UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DIRIGIDA POR MODELOS PARA PORTABILIDADE ENTRE PLATAFORMAS DE COMPUTAÇÃO EM NUVEM

ELIAS ADRIANO NOGUEIRA DA SILVA

ORIENTADOR: PROF. DR. DANIEL LUCRÉDIO

São Carlos – SP

Maio/2013

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DIRIGIDA POR MODELOS PARA PORTABILIDADE ENTRE PLATAFORMAS DE COMPUTAÇÃO EM NUVEM

ELIAS ADRIANO NOGUEIRA DA SILVA

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

Orientador: Prof. Dr. Daniel Lucrédio

São Carlos – SP

Maio/2013

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

S586ad

Silva, Elias Adriano Nogueira da.
Uma abordagem dirigida por modelos para portabilidade entre plataformas de computação em nuvem / Elias Adriano Nogueira da Silva. -- São Carlos : UFSCar, 2013.
96 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2013.

1. Engenharia de software. 2. Computação em nuvem. 3. Desenvolvimento orientado por modelos. 4. Plataforma como serviço (PaaS). 5. Linguagem específica de domínio. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Uma Abordagem Dirigida por Modelos para Portabilidade Entre Plataformas de Computação em Nuvem"

Elias Adriano Nogueira da Silva

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Daniel Lucrédio

Prof. Dr. Daniel Lucrédio
(Orientador - DC/UFSCar)

RAPenteado

Profa. Dra. Rosângela Aparecida Dellosso Penteado
(DC/UFSCar)

MDelFabro

Prof. Dr. Marcos Didonet Del Fabro
(UFPR)

São Carlos
Maio/2013

De maneira geral dedico este trabalho a todos que acreditaram em meu potencial,
em especial à minha noiva Jéssica e minha mãe Abma.

AGRADECIMENTOS

Foram muitos os que, direta ou indiretamente, me ajudaram a concluir este trabalho. Meus sinceros agradecimentos...

... a Deus por me abençoar com a vida, força, fé e saúde para que eu pudesse concluir mais esta importante etapa da minha vida. Obrigado por estar sempre ao meu lado nesta caminhada e por atender todas as minhas súplicas.

...a minha mãe Abma e minha noiva Jéssica pelo amor e apoio incondicional. Amo muito vocês!!!

...a minha família de uma maneira geral pela torcida e por acreditar que este grande sonho um dia se realizaria.

...ao meu orientador Daniel Lucrédio, por ter me acolhido, acreditado em mim, compreendido minhas dificuldades e ajudado a realizar este trabalho. Além das tarefas de orientador, foi amigo, dedicado e paciente. Somente Deus poderá recompesá-lo por toda ajuda que a mim foi dada. Como diria o chaves: muitississsisisisissimooo obrigado!

...ao meu grande amigo Marcos Prado por ter acompanhado toda essa trajetória. Que Deus continue iluminando sua vida e permita que todos seus anseios se realizem. Muito obrigado marcão!!!

...aos meus amigos e colegas de mestrado os quais compartilhei as alegrias e dificuldades deste percurso. Obrigado por me suportar e por todos os momentos de alegria que vivemos Rafael Dias, Paulo Papotti, Du Cirilo, Fernando, Carlão, Vitinho, André(zé), demais colegas, professores e funcionários do DC-UFSCar.

... aos membros da minha banca de qualificação Dra. Rosângela Penteado e Dr. Hermes Senger pela importante contribuição ao trabalho.

... a todos os contribuintes brasileiros que através da CAPES financiaram esta pesquisa.

Enfim... a todos que, de alguma maneira, contribuiram para realização deste trabalho!

Nós só podemos ver um pouco do futuro, mas o suficiente para perceber que há muito a fazer.

Alan Turing

RESUMO

A computação em nuvem tem potencial para revolucionar a maneira como sistemas são desenvolvidos e comercializados. Entre as diversas lacunas de pesquisa relacionados a esse novo modelo computacional está o ***Lock-In***. Isto é, o aprisionamento do usuário ao provedor devido a dificuldade na migração de dados e aplicativos de uma plataforma de nuvem para outra. Tal aprisionamento ocorre, dentre outros motivos, devido a falta de um padrão para desenvolvimento de aplicações para a nuvem. Este trabalho apresenta uma abordagem dirigida por modelos (Model-Driven Engineering - MDE) para portabilidade de aplicações entre plataformas de Computação em nuvem. Com o MDE os engenheiros de software podem trabalhar em um nivel mais alto de abstração livrando-se de tarefas repetitivas de codificação, que ficam a cargo de transformações automatizadas, e ficando assim protegidos das complexidades requeridas para implementação nas diferentes plataformas. Além da portabilidade o MDE traz benefícios adicionais em relação a abordagens tradicionais de desenvolvimento de sistemas. Este trabalho apresenta o desenvolvimento de uma linguagem textual que possibilita a especificação de aplicações em um alto nível de abstração. Bem como geradores de código para duas conhecidas plataforma de nuvem, a *Google App Engine*(GAE) e a *Windows Azure*, mostrando que a linguagem desenvolvida pode servir de entrada para a geração de grande parte do código necessário para muitas aplicações de nuvem e que facilita seu desenvolvimento. O MDE oferece uma maior produtividade, melhor manutenção e documentação e reúso. A avaliação realizada observou alguns desses benefícios, o que comprova a viabilidade da abordagem MDE.

Palavras-chave: Computação em Nuvem, Desenvolvimento Dirigido por Modelos, Portabilidade.

ABSTRACT

Cloud Computing has potential to revolutionize way that systems are developed and marketed. Among several research gaps related to this new model is the Lock-In. The Lock-In is the difficulty on migrating data and applications from a cloud platform to another. The lack of standardization, as well as other reasons, are causing the problem. This work presents a model-driven(MDE) approach for portability of applications between cloud platforms. With MDE software engineers can work at a high level of abstraction freeing themselves from repetitive tasks related to software implementation and specific details of cloud platforms. Besides portability, MDE brings additional benefits related to traditional software development approaches. The approach presented in this work, basically consists of a DSL and a set of automated transformations for two known cloud platforms *Google App Engine and Microsoft Azure*. The approach allows the development of cloud applications on a high abstraction level, showing that despite being simple, the elements of the approach are enough to generate many cloud applications, as well as facilitates its development. The evaluation confirms the benefits provided by MDE technologies.

Keywords: Cloud Computing, Model-Driven Engineering, Portability

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

*Este capítulo apresenta **o contexto** em que este trabalho está inserido, a motivação que o originou, o objetivo e a metodologia utilizada em seu desenvolvimento. Também destaca as suas principais contribuições. Em seguida é descrita a organização desta dissertação.*

1.1 Visão Geral

A computação em nuvem tem se configurado como uma importante evolução tecnológica recente. Apesar de ainda em estágio de amadurecimento, o tema se tornou rapidamente um dos principais objetos de pesquisa em Engenharia de Software na atualidade (BREITMAN; VIRTEBO, 2010; SILVA; LUCRéDIO, 2012). Porém, ainda não há um padrão para desenvolvimento de aplicações para esse modelo, o que acaba gerando uma ampla diversidade de propostas de tecnologias e plataformas disponíveis no mercado.

Para o desenvolvedor, isso resulta em uma série de problemas, entre eles o *Lock-In* (ARMBRUST et al., 2009). O *Lock-In* é o aprisionamento do usuário ao provedor de serviço devido à dificuldade de extração de dados e programas de uma plataforma para utilizá-los em outra, causada pela grande diferença entre as plataformas. A falta de portabilidade que gera esse problema está dificultando a adoção do modelo de nuvem. Isso porque a comunidade de desenvolvimento tem receio em criar seus sistemas levando-se em consideração uma tecnologia específica e em seguida serem cobradas taxas abusivas pela sua utilização (ARMBRUST et al., 2009), ou mesmo ficar presa a uma tecnologia que pode vir a se tornar inadequada e precisa ser abandonada.

Algumas iniciativas para resolver esse problema são apresentadas no Capítulo 3. A maioria delas associadas à padronização de APIs para o desenvolvimento de tecnologias relacionadas à nuvem. 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. Todavia, os provedores de nuvem podem querer utilizar tecnologias específicas para criar soluções que estejam aliadas aos seus requisitos de negócio e não optar por seguir um padrão. Sendo assim, é importante que sejam pesquisadas abordagens alternativas, não apenas para explorar um caminho diferente, mas para ajudar a enriquecer o modelo de nuvem e consequentemente facilitar sua adoção.

A abordagem utilizada neste trabalho, faz uso de desenvolvimento de software dirigido por modelos (*Model Driven Engineering* ou MDE). A ideia em torno do MDE aplicado ao desenvolvimento de sistemas para nuvem é identificar as abstrações que permitem criar sistemas, independentemente das tecnologias das plataformas (BRUNELIèRE; CABOT; JOUAULT, 2010). Desta maneira, os engenheiros de software podem trabalhar em um nível mais alto de abstração, em modelos que melhor captam a essência e a lógica da aplicação. Consequentemente, é possível reduzir as tarefas repetitivas de codificação, que ficam a cargo de transformações automatizadas. Além disso, o MDE possibilita que o desenvolvedor fique protegido das complexidades requeridas para implementação nas diferentes plataformas (FRANCE; RUMPE, 2007).

Neste sentido, este trabalho envolveu, primeiramente, o desenvolvimento de uma linguagem textual que possibilita a especificação de aplicações em um alto nível de abstração. Também foram construídos geradores para duas conhecidas plataforma de nuvem, a *Google App Engine*[1] (GAE) e a *Windows Azure*[2], demonstrando que a linguagem desenvolvida, apesar de simples, pode servir de entrada para a geração de grande parte do código necessário para muitas aplicações de nuvem e que facilita seu desenvolvimento. No momento, os geradores construídos permitem a portabilidade de aplicações entre a GAE e a Azure. Somente essas duas plataformas foram abordadas, pois considerou-se que são suficientes para análise da portabilidade. Todavia, no futuro, é possível que sejam construídos geradores para outras plataformas a partir da mesma linguagem, ajudando assim, a resolver o problema da falta de portabilidade entre diversos provedores.

1.2 Motivação e Objetivo

Em termos de mercado, o International Data Corporation - IDC (2012) estima que, somente em 2011, serviços de nuvem ajudaram organizações de todos os portes ao redor do mundo a gerar mais que 400 bilhões de dólares em receita e 1.5 milhões de novos empregos. Além disso, a previsão é de que até 2015 o número de novos empregos gerados em decorrência do uso desse

[1] https://developers.google.com/appengine/
[2] http://www.windowsazure.com

modelo computacional ultrapassará 8.8 milhões (IDC, 2012). Portanto, faz-se necessário que conceitos e tecnologias relacionadas a esse modelo computacional sejam melhor explorados.

Neste contexto, levando-se em consideração o problema do *Lock-In* e com a motivação de melhorar o desenvolvimento de aplicações para nuvem e apresentar uma abordagem alternativa para facilitar a portabilidade entre as diversas plataformas existentes no mercado, este trabalho teve por objetivo propor: **uma abordagem dirigida por modelos (*Model-Driven Engineering - MDE*) para desenvolvimento de aplicações portáveis entre plataformas da Computação em Nuvem**. Tal abordagem é também um passo inicial para o desenvolvimento de uma abordagem MDE para interoperabilidade, algo a ser explorado em trabalhos futuros.

Em síntese, o objetivo deste trabalho pode ser desmembrado nos seguintes itens:

- Estudar as características dos provedores de serviços de nuvem;
- Desenvolver uma linguagem específica de domínio;
- Desenvolver de um conjunto de transformações para geração de aplicações específicas de plataforma; e
- Avaliar da portabilidade das aplicações geradas.

1.3 Metodologia

Para levantamento do estado da arte relacionado à Computação em Nuvem (apresentado no Capítulo 2), definição do tópico de pesquisa e dos trabalhos correlatos (apresentados no Capítulo 3) apresentados neste trabalho, foi feita uma revisão sistemática de literatura de acordo com o processo proposto por Kitchenhan (2007). Uma revisão sistemática supostamente apresenta uma avaliação justa do tópico de pesquisa e utiliza uma metodologia de revisão de literatura rigorosa, confiável e passível de auditagem (MAFRA; TRAVASSOS, 2005). Para possibilitar a documentação da revisão e facilitar o processo, foi utilizada a ferramenta *State of the Art through Systematic Review - START*[3]. Como parte do processo, foi desenvolvido um protocolo que apresenta um plano de pesquisa que foi seguido. 122 artigos foram analisados, dos quais 100 foram selecionados e seus dados foram extraídos e analisados de acordo com os critérios previstos no protocolo.

Toda a revisão foi documentada, desde a sua concepção à extração dos resultados, para permitir que outros pesquisadores interessados nos estudos possam fazer sua repetição ou ainda

[3]START: http://lapes.dc.ufscar.br/software/start-tool

tarefas de auditagem. O estudo foi conduzido em um período de 8 (oito) meses. Os resultados encontram-se publicados em Silva e Lucrédio (2012). Além dos conceitos apresentados no Capítulo 2, a revisão apresenta 10 (dez) lacunas de pesquisa relacionadas à Engenharia de Software para Computação em Nuvem (entre eles o *Lock-In*) e pode ser utilizada como uma importante ferramenta de pesquisa relacionada ao tema.

A partir da definição do tópico de pesquisa como a portabilidade de aplicações da nuvem e a MDE como uma alternativa e consequentemente uma hipótese para solução, a metodologia utilizada para desenvolvimento da abordagem explorou estudos de caso e prototipação de uma linguagem e transformações para geração automática de código para as plataformas de Computação em Nuvem *Google App Engine* e *Windows Azure*. Inicialmente, por meio dos estudos de caso, foram captadas características conceituais das plataformas. Os estudos de caso consistiram no desenvolvimento de aplicações de exemplo que serviram como referência para o desenvolvimento das transformações necessárias para geração de código para cada plataforma.

O domínio selecionado como objeto de estudo dessa abordagem foi o domínio de CRUD, que representa operações típicas de acesso a bases de dados (*Create, Retrieve, Update, Delete*), e que constitui parte importante da maioria das aplicações. Em seguida, foi feito o projeto do metamodelo do domínio. Junto com especificações de sintaxe textual concreta, esse metamodelo deu origem a uma linguagem específica de domínio (*Domain Specific Language* ou DSL) que descreve os conceitos do domínio em alto nível de abstração. Essa DSL constitui a base para a construção de modelos independentes de plataforma de nuvem. Logo após essa etapa, a partir do conhecimento adquirido nos estudos de caso, foram definidas transformações que são utilizadas para gerar código especifico de plataforma. A implementação da DSL foi feita utilizando a ferramenta Xtext[4] e o conjunto de transformações foi definido utilizando a ferramenta de geração Xtend[5].

A análise da portabilidade foi dividida em duas partes. Primeiro (Parte 1) foi feito um estudo de caso com base em uma situação-problema onde foram geradas aplicações para as plataformas utilizadas neste estudo (no caso a GAE e Azure) e uma análise e comparação do código e artefatos de implementação gerados pelo conjunto de transformações a partir de um modelo de domínio.

Em seguida, foi feita a análise das aplicações sob o ponto de vista do usuário (Parte 2). Basicamente, os testes consistiram em fazer com que usuários utilizassem as diferentes versões do sistema e respondessem se eles consideram que as aplicações foram portadas. Casos de teste

[4]Xtext: `http://www.eclipse.org/Xtext/`
[5]Xtend: `http://www.eclipse.org/Xtend/`

direcionaram essa utilização, de modo a prover uma boa cobertura das funcionalidades implementadas. Testes relacionados ao processo de desenvolvimento e demais testes relacionados à abordagem serão realizados em trabalhos futuros.

1.4 Contribuições

Em relação aos avanços no estado da arte do tópico de pesquisa abordado neste trabalho. Podem ser destacadas as seguintes contribuições:

- **Abordagem para redução do esforço da portabilidade:** As tarefas repetitivas de programação ficam a cargo de transformações automatizadas. Consequentemente, diminui-se o esforço para portabilidade de aplicações entre provedores de computação em nuvem.
- **Uma abordagem alternativa a padronização de APIs:** a maioria das iniciativas para se resolver o problema da falta de interoperabilidade giram em torno da padronização de APIs. A abordagem apresentada neste trabalho, diferentemente das demais, utiliza MDE como uma solução.
- **Benefícios adicionais oferecidos pelo MDE, em relação aos padrões:** o uso de MDE no desenvolvimento de software oferece benefícios adicionais em relação a abordagens tradicionais. Com mde é possível automatizar diverssas tarefas, aumentar o reuso da modelagem,etc.
- **Revisão sistemática:** a revisão sistemática desenvolvida fornece uma visão atualizada do tópico de pesquisa de Engenharia de Software para Computação em Nuvem. A mesma pode ser utilizada como uma importante ferramenta para os pesquisadores da área.
- **Facilitar adoção de nuvem por parte de desenvolvedores:** Uma vez que a portabilidade entre provedores é facilitada, o problema do aprisionamento em um provedor específico se torma mais fácil de se resolver. Isso pode ajudar a adoção do modelo de nuvem por parte dos desenvolvedores.

1.5 Organização do trabalho

Esta dissertação está organizada da seguinte forma:

- O Capítulo 2 apresenta o levantamento bibliográfico acerca dos conceitos de Computação em Nuvem, bem como portabilidade, interoperabilidade e desenvolvimento dirigido por modelos.
- O Capitulo 3 apresenta trabalhos que possuem certa correlação com este.
- A capítulo 4 apresenta o desenvolvimento da abordagem.
- O capítulo 5 apresenta a avaliação realizada.
- O capítulo 6 apresenta as conclusões e trabalhos futuros.

Capítulo 2

LEVANTAMENTO BIBLIOGRÁFICO

Este capítulo descreve os principais conceitos relacionados ao trabalho desenvolvido. Primeiramente, são definidos os conceitos da Computação em Nuvem. Em seguida, são apresentadas as formas de utilização dos serviços de nuvem sob uma perspectiva de Engenharia de Software e alguns provedores de nuvem e seus respectivos modelos de serviço. Por fim, são apresentados os conceitos de Desenvolvimento dirigido por modelos (MDE).

2.1 Computação em Nuvem

A Computação em Nuvem é uma das mais promissoras evoluções tecnológicas que ainda estão se desenvolvendo. Esta seção apresenta conceitos relacionados a essa tecnologia, bem como vantagens e desvantagens de sua utilização.

2.1.1 Conceito

A 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 através da Internet. A ideia principal desse conceito é permitir que empresas adquiram recursos computacionais sob demanda e o pagamento desse serviço seja feito de acordo com o volume de utilização (ARMBRUST et al., 2009). Embora a definição do *NIST* [1] seja a mais utilizada na literatura, não há ainda uma definição acadêmica da Computação em Nuvem amplamente aceita pela comunidade científica. Vaquero et al. (2008) fizeram um rigoroso levantamento de várias definições propostas na literatura no sentido de alcançar uma definição completa e elaboraram uma definição própria. Apesar de ser mais

[1] NIST: Instituto Nacional de Padrões e Tecnologia dos Estados Unidos - `http://www.nist.gov`

uma proposta, a definição alcançada se mostrou bastante coerente e completa, e, portanto foi utilizada neste trabalho de mestrado.

> Nuvens são grandes repositórios de recursos virtualizados (hardware, plataformas de desenvolvimento/ou serviços), facilmente acessíveis. Esses recursos podem ser reconfigurados dinamicamente de modo a se ajustar a cargas variadas, otimizando a utilização desses mesmos recursos. Esse repositório de recursos é tipicamente explorado utilizando-se um modelo do tipo pagamento-por-uso, onde os fornecedores de infra-estrutura oferecem garantias no formato de SLAs (service level agreements) customizadas (VAQUERO et al., 2008).

Essa definição cita o termo recursos, de uma forma genérica, como os elementos que compõem a infrestrutura da nuvem. Esse é o ponto de destaque com relação ao presente trabalho, pois sugere uma grande heterogeneidade nos ambientes de nuvens. Já que os recursos virtualizados podem ser de diferentes tipos, as aplicações que os utilizam também precisam ser diversificadas, o que acaba aumentando sua complexidade e tornando as aplicações mais difíceis de serem reutilizadas e/ou portadas. Na prática, o que acaba acontecendo é uma alta especialização das aplicações com relação a um tipo específico de recurso (hardware, plataforma e/ou conjunto de serviços), originando o problema denominado Lock-In (ARMBRUST et al., 2009), isto é, as aplicações ficam "presas" a um determinado provedor de nuvem. Esse problema será melhor discutido mais adiante neste trabalho.

Outro elemento característico da nuvem contido nessa definição é a maneira com que os recursos são tipicamente explorados. O modelo pagamento-por-uso gera uma grande flexibilidade no gerenciamento e manutenção das aplicações, tanto para o provedor de nuvem quanto para o fornecedor do software, o que é também conhecido como elasticidade. Para o provedor de nuvem, elasticidade significa um aproveitamento melhor dos recursos disponíveis, que no geral ficam menos ociosos devido à alta rotatividade das demandas. Para o fornecedor do software, elasticidade significa maior agilidade no atendimento às demandas pelo uso do software, que muitas vezes são variadas e exigem rápida reconfiguração dos recursos disponíveis. Essa característica também será discutida mais adiante, no entanto não tem relação direta com o objetivo deste trabalho.

Em relação à oferta de serviços de nuvem, várias taxonomias da Computação em Nuvem foram sugeridas, com o objetivo de caracterizar quais serão oferecidos sob o modelo pagamento por uso. Muitas delas foram criadas sob uma perspectiva de negócios (RIMAL; CHOI; LUMB, 2009), algumas apresentam termos como HaaS (Hardware as a Service), PaaS (Platform as a Service), DaaS ([Development, Database, Desktop] as a Service), IaaS (Infrastructure as a Service), BaaS (Business as a Service), FaaS (Framework as a Service), ou mesmo o termo XaaS (Everything as a Service), que engloba o conceito de serviço de forma genérica. As

definições desses termos são muito amplas e muitas vezes mal compreendidas (ARMBRUST et al., 2009). No entanto, observando-se a literatura é possível enquadrar as principais soluções nas seguintes categorias:

- **Software como Serviço – SaaS:** refere-se a aplicações que funcionam sobre a infraestrutura da nuvem e são providas como serviço aos consumidores através da Internet (ARMBRUST et al., 2009; MELL; GRANCE, 2009). Um SaaS normalmente opera sobre a infraestrutura da nuvem, mas nem sempre isso é necessário. Ex.: Google Drive[2], Flickr[3], Picasa[4], entre outras.

- **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). Isto inclui hospedagem, testes e entrega de sistemas. Ex.: Google App Engine[5], Microsoft Azure[6], entre outras.

- **Infraestrutura como Serviço – IaaS:** consiste na entrega de infraestrutura computacional como serviço (RIMAL; CHOI; LUMB, 2009). O cliente adquire, sob demanda, serviços de processamento, armazenamento e demais recursos computacionais fundamentais onde é possível implantar e rodar qualquer software. Essa oferta trata-se basicamente do aluguel de um computador virtual na infraestrutura do provedor. Ex.: Amazon[7], GoGrid[8], IBM SmartCloud [9], entre outras.

Youseff, Butrico e Da Silva (2008) propuseram uma ontologia [10] (Figura 2.1) que ilustra a interdependência e composição entre os diferentes tipos de serviços oferecidos pelos provedores. As camadas mais inferiores são responsáveis por virtualizar servidores [11] que serão utilizados pela infraestrutura, **são o que efetivamente podemos chamar de nuvem computacional** (ARMBRUST et al., 2009). O termo "serviço" no, contexto da definição dos conceitos, faz alusão ao modelo de entrega denominado *Utility Computing* , isto é, aquisição de recursos computacionais sob demanda. Para um melhor entendimento do que é aquisição sob demanda,

[2]Google Drive: `http://drive.google.com/`
[3]Flickr: `http://www.flickr.com/`
[4]Picasa: `http://picasa.google.com/`
[5]Google App Engine: `http://code.google.com/appengine`
[6]Windows Azure: `http://www.windowsazure.com/`
[7]Amazon:`http://aws.amazon.com/pt/ec2/`
[8]GoGrid: `http://www.gogrid.com/`
[9]`http://www.ibm.com/cloud-computing/`
[10]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)
[11]A virtualização de servidores é a técnica de execução de um ou mais servidores virtuais sobre um servidor físico (DANIELS, 2009)

pode-se fazer uma analogia com os modelos de entrega dos serviços de água, energia elétrica e gás. Portanto, SaaS seria um serviço de software, IaaS um serviço de infraestrutura e PaaS um serviço de plataforma de desenvolvimento e hospedagem de aplicações.

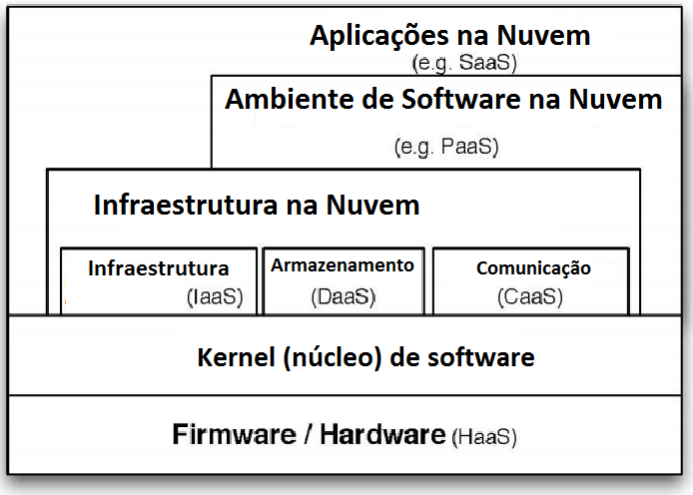

Figura 2.1: Ontologia da Computação em Nuvem(adaptado de Youseff, Butrico e Da Silva (2008))

Outra classificação que pode ser encontrada na literatura é relacionada ao modelo de implantação da nuvem. Quando os serviços são disponibilizados ao público em geral, normalmente por uma empresa especializada em comercializar serviços de nuvem, denomina-se **nuvem pública** (ARMBRUST et al., 2009; MELL; GRANCE, 2009). Alguns exemplos são as nuvens da Microsoft[12], Google[13] e Amazon[14]. 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, 2009). Na prática, é uma infraestrutura de nuvem dedicada a uma empresa, se comportando de forma semelhante a um ambiente de computação em nuvem pública (RIMAL; CHOI; LUMB, 2009). Neste 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. O modelo de **nuvem híbrida** prevê uma utilização mista, porém integrada, dos demais paradigmas, isto é, a combinação de serviços de nuvens públicas com recursos de nuvens privadas (ARMBRUST et al., 2009).

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 API's.

[12]Windows Azure: `http://www.windowsazure.com/`
[13]Google App Engine: `http://code.google.com/appengine`
[14]Amazon:`http://aws.amazon.com/pt/ec2/`

2.1.2 Vantagens

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 magros [15] e incorporar dispositivos móveis, além de hardware especifico que porventura venha a surgir. Para isto, basta que eles estejam conectados à Internet (VELTE; VELTE; ELSENPETER, 2009).

- **Maior facilidade de gerenciamento**: não é necessário manter equipes especializadas para manter a infra-estrutura de suporte aos aplicativos da empresa. Todas as tarefas de manutenção e backup são feitas por quem está sendo contratado para fornecer os serviços de nuvem (ARMBRUST et al., 2009);

- **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. Desta maneira, a empresa economiza com energia, aquisição de infraestrutura e contratação de pesssoal. 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 do uso da computação em nuvem antes da sua completa adoção (SILVA; LUCRéDIO, 2012). Na literatura é possível encontrar alguns trabalhos relacionados a adoção do modelo de nuvem (ZARDARI; BAHSOON, 2011; SARIPALLI; PINGALI, 2011; KHAJEH-HOSSEINI et al., 2011, 2012; YAM et al., 2011). No trabalho de Narasimhan e Nichols (2011) é possível encontrar um estudo da nuvem sobre a perspectiva das empresas que a adotam. No entanto não foi encontrado um processo formal específico para apoiar essa tarefa.

[15] 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).

2.1.3 Desvantagens

Há muitos quesitos que ainda não estão bem estabelecidos ou ainda em fase de pesquisa (SILVA; LUCRéDIO, 2012) e consequentemente 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 flexivel e permite negociar condições adicionais com usuários que não se convencerem das políticas adotadas pelos provedores (KANDUKURI; PATURI; RAKSHIT, 2009). Há muitas pesquisas e considerações sendo feitas em torno desse tema. A Cloud Security Alliance [16] reúne esforços para promover melhores práticas na garantia de segurança e oferecer treinamentos para ajudar a proteger todas as formas de computação. Certamente, no futuro as aplicações de Computação em Nuvem serão muito 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 pesquisas em tecnologias *open source* (SILVA; LUCRéDIO, 2012). Até o momento, boa parte dos SaaS disponibilizados só funcionam em uma plataforma. Isto gera uma dependência direta do usuário à plataforma, fazendo com que o usuário fique preso ao provedor.

- **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 auferir lucro. Neste quesito, pode-se citar o Gmail[17] que utiliza informações do usuários para redirecionar propogandas personalizadas.

[16] Cloud Security Alliance: `https://cloudsecurityalliance.org/`
[17] GMail: `http://mail.google.com/`

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. Algumas ofertas têm como público alvo desenvolvedores; outras, usuários finais ou ambos. A seguir apresenta-se como os principais tipos de serviços podem ser utilizados. Antes, no entanto, discute-se a diferença entre SaaS e Service-Oriented Architecture (SOA), para elucidar uma confusão referente a esses termos.

2.2.1 SaaS vs SOA

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. Os autores propuseram uma abordagem para interoperabilidade entre aplicações de nuvem que utiliza MDE para gerar *Web Services Description Language*[18]*(WSDL)* 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 a criação de serviços para compor uma arquitetura mais flexível para uma aplicação, SaaS se refere a entrega da funcionalidade de software em forma de serviços, que são providos independentemente do estilo arquitetural do software.

Para uma melhor compreensão da 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 US$ 5,00 por mês para cada pessoa em sua conta [19]. 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 Submarino [20], por exemplo, utiliza um web service dos Correios[21] para efetuar cálculo do valor do frete. Para isso, o sistema faz uma

[18]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`

[19]Google Apps: `http://www.google.com/apps/intl/en/business/index.html`

[20]Submarino: `http://www.submarino.com.br`

[21]http://www.correios.com.br/

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 explicito o uso de serviços para proporcionar mais flexibilidade e permitir que modulos específicos sejam construídos (PAPAZOGLOU et al., 2007, 2008).

Ainda com relação a esse assunto, cabe fazer uma comparação entre as definições de SaaS e o modelo de provedor de aplicações (*Application Server Provider - 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 baseada na 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) através da Internet e paga sob demanda. Já sistemas como o famoso ERP[22] JD Edwards[23] 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 fornecer componentes para SOA, e SOA ajuda a realizar SaaS mais rapidamente (LAPLANTE; ZHANG; VOAS, 2008).

2.2.2 Utilização do Modelo SaaS

Como já discutido anteriormente, a Computação em Nuvem dá aos provedores de aplicação a escolha de negociar seus produtos como SaaS (ARMBRUST et al., 2009). Para os clientes não é necessário a aquisição de uma grande infra-estrutura. Até computadores com pequeno poder computacional e hardware que eventualmente venham a surgir podem ser utilizados como clientes. O modelo de entrega SaaS poderá revolucionar a forma de como as empresas vão investir em sistemas de informação. 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

[22] Enterprise Resource Planning

[23] JDEdwards:http://www.oracle.com/br/products/applications/jd-edwards-enterpriseone/index.html

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. Com isto, empresas que possuem software de alto custo poderão se inserir em um novo segmento de mercado (TAURION, 2009).

Do ponto de vista da Engenharia de Software, para adotar esse modelo 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 sistema 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 às vezes devem ser consideradas pelo engenheiro de software, que deve prover funções administrativas de apoio a esse tipo 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 isso é necessário que o sistema seja independente de plataforma.

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 incumbe 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 um maior 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 resolvido, as demandas de serviços computacionais passarão a ser mais um item nas listas de licitações.

Na perspectiva de Engenharia de Software, 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, no entanto, há o diferencial de que a entrega ao usuário final sempre ocorre através da Internet, o que na prática já acontecia na maioria dos casos onde nuvens não eram utilizadas. Cabe lembrar que não necessariamente o sistema será entregue sob a modalidade SaaS e tão pouco 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.

Nesse modelo 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 al-

guma 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 apolicações para nuvem (SILVA; LUCRéDIO, 2012). Por exemplo: o Google App Engine provê muitas 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 visão da Engenharia de Software, para adotar esse modelo a equipe de desenvolvimento utiliza ferramentas e bibliotecas fornecidas pela plataforma para desenvolver suas aplicações. Normalmente, isso exige um certo treinamento para que os desenvolvedores possam conhecer e utilizar a plataforma adequadamente. 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 Seção 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 (ESPARZA; ESCOÍ, 2011).

2.3 Provedores de Nuvem

Para a Engenharia de Software, um provedor de Computação em Nuvem é um fornecedor de recursos virtualizados, seguindo principalmente os modelos PaaS e IaaS. Cada provedor dita a forma como um ambiente é disponibilizado, ou seja, quais tecnologias, padrões e aplicações devem ser utilizadas pelos desenvolvedores para criar ou prover software para os clientes. Existe uma grande quantidade de provedores, alguns dos quais são listados na Tabela 2.1.

Conforme abordado na Seção 2.2, para utilizar o modelo IaaS, normalmente o desenvol-

Tabela 2.1: Provedores de Computação em Nuvem e seus respectivos modelos de serviço

Modelo	*Nome do Provedor*	*Website*
SaaS	- Google Docs - Salesforce CRM - Flickr - Picasa	-http://docs.google.com/ -http://www.salesforce.com/ -http://www.flickr.com/ -http://picasa.google.com/
PaaS	- Google App Engine - Microft Azure - Cloud Foundry	-http://www.windowsazure.com/ -http://cloud.google.com/appengine/ -http://www.cloudfoundry.com/
IaaS	- Eucalyptus - GoGrid - Flexiscale - Amazon AWS EC2	-http://www.eucalyptus.com/ -http://www.gogrid.com/ -http://www.flexiscale.com/ -http://aws.amazon.com/ec2/

vedor cria suas aplicações da maneira tradicional, as implanta no provedor e as disponibiliza aos clientes. Ou seja, do ponto de vista da Engenharia de Software, não há muita diferença com relação ao desenvolvimento tradicional, exceto pelo fato de que, uma vez implantadas, as aplicações podem colher os benefícios de escalabilidade associados à computação em nuvem. Já o modelo PaaS apresenta algumas particularidades. Seu objetivo é ocultar as complexidades do desenvolvimento de sistemas para nuvem, provendo uma plataforma consistente e de alto nível para desenvolver aplicações escaláveis. 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). Por esses motivos, o modelo PaaS exige que o desenvolvedor adote um conjunto de ferramentas, bibliotecas, tecnologias e padrões disponibilizados pelo provedor da plataforma para criar suas aplicações

É justamente neste ponto que surge o problema da portabilidade de aplicações no contexto de PaaS (conforme será melhor detalhado na Seção 2.5). Ao optar por uma plataforma, o desenvolvedor fica atrelado às suas tecnologias particulares, o que dificulta a migração da aplicação para um provedor diferente. No entanto, independentemente das diferenças entre as plataformas, há um núcleo conceitual comum, que pode ser reaproveitado para proporcionar maior portabilidade.

Por esse motivo foi feito um estudo sobre quais são os elementos desse núcleo conceitual comum. O resultado desse estudo é apresentado a seguir, com uma visão geral das principais características de dois dos principais provedores de nuvem na atualidade, o Google App Engine e a Microsoft Windows Azure. Ainda que não tenham sido feitos estudos comparativos com outras plataformas, acredita-se que, salvo algumas diferenças de menor impacto, as mesmas

sejam comuns a outras plataformas que utilizam o mesmo modelo de oferta de serviço (PaaS).

2.3.1 Google App Engine

O *Google App Engine* (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 ambiente Java [24], entretanto, qualquer outro poderia ser utilizado. O ambiente Java do GAE fornece uma máquina virtual Java, uma interface de Java *Servlets*, e serviço de armazenamento de dados.

No momento da escrita deste trabalho, 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, 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 VRaptor[25], Struts[26], entre outros.

Os aplicativos podem usar os serviços de armazenamento de dados do GAE, entre eles o *Datastore*. Há suporte para duas interfaces Java padrão: *Java Data Objects* (JDO)[27] 2.3 e *Java Persistence API*[28] (JPA). Essas interfaces são implementadas usando a Plataforma de acesso *DataNucleus*[29], uma implementação de código aberto desses padrões.

Não é necessário nenhum investimento inicial para começar a desenvolver aplicações utilizando o *App Engine*. 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`".

[24] Para maiores informações a respeito das demais linguagens suportadas pelo GAE acesse: `https://developers.google.com/appengine/`

[25] Vraptor: `http://vraptor.caelum.com.br/`

[26] Struts:`http://struts.apache.org/`

[27] JDO:`http://www.oracle.com/technetwork/java/index-jsp-135919.html`

[28] JPA:`http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html`

[29] `http://www.datanucleus.org/`

Como auxílio ao desenvolvedor, existe um suporte personalizado na forma de *plugins* para a IDE Eclipse, de forma a facilitar o desenvolvimento de aplicações para o GAE.

2.3.2 Microsoft Windows Azure

O *Windows Azure* é a plataforma (PaaS) de nuvem da Microsoft. Para criar aplicações e serviços, os desenvolvedores podem usar o conhecimento já adquirido no *Microsoft Visual Studio*. Além disso, são suportados protocolos e padrões populares, como o protocolo simples de acesso a objetos - SOAP, a Transferência de Estado Representacional - REST e *eXtensible Markup Language* - XML. A plataforma combina recursos de desenvolvimento baseados na nuvem com serviços de infraestrutura de rede, computação e armazenamento. Assim como o GAE, a *Azure* tem um ambiente de desenvolvimento próprio para linguagens específicas. As linguagens que possuem suporte nativo são .Net, Node.js, Java e PHP. Porém, aplicações escritas em outras linguagens como por exemplo Python podem também ser implantadas através do uso de virtualização.

A plataforma disponibiliza um domínio de rede do tipo: "`http://<nomedaaplicacao>.cloudapp.net`" e, no momento da escrita desta dissertação, a política de gratuidade se estende aos três primeiros meses de uso, sendo que após esse período é necessário habilitar e efetuar o pagamento. O *Windows Azure* provê ainda uma tecnologia própria para armazenamento de dados. São três as formas de persistência: tabelas, filas e *blobs* (grandes blocos de dados em formato binário). Além disso há um ambiente de dados e processamento de consultas baseado no *SQL Server*, chamado de *SQL Azure*.

Assim como no caso do GAE, a IDE Eclipse pode ser utilizada para desenvolver aplicações para essa plataforma.

2.4 Conceitos de Desenvolvimento Dirigido Por Modelos

Apesar da evolução das técnicas e ferramentas de desenvolvimento de sistemas, preocupações com modelagem, reuso, 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). As abordagens MDE 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. O desenvolvedor

não precisa interagir manualmente com todo o código-fonte. Pode concentrar-se em modelos de maior nível de abstração, o que o protege das complexidades inerentes às diferentes plataformas (FRANCE; RUMPE, 2007). A ideia principal é reconhecer a importância dos modelos no processo de desenvolvimento de software e, não apenas utilizá-los como "um guia", mas torná-los parte integrante do software assim como o código fonte. Mecanismos de transformação 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).

Nesse contexto, um modelo é uma descrição ou especificação abstrata do sistema, sendo geralmente representado como uma combinação de elementos gráficos e textuais (OMG, 2003). No desenvolvimento de sistemas de maneira tradicional, os modelos são originados em fase 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 o fato de que, muitas vezes, mudanças são feitas diretamente no código, considerando geralmente as restrições de tempo. Além disso, criar e manter a documentação atualizada são normalmente tarefas manuais não muito apreciadas por desenvolvedores. Dessa maneira, os modelos rapidamente se tornam 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).

Na MDE o foco nos modelos busca 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).

2.4.1 Vantagens e desvantagens do desenvolvimento dirigido por modelos

As principais vantagens da abordagem MDE estão relacionadas aos problemas recorrentes no desenvolvimento de software. A Tabela 2.2 apresenta uma compilação delas.

Todavia, a MDE não oferece somente benefícios. A Tabela 2.3 apresenta algumas desvan-

Tabela 2.2: Principais Vantagens da Prática de MDE (CIRILO, 2011)

Vantagem	*Caracterização*
Produtividade	- Automatização da geração de código a partir de modelos através 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; - A documentação permanece atualizada, o que facilita as tarefas de manutenção.
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 de 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.
Verificações e Otimizações	- Os modelos facilitam a análise por verificadores semânticos, conforme a sintaxe de seu metamodelo, e a execução de otimizações automáticas; - Minimização da 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.

Tabela 2.3: Desvantagens do MDE (CIRILO, 2011)

Desvantagem	*Caracterização*
Rigidez	- Grande parte do código é gerado e fica longe do alcance do desenvolvedor;
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 orientada a modelos requer mais tempo e esforço.

tagens dessa abordagem (AMBLER, 2003; THOMAS, 2004).

Para que essa abordagem ocorra de fato, são necessários alguns elementos. A Figura 2.2 apresenta os principais elementos da MDE e como eles estão presentes no processo de desenvolvimento.

- **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 [30](Domain Specific Language - Linguagem específica de domínio).

- **Ferramenta para definir transormações**: são utilizadas para gerar outros modelos e código fonte a partir de modelos recebidos como entrada. Através delas são construídas as regras de mapeamento.

- **Modelos**: servem de entrada para transformações e representam o conhecimento do do-

[30]DSL: uma DSL é uma linguagem projetada para ser útil para um conjunto específico de tarefas em um domínio específico (DEURSEN et al., 2000)

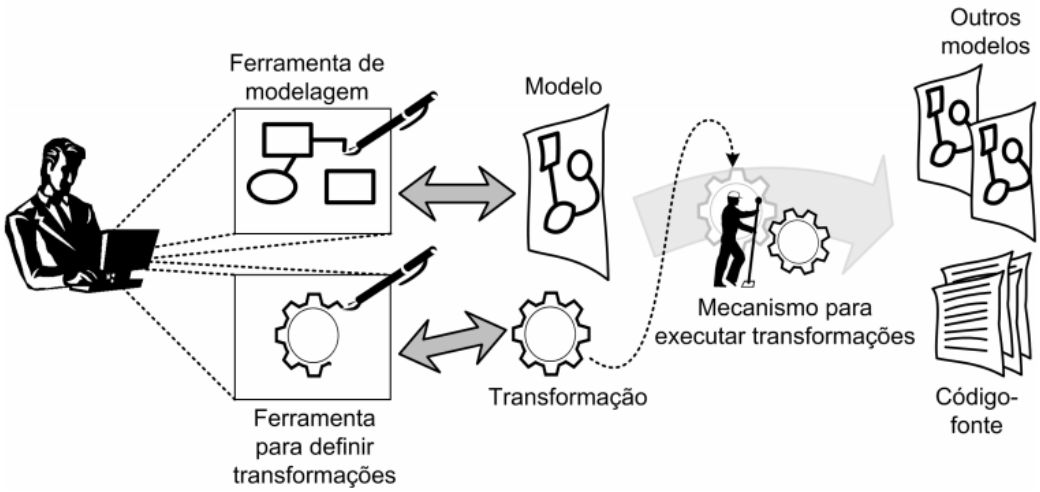

Figura 2.2: Principais elementos da MDE (LUCRéDIO, 2009)

mínio da aplicação.

- **Mecanismos para executar transformações**: executa as transformações seguindo as regras de mapeamento de modelo para modelo ou de modelo para código. Mantêm informações de rastreabilidade, possibilitando muitas vezes saber a origem de cada elemento gerado, seja modelo ou código fonte.

- **Outros Modelos e código fonte**: são resultantes do processo de transformação.

2.4.2 Metamodelagem

As principais abordagens MDE baseiam-se no conceito de metamodelagem, que dá suporte a 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 trata-se de um modelo que fornece bases para 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). A Figura 2.3 apresenta a arquitetura de metamodelagem comumente utilizada pela comunidade de desenvolvimento.

- **Nível M3**: a camada meta-metamodelo é utilizada para definir linguagens de modelagem (como a UML). Normalmente, o meta-metamodelo é uma instância de si próprio. Existem alguns meta-metamodelos utilizados na indústria, adotados por diferentes ferramentas de modelagem e transformação. Na figura, é citado o MOF (OMG, 2003), um padrão estabelecido pelo *OMG (Object Management Group)*.

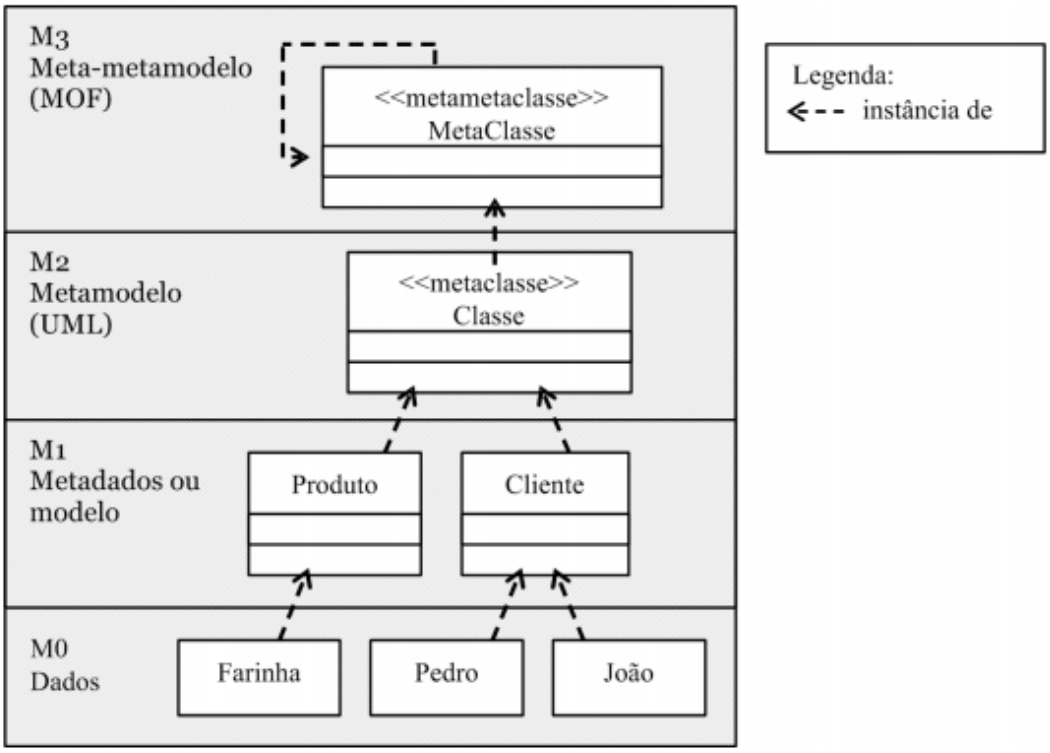

Figura 2.3: Arquitetura de metamodelagem. A figura acima foi extraída de (LUCRéDIO, 2009). No entanto, a mesma aparece recorrentemente em diversas fontes bibliográficas.

- **Nível M2**: a camada metamodelo 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 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.

2.4.3 Abordagens da indústria para a MDE

Há na indústria diversas abordagens para o desenvolvimento dirigido por modelos, entre elas está a *Model-Driven Architecture (MDA)*. A MDA apresenta conceitos como CIM (Computation Independent Model - Modelo Independente de Computação), PIM *(Platform Independent Model - Modelo Independente de Plataforma)* e PSM *(Platform-Specific Model - Modelo Específico de Plataforma)*. Um CIM é uma visão do sistema de um ponto de vista que não depende de computação. Ele não mostra detalhes da estrutura dos sistemas. É também conhecido como modelo do domínio e utiliza um vocabulário familiar aos profissionais e especialistas

no domínio em questão. O CIM pode ser mapeado para um PIM e um PSM. Um PIM é uma visão do sistema de forma independente da plataforma de implementação. Um PSM é uma visão do sistema do ponto de vista de uma plataforma específica. Combina as especificações de um PIM com detalhes sobre como o sistema utiliza aquele tipo particular de plataforma (OMG, 2003).

Na MDA, transformações são utilizadas para transformar um modelo em outro, sucessivamente, até o código-fonte. O meta-metamodelo utilizado nesta abordagem é *Meta Object Facility (MOF)*. A transformação entre CIM e PIM é menos passível de automação, pois envolve mais decisões e maiores possibilidades de interpretação dos requisitos. Já a transformação entre PSM e código-fonte é mais passível de automação, já que o PSM está intimamente ligado com a plataforma de implementação (LUCRéDIO, 2009).

A empresa Sun Microsystems, comprada pela Oracle em 2009, também fornece meios de se trabalhar com MDE. A Sun através do seu ambiente de desenvolvimento NetBeans, busca oferecer ao desenvolvedor a opção de criar sistemas simultaneamente no modelo e código-fonte. A empresa oferece ferramentas que seguem os padrões OMG, e sua principal iniciativa é baseada no *Java Metadata Interface (JMI)*.

Outra abordagem muito 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 JMI, mas sim 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 GMF (Graphical Modeling Framework). Esse projeto permite a definição completa de um ambiente de modelagem para uma linguagem visual específica para um determinado domínio. 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 Xtext, 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 JET e Xpand.

2.5 Portabilidade e Interoperabilidade

O principal problema da falta de portabilidade é que o trabalho empenhado em tarefas específicas de plataforma não pode ser reaproveitado em outras plataformas. Idealmente, o desenvolvimento de software deveria ser mais conceitual e menos focado em esforço repetitivo (LUCRéDIO; ALMEIDA; FORTES, 2012).

A norma ISO 9126 (1991), desenvolvida para identificar atributos de qualidade de software, define portabilidade como a facilidade com a qual o software pode ser transposto de um ambiente para outro conforme os seguintes sub-atributos:

- **Adaptabilidade:** representa a capacidade do software de se adaptar a diferentes ambientes sem a necessidade de ações adicionais (configurações);
- **Capacidade para ser Instalado:** identifica a facilidade com que se pode instalar o sistema em um novo ambiente;
- **Coexistência:** mede o quão facilmente um software convive com outros instalados no mesmo ambiente; e
- **Capacidade para Substituir:** facilidade de o sistema ser substituído por outro.

Sendo a portabilidade um atributo de qualidade de software, tal característica se torna fundamental para melhora e disseminação do modelo de nuvem. A portabilidade entre plataformas de nuvem é um tópico atual e tem sido explorado por diversos trabalhos na literatura (conforme será melhor apresentado no Capítulo 3). Existem alguns tipos de portabilidade, dentre os quais são destacados o seguintes:

- **Portabilidade de máquinas virtuais entre provedores de nuvem:** o modelo IaaS normalmente se utiliza da virtualização de servidores [31] para prover recursos computacionais aos clientes. Geralmente, as máquinas virtuais são gerenciadas pelos próprios clientes.

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

Apenas migrar uma máquina virtual de um provedor para o 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[32]) consiste em uma alternativa para padronizar as máquinas virtuais e consequentemente permitir que as mesmas sejam executadas por diferentes provedores que suportam o formato.

- **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 Locaweb[33], 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 na Seção 2.2.4, 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. É neste cenário que o presente trabalho se enquadra, conforme discutido no Capítulo 4.

- **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, onde os sistemas instalados pelo o usuário podem ser diversos; ou tecnologia previamente estabelecida, onde o provedor suporta apenas SGBDs específicos. Sendo assim, da mesma maneira que na portabilidade de aplicações,

[32]OVF: http://www.dmtf.org/standards/ovf
[33]Locaweb: `http://www.locaweb.com.br`

antes de fechar o contrato com o provedor, é importante verificar se o mesmo suporta o SGBD em questão. Shirazi et al. (2012) apresenta uma solução baseada em padrões de projeto para possibilitar portabilidade de dados entre alguns banco dados de nuvem. No entanto, a solução ainda está em fase de pesquisa e mais estudos ainda são necessários.

2.5.1 Portabilidade e interoperabilidade no contexto de 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. Não é dificil ver porque o **Lock-In** está dificultando a adoção do modelo de nuvem. A possibilidade de ficar preso 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 possivel 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 2.4 ilustra uma possível utilização da MDE, em sua proposta original, para o desenvolvimento de aplicações portáveis entre plataformas de Computação em Nuvem.

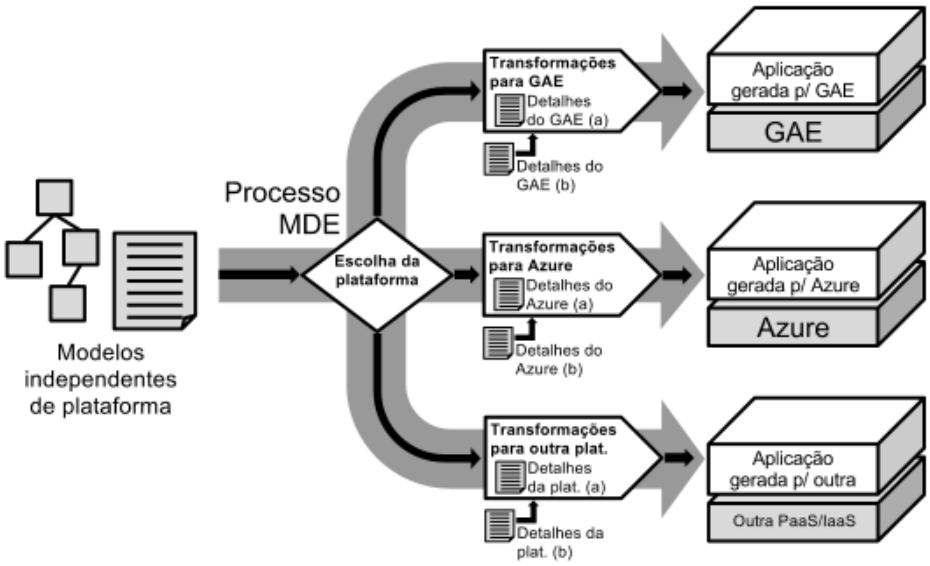

Figura 2.4: MDE para geração de aplicações portáveis

Através de um processo MDE devem ser geradas aplicações para diferentes plataformas de Computação em Nuvem. Como há várias ofertas de serviço, é necessário construir mecanismos que as explorem. Isto é, a partir de um mesmo modelo genérico deve ser possível gerar sistemas (ou parte deles) específicos para PaaS ou demais tecnologias relacionadas a nuvem. 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. É importante ressaltar que os detalhes específicos de plataforma podem ser especificados direto nas transformações/processo (Figura 2.4a) ou como modelos extras (Figura 2.4b). Ou seja, esses detalhes de plataforma podem ficar encapsulados nas transformações ou serem representados através de outros modelos de alto nível que também podem servir de entrada para os transformadores.

Outro conceito relacionado à portabilidade, porém complementar, é o de interoperabilidade. No contexto de computação em nuvem, interoperabilidade consiste em fazer com que uma mesma aplicação execute sobre uma combinação heterogênea de plataformas e provedores, mantendo uma funcionalidade global consistente.

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 (KLEPPE; JOS; WIM, 2003). A Figura 2.5 ilustra a 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 podem servir de entrada para os transformadores. Todavia, neste caso 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.

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 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 de forma independente.

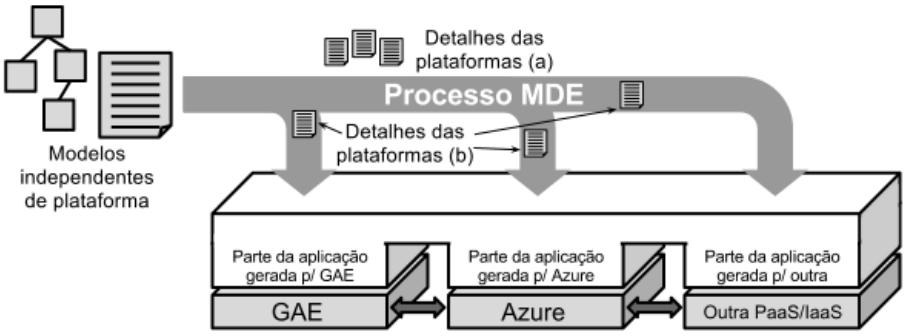

Figura 2.5: MDE para geração de aplicações interoperáveis

Com relação a esses dois conceitos (portabilidade e interoperabilidade) ARMBRUST et al. (2009) sugeriram que fosse criado um padrão para desenvolvimento de sistemas para nuvem de modo que seja 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.

Nessa linha, o Opencloud manifesto[34] está reunindo esforços no sentido de estabelecer um conjunto de princípios para garantir que as organizações tenham liberdade e flexibilidade na escolha das ofertas de nuvem.

No quesito interoperabilidade o manifesto apresenta a seguinte argumentação:

> "É importante que dados e aplicações exponham interfaces padronizadas. As organizações irão querer a flexibilidade para criar novas soluções e aplicativos que interagem uns com os outros, independentemente de onde eles estejam (nuvens públicas, privadas, em ambientes tradicionais de TI ou uma combinação). Provedores de nuvem precisam suportar padrões de interoperabilidade para que as empresas possam combinar capacidades de qualquer fornecedor de nuvem e suas soluções."

Quando se trata da portabilidade, o manifesto justifica sua necessidade afirmando que para os desenvolvedores seria difícil tomar a decisão de abandonar o modelo de nuvem para voltar ao modelo tradicional ou ainda migrar de um provedor para outro, uma vez que os sistemas seriam construídos utilizando-se bibliotecas e padrões específicos adotados por cada provedor de nuvem e uma mudança como essa geraria muito custo.

[34] `http://www.opencloudmanifesto.org/`

2.6 Considerações finais

Este capítulo apresentou os conceitos relacionados a computação em nuvem, desenvolvimento dirigido por modelo, bem como portabilidade e interoperabilidade no contexto de computação em nuvem e MDE. Tais conceitos são necessários para entendimento do tópico de pesquisa abordado neste trabalho, bem como configura uma visão do estado-da-arte das áreas de pesquisa envolvidas no estudo. No capítulo seguinte serão apresentados trabalhos que possuem certa correlação com o trabalho desenvolvido nesta pesquisa.

Capítulo 3

TRABALHOS CORRELATOS

Este capítulo apresenta alguns dos trabalhos que possuem relação com o trabalho desenvolvido nesta pesquisa. A organização do capítulo está conforme segue: a seção 3.2 apresenta uma abordagem dirigida por modelos proposta por Sharma e Sood 2011. A seção 3.3 apresenta o uma abordagem proposta por Brunèriere et al2010. Por fim, a seção 3.4 apresenta outras abordagens encontradas na literatura.

3.1 Considerações Iniciais

Na literatura foram encontrados diversos trabalhos relacionados ao problema da falta de portabilidade e interoperabilidade em aplicações da Computação em Nuvem. O tema tem sido bastante abordado pela comunidade acadêmica, o que demonstra sua relevância. As propostas para desenvolvimento de aplicações para nuvem giram basicamente em torno de duas abordagens: padronização de APIs e MDE. A seguir são apresentadas as principais abordagens relacionadas com este trabalho.

3.2 Abordagem de Sharma e Sood (2011)

Sharma e Sood (2011) propuseram uma abordagem dirigida por modelos para interoperabilidade em SaaS. Os autores definem os modelos em diferentes níveis de abstração baseados em separação de interesses - Computation Independent Model (CIM), Platform Independent Model e um Platform Specific Model (PSM) tomando por base a MDA. Cada nível pode ser composto de um ou mais modelos para especificar aspectos estruturais, funcionais e comportamentais do sistema.

Na modelagem do PIM é utilizada uma definição formal das operações oferecidas pelo serviço que podem ser acessadas através de uma interface que posteriormente deve ser composta com outros serviços para montar um sistema completo. As regras de negócio dessa interface são especificadas através da declaração de restrições, pré-condições e pós-condições e invariantes em Object Contraint Language[1] (OCL). A Figura 3.1 apresenta o modelo de uma funcionalidade de um Sistema de Reservas de Hotel; utilizada pelos autores como exemplo para especificação do PIM.

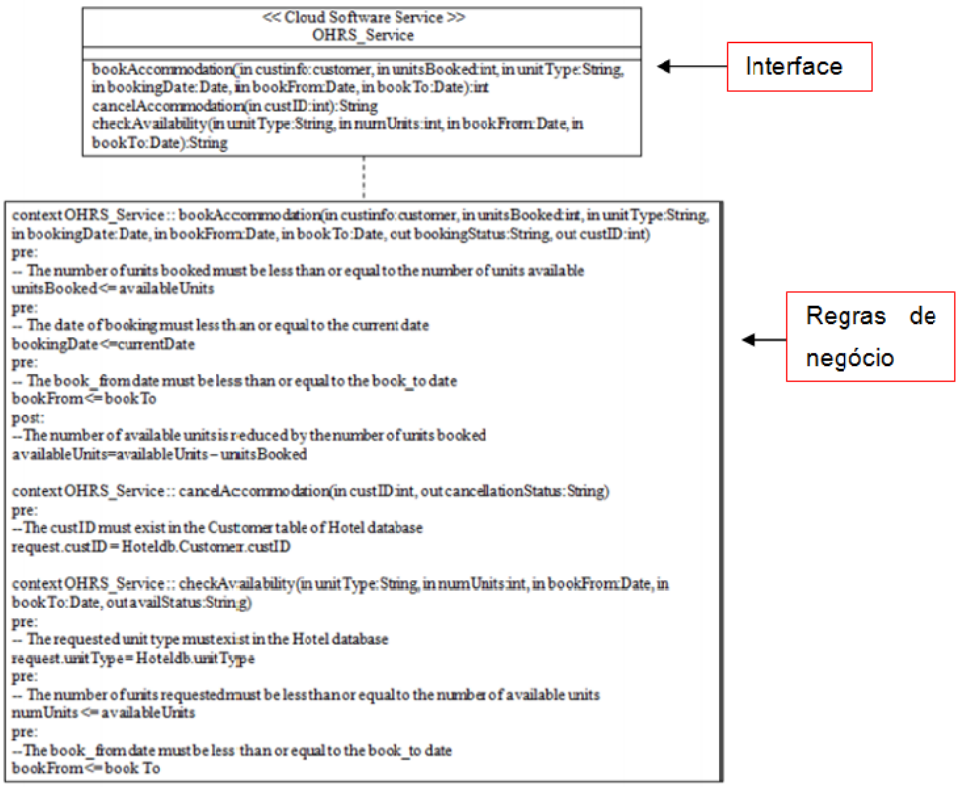

Figura 3.1: Um modelo formal (PIM) de um serviço de negócio (SHARMA; SOOD, 2011)

Após algumas transformações o PIM é então convertido em um WSDL [2] PSM, que consiste em um modelo específico para a linguagem de descrição de web services. A Figura 3.2 apresenta o exemplo utilizado após as transformações.

A etapa final é a transformação do WSDL PSM em uma WSDL. Os autores afirmam que, no contexto de MDE, a WSDL representa a PSM do SaaS e que em um Cloud SaaS esta mesma linguagem é responsável pela troca de mensagens e comunicação entre a interação dos serviços.

[1] A linguagem OCL é uma linguagem de texto precisa que possibilita a expressão de restrições em um modelo orientado a objeto que não possam ser especificadas através dos diagramas (OMG, 2012).

[2] WSDL: Uma Web Services Description Language (WSDL) é uma linguagem baseada em XML utilizada para descrever Web Services funcionando como um contrato do serviço. Trata-se de um documento escrito em XML que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis (W3C, 2012).

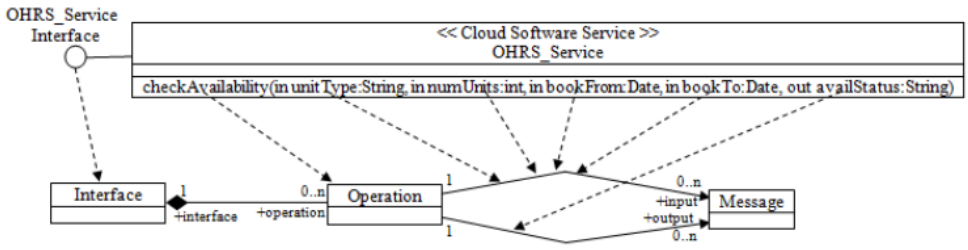

Figura 3.2: Serviço de nuvem (PIM) mapeado para um WSDL PSM (SHARMA; SOOD, 2011)

Segundo os autores, isso garante a interoperabilidade no nível PSM na abordagem orientada a modelos. E como a WSDL é independente de plataforma e pode ser implantada em qualquer provedor que dê suporte a essa tecnologia, a portabilidade também estaria garantida.

No entanto, é possível observar que a partir de um modelo de entrada e de seguidas transformações o resultado obtido foi uma WSDL (A Figura 3.3 apresenta a sequência de tarefas que foram realizadas até o resultado final). Uma WSDL é uma forma de implementação de SOA através de Web services e não necessariamente uma especificação de SaaS. Como foi abordado na Seção 2.2.1 desse trabalho, SaaS refere-se ao modelo de entrega, ou seja, a maneira com que o sistema será negociado e disponibilizado ao cliente. Portanto, sendo a WSDL uma maneira de implementar SOA e pelo fato da abordagem não tratar geração de sistemas para o modelo de PaaS, é possivel chegar a conclusão que a abordagem apresentada refere-se à descrição de serviços utilizando modelos e que a mesma não trata da geração de código para todos os tipos de ofertas de serviços de nuvem.

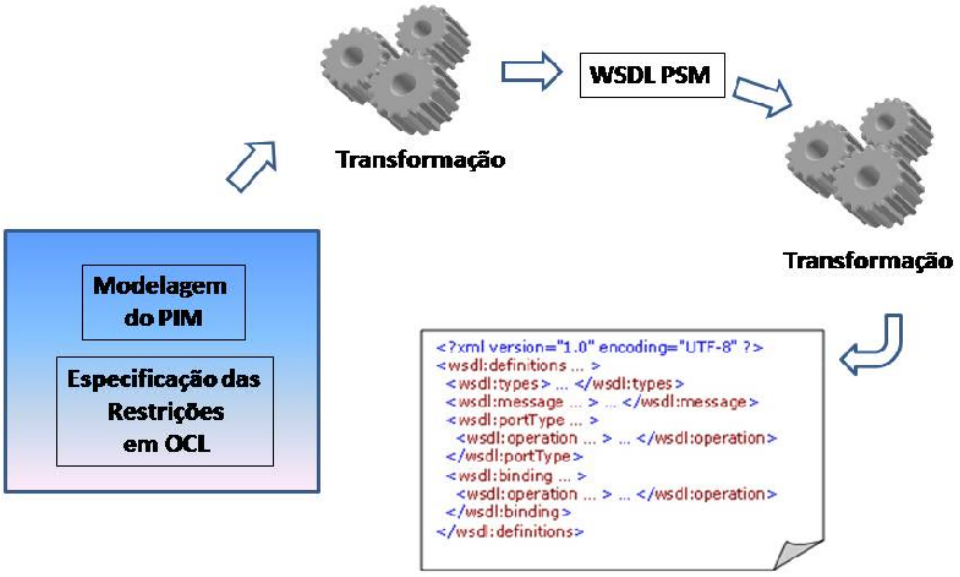

Figura 3.3: Tarefas da abordagem proposta por Sharma e Sood (2011)

3.3 Abordagem de Brunelièle, Cabot e Jouault (2010)

Brunelièle, Cabot e Jouault (2010) propuseram o conceito de **Modelagem como serviço - MaaS**. Similar ao SaaS, MaaS permitiria a implantação e execução sob demanda de modelagem e serviços orientados a modelos, sobre a infraestrutura da Internet. A ideia é criar um repositório de modelos e ferramentas de desenvolvimento orientado a modelos e que seja oferecido à equipe de desenvolvimento como um serviço. Entre as diversas aplicações desse conceito, destacam-se:

- Criação de uma ferramenta de modelagem colaborativa e distribuída, ou seja, um instrumento para permitir a especificação e compartilhamento de modelos de software entre os membros da equipe em tempo real;
- Definição de mashups [3] de modelagem como uma combinação de serviços de MDE de diferentes fornecedores;
- Ferramentas de transformação de modelos na nuvem para facilitar o gerenciamento de PIM;
- Geração de código na nuvem. A transferência dos serviços de geração de código e simulação para nuvem facilitará a implantação e evolução de aplicações de software. Desenvolvedores não precisariam se preocupar com configuração da infraestrutura para compilar e implantar os aplicativos e confiariam na nuvem para isso;
- A troca de dados e metadados[4] entre as ferramentas de MDE é um dos grandes desafios da atualidade. Até o momento, o problema tem sido resolvido através da definição de pontes entre as ferramentas. MaaS ofereceria uma solução mais transparente e global para esse problema. Por exemplo, as pontes poderiam ser definidas como serviços executados automaticamente na demanda por outros serviços, quando problemas de incompatibilidade são detectados; e
- Gerenciamento global e distribuído dos modelos. Projetos complexos na MDE abrangem vários modelos, transformações, entre outros artefatos. MaaS poderia facilitar a manipulação de todos esse artefatos em um ambiente distribuído.

[3] Mashup: é um site personalizado ou uma aplicação web que usa conteúdo de mais de uma fonte para criar um novo serviço completo (YU et al., 2008)

[4] Metadados: são dados sobre outros dados.

3.4 Outras abordagens

Se cada provedor de nuvem aplicasse padrões bem estabelecidos em suas plataformas, teoricamente uma aplicação poderia ser facilmente portada. Na literatura podem ser encontradas algumas iniciativas em torno da padronização. Armbrust et al. (ARMBRUST et al., 2009) sugere que a padronização como uma maneira de resolver o aprisionamento ao provedor. Entre as iniciativas que visam a padronização está o **DMTF (Distributed Management Task Force)**. O DMTF trabalha com a padronização da interação entre ambientes de nuvem. A iniciativa desenvolveu a especificação CIMI - Cloud Infrastructure Management Interface [5]. A especificação padroniza a interação entre ambientes de nuvem para alcançar uma infraestrutura de nuvem interoperável. Possibilita assim que os usuários administrem sua infraestrutura e a utilizem sem complexidade.

Alguns sistemas, comumente chamados de sistemas legados, apesar de serem bastante antigos, fornecem serviços essenciais. Pelo grau de criticidade e custo para modernização, muitos continuam ativos nas empresas. E muitos deles não estão preparados para serem migrados para nuvem, muitas vezes por não serem sistemas Web ou fazer uso de tecnologias não suportada por plataformas de nuvem. As plataformas não suportam todos os sistemas, normalmente, o provedor especifica qual conjunto de padrões e/ou linguagens que suportam. Isso acontecia também como os provedores do modelo application server provider(ASP), alguns deles hospedavam somente aplicações em Java, outros em PHP, etc. Tendo como foco essa problemática, o projeto REMICS[6] propõe uma abordagem MDE para migração de sistemas legados para nuvem (MOHAGHEGHI et al., 2010; MOHAGHEGHI; ANDTHER, 2011). Formado por um consórcio entre diversas intituições de pesquisa, consultoria e usuários da nuvem, o REMICS é um projeto robusto e tem previsão de funcionamento até o segundo semestre de 2013 [7]. Seu principal objetivo é especificar, desenvolver e avaliar uma ferramenta MDE para migração de serviços. O processo de migração proposto consiste no entendimento dos sistemas legados em termos de sua arquitetura e funcionalidade, e no projeto de uma nova aplicação SOA que provê a mesma ou melhor funcionalidade.

Na literatura é também possível encontrar algumas outras abordagens para tratar algum conceito ou problema relacionado à Computação em Nuvem. Peidro e Escoí (2011) ressaltam a importância de uma linguagem de modelagem abrangente e escalável e indica novas linhas de pesquisa para a concepção de plataformas de nuvem auto-gerenciáveis. Os autores destacam

[5] `http://www.dmtf.org/standards/cloud`

[6] REMICS Website: http://www.remics.eu/

[7] A lista completa das publicações relacionadas ao projeto REMICS pode ser acessada através do site: `http://www.remics.eu/publications`

que o ideal seria se o desenvolvedor captasse os aspectos da aplicação a partir de modelos e a plataforma fosse responsável pelas demais tarefas necessárias para que o sistema funcione. Ou seja, ela se encarregaria de tratar da distribuição, implantação, execução, monitoramento, escalabilidade, elasticidade, etc. Mais questões de pesquisa e abordagens relacionadas ao desenvolvimento de sistemas para o modelo de nuvem podem ser encontradas em Armbrust et al. (2009) e Silva e Lucrédio (2012).

3.5 Considerações finais

Na tabela 3.1 é apresentada uma síntese dos trabalhos correlatos apresentados neste capítulo, bem como as características da abordagem proposta. A Computação em Nuvem ainda está evoluindo, e oportunidades de pesquisas relacionadas tem sido exploradas (SILVA; LUCRéDIO, 2012). O conceito de nuvem foi criado de uma iniciativa mercadológica e somente há pouco tempo tem sido pesquisado com maior atenção pela comunidade científica. Por esse motivo ainda há relativamente poucas publicações científicas relacionadas. Todavia, grandes instituições de pesquisa no Brasil e no mundo estão começando a explorar essa tecnologia e o número de trabalhos relacionados à nuvem é crescente(SILVA; LUCRéDIO, 2012).

Tabela 3.1: Provedores de Computação em Nuvem e seus respectivos modelos de serviço

Abordagem	***Iniciativa***	***Características***
- Sharma e Sood	- MDE	- Geração de WSDL - Uso de SOA - Interoperabilidade - Modelagem indepentende de plataforma
- Brunelièle, Cabot e Jouault	- MDE	- Conceito de MaaS - Modelagem Colaborativa - Ferramentas de transformação na nuvem - Geração de código na nuvem - Troca de dados e metadados entre ferramentas MDE - Gerenciamento global e distribuido de modelos
- DMTF	- Padronização	- Interação entre ambientes - Especificação CIMI - Administração de Infraestrutura
- Remics	- MDE	- Migração de Sistemas Legados - MDE para migração de serviços - Projeto de uma aplicação SOA para prover a mesma ou melhor funcionalidade
- Abordagem proposta	- MDE	- Desenvolvimento de novos sistemas - Modelagem independente de plataforma - Portabilidade através da geração a partir do mesmo modelo - Baseada em linguagem específica de domínio - Foco no modelo de serviço PaaS - Geração de sistemas específicos de plataforma - Reuso da modelagem - Automatização de tarefas repetitivas - Aplicação gerada de acordo com uma implementação de referência

Capítulo 4

UMA ABORDAGEM DIRIGIDA POR MODELOS PARA DESENVOLVIMENTO DE APLICAÇÕES PORTÁVEIS ENTRE PLATAFORMAS DE COMPUTAÇÃO EM NUVEM

Este capítulo apresenta uma Abordagem Dirida por Modelos para Desenvolvimento de Aplicações Portáveis Entre Plataformas de Computação em Nuvem. A organização do capítulo está conforme segue: na seção 4.1 são apresentadas algumas considerações iniciais, na seção 4.2 são apresentados o projeto e desenvolvimento do modelo independente de plataforma. Na seção 4.3 é apresentado o desenvolvimento das tranformações para cada provedor.

4.1 Considerações iniciais

Conforme apontado no capítulo introdutório desta dissertação, 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 para que o modelo possa se estabelecer de fato. Dentre tais problemas, está o ***Lock-In***, que é definido como o aprisionamento do usuário a plataforma de nuvem devido a falta de portabilidade entre os provedores.

Para solucionar o problema, foi utilizada a abordagem dirigida por modelos apresentada neste capítulo. O domínio selecionado como objeto de estudo dessa abordagem foi o domínio de CRUD, que representa operações típicas de acesso a bases de dados (*Create, Retrieve, Update, Delete*), e que constitui parte importante da maioria das aplicações. A partir do domínio 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 (*Domain Specific Language* ou DSL) que descreve os conceitos do domínio em alto nível de abstração. Essa DSL constitui a base para a construção de modelos independentes de plataforma de nuvem. Logo após essa etapa, a partir do conhecimento adquirido em estudos de caso, foram definidas transformações que são utilizadas para gerar código especifico de plataforma.

Considerou-se que era necessário e suficiente a contrução de geradores para pelo menos duas plataformas de nuvem. Portanto, foram escolhidas as plataformas Google App Engine (GAE) Java e Windows Azure (Azure) Java[1]. A Seção 2.3 apresentou uma visão geral dessas plataformas. As seções a seguir apresentam um aprofundamento sobre suas características, bem como os conceitos que foram utilizados para o desenvolvimento da DSL.

Para aprofundamento e conhecimento das características das plafatormas, foram desenvolvidos estudos de caso que levaram à construção de implementações de referência. Essas implementações são aplicações base a partir das quais foram projetadas e construídas as transformações que levam em conta suas características. Em termos de questões arquiteturais das implementações, foi utilizado o modelo Model-View-Controller (MVC). O MVC possibilita a implementação de sistemas com clareza e objetividade, permitindo que a separação em camadas permita a modificação e extensão dos sistemas para atender novas exigência sem que uma camada interfira na outra. Portanto, o conhecimento embutido nas tranformações desenvolvidas, além dos detalhes de plataforma, levam em consideração o modelo citado.

4.2 Projeto e Desenvolvimento do Modelo Independente de Plataforma (PIM)

Uma linguagem específica de domínio (*Domain Specific Language* - DSL) é uma linguagem pequena, normalmente declarativa, focada em um domínio de problema em particular (DEURSEN et al., 2000). DSLs existem há um longo tempo. Mernik, Heering e Sloane (2005) citam os exemplos da linguagem APT para controle numérico, que data de 1957-58, e da mais famosa BNF, ou *Backus-Naur Form*, linguagem para especificação de gramáticas criada em 1959. Desde então, diversas linguagens vêm sendo desenvolvidas e utilizadas, e a literatura nesta área é bastante rica (DEURSEN et al., 2000; MERNIK; HEERING; SLOANE, 2005) . Uma linguagem específica de domínio pode ser textual (permitindo especificar programas) ou visual (permitindo

[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.

especificar modelos ou diagramas).

A definição da linguagem normalmente requer um metamodelo que seja capaz de obter 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 visual ou representação textual), a existência de um metamodelo como esquema conceitual, em geral, indica que uma maior quantidade de instâncias pode ser expressa.

Como parte do desenvolvimento da abordagem proposta neste trabalho, uma DSL para o domínio de CRUD foi implementada. Esse domínio foi selecionado pelo fato de englobar diversos conceitos necessários ao desenvolvimento de aplicações para nuvem. No entanto, é possível estender a abordagem para tratar outros domínios. Operações CRUD nas plataformas estudadas são realizadas seguindo um modelo orientado a objetos de entidades persistentes. Portanto, a DSL obtida tem esse foco e é bastante simples. Para desenvolvê-la foi utilizada a ferramenta Xtext. Portanto, sua representação é textual. Tal representação foi utilizada devido à sua simplicidade, facilidade de aprendizado e flexibilidade das ferramentas utilizadas em seu desenvolvimento. Todavia, o metamodelo projetado pode ser igualmente representado de forma gráfica. A Figura 4.1 apresenta o metamodelo da DSL criada para representar o modelos independentes de plataforma.

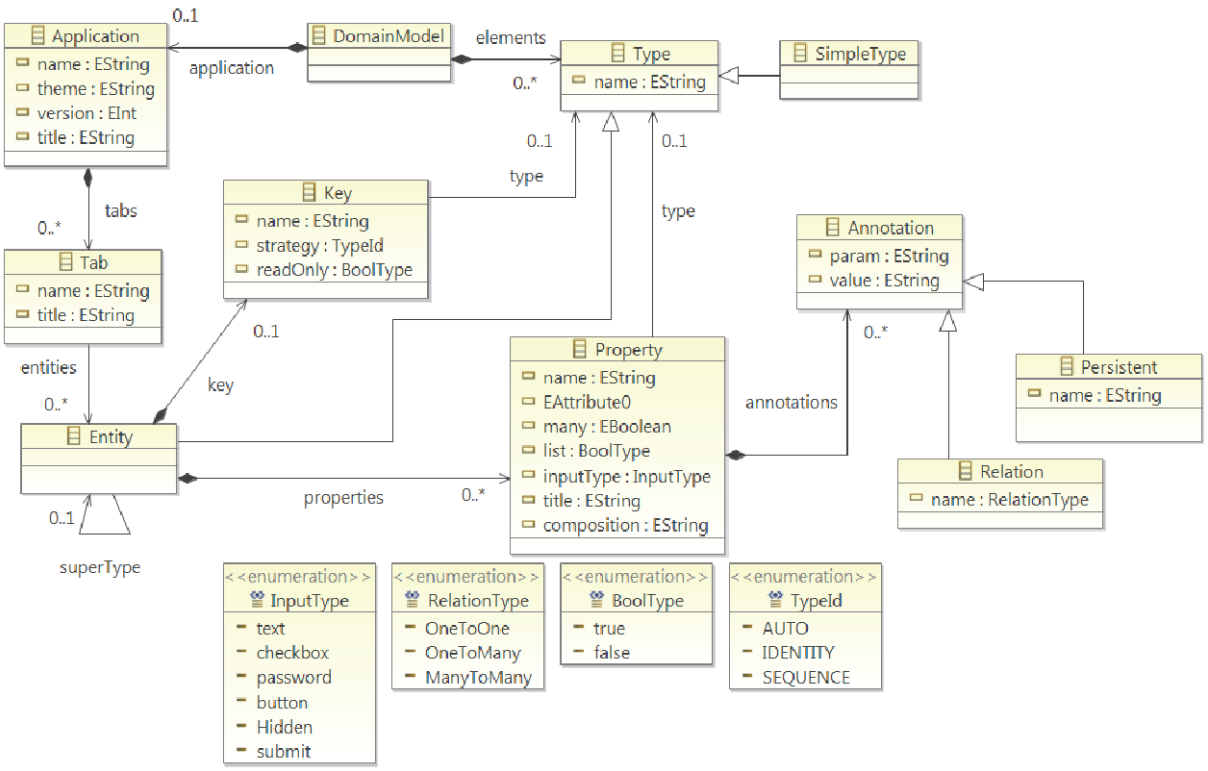

Figura 4.1: Metamodelo da DSL desenvolvida

No metamodelo projetado, uma aplicação é composta por módulos. Cada módulo (*Module*), além de atributos como nome e título, agrupa zero ou mais entidades. Uma entidade (*Entity*) representa uma classe persistente, e pode possuir propriedades (*Property*), que representam os seus atributos persistentes. Nesse contexto, os módulos apenas são utilizados para melhor organizar a camada de apresentação da aplicação, mas pode ser útil também para organizar o código gerado em módulos de software para efeito de reutilização.

A partir da gramática Xtext relacionada ao metamodelo apresentado na figura 4.1 foi gerada a linguagem que permite a especificação de modelos de domínio. Esses modelos são utilizados como entrada para o conjunto de transformações que geram o código específico de plataforma para cada provedor.

4.3 Projeto e Desenvolvimento de transformações para plataformas GAE e Azure

Com base nas análises realizadas, concluiu-se que em se tratando de processo de desenvolvimento, criar aplicações fazendo uso de PaaS é similar ao desenvolvimento tradicional. Mas, em termos de tecnologia, devem ser levados em consideração as tecnologias que são suportadas, bem como as restrições impostas pelas plataformas.

No caso do domínio estudado, de operações do tipo CRUD, e considerando que as aplicações geradas estarão de acordo com a implementação de referência e utilizam o modelo arquitetural MVC, três pontos precisaram de atenção especial: persistência em banco de dados, classes controladoras e interface. Na tabela 4.1 são apresentados todos artefatos desenvolvidos na abordagem. Na seção seguinte são apresentados maiores detalhes sobre o desenvolvimento desses elementos.

4.3.1 Persistência em banco de dados

A plataforma ***Google App Engine***[2] prevê algumas opções para armazenamento de dados:

- **App Engine Datastore**: provê um armazenamento NoSQL com um mecanismo de consulta e transações atômicas.

[2]Todas as informações relacionadas a plataforma estão de acordo com a documentação que pode ser acessada através do site: `https://developers.google.com/appengine/`

Tabela 4.1: Transformações desenvolvidas

Transformação	***Descrição***
Cadastro Azure	- Transformação para geração de telas de cadastro para plataforma Azure;
Cadastro GAE	- Transformação para geração de telas de cadastro para plataforma GAE;
Controlador Azure	- Transformação para geração da camada de controle para plataforma Azure;
Controlador GAE	- Transformação para geração da camada de controle para plataforma GAE;
Listagem Azure	- Transformação para geração de listagens utilizando plataforma Azure;
Listagem GAE	- Transformação para geração de listagens utilizando plataforma GAE;
Modelo Azure	- Transformação para geração de entidades na camada de modelo levando em consideração as tecnologias específicas da camada de modelo Azure;
Modelo GAE JPA	- Transformação para geração de entidades na camada de modelo levando em consideração o padrão JPA para persistência de entidades;
Modelo GAE JDO	- Transformação para geração de entidades na camada de modelo levando em consideração o padrão JDO para persistência de entidades;
Configuração GAE	- Transformação para geração do arquivo de configuração appengine-web.xml;
Linguagem específica de domínio e editor para criação de modelos idendependetens de plataforma	- Linguagem específica de domínio e plugin para ferramenta eclipse para desenvolvimento dos modelos independentes de plataforma;

- **Google Cloud SQL**[3]: provê um banco de dados relacional baseado em MySQL[4].
- **Google Cloud Storage**: oferece um serviço de armazenamento de objetos e arquivos de terabytes de tamanho. No momento da escrita deste trabalho, esse serviço ainda está em fase experimental e é um nova funcionalidade que está sendo oferecida pela plataforma.

Para desenvolvimento deste estudo, a modalidade de armazenamento de dados escolhida foi o *App Engine Datastore*. O Datastore é nativo ao App Engine e até que seja atigindo os limites dos níveis gratuitos não é necessário efetuar contrato para pagamento. Além disso, o armazenamento NoSQL é mais barato e possui características de escalabildade gerenciada pelo próprio sistema.

O armazenamento de dados na plaforma GAE prevê o uso dos padrões Java Data Objects [5](JDO) e Java Persistence API[6] (JPA) para especificar as entidades que serão armazenadas no banco. Essas interfaces são implementadas usando a Plataforma de acesso *DataNucleus*[7], a implementação de código aberto desses padrões. O desenvolvedor precisa aprender a sintaxe e as restrições impostas pela plataforma. O GAE segue um modelo orientado a objetos para as entidades persistentes.

A ***Windows Azure*** suporta 4 (quatro) principais opções para armazenamento de dados:

- **SQL Database**: esta opção se encaixa no modelo de serviço de PaaS, o que significa que o Azure gerencia a solução e o usuário não precisa se preocupar com questões relacionadas a configuração de software e/ou hardware;
- **SQL Server em máquina virtual**: trata-se do SGBD SQL Server executando em uma máquina virtual criada utilizando a máquina virtual do Windows Azure. Esta abordagem permite ao usuário utilizar todos os recursos do SQL Server[8], incluindo aspectos que não são providos pelo SQL Database. No entanto, requer que o usuário se ocupe das tarefas de gerenciamento do servidor de banco e da máquina virtual;
- **Blob Storage**: armazena coleções de bytes não-estruturados, tais como áudio e vídeo; e
- **Table Storage**: armazenamento chave-valor NoSQL. O Table Storage não suporta relacionamento entre objetos e impõe muitas restrições na especificação da entidade a ser

[3]Os planos de pagamento podem ser acessados através da página: `https://developers.google.com/cloud-sql/docs/billing`

[4]MySQL: http://www.mysql.com/

[5]JDO:`http://www.oracle.com/technetwork/java/index-jsp-135919.html`

[6]JPA:`http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html`

[7]`http://www.datanucleus.org/`

[8]`http://www.microsoft.com/en-us/sqlserver/default.aspx`

persitida.

A Figura 4.2 apresenta as diferenças que mais influenciam na tomada de decisão sobre qual tipo de armazenamento adotar[9].

	Tipo de Armazenamento	Mecanismo de Acesso	Suporte a Transações	Tamanho Máximo	Preço(US Dollars)
SQL Database	Tabelas Relacionais	Protocolo: TDS Interfaces: ADO.NET, Entity Framework, etc.	Em todas as tabelas do Banco	150 GB por Banco, possibilidade de aumentar com federação	$ 9.99 a $ 0.999/GB por Mês
SQL Server na VM	Tabelas Relacionais	Protocolo: TDS Interfaces: ADO.NET, Entity Framework, etc.	Em todas as tabelas do Banco	Vários TBs por banco, dependento do tamanho da VM	$ 2.02 a $ 0.125 por hora, mais blob storage para discos de dados
Blob Storage	Objetos Binarios	Protocolo: REST Interfaces: Windows Azure Storage Client, Windows File I/O.	Nenhum	200 GB por bloco blob, 1TB por página blob.	$ 0.125 a $ 0.037/GB por mês, $0.01/100.000 operações
Table Storage	Armazenamento Chave/Valor	Protocolo: Odata Interfaces: Windows Azure Storage Client	Em entidadedes na mesma partição em uma Tabela	100 TB por Table	$ 0.125 a $ 0.037/GB por mês, $0.01/100.000 operações

Figura 4.2: Caracterísiticas do armazenamento de dados Azure

Para desenvolvimento deste estudo, a modalidade de armazenamento escolhida foi a Table Storage. Conforme apontado anteriomente, o Table Storage é a opção de armazenamento NoSQL da plataforma Windows Azure. O armazenamento no Table Storage é muito mais barato que as demais ofertas. Além disso, as questões de escalabilidade e gerenciamento da infraestrutura são transparentes aos usuários.

Diferentemente do GAE, que utiliza os padrões JDO e JPA para mapeamento das classes persistentes, o Azure Table Storage não suporta relacionamentos entre entidades e utiliza uma Api REST[10] para comunicação entre camada de controle e armazenamento. Além disso, a

[9]Para maiores detalhes relacionados as ofertas de armazenamento na plataforma Windows Azure visite `http://www.windowsazure.com/en-us/pricing/details/`

[10]http://www.w3.org/2001/sw/wiki/REST

plataforma impõe limitações nas propriedades das classes de entidade. Uma entidade pode ter até 255 propriedades, incluindo 3 propriedades específicas. Portanto, o desenvolvedor pode incluir até 252 propriedades. Todos os dados em em uma entidade não podem exceder 1 MB.

O Table Service suporta apenas um conjunto de 8 (oito) tipos de dados definidos pela especificação WCF Data Services[11]. Os tipos são[12]: byte[], bool, DateTime, double, Guid, Int32 ou int, Int64 ou long e String.

O GAE e a DSL projetada suportam relacionamentos entre classes. Além disso, o GAE faz o gerenciamento automático na recuperação de entidades. Na plataforma Azure, foi necessário criar uma camada que trata do relacionamento entre as classes. Ou seja, o código gerado pelas transformações a partir da DSL trata relacionamento entre entidades mesmo a Azure não suportando diretamente esse tipo de operação.

As restrições e características específicas de plataforma anteriormente descritas foram analisadas e a partir delas foram criadas transformações que encapsulam esse conhecimento e geram os elementos necessários para a aplicação final. Conforme apontado anteriormente, o mapeamento entre classes e banco de dados na plataforma GAE pode ser feito utilizando as APIs JPA e JDO. Especificar uma entidade em algum desses padrões requer conhecimento do conjunto de anotações[13] adotadas e das restrições impostas pela plataforma. Portanto, com a finalidade de automatizar a geração das entidades, e evitar que o desenvolvedor gaste tempo e se preocupe com sintaxe das anotações das classes, foram desenvolvidas transformações que encapsulam esse conhecimento. Tais transformações permitem que a partir do modelo de alto nível sejam geradas entidades de acordo com o padrão desejado.

Para um melhor entendimento é possível citar a seguinte situação:

"Um Website necessita de funcionalidades relacionadas ao cadastro de produtos. Será necessário listar, cadastrar, atualizar e remover produtos (operações CRUD). Identificou-se a necessidade de persistir o seguintes atributos: identificador (id), nome (nome), descrição (descricao) e preço (preco)"

Para solucionar a situação, seria necessário criar uma entidade produto, camada de controle e artefatos na camada de visualização. Utilizar a plataforma GAE com o padrão JDO resultaria na classe produto anotada conforme consta na Listagem 4.1.

[11] WCF: `http://msdn.microsoft.com/en-us/library/cc668772.aspx`

[12] A tabela completa pode ser acessada através do endereço: http://msdn.microsoft.com/en-us/library/windowsazure/dd179338.aspx

[13] Uma anotação é um metadado sobre a classe que permite afetar a semântica dos programas em execução. Normalmente é escrita na forma: @nomedaanotacao[(...)].

Listagem 4.1: Classe Produto anotada com o padrão JDO

```
1
2  package com.gaeapplication.model;
3
4  import java.util.*;
5  import com.google.appengine.api.datastore.Key;
6  //annotations import
7  import javax.jdo.annotations.IdGeneratorStrategy;
8  import javax.jdo.annotations.PersistenceCapable;
9  import javax.jdo.annotations.Persistent;
10 import javax.jdo.annotations.PrimaryKey;
11
12 /* detachable="true" means that the
13 object can be managed after the persistence manager was closed*/
14
15 @PersistenceCapable(detachable="true")
16 public class Produto  {
17         @PrimaryKey
18         @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
19         private Key id;
20         private String nome;
21         private String descricao;
22         private Double preco;
23
24         ... Construtores ...
25
26         ... Gets e Sets...
27 }// fim da classe
```

O código apresentado na listagem refere-se a uma classe de entidade denominada Produto anotada com o padrão JDO do GAE. As linhas 15, 17 e 18 referem-se anotações utilizadas pela API JDO. As linhas de 7 a 10 mostram o impo import de elementos da JDO.

Para desenvolver as transformações que geram as classes anotadas no padrão JDO, foi necessário um estudo aprofundado sobre toda a API JDO suportada pela plataforma. Isto é, toda a sintaxe, as restrições e questões específicas da JDO do GAE. Além disso, foi necessário encapsular esse conhecimento em termos de metamodelo. Isso significa que as informações do modelo especificado utilizando-se a DSL desenvolvida preenchem informações necessárias às transformações.

De igual modo, para desenvolver as transformações necessárias à camada de persistência de dados da Windows Azure, foi necessário estudar o funcionamento da API e expressar esse conhecimento em termos de metamodelo. Na Listagem 4.2 é apresentado o código da entidade produto conforme os padrões estabelecidos pela plataforma.

Listagem 4.2: Classe Produto conforme o Windows Azure

```
1
2  package com.azureapplication.model;
3
4  import java.util.*;
5
6  import com.microsoft.windowsazure.services.table.client.TableServiceEntity;
7
8  //Generated Class
9
10 public class Produto extends TableServiceEntity {
11
12         // atributes
13         private String nome;
14         private String descricao;
15         private Double preco;
16         // constructors
17
18         public Produto() {
19                 this.partitionKey = "Produto";
20         }
21         ...
22
23         // ... gets e sets ...
24
25         }
```

Como pode ser visto na listagem, a classe produto não está anotada com nenhum dos padrões citados anteriormente(JDO ou JPA). Trata-se apenas de uma classe simples que herda da classe TableServiceEntity(linha 10). As demais linhas(de 13 a 23) apresentam os atributos persistentes, construtores, métods get e set. A TableServiceEntity é uma classe referente ao serviço de armazenamento do Table Service específico da plataforma Azure. Todas entidades que serão armazenadas no serviço precisam herdar dessa classe.

Analisando as listagens 4.1 e 4.2, pode-se perceber a diferença entre o código específico de cada plataforma. Apesar do requisito ser o mesmo e da implementação ter sido feita utilizando

a mesma linguagem (no caso a linguagem Java), o código gerado para cada plataforma é diferente. Por exemplo, a classe produto da plataforma Azure deve herdar da TableServiceEntity enquanto a classe Produto na plataforma GAE deve ser anotada com os padrões JDO ou JPA. Portanto, para possibilitar a portabilidade neste contexto, foi necessário desenvolver transformações para geração da camada de modelo conforme o padrão JPA e JDO na plataforma GAE e transformação conforme o modelo Table Storage da plataforma Windows Azure. A utilização de um dos modelos JPA ou JDO na plataforma GAE é alternativa, ou seja, o desenvolvedor pode escolher qual o padrão deseja usar. Para isso basta selecionar a partir de qual transformação deseja gerar o código. Na Listagem 4.3 é apresentado um trecho do código Xtend para transformação da JDO.

Listagem 4.3: Código da Transformação JDO do Google App Engine

```
1   ...
2   def compile(Entity entity)'''
3       package com.gaeapplication.model;
4       // dynamic imports - importa todas as classes necessárias
5       «FOR property:entity.properties»
6           «compileImports(property.type)»
7       «ENDFOR»
8       //faz a geração da entidade
9       @PersistenceCapable(detachable="true")
10      public class «entity.name»
11      «IF entity.superType != null»
12          extends «entity.superType.name»
13      «ENDIF» {
14          //gera a chave primária
15          @PrimaryKey
16          «compileKey(entity.key)»
17          «FOR property:entity.properties»
18              «IF property.many==true»
19              //chamada a um método que cria as propriedades
20              «compileProperties(property)»
21          «ELSE»
22              //chamada a um método que cria as propriedades
23              «compileProperty(property)»
24              «ENDIF»
25          «ENDFOR»
26          //gerando gets e sets
27          «compileKeyGetsAndSets(entity.key)»
28          «FOR property:entity.properties»
```

```
29                          «IF property.many==true»
30                          «compileGetList(property)»
31                                  «compileSetList(property)»
32                  «ELSE»
33                                  «compileGets(property)»
34                                  «compileSets(property)»
35                          «ENDIF»
36                  «ENDFOR»
37          }
38          ```
```

Ao analisar o código da transformação é possível perceber código específico da JDO embutido(linha 15) e o uso dos elementos de domínio para preencher informações no código (linhas 10, 12, 16, 18, 27, 28 e 29). Com a transformação, detalhes podem ser abstraídos do desenvolvedor e evitar o esforço repetitivo no mapeamento utilizando a JDO. O mesmo acontece com o padrão JPA. Na Listagem 4.4 é apresentado um trecho do código da transformação JPA.

Listagem 4.4: Código da Transformação JPA do Google App Engine

```
1
2   package gae.transformations.jpamodeltransformation
3           ...
4       def compile(Entity entity)```
5                   package com.gaeapplication.model;
6                   import javax.persistence.*;
7                   import com.google.appengine.api.datastore.Key;
8                   import java.util.*;
9                   //anotação JPA
10                  @Entity
11                  public class «entity.name»
12                  «IF entity.superType != null» extends
13                  «entity.superType.name»«ENDIF» {
14                          @Id
15                          «compileKey(entity.key)»
16                          «FOR property:entity.properties»
17                                  «compileProperty(property)»
18                          «ENDFOR»
19                          «compileKeyGetsAndSets(entity.key)»
20                          «FOR property:entity.properties»
21                                  «compileGets(property)»
22                                  «compileSets(property)»
23                          «ENDFOR»
24          }
```

```
25      ‘‘‘
26      ...
27  }
```

Ao analisar o código da tranformação é possível perceber código JPA embutido entre o codigo xTend(linhas 6, 10 e 14). A linha 15 faz uma chamada para um método que vai gerar o identificador, a linha 17 faz uma chamada para um métodos que vai gerar os atributos da classe, a linha 19 faz uma chamada para um método que irá gerar os métodos get e set do identificadosr e as linhas 21 e 22 faz chamada para um método que vai gerar os métodos gets e sets(respectivamente) dos atributos.

É importante ressaltar que, muitas vezes, o código da transformação pode ser mais difícil de desenvolver e ficar mais complexo que o código final da aplicação (LUCRéDIO; ALMEIDA; FORTES, 2012). Uma discussão mais aprofundada com relação a esse assunto é apresentada na Seção 5.2.

4.3.2 Camada de controle

As plataformas selecionadas trabalham segundo o conceito de requisição/resposta. Para facilitar a criação da camada de controle, manter o código mais enxuto e tornar as aplicações Restful[14], foi utilizado o framework VRaptor. O VRaptor provê um conjunto de ferramentas que simplificam o desenvolvimento de tais aplicações, ajudam a aumentar a legibilidade, reduzir a complexidade e o número de linhas de código da aplicação. Além disso, eliminam a necessidade da configuração do descritor de implantação (Web.xml) para cada controlador criado. O framework provê acesso aos controladores através de chamadas do tipo REST.

Para dar suporte ao VRaptor, foi necessária a criação de transformações capazes de capturar os elementos de código utilizados pelo framework, tais como anotações, nomenclatura de classes, classes controladoras, tags suportadas na camada de visualização, etc. Também foi necessário cuidado para respeitar as diferenças entre as plataformas. Outros frameworks e/ou ferramentas suportadas pelas plataformas poderiam ter sido utilizadas no desenvolvimento. Para isso, seria necessário utilizá-los na implementação de referência e em seguida abstrair o código para o nível de metamodelo levando em consideração as características da aplicação desenvolvida.

Na camada de controle foram gerados recursos REST conforme semântica do framework

[14]Web Services RESTful utilizam a arquitetura REST (REpresentational State Transfer). Aplicações Restful focam nos recursos do serviço e são transferidos por HTTP, além de usarem os métodos do protocolo HTTP (GET, PUT, POST, DELETE) para fazer a comunicação entre Cliente e Provedor (FIELDING; TAYLOR, 2002).

utilizado(no caso o VRaptor). Esses recursos permitem a comunicação entre as camadas de visualização e controle de modo a fazer as operações de CRUD necessárias à aplicação. Na listagem 4.5 é apresentado um trecho de código da camada de controle gerada para o cadastro de produto citado na seção anterior.

Listagem 4.5: Código da camada de controle para operações CRUD do cadastro de produtos

```
1  ...imports...
2
3  @Resource
4  @Path("/produto")
5  public class ProdutoController{
6
7          private Result result;
8
9          public ProdutoController(Result result) {
10                 this.result = result;
11         }
12
13         public void lista(){
14                 ...código para recurerar lista de produtos...
15         }
16
17         public void excluir(int id){
18                 ... código para excluir obj...
19         }
20
21         public void novo() {
22                 ...código redirecionando para cadastro de produto...
23         }
24
25         public void alterar(int id){
26                         ...código para alterar produto...
27         }
28
29         public void cadastro(Produto obj) {
30                 ... código para cadastro de produto...
31         }
32         public void salvar(Produto obj , int id){
33                         ...código para salvar entidade...
34     }
35
36     private void validacao(Produto obj){
```

```
37
38          }
39      }
```

A linha 3 é uma anotação que "indica" ao framework que aquela classe será um recurso REST. A linha 4 nomeia o caminho daquele recurso. A camada de controle deve receber uma requisição e um conjunto de parâmetros da camada de visualização e executar sua função, que pode ser Cadastrar, Recuperar, Atualizar ou Excluir. Como pode ser observado no código do controlador gerado para operações CRUD da classe produto, são disponibilzados os métodos básicos para que a camada de visualização possa fazer as requisições. Todos os métodos públicos apresentados na listagem são operações disponíveis no recurso produto. O método excluir, por exemplo, é uma operação que exclui produtos do banco e recebe um parâmetro que no caso é o identificador de um produto a ser removido.

Os artefatos MDE desenvolvidos geram aplicações CRUD básicas que funcionam nas plataformas especificadas de acordo com a delimitação do escopo da pesquisa. Na Listagem 4.6 é apresentado um trecho do código da transformação que gera a camada de controle para a plataforma GAE.

Listagem 4.6: Código da transformação para camada de controle das aplicações da plataforma GAE

```
1
2   ... imports ...
3
4       def compile(Entity entity)'''
5           package com.gaeapplication.controllers;
6
7           //imports
8
9           @Resource
10          @Path("/«entity.name.toFirstLower»")
11          public class «entity.name.toFirstUpper»Controller{
12
13                  private Result result;
14
15                  public «entity.name.toFirstUpper»Controller(Result result){
16                  this.result = result;
17          }
18          ...
19
20          public void excluir
```

```
21                  («IF entity.key.type.name!="String"»
22                  int«ELSE»String«ENDIF» «entity.key.name»)
23                  {...}
24                  ...
25      }
```

As linhas 9 e 10 apresentam as anotações utilizadas para tornar a classe um recurso REST. A listagem refere-se basicamente é uma metaclasse que com as informações do modelo de domínio irá gerar classes controladoras. Os códigos Java são elementos estáticos, os demais códigos entre "« »" serão substituidos por informações que virão do modelo de domínio, tais como nome da classe, nome dos atributos, tipo dos parâmetros, etc. e servirão para gerar as classes finais.

Assim como para a GAE, foi necessário desenvolver transformações para a camada de controle para a plataforma Azure. As operações disponibilizadas são as mesmas, o que muda é o uso das bibliotecas utilizadas pela plataforma.

4.3.3 Interface

O GAE e Azure trabalham com tecnologias de páginas geradas no servidor (JSP). Assim, o suporte para modelagem e geração de código referente à interface deve seguir esse modelo. Para proporcionar uma melhor experiência do usuário em relação ao uso da aplicação, foi utilizada a biblioteca JQuery[15] no desenvolvimento da Interface gráfica. Além de tecnologias como Asynchronous JavaScript and XML[16](AJAX) e Cascading Style Sheets (CSS). Pontanto, as transformações, de modo semelhante às demais camadas, encapsulam o conhecimento relacionado a essas tecnologias e geram código fazendo uso das mesmas. Foram criadas transformações para gerar a camada de visualização. Na Figura 4.3 são apresentadas as interfaces geradas para o exemplo citado nas seções anteriores. Além de interfaces que permitem ao usuário listar e excluir produtos, foram geradas funcionalidades referentes a cadastros e atualizações.

As transformações desenvolvidas, assim como nas demais camadas, levam em consideração as tecnologias e um *layout* predefinidos nas implementações de referência. A Listagem 4.7 apresenta um pequeno trecho do código referente a um tela de listagem de dados. Nessa transformação,o código Xtend é misturado com o HTML e com os demais códigos referentes à camada de visualização para gerar a funcionalidade necessária aos requisitos da aplicação.

[15]JQuery: `http://jquery.com/`
[16]AJAX: `http://www.w3schools.com/ajax/default.ASP`

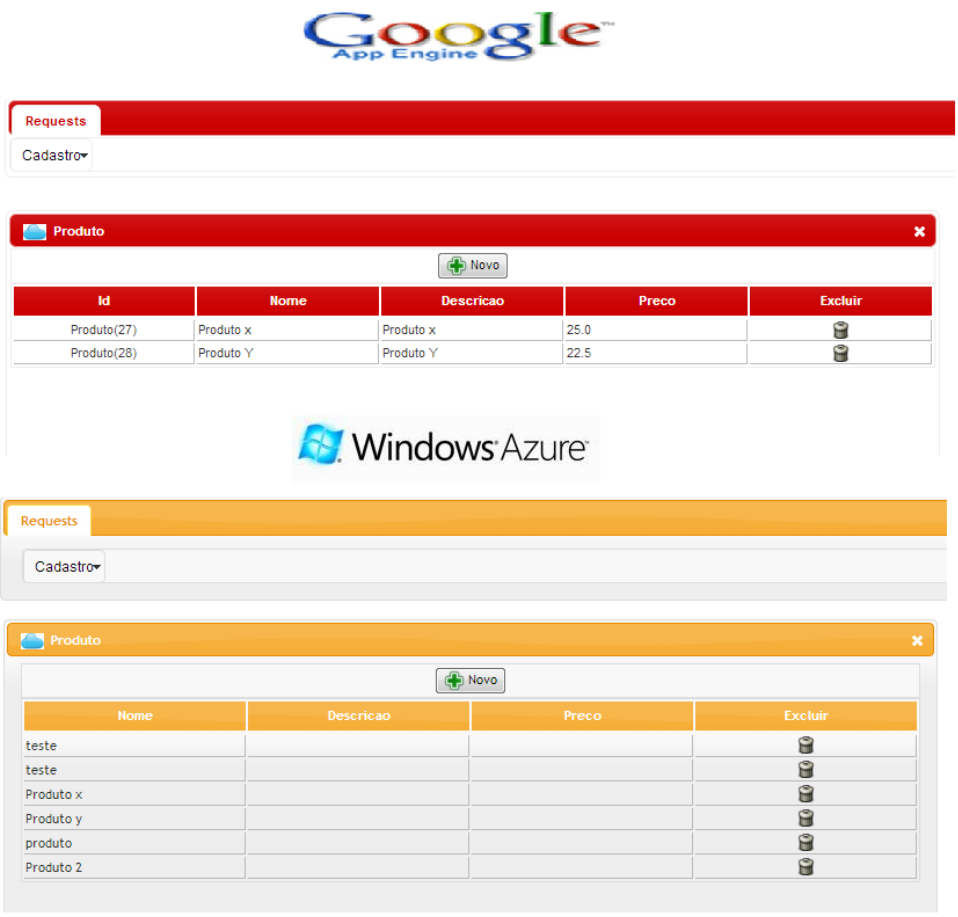

Figura 4.3: Interfaces geradas cadastro de produto

Listagem 4.7: Código da transformação para camada de controle das aplicações da plataforma GAE

```
1  def compile(Entity entity)'''
2          <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3          <!DOCTYPE html PUBLIC"-//W3C//DTD␣HTML␣4.01␣Transitional//EN"
4          "http://www.w3.org/TR/html4/loose.dtd">
5          <html>
6      <!-- Generated From Cloud Portability Approach -->
7
8      <head>
9          <meta http-equiv="Content-Type" content="text/html;␣charset=UTF-8">
10         <title>«entity.name»</title>
11         ...
12     '''
```

As linhas de 2 a 9 apresentam elementos fixos que serão gerados no arquivo .jsp referente a uma tela de cadastro de produtos. A linha 10 apresenta um atributo(«entity.name») que será substituido pela informação "name" que virá da modelagem.

4.4 Considerações Finais

Neste trabalho, diferentemente das demais abordagens encontradas na literatura, foi tratado o problema da portabilidade de aplicações entre os provedores GAE e Azure através de uma abordagem MDE. Mais especificamente, foi desenvolvida uma linguagem específica de domínio e um o conjunto de transformações que permitem a geração de aplicações Web de acordo com uma implementação base (implementação de referência) desenvolvida para cada plataforma. Em resumo, os seguintes passos foram seguidos:

1. Desenvolvimento de uma aplicação de exemplo (implementação de referência) para a plataforma GAE;

2. Desenvolvimento do metamodelo e DSL para o domínio CRUD, como foco na plataforma GAE;

3. Desenvolvimento de um conjunto de transformações para gerar código CRUD, camada de controle e interface para a plataforma GAE, reutilizando grande parte do código da implementação de referência desenvolvida no passo 1;

4. Desenvolvimento de uma aplicação de exemplo (implementação de referência) para a plataforma Azure, aproveitando o conhecimento obtido no passo 1;

5. Análise do metamodelo e DSL desenvolvidos no passo 2, para verificar a necessidade de refinamento para oferecer suporte também à plataforma Azure; e

6. Desenvolvimento de um conjunto de transformações para gerar código CRUD, camada de controle e interface para a plataforma Azure, reutilizando grande parte do código da implementação de referência desenvolvida no passo 4, e também aproveitando o conhecimento obtido no passo 3.

Como pode ser constatado, houve alguma iteratividade, onde os primeiros passos serviram como aprendizado e aquisição de conhecimento, que depois pode ser aproveitado, inclusive em forma de código, nos passos posteriores. Em particular, o desenvolvimento para a Azure foi consideravelmente mais fácil uma vez que o mesmo procedimento já havia sido feito para o GAE. Pelo fato do domínio selecionado como objeto de estudo da abordardagem ter sido o domínio de CRUD, não foi necessário um refinamento ou modificação da DSL para atender as especificidades da plataforma Azure. As diferenças entre as plataformas ficam encapsuladas nas transformações. A figura 4.4 apresenta em síntese todos os elementos desenvolvidos neste trabalho. O resultado desse esforço, ou seja, da criação de todos os elementos da abordagem é a portabilidade que pode ser alcançada pela geração de aplicações a partir do mesmo modelo de domínio. Para avaliar esse resultado, foram realizados alguns estudos, apresentados no próximo capítulo.

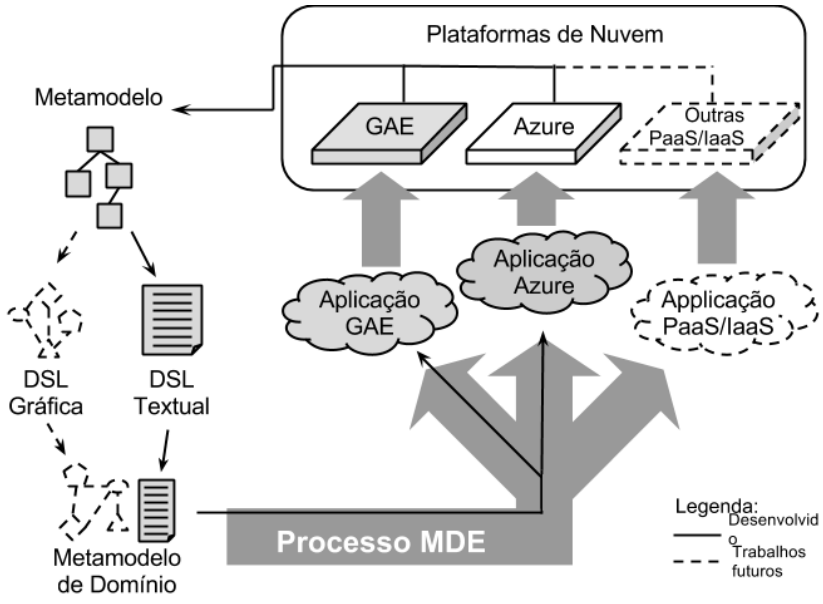

Figura 4.4: Abordagem dirigida por modelo para portabilidade

Capítulo 5

AVALIAÇÃO

Este capítulo apresenta a avaliação da abordagem MDE utilizada para resolução do problema da dificuldade de portabilidade de aplicações na computação em nuvem. A organização do capítulo está conforme segue: a seção 5.2 apresenta um estudo de caso utilizando a abordagem proposta, a seção 5.3 apresenta uma avaliação utilizando uma abordagem experimental que visou analisar a portabilidade sob o ponto de vista dos usuários finais.

5.1 Considerações Iniciais

Novas técnicas e produtos de software não deveriam ser simplesmente propostos sem passar por um processo de avaliação (JURISTO; MORENO, 2010). A Engenharia de Software alcançou um estágio em que é necessário fornecer evidências de que as técnicas e procedimentos de desenvolvimento são válidos. A avaliação desses elementos é de fundamental importância para que o desenvolvimento de software tenha por base elementos concretos e dados confiáveis (TICHY, 1998), não apenas suposições e crenças que, a longo prazo, podem se mostrar equivocadas e consequentemente gerar um grande custo ou levar um projeto ao fracasso (JURISTO; MORENO, 2010).

Um experimento pode somente mostrar existência de uma falha em uma teoria, mas não sua ausência. Isso significa que experimentos não devem ser vistos como provas, mas como evidências. Seus resultados são válidos para o conjunto o qual foi objeto de pesquisa (TRAVASSOS; GUROV; AMARAL, 2002). A idéia é utilizá-los para indução visando generalizar os resultados e derivar teorias a partir da observação(TICHY, 1998). Experimentos testam convicções contra realidade. A comunidade gradualmente aceita uma teoria se todos os fatos conhecidos fora do seu domínio pode ser deduzido dessa teoria e se há um numeroso conjunto de estudos experimentais que a evidencia(TICHY, 1998). Experimentos provêm um método sistemático e controlado para

a avaliação das técnicas e ferramentas de software (TRAVASSOS; GUROV; AMARAL, 2002). O tipo de experimento mais adequado vai depender dos objetivos do estudo, das propriedades do processo ou dos resultados finais esperados (WOHLIN et al., 2000). Dependendo das condições da investigação empírica, há 3 (três) tipos de estratégias que podem ser desenvolvidas: Survey, Estudo de Caso e Experimento controlado (WOHLIN et al., 2000; TRAVASSOS; GUROV; AMARAL, 2002).

A abordagem desenvolvida neste trabalho propõe o uso de modelos para facilitar a portabilidade de aplicações entre plataformas de Computação em Nuvem (mais especificamente para plataformas PaaS). Para atingir o objetivo de avaliar a portabilidade dos sistemas de maneira efetiva foi necessário desenvolver um sistema utilizando os elementos MDE da abordagem e portá-lo entre pelo menos duas plataformas. Isto é, gerar o mesmo sistema para ambas plataformas e verificar sua equivalência em relação ao conjunto de requisitos da aplicação. Além disso, o objetivo não é somente possibilitar a portabilidade, e sim torná-la menos custosa para o desenvolvedor. Sendo assim, o objetivo da avaliação foi duplo:

- O_1**.** Verificar se houve efetivamente a portabilidade; e
- O_2**.** Verificar se a abordagem desenvolvida, usando MDE, facilita a tarefa do desenvolvedor na portabilidade.

Buscando atender a esses objetivos, avaliação foi dividida em duas partes: estudo de caso e experimento controlado, descritos a seguir.

5.2 Estudo de Caso Preliminar

O estudo de caso é utilizado para monitorar projetos, atividades e atribuições (WOHLIN et al., 2000). É a técnica onde os fatores chave que podem afetar os resultados da 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. O Estudo de caso é um método utilizado em estudos empíricos em várias ciências. Na Engenharia de software pode ser usado para para determinar qual o melhor entre dois métodos (YIN, 2008).

O estudo de caso desenvolvido neste trabalho visou, de maneira exploratória, **(i)** comparar as duas aplicações geradas a partir do mesmo modelo de domínio com o objetivo de verificar e

equivalência dos sistemas através da análise do código gerado (Objetivo O_1). Também buscou **(ii)** avaliar os benefícios da abordagem no processo de portabilidade (Objetivo O_2).

A partir de uma situação-problema foi feita a análise de requisitos e utilizando a abordagem foram gerados sistemas para as duas plataformas que são objeto de estudo deste trabalho. O estudo foi realizado pelo próprio pesquisador e autor deste trabalho. As seções a seguir apresentam o desenvolvimento do estudo de caso.

5.2.1 Análise de requisitos

O domínio de laboratório de análises clínicas foi selecionado para desenvolvimento do estudo de caso. Foi desenvolvido um sistema levando em consideração a seguinte situação:

"Um Website de um laboratório de análises clínicas deve permitir que seus usuários solicitem coleta domiciliar de material para análise. Para isso é necessário armazenar dados do cliente, do médico solicitante e do tipo de exame a ser feito, bem como o material a ser examinado. Basicamente, o usuário deve efetuar seu cadastro e escolher quais exames deseja realizar. Feito isso, o pessoal do laboratório entra em contato com o solicitante e leva os coletores específicos para recolher o material."

Para representar a análise de requisitos da aplicação foram selecionados os diagramas de caso de uso e de classes.

5.2.1.1 Diagrama de Caso de Uso

Um diagrama de caso de uso[1] é a representação das características externamente observáveis do sistema e dos elementos externos que interagem com ele. Pontanto, é um modelo de análise que representa um refinamento dos requisitos funcionais do sistema em desenvolvimento. O modelo de caso de uso é importante pois direciona diversas tarefas posteriores do desenvolvimento do sistema e força os desenvolvedores a moldarem o sistema de acordo com as necessidades do usuário. Um caso de uso é a especificação de uma sequência completa de uma interação entre um sistema e os agentes externos (BEZERRA, 2007). A Figura 5.1 apresenta o diagrama de casos de uso da aplicação. Algumas tarefas são acessíveis apenas pelo administrador do sistema e consequentemente algumas funcionalidades não estarão disponíveis aos usuários finais.

[1]Maiores informações sobre UML podem ser encontradas no endereço: `http://www.omg.org/spec/UML/`

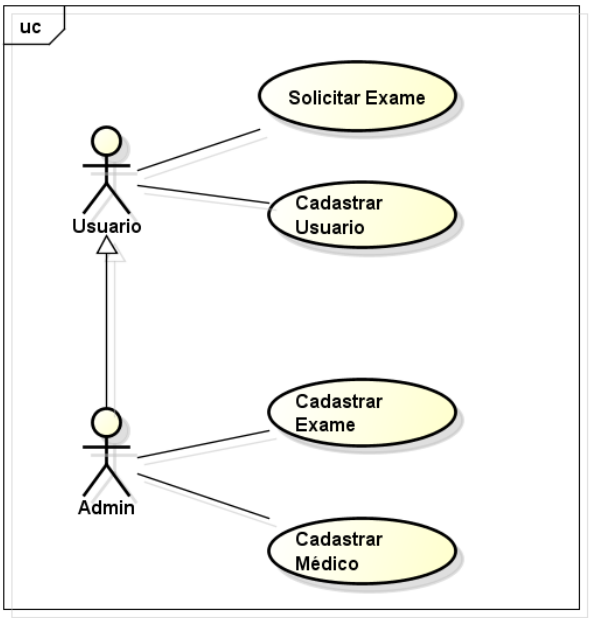

Figura 5.1: Diagrama de Caso de Uso

5.2.1.2 Diagrama de Classes

Por definição, um modelo de classes de análise não leva em consideração restrições inerentes à tecnologia utilizada na solução de um problema. O modelo é composto por objetos identificados na análise do domínio e na análise de aplicação. O diagrama de classes é utilizado na construção do modelo de classes (BEZERRA, 2007). A Figura 5.2 apresenta o diagrama de classes do modelo de domínio da aplicação. Um cliente pode solicitar diversos exames. Uma solicitação de exame, por sua vez, deve informar o médico solicitante que deve já estar previamente cadastrado no sistema.

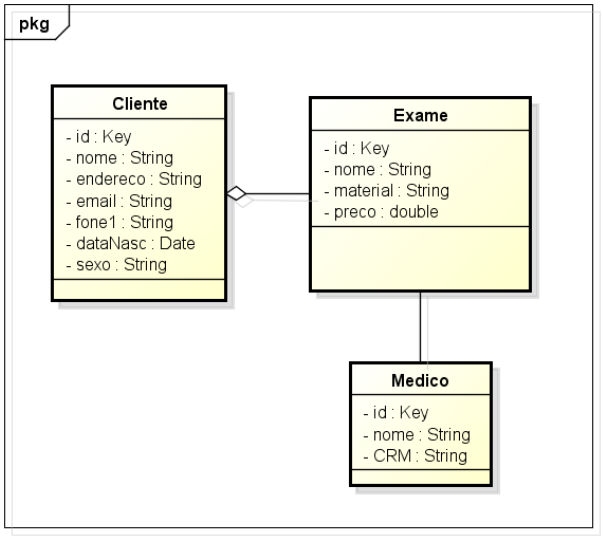

Figura 5.2: Diagrama de Classes

5.2.2 Desenvolvimento da Solução

Com base nos modelos apresentados na seção anterior, foi desenvolvida uma solução utilizando o componente de modelagem independente de domínio criado na abordagem. Para isso, as classes apresentadas na Figura 5.2 foram modeladas utilizando a DSL. A abordagem desenvolvida não trata o controle de permissão de usuários conforme indicado na diagrama de casos de uso da Figura 5.1. Esse requisito foi implementado manualmente após a geração da aplicação.

O modelo de domínio criado serviu de entrada para o conjunto de transformações definidos para as plataformas. As transformações geraram código básico de aplicações CRUD funcionais para cada plataforma especifica. A Figura 5.3 apresenta a estrutura de pacotes das duas versões da aplicação, sinalizando as semelhanças entre as duas aplicações.

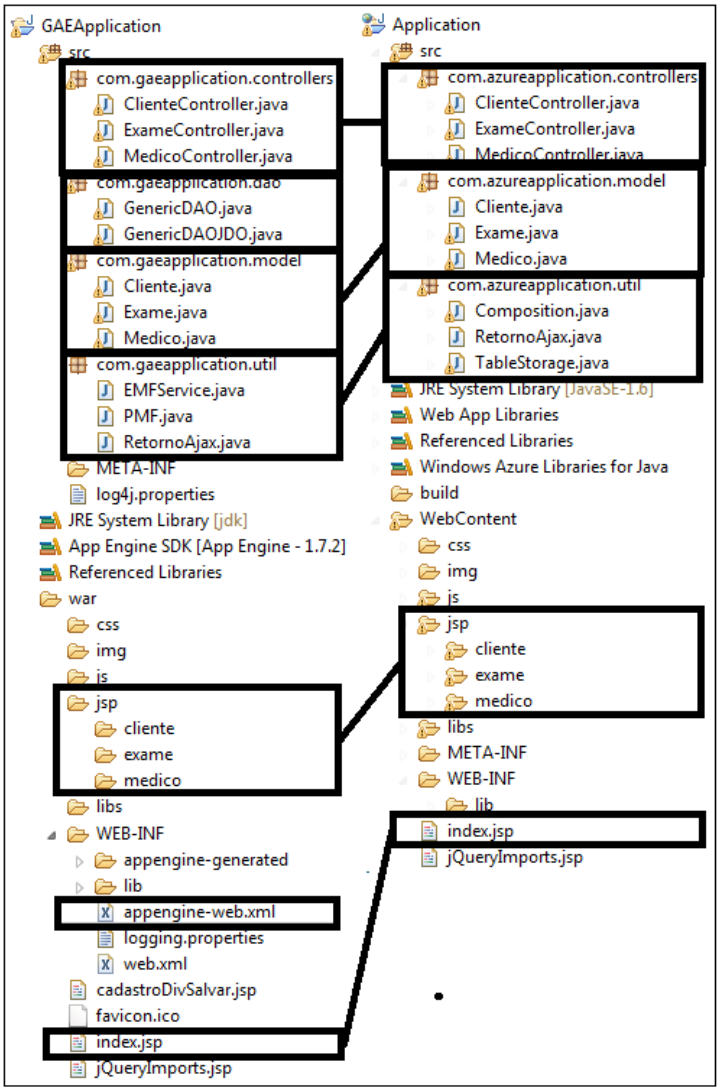

Figura 5.3: Aplicações geradas

Na figura 5.3 é possivel perceber a igualdade entre muitos elementos das duas aplicações, bem como outros elementos que diferem. Embora em termos de funcionalidade as aplicações sejam equivalentes, quando se trata de detalhes de implementação, as mesmas são bastante diferentes. Conforme apontado anteriormente ao longo do texto, cada plataforma possui seus detalhes específicos e suporta o uso de determinadas tecnologias de desenvolvimento. Para ilustrar esse ponto, a Figura 5.4 destaca a diferença entre as classes da camada de modelo da aplicação.

```java
package com.azureapplication.model;

import java.util.*;
import com.microsoft.windowsazure.services.table.client.TableServiceEntity;
//Generated Class

public class Cliente extends TableServiceEntity {

    private   String  nome ;
    private   String  endereco ;
    private   String  email ;
    private   String  fone1 ;
    private   String  fone2 ;
    private  Date  dataNasc ;
    private   String  sexo ;
    private   String  exame ;

    public Cliente( ) {
            this.partitionKey = "Cliente";
        }

    public Cliente(String id) {
            this.partitionKey = "Cliente";
            this.rowKey = id;
        }

    public Cliente(String partitionKey, String rowKey) {
            this.partitionKey = partitionKey;
            this.rowKey = rowKey;
```

(a)

```java
package com.gaeapplication.model;

import java.util.*;
import com.google.appengine.api.datastore.Key;

//annotations import
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

//Generated Class

@PersistenceCapable(detachable="true")
public class Cliente
 {
    @PrimaryKey
    @Persistent(valueStrategy =
        IdGeneratorStrategy.IDENTITY)
    private Key id;
    private String nome;
    private String endereco;
    private String email;
    private String fone1;
    private String fone2;
    private Date dataNasc;
    private String sexo;
    private List<Exame> exame;

```

(b)

Figura 5.4: Classes equivalentes da camada de modelo. O lado esquerdo (a) é uma classe da plataforma Azure, e o lado direito (b) é uma classe da plataforma GAE.

Apesar de ambas implementações terem sido geradas na mesma linguagem (no caso Java) destino, é possível claramente notar as diferenças entre as implementações na camada de modelo. Por exemplo, na figura (b) é possível ver anotações JDO(linhas 14, 17 e 18) e na figura (a) a herança necessária para utilizar o serviço Table Storage da Azure(linha 7). Há também diversos outros elementos que diferem entre as aplicações. A maioria deles relacionados a persistência, como por exemplo as Classes GenericDAOJDO e TableStorage apresentadas na figura 5.3.

Conforme apontado na seção 2.4, a portabilidade no contexto de MDE refere-se à geração de código (utilizando-se elementos do MDE) para diferentes plataformas a partir de um mesmo modelo (KLEPPE; JOS; WIM, 2003). Como uma primeira análise de facilidade de portabilidade, analisou-se a porcentagem de código gerado em relação ao código total das aplicações. A Tab ela 5.1 mostra esses dados. As maioria das modificações feitas a mão (1.2% no Azure e 2.9% no GAE) foram adaptações de interface.

Tabela 5.1: Dados do código das aplicações

Plataforma	*Linhas de Código total*	*Linhas de código geradas*	*% código modificado*
GAE	1874	1820	2.9%
Azure	1567	1551	1.2%

Como análise complementar em termos de produtividade, foi calculada a **razão entre a especificação e código**. Essa métrica permite determinar a relação entre um elemento de especificação e o código gerado correspondente (LUCRéDIO, 2009). Por exemplo, se a especificação de uma entidade de 10 linhas produz 1000 linhas de código, tem-se uma relação 1 para 100. Isso significa que cada elemento de domínio gera em torno de 100 linhas e a partir de então é possível verificar o ganho de produtividade em termos de linhas de código. Esta métrica é calculada da seguinte forma:

$$REC = \sum LOC(cod.\ gerado) / \sum NEE(modelos) \tag{5.1}$$

em que: onde NEE(modelo) corresponde ao número de elementos de especificação do modelo.

Para execução do cálculo, foi definido que um elemento de especificação é uma linha de código da DSL. A solução escrita possui 120 linhas de implementação. Porém, foram consideradas apenas 32 descontando comentários e linhas em branco. Portanto, o cálculo se deu da seguinte maneira:

$$REC = ((1820\ linhasGAE) + (1551\ linhas\ Azure))\ /32\ \therefore REC \cong 105 \tag{5.2}$$

O resultado da equação mostra que, em média, para cada linha de código escrita na DSL foram geradas 105 linhas de código de implementação, incluindo as duas aplicações.

Discussão

A comparação entre as aplicações de um ponto de vista meramente estrutural (Figuras 5.3 e 5.4) não permite avaliar o objetivo O_1 em sua plenitude, pois não considera os aspectos dinâmicos da execução do código. No entanto, foi possível perceber um alto grau de similaridade entre as aplicações geradas, o que indica que a portabilidade, ao menos em termos, está provavelmente sendo alcançada. Uma análise mais detalhada nesse aspecto é apresentada na Seção 5.3.

Com relação ao objetivo O_2, e a partir do desenvolvimento das aplicações apresentado neste estudo de caso, foi possível concluir que a abordagem auxiliou na portabilidade entre as plataformas selecionadas através da geração de aplicações semelhantes para cada plataforma. A alta porcentagem de reutilização de código gerado e baixa porcentagem de código escrito à mão (Tabela 5.1) são indicativos de que, pelo menos no domínio escolhido, a portabilidade pode ser realizada de forma bastante facilitada pela automação. Com relação à produtividade, verificou-se que para cada elemento escrito na DSL foram geradas 105 linhas de código, o que significa que o desenvolvedor gasta menos tempo escrevendo código.

Uma primeira e mais evidente ameaça à validade desse estudo é a sua realização pelo próprio pesquisador autor do trabalho. Seu conhecimento privilegiado 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 possam aprender a utilizá-la e, depois de algum treinamento, chegar ao mesmo nível de eficiência no seu uso.

No contexto da geração de código, pode-se argumentar que o uso da quantidade de linhas de código para mensurar a produtividade pode não ser eficiente pois geradores normalmente produzem um código muito denso. No entanto, vale ressaltar que nesta abordagem a construção de geradores é feita a partir de uma implementação de referência, o que significa que o código gerado segue os mesmos padrões e formato utilizados por programadores humanos.

Medir produtividade utilizando a razão entre a especificação e código também apresenta um risco. No domínio selecionado como objeto de estudo dessa abordagem, as funções geradas possuem muito em comum e boa parte do código pode ser reutilizado entre uma funcionalidade e outra. Ou seja, na prática para desenvolver uma funcionalidade de cadastro de médicos, reaproveita-se o código de uma funcionalidade já desenvolvida anteriormente e não se escreve todo código referente a funcionalidade. É possível encontrar na literatura estudos que exploraram essa questão(reuso). Tais estudos evidenciam que, de uma maneira geral, abordagens MDE, quando comparadas com o desenvolvimento de software de maneira tradicional, ajudam a aumentar de maneira significativa a reusabilidade e produtividade através da geração de código(LUCRéDIO; ALMEIDA; FORTES, 2012; LUCRéDIO, 2009; CIRILO, 2011; MOHAGHEGHI; DEHLEN, 2008)

Uma outra argumentação possível é relacionada ao tempo gasto para construção das transformações. Foi gasto um tempo de 3,2 meses no desenvolvimento desses metaprogramas. Apesar do custo inicial elevado na construção desses artefatos (LUCRéDIO; ALMEIDA; FORTES, 2012; KROGMANN; BECKER, 2007), esses elementos podem ser reutilizados para desenvolvimento de

outros sistemas para as plataformas selecionadas utilizando a linguagem Java. Dessa maneira, em novos projetos, seu tempo de construção já não mais seria contabilizado. No entanto, um estudo quantitativo com relação a essa análise custo-benefício está fora do escopo deste trabalho.

Outro questionamento frequente quando se trata de abordagens MDE é relacionado às modificações feitas diretamente no código (normalmente operações de manutenção e adaptações) e que não são atualizadas nos modelos. Essas modificações, normalmente, são necessárias pois nem sempre é possível gerar todo o código conforme desejado. Na abstração há perda de informações e normalmente essas informações são inseridas manualmente. Para contornar o problema da perda de informações através da representação nos PIM é necessário construir uma linguagem de modelagem suficiente para expressar todas as peculiaridades do problema e isso pode ser custoso e mais caro que propriamente desenvolver os sistema utilizando uma linguagem de programação tradicional. Existem linhas de pesquisa que exploram o problema da inconsistência dos modelos em relação ao código. As pesquisas que mais se aproximam desse problema envolvem a chamada engenharia ida-e-volta (*round-trip engineering* ou RTE)(ANTKIEWICZ; CZARNECKI, 2006; HETTEL; LAWLEY; RAYMOND, 2008). As pesquisas relacionadas à RTE exploram abordagens para manter o sincronismo entre artefatos de origem e destino, mas estão fora do escopo deste trabalho.

A abordagem apresentada neste trabalho pode ser utilizada tanto para desenvolvimento de aplicações para um provedor específico como para portabilidade de aplicações entre provedores. Além disso, é possível também utilizá-la para geração de interesses específicos do sistema, como somente a camada de modelo, ou a camada de controle, ou ainda interface. Isso evitando que o desenvolvedor gaste seus esforços com tarefas repetitivas de codificação.

Em resumo, com relação ao objetivo O_2, no estudo de caso observou-se que a abordagem, assim como outros relatos relativos a MDE na literatura, pode facilitar o processo de portabilidade, principalmente por tornar possível a geração automática de grande parte do código.

Com relação ao objetivo O_1, como já discutido, foram obtidos apenas indícios de similaridade estrutural, mas que são insuficientes para assegurar uma portabilidade minimamente aceitável. Neste sentido, foi realizada uma avaliação complementar, apresentada a seguir.

5.3 Avaliação da Portabilidade

A avaliação da portabilidade sob a perspectiva do usuário foi conduzida utilizando uma abordagem experimental. O processo de execução de um experimento presume a realização

de diferentes atividades (WOHLIN et al., 2000). O número e a complexidade dessas atividades podem variar de acordo com as características do estudo (TRAVASSOS; GUROV; AMARAL, 2002). A literatura apresenta cindo fases gerais que sempre estão presentes num processo de experimentação (WOHLIN et al., 2000):

- **Definição**: é a primeira fase onde o experimento é expresso em termos de problemas objetivos. O objetivo é formulado a partir do problema a ser resolvido;
- **Planejamento**: é onde o projeto do experimento é determinado, a instrumentação é considerada e os aspectos da validade são avaliados;
- **Execução**: esta fase, em princípio, consiste de três etapas: preparação, execução e validação dos dados coletados. No passo de preparação, os participantes do experimento são preparados e o material necessário que compõe a instrumentação é elaborado. Na preparação, os participantes devem ser informados sobre a intenção do estudo, e devem dar consentimento e comprometerem-se em participar do experimento. A principal preocupação dessa fase é garantir que o experimento seja conduzido de acordo com o plano e que os dados sejam coletados de maneira correta;
- **Análise e interpretação dos resultados**: os dados coletados durante a fase de operação servem de entrada para essa etapa. A idéia é analisar os dados para entendê-los e interpretá-los formalmente; e
- **Apresentação e empacotamento:** a ideia dessa fase é organizar adequadamente os dados de maneira que possam ser publicados. É tambem necessário empacotar os dados de maneira que o experimento possa ser replicado. O empacotamento dos dados experimentais pode servir como base para a criação de bibliotecas de experimentação para facilitar o reúso das descobertas em estudos futuros, classificar os dados experimentais e criar relatórios detalhados com os dados confiáveis (TRAVASSOS; GUROV; AMARAL, 2002).

O teste da abordagem proposta neste trabalho seguiu as fases experimentais propostas por Wohlin et al (2000) apresentadas anteriormente e foi conduzida no primeiro semestre de 2013. Quando se trata da portabilidade de aplicações sob a perspectiva do usuário final, espera-se que os participantes que utilizaram o sistema numa plataforma A não percebam a diferença em utilizá-lo em uma plataforma B. Para avaliar este quesito foi elaborado um experimento baseado em casos de teste do tipo caixa preta nos sistemas desenvolvidos no estudo de caso, a exemplo de outros relatos da literatura (SILVA; SOARES, 2009). Os testes foram submetidos a usuários diferentes, em plataformas diferentes e considerou-se que se caso eles conseguissem seguir os casos de teste com sucesso em ambas as plataformas, a portabilidade foi bem sucedida.

5.3.1 Definição

O objetivo do estudo foi:

- **Analisar** as aplicações desenvolvidas utilizando uma abordagem dirigida por modelos;
- **Com o propósito de** avaliar a portabilidade;
- **Do ponto de vista dos** usuários finais;
- **Com respeito à** similaridade de aplicações;
- **No contexto de** usuários aleatórios e das aplicações executando nas plataformas de computação em nuvem GAE e Azure.

O contexto é o ambiente que o experimento irá executar. O contexto brevemente define o pessoal e os artefatos envolvido nos experimentos.

5.3.2 Planejamento

O planejamento do experimento envolveu as seguintes etapas:

a) **Seleção do Contexto.** O experimento ocorreu em ambiente acadêmico, sendo realizado no Laboratório de Engenharia de Software e Banco de dados - LABDES com estudantes selecionados aleatoriamente no âmbito do departamento de Computação da Universidade Federal de São Carlos (UFSCar).

b) **Formulação da Hipótese.** É possível portar (no contexto de MDE) aplicações entre plataformas de Computação em Nuvem sem que os usuários finais percebam as diferenças na utilização dos sistemas em cada plataforma.

c) **Seleção dos Participantes:** A seleção dos participantes foi feita através de amostragem não probabilística por conveniência (WOHLIN et al., 2000), selecionando os indivíduos disponíveis mais próximos para participarem do experimento. Participaram do estudo 10 (dez) alunos de mestrado do departamento de computação da Universidade Federal de São Carlos.

d) **Projeto**: O projeto do experimento descreve como os testes experimentais são organizados e executados. Um caso de teste inclui não só dados de entrada, mas condições e e procedimentos relevantes para sua execução e uma maneira de determinar se o programa passou ou falhou no teste (YOUNG, 2008). Tradicionalmente, um teste verifica se um requisito foi

implementado conforme a especificação. A definição dos casos de teste para este estudo foi feita com base nos requisitos de análise das aplicações. Para isso foram definidos casos de teste isolados, conforme exemplo apresentado na Tabela 5.2. Uma relação completa dos casos de teste utilizados encontram-se no Apêndice A

Tabela 5.2: Exemplo da descrição de um caso de teste

Descrição	Efetuar o cadastro de um médico
Pré-condição	
Etapas	1. Clique na aba cadastro/Médico 2. Clique no botão novo 3. Preencha os dados solicitados 4. Clique em Salvar
Resultados Esperados	1. O item deve ser inserido adequadamente.

O apêndice A apresenta os casos de teste executados.

5.3.3 Execução do experimento

O experimento foi executado em 2 passos:

(a) **Preparação.** Nesta fase os materiais definidos na instrumentação do experimento foram elaborados:

- **Diretrizes**: Os seguintes documentos foram produzidos para serem utilizados no estudo: 1) Descrição da tarefa, com as instruções da sua execução; e 2) Descrição da aplicação e material de apoio.
- **Instrumentos de coleta de dados**: Como o que estava sendo testado através dos casos de teste era a funcionalidade da aplicação, o instrumento de medida era o sucesso ou falha de um caso de teste. Havia no documento um campo que possibilitava o usuário afirmar se o caso de teste foi concluído com sucesso ou não. Os testes serviram para que os usuários entrassem em contato com as aplicações para em seguida responder se segundo seu julgamento a aplicação foi de fato portada.

e) **Execução**: Inicialmente, utilizando a abordagem MDE, foram geradas aplicações para cada plataforma. Logo após, os usuários executaram os testes experimentais conforme previso no projeto. Cada usuário realizou os testes primeiro em uma plataforma, e depois os reproduziu na outra. Os usuários foram informados sobre quais plataformas estariam usando.

5.3.4 Análise dos dados

Nos testes executados, os usuários utilizaram o sistema de acordo com um conjunto de casos de teste previamente desenvolvidos. Os casos de teste foram todos executados com sucesso em ambas as plataformas. Para avaliação complementar, ao final de cada teste os usuários responderam questões avaliativas. Conforme apontado na seção 2.5, a norma ISO 9126 (1991) desenvolvida para identificar atributos de qualidade de software define portabilidade como a facilidade com a qual o software pode ser transposto de um ambiente para outro conforme os seguintes sub-atributos: Adaptabilidade, Capacidade para ser Instalado, Coexistência e Capacidade para Substituir. É importante ressaltar, que em se tratando da norma ISO 9126, a avaliação levou em consideração o item capacidade de ser substituído. A norma é específica para portabilidade de sistemas entre plataformas computacionais, e esse quesito é o mais apropriado para analisar a portabilidade no contexto desta pesquisa. Portanto, as questões são relacionadas a esse item.

- **Você considera que em termos de funcionalidades os sistemas são equivalentes?** Esta questão avalia a equivalência entre os sistemas em termos de funcionalidade. Para que dois sistemas A e B sejam equivalentes o conjunto de funcionalidades implementadas em A deve ser igual ao conjunto de funcionalidades implementadas em B. Neste quesito todos os 10 participantes consideraram que os sistemas eram equivalentes.

- **Você considera que em termos de interface os sistemas são equivalentes?** Um sistema pode ser equivalente em termos de funcionalidades mas possuir interfaces de acesso diferentes. Esta questão avalia se em termos de interface com o usuário os sistemas gerados eram equivalentes. Neste quesito todos os 10 participantes consideraram que os sistemas eram equivalentes.

- **Você considera que para as mesmas entradas em cada sistema, foram obtidas as mesmas saídas?** Esta questão avalia se os sistemas são equivalentes em termos de saídas para as mesmas entradas. Neste quesito todos os 10 participantes consideraram que os sistemas eram equivalentes.

- **Você considera que o sistema A (Azure) pode ser substituído pelo Sistema B (GAE)?** Esta questão avalia se o sistema A pode ser substituido pelo sistema B. Neste quesito todos os 10 participantes consideraram que A pode ser substituido pelo sistema B.

5.3.5 Interpretação dos Resultados

A partir das avaliações feitas, é possível concluir que foi possível portar (no contexto de MDE) aplicações entre plataformas de Computação em Nuvem sem que os usuários finais percebessem as diferenças na utilização dos sistemas em cada plataforma. A abordagem apresentada neste trabalho configura uma alternativa para a portabilidade e consequentemente para resolução do problema do *Lock-In*.

5.3.6 Ameaças à Validade

Como ameaças a validade dos estudos experimentais executados é possível citar:

- Baixo número de usuários: um pequeno número de ususários participaram dos experimentos. Um maior número de pessoas utilizando o sistema pode levar a descoberta de mais falhas e diferênças.
- Domínio e casos de teste simplificados: a aplicação utilizada no estudo de caso foi relativamente pequena e apresenta um domínio de problema simplificado. Tais elementos podem facilitam a análise e é possível que pode não ter se testado tudo.
- Tratamento estatístico: não foi feito nenhum tipo de tratamento estatístico nas análises realizadas.

5.4 Considerações finais

O objetivo da abordagem desenvolvida é facilitar a portabilidade de aplicações entre plataformas de nuvem, fazendo uso dos benefícios da MDE. Com isso, a expectativa era não somente possibilitar a portabilidade (objetivo O_1), mas também facilitá-la (O_2).

Na avaliação, ambos os objetivos foram considerados. Em um estudo de caso observou-se benefícios oriundos da MDE na portabilidade, em termos de quantidade de código gerado. Resultados similares são encontrados na literatura(LUCRéDIO; ALMEIDA; FORTES, 2012), portanto há fortes indícios de que o objetivo O_1 foi atingido. No estudo de caso, também foi feita uma avaliação comparativa das estruturas das aplicações geradas para as duas plataformas. Essa avaliação indicou certa similaridade, mas insuficiente para considerar o objetivo O_2 como atingido.

Em um experimento envolvendo 10 (dez) usuários, as aplicações geradas para as duas plataformas (GAE e Azure) foram avaliadas em termos de sua funcionalidade para o usuário final.

Após a realização de um conjunto de casos de teste, todos os usuários consideraram as aplicações suficientemente similares, e portanto considera-se o objetivo O_2 como atingido, ao menos no domínio escolhido (CRUD).

Capítulo 6

CONCLUSÃO

A computação em nuvem tem potencial para revolucionar a indústria de software. Todavia, esse modelo computacional ainda está emergindo e há muita confusão a respeito das definições dos termos e serviços que surgiram em sua decorrência. Grande parte das plataformas de computação em nuvem são proprietárias e, portanto, fechadas. A falta de uma tecnologia padrão para desenvolvimento de sistemas para nuvem gera o problema de Lock-In. Esse problema está dificultando a adoção do modelo de nuvem, uma vez que os usuários ficam vulneráveis ao aumento de preço, problemas de confiabilidade e dificuldade extra para abandonar o modelo ou migrar de um provedor de nuvem para outro. No sentido de resolver esse problema este trabalho apresentou uma abordagem alternativa à padronização, que é o caminho proposto pela indústria. A abordagem apresentada faz uso do desenvolvimento dirigido por modelos (MDE). As tecnologias de nuvem e MDE, aliadas, oferecem muitos benefícios aos desenvolvedores de sistemas. O MDE oferece uma maior produtividade, melhor manutenção e documentação e reúso. A avaliação realizada observou alguns desses benefícios, o que comprova a viabilidade da abordagem MDE.

No contexto do domínio delimitado como escopo desta pesquisa, foi possível portar aplicações entre os provedores objetos de estudo utilizando a abordagem proposta. A portabilidade é um atributo de qualidade de software e certamente ajudará ainda mais na disseminação do modelo de nuvem.

Trabalhos Futuros

Em trabalhos futuros serão realizadas mais avaliações da abordagem. Tais estudos visarão verificar a viabilidade da construção dos artefatos, bem como avaliar processos de desenvolvimento utilizando a DSL desenvolvida neste trabalho. No momento da escrita desta dissertação,

já se encontra em andamento um estudo de caso que visa explorar a infraestrutura de geração de código e portabilidade com outros desenvolvedores. Diferentemente do estudo apresentado na Seção 5.2, espera-se com isso identificar pontos de melhoria não percebidos pelo autor deste trabalho.

Além disso, para dar continuidade ao trabalho e chegar à resolução do problema da interoperabilidade, será desenvolvida uma abordagem dirigida por modelos para interoperabilidade entre plataformas de nuvem. Os artefatos desenvolvidos neste trabalho passarão por melhorias e serão aproveitados no desenvolvimento de referida abordagem. Espera-se demonstrar a MDE como uma opção alternativa para o problema da interoperabilidade no cenário de computação em nuvem, possibilitando a construção de sistemas que tirem melhor proveito de cada oferta de serviço de nuvem.

PUBLICAÇÕES E PROJETOS

A partir da pesquisa apresentada nesta dissertação, foram publicados os seguintes artigos:

SILVA, E. A. N.; LUCRéDIO, D. Software Engineering for the cloud: A Research Roadmap. SBES - Software Engineering Brazilian Symposium, Brazil, UFSCar, 2012 (Qualis CAPES B2). Esse artigo tem por base a revisão bibliográfica(revisão sistemática) deste trabalho de mestrado, sendo um dos pioneiros da área de engenharia de software para computação em nuvem no Brasil.

Silva, E. A. N.; Lucrédio, D. Em direção à portabilidade entre plataformas de Computação em Nuvem usando MDE. III WB-DSDM - Brazilian Workshop on Model-Driven Software Development, 2012. Este artigo descreve a ideia inicial do trabalho, já com os conceitos que (na época) estavam sendo implementados.

Os seguintes trabalhos também foram submetidos:

Silva, E. A. N.; Fortes, R. P. M. Uma abordagem dirigida por modelos para interoperabilidade entre plataformas de computação em nuvem. Projeto FAPESP Nº 2012/24487-3 - Bolsa de Doutorado Regular. Aprovado.

Silva, E. A. N.; Lucrédio, D.; Fortes, R. P. M. A model-driven approach for promoting cloud interoperability. Software Engineering Innovation Foundation - SEIF Microsoft Research Award 2013. Não aprovado.

Silva, E. A. N.; Lucrédio, D.; Silva, V. G. A model-driven approach for promoting Cloud PaaS portability. 2013. Submetido à XXXIX Latin American Computing Conference -CLEI 2013. Em avaliação.

REFERÊNCIAS

AMBLER, S. W. Agile model driven development is good enough. *IEEE Software*, v. 20, p. 71–73, 2003.

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

ARMBRUST, M. et al. 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>.

BEZERRA, E. *Princípios de Análise e Projeto de Sistemas com UML*. [S.l.]: Elsevier, 2007.

BITTAR, T. J. et al. 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.

BREITMAN, K.; VIRTEBO, J. Computação na Nuvem - Uma Visão Geral. In: . [S.l.]: CONSEGUI, 2010.

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.pdf>.

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.

CIRILO, C. E. *Model Driven RICHUB - Processo Dirigido a modelos para a contrução de interfaces ricas de aplicações ubíquas sensíveis ao contexto.* Tese (Doutorado), 2011.

DANIELS, J. Server virtualization architecture and implementation. *Crossroads*, ACM, New York, NY, USA, v. 16, n. 1, p. 8–12, 2009. ISSN 1528-4972. Disponível em: <http://doi.acm.org/10.1145/1618588.1618592>.

DEURSEN, V. et al. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, v. 35, n. 6, p. 26–36, 2000.

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.

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern Web architecture. *ACM Trans. Internet Technol.*, ACM, New York, NY, USA, v. 2, n. 2, p. 115–150, maio 2002. ISSN 1533-5399.

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. Disponível em: <http://dx.doi.org/10.1109/FOSE.2007.14>.

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

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.

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

IDC, I. D. C. Cloud Computing's Role in Job Creation. 2012. Disponível em: <http://www.microsoft.com/en-us/news/download/features/2012/IDC_Cloud_jobs_White_Paper.pdf>.

ISO/IEC9126. Software product evaluation - Quality characteristics and guidelines for their use. . In: . [S.l.]: Norma, 1991.

JURISTO, N.; MORENO, A. M. *Basics of software engineering experimentation*. [S.l.]: Springer Publishing Company, Incorporated, 2010.

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.

KHAJEH-HOSSEINI, A. et al. The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise. *Software: Practice and Experience*, John Wiley & Sons, Ltd, v. 42, n. 4, p. 447–465, 2012. ISSN 1097-024X. Disponível em: <http://dx.doi.org/10.1002/spe.1072>.

KHAJEH-HOSSEINI, A. et al. Decision Support Tools for Cloud Migration in the Enterprise. In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. [S.l.: s.n.], 2011. p. 541–548. ISSN 2159-6182.

KLEPPE, A.; JOS, W.; WIM, B. MDA Explained, The Model-Driven Architecture: Practice and Promise. In: . [S.l.]: Addison-Wesley, 2003.

KROGMANN, K.; BECKER, S. A case study on model-driven and conventional software development: The palladio editor. *Proc. of tSoftware Engineering*, p. 169–176, 2007.

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, maio 2008. ISSN 1520-9202.

LUCRéDIO, D. Uma Abordagem Orientada a Modelos para Reutilização de Software. In: . [S.l.]: Tese de Doutorado, 2009.

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.

MAFRA, S. N.; TRAVASSOS, G. H. Técnicas de Leitura de Software: Uma Revisão Sistemática. *XIX Simpósio Brasileiro de Engenharia de Software (SBES 2005)*, 2005. Disponível em: <http://tcc-tonismar.googlecode.com/svn/trunk/TCC/Bibliografia/05 - 9612.pdf>.

MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing . In: . [s.n.], 2009. Disponível em: <http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf/>.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, ACM, v. 37, n. 4, p. 316–344, 2005.

MOHAGHEGHI, P.; ANDTHER, T. S. Software Engineering Challenges for Migration to the Service Cloud Paradigm: Ongoing Work in the REMICS Project. In: *Services (SERVICES), 2011 IEEE World Congress on*. [S.l.: s.n.], 2011. p. 507–514.

MOHAGHEGHI, P. et al. REMICS-REuse and Migration of Legacy Applications to Interoperable Cloud Services. *Towards a Service-Based Internet*, Springer, p. 195–196, 2010.

MOHAGHEGHI, P.; DEHLEN, V. Where is the proof?-A review of experiences from applying MDE in industry. In: *Model Driven Architecture–Foundations and Applications*. [S.l.: s.n.], 2008. p. 432–443.

MOORE, B. et al. *Eclipse Development using the Graphical Editing Framework and the EclipseModeling Framework. [S.l.]: ibm.com/redbooks.* [S.l.: s.n.], 2004.

NARASIMHAN, B.; NICHOLS, R. State of Cloud Applications and Platforms: The Cloud Adopters' View. *Computer*, v. 44, n. 3, p. 24–28, mar. 2011. ISSN 0018-9162. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5719574>.

OMG. MDA Guide Version 1.0. 1. 2003.

OMG. *Object Constraint Language (OCL)*. [S.l.], 2012. Disponível em: <http://www.omg.org/spec/OCL/2.3.1/>.

PAPAZOGLOU, M. P. et al. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, v. 40, n. 11, p. 38–45, 2007. ISSN 0018-9162.

PAPAZOGLOU, M. P. et al. Service-oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, v. 17, n. 2, p. 223–255, 2008. Disponível em: <http://www.worldscinet.com/ijcis/17/1702/S0218843008001816.html>.

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.

SARIPALLI, P.; PINGALI, G. MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds. In: *2011 IEEE 4th International Conference on Cloud Computing*. [S.l.]: IEEE, 2011. p. 316–323. ISBN 978-1-4577-0836-7. ISSN 2159-6182.

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.

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.

SILVA, E. A. N.; LUCRéDIO, D. Software Engineering for the cloud: A Research Roadmap. *SBES - Software Engineering Brazilian Symposium*, Brazil, UFSCar, 2012.

SILVA, L.; SOARES, S. Analyzing structure-based techniques for test coverage on a J2ME software product line. In: IEEE. *Test Workshop, 2009. LATW'09. 10th Latin American*. [S.l.], 2009. p. 1–6.

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.

TANENBAUM, A. S.; Van Steen, M. *Distributed systems: principles and paradigms*. [S.l.]: Prentice Hall Upper Saddle River, NJ:, 2002.

TAURION, C. *Computação em Nuvem - Transformando o Mundo da Tecnologia da Informação*. [S.l.: s.n.], 2009.

THOMAS, D. Revenge of the modelers or uml utopia? *IEEE Software*, v. 21, p. 15–17, 2004.

TICHY, W. F. Should computer scientists experiment more? *Computer*, IEEE, v. 31, n. 5, p. 32–40, 1998.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. *Introdução à engenharia de software experimental - Relatório Técnico*. [s.n.], 2002. Disponível em: <http://www2.ufpa.br/cdesouza/teaching/topes/4-ES-Experimental.pdf>.

UNIVERSITY, K. Guidelines for performing Systematic Literature Reviews in Software Engineering. 2007.

VAQUERO, L. M. et al. A break in the clouds. *ACM SIGCOMM Computer Communication Review*, v. 39, n. 1, p. 50, 2008. ISSN 01464833. Disponível em: <http://dl.acm.org/citation.cfm?id=1496091.1496100>.

VELTE, T.; VELTE, A.; ELSENPETER, R. *Cloud Computing, A Practical Approach*. [S.l.]: McGraw-Hill, Inc., 2009.

W3C. *Web Services Description Language (WSDL) 1.1*. 2012. Disponível em: <http://www.w3.org/TR/wsdl>.

WOHLIN, C. et al. Experimentation in software engineering: an introduction. Kluver Academic Publishers, 2000.

YAM, C.-Y. et al. Migration to Cloud as Real Option: Investment Decision under Uncertainty. In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. [S.l.: s.n.], 2011. p. 940–949.

YIN, R. K. *Case study research: Design and methods*. [S.l.]: Sage Publications, Incorporated, 2008.

YOUNG, M. *Software Testing and Analysis: Process, Principles, and Techniques*. [S.l.]: John Wiley & Sons, 2008.

YOUSEFF, L.; BUTRICO, M.; Da Silva, D. Toward a Unified Ontology of Cloud Computing. In: *Grid Computing Environments Workshop, 2008. GCE '08*. [S.l.: s.n.], 2008. p. 1–10.

YU, J. et al. Understanding mashup development. *Internet Computing, IEEE*, IEEE, v. 12, n. 5, p. 44–52, 2008.

ZARDARI, S.; BAHSOON, R. Cloud adoption: a goal-oriented requirements engineering approach. In: *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing*. New York, NY, USA: ACM, 2011. (SECLOUD '11), p. 29–35. ISBN 978-1-4503-0582-2. Disponível em: <http://doi.acm.org/10.1145/1985500.1985506>.

GLOSSÁRIO

AJAX – *Asynchronous JavaScript and XML*

API – *Application Programming Interface*

ASP – *Application Server Provider*

BaaS – *Business as a Service*

CIM – *Computation Independent Model*

CRUD – *Create, Retrieve, Update e Delete*

CSS – *Cascading Style Sheets*

DMTF – *Distributed Management Task Force*

DSL – *Domain Specific Language*

DSL – *Domain Specific Language*

DaaS – *[Development, Database, Desktop] as a Service*

EMF – *Eclipse Modeling Framework*

ERP – *Enterprise Resource Planning*

FaaS – *Framework as a Service*

GAE – *Google App Engine*

GMF – *Graphical Modeling Framework*

HaaS – *Hardware as a Service*

IDE – *Integrated development environment*

IaaS – *Infrastructure as a Service*

JDO – *Java Data Objects*

JET – *Java Emitter Template*

JMI – *Java Metadata Interface*

JPA – *Java Persistence API*

JSP – *Java Server Pages*

JVM – *Java Virtual Machine*

M2T – *Modelo to text*

MDA – *Model-Driven Architecture*

MDE – *Model-Driven Engineering*

MOF – *Meta Object Facility*

MVC – *Model-View-Controller*

MaaS – *Modeling as a Service*

NIST – *Instituto Nacional de Padrões e Tecnologia dos Estados Unidos*

OMG – *Object Management Group*

OVF – *Open Virtualization Format*

OVF – *Open Virtualization Format*

PIM – *Platform Independent Model*

PSM – *Platform Specific Model*

PaaS – *Platform as a Service*

PaaS – *Platform as a Service*

REST – *Representational State Transfer*

SDK – *Software Development Kit*

SLA – *Service Level Agreements*

SOAP – *Simple Object Access Protocol*

SOA – *Service-Oriented Architecture*

SQL – *Structured Query Language*

SaaS – *Software as a Service*

UML – *Unified Modeling Language*

WSDL – *Web Services Description Language*

XML – *Extensible Markup Language*

XaaS – *Everything as a Service*

Apendice A

CASOS DE TESTE

Casos de Teste

Casos de Teste – Bloco 1 - Cadastro

1. Cadastrar um novo Médico
    a. **Descrição:** Cadastrar um novo médico no sistema
    b. **Pré-condições:**
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Medico -> Abrir a listagem
        - Usuário Clica na Opção Novo 
        - Usuário preenche todos os dados (Sugestões na folha anexa)
        - Usuário clica no botão salvar 
    d. **Resultados esperados:**
        - Deverá aparecer uma mensagem de que o médico foi cadastrado com Sucesso.
        - O novo medico deve ser persistido e aparecer na listagem de médicos.

Resultado Azure: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

---

2. Cadastrar um novo Médico
    a. **Descrição:** Cadastrar um novo médico no sistema
    b. **Pré-condições:**
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Medico -> Abrir a listagem
        - Usuário Clica na Opção Novo 
        - Usuário preenche todos os dados (Sugestões na folha anexa)
        - Usuário clica no botão salvar 
    d. **Resultados esperados:**
        - Deverá aparecer uma mensagem de que o médico foi cadastrado com Sucesso.
        - O novo medico deve ser persistido e aparecer na listagem de médicos.

Resultado Azure: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

---

3. Cadastrar um novo Exame
    a. **Descrição:** Cadastrar um novo exame no sistema
    b. **Pré-condições:**
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Exame -> Abrir a listagem
        - Usuário Clica na Opção Novo 
        - Usuário preenche todos os dados (Sugestões na folha anexa)
        - Usuário clica no botão salvar 
    d. **Resultados esperados:**
        - Deverá aparecer uma mensagem de que o exame foi cadastrado com Sucesso.
        - O novo exame deve ser criado e aparecer na listagem de clientes.

Resultado Azure: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

---

4. Cadastrar um novo Exame
    a. **Descrição:** Cadastrar um novo exame no sistema
    b. **Pré-condições:**
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Exame -> Abrir a listagem
        - Usuário Clica na Opção Novo 
        - Usuário preenche todos os dados (Sugestões na folha anexa)
        - Usuário clica no botão salvar
    d. **Resultados esperados:**
        - Deverá aparecer uma mensagem de que o exame foi cadastrado com Sucesso.
        - O novo exame deve ser criado e aparecer na listagem de clientes.

Resultado Azure: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

---

5. Cadastrar um novo Cliente
    a. **Descrição:** Cadastrar um novo cliente no sistema
    b. **Pré-condições:** deverá haver médicos e exames pré-cadastrados no sistema.
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Cliente -> Abrir a listagem
        - Usuário Clica na Opção Novo 
        - Usuário preenche todos os dados (Sugestões na folha anexa)
        - Usuário clica no botão salvar 
    d. **Resultados esperados:**
        - Deverá aparecer uma mensagem de que o cliente foi cadastrado com Sucesso.
        - O novo cliente deve ser criado e aparecer na listagem de clientes.

Resultado Azure: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

---

6. Efetuar uma solicitação
    a. **Descrição:** Solicitar uma exame a ser feito por um cliente específico.
    b. **Pré-condições:** o cliente deverá estar pré-cadastrado no sistema.
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Cliente -> Abrir a listagem
        - Usuário seleciona o cliente
        - Usuário efetua solicitações de exames.
        - Usuário clica no botão salvar exame 
    d. **Resultados esperados:**
        - O exame cadastrado deverá aparecer na listagem de exames para o cliente específico.

Resultado Azure: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Casos de Teste – Bloco 2 – Atualização

1. Atualizar o endereço do Cliente
    a. **Descrição:** atualização do endereço do cliente
    b. **Pré-condições:** o cliente deverá estar pré-cadastrado no sistema.
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Cliente -> Abrir a listagem
        - Usuário Seleciona o cliente
        - Usuário altera o endereço (Sugestões na folha anexa)
        - Usuário clica no botão salvar 
    d. **Resultados esperados:**
        - Deverá aparecer uma mensagem de que o cliente foi atualizado com Sucesso.
        - O novo valor deverá ser persistido e aparecer na listagem de clientes.

Resultado Azure: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

---

2. Atualizar o CRM de um Médico
    a. **Descrição:** atualização do crm do médico
    b. **Pré-condições:** o médico deverá estar pré-cadastrado no sistema.
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Médico -> Abrir a listagem
        - Usuário Seleciona o médico

- Usuário altera o crm (Sugestões na folha anexa)
- Usuário clica no botão salvar 

d. **Resultados esperados:**
   - Deverá aparecer uma mensagem de que o médico foi atualizado com Sucesso.
   - O novo valor deverá ser persistido e aparecer na listagem de médicos.

Resultado Azure: a) Sucesso b) Falha
- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha
- Em Caso de Falha, motivo:_____

---

3. Atualizar o preço de um exame
   a. **Descrição:** atualização do preço de um exame
   b. **Pré-condições:** o exame deverá estar pré-cadastrado no sistema.
   c. **Etapas:**
      - Usuário Clica na Opção Cadastro/Exames -> Abrir a listagem
      - Usuário Seleciona o exame
      - Usuário altera preço (Sugestões na folha anexa)
      - Usuário clica no botão salvar
   d. **Resultados esperados:**
      - Deverá aparecer uma mensagem de que o exame foi atualizado com Sucesso.
      - O novo valor deverá ser persistido e aparecer na listagem de médicos.

Resultado Azure: a) Sucesso b) Falha
- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha
- Em Caso de Falha, motivo:_____

Casos de Teste – Bloco 3 – Remoção

1. Remover um exame
    a. **Descrição:** remoção de um registro de exame do sistema
    b. **Pré-condições:** o exame deverá estar pré-cadastrado no sistema.
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Exames -> Abrir a listagem
        - Usuário Seleciona o exame
        - Usuário clica no botão remover 
    d. **Resultados esperados:**
        - O cliente deverá ser removido com Sucesso.
        - O novo valor deverá ser persistido e aparecer na listagem de clientes.

Resultado Azure: a) Sucesso b) Falha
- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha
- Em Caso de Falha, motivo:_____

---

2. Remover uma solicitação
    a. **Descrição:** remoção de um registro de solicitação do sistema
    b. **Pré-condições:** a solicitação deverá estar inserida na lista de solicitações do usuário
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Cliente -> Abrir a listagem
        - Usuário seleciona o cliente.
        - Usuário seleciona a solicitação.
        - Usuário clica no botão remover solicitação. 
    d. **Resultados esperados:**
        - A solicitação deverá ser removida com Sucesso.
        - O novo valor deverá ser persistido e aparecer na listagem de clientes.

Resultado Azure: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

---

3. Remover um médico
    a. **Descrição:** remoção de um registro de médico do sistema
    b. **Pré-condições:** o médico deverá estar inserido na lista de solicitações do usuário
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Medico. -> Abrir a listagem
        - Usuário seleciona o médico.
        - Usuário clica no botão remover médico. 
    d. **Resultados esperados:**
        - A solicitação deverá ser removida com Sucesso.
        - O novo valor deverá ser persistido e aparecer na listagem de clientes.

Resultado Azure: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

Resultado GAE: a) Sucesso b) Falha

- Em Caso de Falha, motivo:_____

---

4. Cadastrar um novo Exame(Forçar um Erro)
    a. **Descrição:** Cadastrar um novo exame no sistema
    b. **Pré-condições:**
    c. **Etapas:**
        - Usuário Clica na Opção Cadastro/Exame -> Abrir a listagem
        - Usuário Clica na Opção Novo 
        - Usuário preenche todos os dados (Sugestões na folha anexa)
            1. Coloque uma letra no campo de valores para analisar como o sistema se comporta.
        - Usuário clica no botão salvar
    d. **Resultados esperados:**
        - Deverá aparecer uma mensagem de que o exame foi cadastrado com Sucesso.

- O novo exame deve ser criado e aparecer na listagem de clientes.

e. **Resultado:** a) Sucesso b) Falha
   - Em Caso de Falha, motivo:_____
   - Obs: _____

Questionário

1. Você considera que em termos de funcionalidades o sistemas são equivalentes?

_____
_____

2. Você considera que em termos de Interface os sistemas são equivalentes?

_____
_____

3. Você considera que para as mesmas entradas em cada sistema, foram obtidas as mesmas saídas?

_____
_____

4. Você acha que o sistema A(Azure) pode ser substituído pelo Sistema B(GAE)

_____
_____

1. Você considera que em termos de funcionalidades o sistemas são equivalentes?

_____
_____

2. Você considera que em termos de Interface os sistemas são equivalentes?

_____
_____

3. Você considera que para as mesmas entradas em cada sistema, foram obtidas as mesmas saídas?

_____
_____

4. Você acha que o sistema A(Azure) pode ser substituído pelo Sistema B(GAE)

_____
_____

Sugestões

Sugestões Médicos

Dr(a). Carla Minozzo
Dr(a). Maria Ferreira De Freitas
Dr(a). Mario Jose Silva Gomes
Dr(a). Thais Nogueira Lobo
Dr(a). Marcos Elias Castro
Dr(a). Augusto Cesar Freitas

Sugestões Exames

Ácido Úrico - Urina R$ 3.00

Espermograma – Esperma R$ 10.00

Espermograma Após Vasectomia – Eperma R$ 10.00

Glicosúria Amostra – Sangue R$ 5.00

Helicobacter Pylori – Fezes R$ 8.00

Hematócrito – Sangue R$ 3.50

Sugestões Clientes

Carlos Botelho Gama Avenida Minas Gerais, 254 – Centro Fone: 78 8854-9856
Data Nasc: 04/05/1980

Exames: Colesterol, Glicose, Espermograma

Marcos Pires Tejano Av. das Oliveira, 112 – São Joaquim Fone 56 89568542

Data Nasc: 01/08/1994

Exames: Espermograma após vasectomia, Hemograma