponível em: <<http://doi.acm.org/10.1145/1113322.1113336>>. Citado nas páginas 76 e 77.
- TANENBAUM, A. S. Modern Operating Systems. Prentice Hall Press, Upper Saddle River, NJ, USA, 2007. Citado na página 32.
- TANENBAUM, A. S.; Van Steen, M. **Distributed systems: principles and paradigms**. [S.l.]: Prentice Hall Upper Saddle River, NJ., 2002. v. 2. Citado na página 40.
- THIERY, A.; CERQUEUS, T.; THORPE, C.; SUNYÉ, G.; MURPHY, J. A dsl for deployment and testing in the cloud. In: IEEE. **Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on**. [S.l.], 2014. p. 376–382. Citado na página 132.

- TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. **Introdução à engenharia de software experimental - Relatório Técnico**. [S.l.: s.n.], 2002. Citado na página 87.
- VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds. **ACM SIGCOMM Computer Communication Review**, v. 39, n. 1, p. 50, dec 2008. ISSN 01464833. Citado nas páginas 34 e 50.
- VELTE, T.; VELTE, A.; ELSENPETER, R. **Cloud Computing, A Practical Approach**. [S.l.]: McGraw-Hill, Inc., 2009. Citado na página 40.
- WALLIS, P. A Brief History of Cloud Computing: Is the Cloud There Yet? 2008. Disponível em: <http://cloudcomputing.sys-con.com/node/581838>. Citado na página 32.
- WEISER, M. Ubiquitous computing. **Computer**, v. 26, n. 10, p. 71–72, 1993. Citado na página 33.
- WHITTLE, J.; ARAÚJO, J. a.; MOREIRA, A. Composing aspect models with graph transformations. In: **Proceedings of the 2006 International Workshop on Early Aspects at ICSE**. New York, NY, USA: ACM, 2006. (EA '06), p. 59–65. ISBN 1-59593-405-7. Disponível em: <http://doi.acm.org/10.1145/1137639.1137649>. Citado na página 138.
- WOHLIN, C.; RUNESON, P.; HOST, M.; OHLSSON, C.; REGNELL, B.; WESSLÉN, A. Experimentation in software engineering: an introduction. Kluwer Academic Publishers, 2000. Citado nas páginas 87 e 158.
- YIN, R. K. **Case study research: Design and methods**. [S.l.]: Sage Publications, Incorporated, 2008. v. 5. Citado na página 87.
- YOUSEFF, L.; BUTRICO, M.; SILVA, D. D. Toward a Unified Ontology of Cloud Computing. In: **Grid Computing Environments Workshop, 2008. GCE '08**. [S.l.: s.n.], 2008. p. 1–10. Citado na página 37.
- ZEGINIS, D.; D'ANDRIA, F.; BOCCONI, S.; Gorrongoitia Cruz, J.; Collell Martin, O.; GOUVAS, P.; LEDAKIS, G.; TARABANIS, K. A. A user-centric multi-PaaS application management solution for hybrid multi-Cloud scenarios. **Scalable Computing: Practice and Experience**, v. 14, n. 1, 2013. Citado nas páginas 64, 65, 66 e 67.
- ZHANG, Z.; WU, C.; CHEUNG, D. W. L. A survey on cloud interoperability: taxonomies, standards, and practice. **ACM SIGMETRICS Performance Evaluation Review**, ACM, v. 40, n. 4, p. 13–22, 2013. Citado nas páginas 47, 52 e 68.