UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação
Mestrado em Ciência da Computação

Dissertação

Uma abordagem para modelagem de aplicações de computação móvel na nuvem

Bruna Gonçalves Ribeiro

Pelotas, 2016

Bruna Gonçalves Ribeiro

Uma abordagem para modelagem de aplicações de computação móvel na nuvem

Dissertação apresentada ao Programa de Pós Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Lisane Brisolara de Brisolara
Coorientador: Adenauer Corrêa Yamin

Pelotas, 2016

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

R484a Ribeiro, Bruna Gonçalves

Uma abordagem para modelagem de aplicações de computação móvel na nuvem / Bruna Gonçalves Ribeiro ; Lisane Brisolara de Brisolara, orientadora ; Adenauer Corrêa Yamin, coorientador. — Pelotas, 2016.

54 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2016.

1. Modelagem. 2. Padrões de projeto. 3. Multiplatafomas. 4. Computação móvel na nuvem. 5. CRUD. I. Brisolara, Lisane Brisolara de, orient. II. Yamin, Adenauer Corrêa, coorient. III. Título.

CDD : 005

Elaborada por Aline Herbstrith Batista CRB: 10/1737

Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Ata de Apreciação de Defesa de Dissertação

Aluno	BRUNA GONÇALVES RIBEIRO
Matrícula	13102506
Curso	Mestrado (PPGC-UFPel)
Título original	Uma abordagem para modelagem de aplicações de computação móvel na nuvem
Tipo de Defesa	Não sigilosa

Apreciação sobre a Dissertação

Em 29 de Março de 2016, os membros abaixo nomeados para a defesa da Dissertação do aluno **BRUNA GONÇALVES RIBEIRO**, matriculado no Programa de Pós-Graduação em Computação, consideraram a Dissertação **aprovada** e estabelecem um prazo máximo de 30 dias para as correções e entrega da versão definitiva, conforme a Ata de Correções, para homologação.

Membros da banca examinadora

Nome	CPF	Instituição	Assinatura
Lisane Brisolara	--	Orientador	
Ana Marilza Pernas	--	PPGC-UFPel	
Cristiano Costa	690.966.270-91	Unisinos	

Agradecimentos

Gostaria de começar agradecendo primeiramente aos meus pais, pois sempre me deram a oportunidade e o incentivo a ir em busca dos meus objetivos, sempre prontos a embarcar nas minhas “loucuras”, nas minhas “indiadas”, e sempre prontos a estender um ombro amigo sempre que preciso.

Agradeço e muito ao meu namorado Jones, por que com certeza se não fosse por ele, nada disso estaria acontecendo. Foi ele quem deu o “empurranzinho” inicial, falando do edital para aluno especial do mestrado, e depois me ajudando a fazer meu projeto de pesquisa para entregar na seleção para aluno regular. Com certeza se não fosse essa ajuda hoje não estaríamos comemorando essa vitória.

Um agradecimento especial aos meus professores da graduação, Dilli e Roger, pelas cartas de recomendação na inscrição do mestrado. Podem ter certeza que essa vitória é dedicada a vocês também.

Agora um agradecimento em conjunto a minha família, mãe, pai, mano e Jones, vocês foram e sempre serão essenciais para a decisão de qualquer caminho que eu queira seguir, qualquer decisão que já tomei na vida. Vocês são meus pilares, as pessoas que sempre estiveram aqui, do meu lado, me apoiando e incentivando a nunca desistir dos sonhos. E olha que a vontade de desistir não foi pouca, só vocês sabem muito bem das minhas angústias. Mas com o apoio e incentivo de vocês, esse sonho se tornou realidade, e hoje estaremos comemorando essa vitória que é nossa.

Agradeço também aos meus amigos que sempre estiveram do meu lado, seja para apoiar nos momentos difíceis, perguntar como estava o mestrado, entre tantas outras coisas. Em especial quero agradecer a Laurinha, minha amiga artista que fez minhas imagens “de boa”, na maior amizade. Laurinha fica aqui meu eterno agradecimento a essa amiga e artista que és. Obrigada!

E aos meus orientadores, sem palavras! Lisane e Adenauer, vocês foram excelentes. Com certeza recebi toda a orientação necessária. Nunca me deixaram na mão esperando respostas. Agradeço demais pela paciência, sei o quanto deve ter sido difícil essa orientação, mas conseguimos.

Enfim, obrigada a todos que de uma forma ou outra me ajudaram a tornar esse sonho realidade!!!

“Às vezes são as pessoas de que ninguém imagina nada, que fazem as coisas que ninguém sequer imagina."

Alan Turing

Resumo

RIBEIRO, Bruna G. Uma abordagem para modelagem de aplicações de computação móvel na nuvem. 2016, 54p. Trabalho acadêmico (Mestrado) - Mestrado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

Aplicações de computação móvel na nuvem (MCC, do inglês *Mobile Cloud Computing*) estão cada vez mais frequentes, servindo como uma estratégia para contornar os problemas relativos à limitação dos dispositivos móveis e transferindo boa parte do processamento e armazenamento para a nuvem. Muitas empresas vêm desenvolvendo plataformas de nuvem e oferecendo serviços e infraestruturas, acessíveis através de APIs específicas. Neste contexto, além dos diferentes sistemas operacionais e APIs, os desenvolvedores também devem se preocupar com detalhes para o uso das diferentes plataformas de nuvem, cada uma com sua infraestrutura e API específica. Devido a esta complexidade, modelos podem ser empregados para abstrair estes detalhes de implementação. Este trabalho propõe uma abordagem de modelagem para aplicações MCC baseada em diagramas UML e SoaML. A abordagem proposta visa construir modelos independentes de plataforma, reduzindo a complexidade do emprego da nuvem e facilitando a definição de uma ferramenta de geração de código multiplataforma. Além disso, este trabalho define também um padrão de projeto para aplicações do tipo CRUD, cujo objetivo é padronizar a modelagem deste tipo de aplicação, abstraindo protocolos específicos de cada plataforma e assim construindo um modelo independente de plataforma. Através de um estudo de caso, a abordagem proposta, bem como o padrão de projeto proposto, são demonstrados e discutidos.

Palavras Chave: Modelagem, Padrões de Projeto, Multiplatafomas, Computação Móvel na Nuvem, CRUD.

Abstract

RIBEIRO, Bruna G. An approach for modeling applications of mobile cloud computing. – 2016, 54p. Mestrado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

Applications of mobile cloud computing (MCC) are frequent, since these can overcome the problems relating to the limitation of mobile devices because much of the processing and storage is done in the cloud. Following the business model of offering cloud services, different companies have developed cloud platforms. In this context, in addition to different operating systems and APIs, developers must also worry about details for the use of different cloud platforms, each one with its infrastructure and specific API. Due to this complexity models can be used to abstract these implementation details. This work proposes a modeling approach for MCC applications based on UML and SoaML diagrams. The proposed approach aims to build platform independent models, reducing the complexity of the cloud usage and facilitating the definition of a cross-platform code generation tool. Additionally, this work also defines a design pattern for CRUD applications, which aims to standardize the modeling of this type of application, abstracting specific protocols for each platform and thus building a platform-independent model. Through a case study, the proposed approach and design pattern are demonstrated and discussed.

Keywords: Modeling, Design Patterns, Cross-platform, Mobile Cloud Computing, CRUD

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CC	*Cloud Computing*
CIM	*Computation Independent Model*
CRUD	*Create, Read, Update, Delete*
ESB	*Enterprise Service Bus*
GAE	*Google App Engine*
IaaS	*Infrastructure as a Service*
MCC	*Mobile Cloud Computing*
MDA	*Model-Driven Architecture*
MIMAC	*Modeling Interaction between Mobile Apps and Cloud*
MV	*Máquina Virtual*
OMG	*Object Management Group*
PaaS	*Platform as a Service*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
SaaS	*Software as a Service*
SDK	*Software Development Kit*
SoaML	*Service Oriented Architecture Modeling*
UML	*Unified Modeling Language*
WSDL	*Web Service Description Language*

Sumário

1 INTRODUÇÃO

Aplicativos móveis são *softwares* desenvolvidos para rodar em dispositivos portáteis, tais como tablets e smartphones. Esses aplicativos são desenvolvidos para apoiar diversas tarefas nas quais a mobilidade do usuário é um aspecto importante. O número e a diversidade dessas aplicações móveis cresce a medida em que as funcionalidades providas pelos dispositivos móveis conseguem substituir os computadores pessoais em diversos usos. O gráfico da Figura 1 ilustra o resultado de uma pesquisa realizada em 2014 pela Global Web Index (2014), a qual demonstra o crescimento no uso de dispositivos móveis ao redor do mundo.

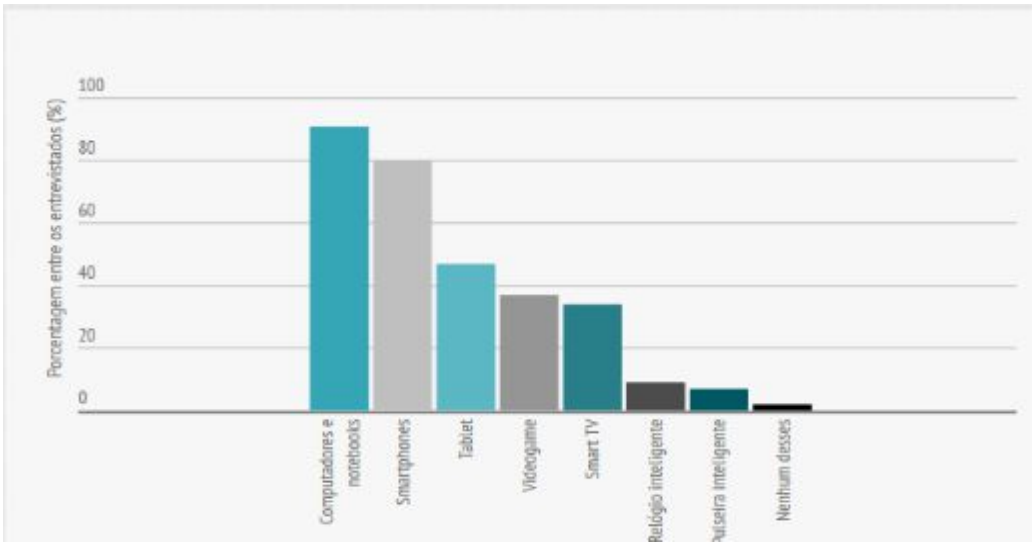

Figura 1 Pesquisa realizada pela Global Web Index
Fonte: (GLOBAL WEB INDEX, 2014)

No entanto, estes dispositivos ainda possuem poder de processamento e armazenamento limitados, o que restringe a complexidade das aplicações que estes podem suportar. A computação na nuvem (CC, do inglês *Cloud Computing*) surge como solução para estes desafios, pois pode fornecer infraestrutura necessária para a parte das aplicações móveis mais complexas ou que exijam mais recursos. Segundo NIST (*National Institute of Standards and Tchnology*) (2011), computação na nuvem é um modelo para habilitar o acesso a um conjunto compartilhada de recursos de computação, como redes, servidores, armazenamento, alicações e serviços, que possam ser providenciados e liberados com o mínimo de esforço de gerenciamento ou interação com o provedor.

Considerando este cenário, *Mobile Cloud Computing* (MCC) foi introduzido como uma integração da computação na nuvem com o ambiente móvel. A MCC pode reduzir os problemas relativos a limitações dos dispositivos, pois todo o processamento e armazenamento de dados é realizado na nuvem, garantindo mais segurança aos dados e diminuindo o consumo de energia dos dispositivos móveis, visto que o processamento é externo aos mesmos (DINH ET AL., 2013).

Muitas empresas vêm desenvolvendo plataformas de nuvem e oferecendo serviços e infraestruturas, acessíveis através de APIs específicas. No domínio de aplicativos móveis, os desenvolvedores já devem se preocupar com aspectos específicos da plataforma móvel alvo. Quando desenvolvendo aplicações de MCC, um nível adicional de complexidade é adicionado referente ao uso de uma plataforma de nuvem específica.

Porém, mesmo com múltiplas plataformas de desenvolvimento, onde cada uma possui infraestrutura e APIs específicas, pode-se observar que há um comportamento que se repete entre as implementações para as diferentes nuvens, principalmente no que se refere a interação do aplicativo com a nuvem. Sendo assim, padrões de projeto podem ser empregados como soluções reutilizáveis, de forma a uniformizar soluções.

Atualmente, UML (*Unified Modeling Language*) (OMG, 2015) é a linguagem de notação comumente empregada pela indústria de *software* para modelar diferentes visões do *software*, incluindo detalhes de projeto. A linguagem UML tem suportado diversas abordagens guiadas por modelos para projeto de *software*, (BRAMBILLA ET AL. 2012) dentre elas a MDA (*Model-driven Architecture*) (OMG, 2014). Em MDA, defende-se que modelos iniciais devem ser independentes de plataforma e que, a partir destes, podem ser derivados modelos dependentes de plataforma, os quais suportam a geração de código. Por não dar suporte a modelagem de aplicações orientadas a serviços, UML permite que extensões da linguagem sejam implementadas, dentre elas destaca-se o SoaML (*Service Oriented Architecture Modeling Language*) (OMG, 2012). Esta extensão tem como objetivo apoiar os diferentes cenários de modelagem de serviços, tais como descrição única de serviços, arquitetura orientada a serviços, ou a definição de contrato de serviços.

Diversas abordagens vêm sendo propostas para aplicações orientadas a serviços, porém, são definidas de forma muito abstrata, não empregando nenhuma linguagem de modelagem padrão. Muitas são definidas seguindo metodologias já conhecidas, como nos padrões Gamma et al. (1995), no entanto possuem um modelo

diferente, empregando notações que não são padronizadas. A definição de um padrão de projeto e seu emprego, combinado com a abstração de modelos e conceitos de MDA, pode facilitar o trabalho de projetistas e desenvolvedores de aplicativos de MCC, suportando a definição da aplicação independente de uma implementação. Da mesma forma que esta estratégia pode auxiliar no mapeamento de funcionalidades da aplicação, a mesma poderá também auxiliar na migração de aplicações existentes para a nuvem, ou na migração de uma dada plataforma de nuvem para outras.

Portanto, este trabalho tem como objetivo principal propor uma abordagem de modelagem para aplicações de MCC, a qual tem por finalidade auxiliar os projetistas a abstrair a complexidade da nuvem e modelar aplicações independente da plataforma de nuvem empregada. A abordagem foca em modelar o comportamento das aplicações, explicitando a interação entre o aplicativo que executa localmente e a infraestrutura da nuvem. Porém, a abordagem também utiliza diagramas estruturais, sendo assim possível modelar tanto a parte comportamental como a parte estrutural das aplicações. Desta forma, a abordagem pode ser integrada a ferramentas de geração automática de código e automatizar o desenvolvimento multiplataforma.

Para avaliar o emprego da abordagem proposta, foram utilizadas aplicações do tipo CRUD (*Create, Read, Update, Delete*), que incluem características encontradas em boa parte das aplicações de MCC (MANJUNATHA ET AL., 2010). Para auxiliar na modelagem dessas aplicações do tipo CRUD, um padrão de projeto foi definido, o qual representa o comportamento dessas aplicações independente da plataforma de nuvem adotada. Através de um estudo de caso, a abordagem e o padrão de projeto proposto são avaliados.

Esta dissertação está organizada em seis capítulos. No segundo capítulo, é apresentada a fundamentação teórica do trabalho, abordando conceitos de computação na nuvem, computação móvel na nuvem, MDA, SoaML e padrões de projeto. O terceiro capítulo discute os trabalhos relacionados que representam o estado da arte na área explorada. O quarto capítulo divide-se em duas seções, onde a primeira apresenta a visão geral da abordagem proposta e a segunda traz o emprego desta abordagem em aplicações do tipo CRUD. Os estudos de caso que demonstram a utilização da abordagem proposta são apresentados e discutidos no quinto capítulo. Por fim, o sexto capítulo conclui o trabalho e discute trabalhos futuros.

2 FUNDAMENTAÇÃO

Este capítulo apresenta os conceitos de Computação Móvel na Nuvem e faz uma revisão de linguagens e notações para modelagem de *software*, além de apresentar conceitos de MDA e Padrões de Projeto. Na seção 2.1 são apresentados os conceitos de Computação Móvel na Nuvem. Na seção 2.2 são revisados os conceitos de MDA, enquanto na seção 2.3 é revisada a SoaML, uma extensão da UML para arquiteturas orientadas a serviço e a seção 2.4 traz os conceitos de Padrões de Projeto.

2.1 Computação Móvel na Nuvem

O termo *Mobile Cloud Computing* (MCC) refere-se a uma infraestrutura onde a maior parte do processamento e armazenamento de dados acontece fora do dispositivo móvel. Neste caso, as aplicações móveis empregam o ambiente de nuvem para execução, ficando livres das limitações de *hardware* dos dispositivos móveis (DINH ET AL., 2013) (FERNANDO ET AL., 2013) (GUAN ET AL., 2011) (KHAN ET AL., 2013).

Aepona (2010) descreve MCC como um novo paradigma onde o processamento e o armazenamento de dados são movidos dos dispositivos móveis para uma plataforma de computação poderosa e localizada na nuvem. Alternativamente, MCC pode ser definida como uma combinação de web móvel e computação na nuvem.

As arquiteturas de MCC permitem que os dispositivos móveis acessem os serviços na nuvem de duas maneiras: através da rede provida pelas operadoras de telefonia (3G, 4G, etc.) ou através de pontos de acesso Wi-Fi conforme é ilustrado na Figura 2.

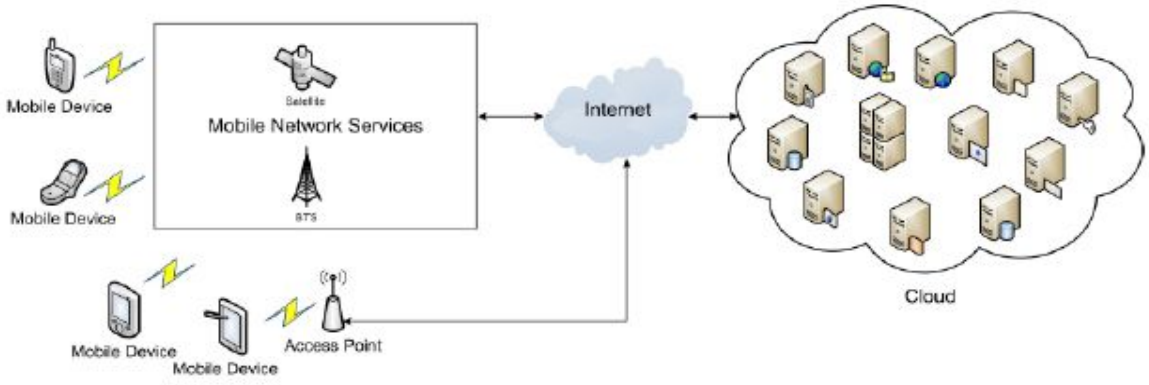

Figura 2 Arquitetura Mobile Cloud
Fonte: Khan et al. (2013)

Ao utilizar a rede de telecomunicações providas pelas operadoras, o celular ou dispositivo móvel é conectado via uma estação base. No caso dos pontos de acesso, os usuários móveis se conectam através de uma rede Wi-Fi. Em ambos os modos, é possível o acesso à Internet e a utilização dos serviços na nuvem, porém a preferência dos usuários é pela rede Wi-Fi, pois ainda fornece conexões de baixa latência e menor consumo de energia dos aparelhos quando comparado a redes 3G ou 4G, por exemplo (KHAN ET AL., 2013).

A utilização de MCC pode ajudar a superar problemas conhecidos da computação móvel (mobilidade, comunicação e portabilidade) (DINH ET AL., 2013) (FERNANDO ET AL., 2013) (KHAN ET AL., 2013), apresentando as seguintes vantagens:

1. Extensão da duração da bateria: a bateria é uma das questões chave para dispositivos móveis e diversas soluções foram apresentadas para melhorar a performance da CPU e gerenciamento do disco e tela de maneira inteligente para reduzir o consumo de energia. A computação na perspectiva da MCC minimiza o impacto computacional decorrente da execução das aplicações nos dispositivos móveis, contribuindo assim para a redução do consumo de energia.

2. Melhoria da capacidade de armazenamento e processamento de dados: capacidade de armazenamento é uma das restrições de dispositivos móveis e os usuários costumam manter muitos arquivos de fotos e vídeos nos mesmos. A MCC permite que estes arquivos sejam processados e mantidos na nuvem logo após a sua captura, economizando espaço de armazenamento nos

dispositivos móveis do usuário. Além disso, a MCC ajuda também na redução do tempo de execução de aplicações que demandam maior poder de processamento, pois o processamento, uma vez que ultrapassa o mesmo, ocorre em servidores potentes localizados na nuvem.

3. Melhoria da confiabilidade: a MCC melhora a confiabilidade pois existem cópias de segurança de dados e aplicações em computadores na nuvem reduzindo o risco de perda destes dados e aplicações. Caso estes fossem mantidos só no aparelho móvel, o risco de perdas seria aumentado devido a danificações (quedas e outros acidentes), perda ou mesmo furto do dispositivo.

2.2 Plataformas e Provedores

A computação na nuvem possui 3 modelos de serviço: *software* como serviço (SaaS), plataforma como serviço (PaaS) e infraestrutura como serviço (IaaS), onde *software* como serviço é a camada composta por aplicações que são executadas no ambiente da nuvem, plataforma como serviço é a camada que oferece serviços como sistemas operacionais, banco de dados entre outros e a infraestrutura como serviço é a camada que fornece os recursos como servidores, armazenamento entre outros (DINH ET AL., 2013) (FERNANDO ET AL., 2013).

Embora as arquiteturas de nuvem em geral sigam um modelo comum de disponibilidade de serviços, cada provedor de serviço na nuvem possui características que variam desde a disponibilidade dos serviços até as linguagens de programação suportadas.

Plataformas para computação na nuvem são muito mais do que ambientes para disponibilização de aplicações ou armazenamento de dados. Tecnologias de alto desempenho estão disponíveis, as quais visam prover serviços de infraestrutura e promover o armazenamento de dados. O Windows Azure (MICROSOFT, 2011), Google APP Engine (GOOGLE DEVELOPERS, 2014), Amazon EC2 (AMAZON, 2010) e IBM (IBM, 2014) são exemplos de provedores de serviço de nuvem. Cada um destes provedores suporta linguagens e por vezes formas distintas de serviços.

Um exemplo neste sentido é o Windows Azure, uma plataforma em nuvem da Microsoft que permite aos seus usuários criar, implantar e gerenciar aplicativos utilizando um ambiente de desenvolvimento em múltiplas linguagens. Windows Azure oferece uma plataforma como serviço (PaaS) a partir de *datacenters* espalhados pelo mundo. A plataforma oferece também uma série de serviços que capacitam as

aplicações com banco de dados, barramentos de serviços, mecanismos de controle de acesso, suporte ao modelo de *software* como serviço (SaaS), entre outros.

Outro exemplo é o Amazon EC2, um serviço que fornece uma capacidade de computação redimensionável na nuvem. O Amazon EC2 permite que os usuários “aluguem” computadores virtuais, nos quais rodam suas próprias aplicações. O EC2 permite a implantação de aplicações escaláveis ao prover um serviço web através do qual um usuário pode criar uma máquina virtual, a qual a Amazon chama de “instância”, destinada a conter o *software* de seu interesse. Um usuário pode criar, lançar e terminar instâncias do servidor, conforme necessário, pagando por hora pelos servidores ativos, daí o termo "elástico" (AMAZON, 2010).

As seções 2.2.1 e 2.2.2 detalham as plataformas da IBM e do Google, respectivamente, as quais são utilizadas nos estudos de caso apresentados neste trabalho, já que são plataformas que, além de disponibilizarem material, são utilizadas pelo grupo de pesquisa.

2.2.1 Plataforma de Nuvem IBM

IBM Cloud (IBM, 2014) tem aplicações de negócios, serviços de desenvolvimento e infraestrutura que auxiliam no desenvolvimento de aplicações de computação na nuvem, oferecendo suporte também para soluções SaaS, IaaS e PaaS. A IBM trabalha com recursos elásticos onde pode-se aumentar ou reduzir a capacidade de forma rápida e fácil para atender a demanda. Assim como outras plataformas, a IBM cobra apenas o que for usado.

Bluemix é uma plataforma da IBM direcionada a computação em nuvem e baseada em padrões abertos, a qual foi desenvolvida para a construção, gerenciamento e execução de aplicações de todos os tipos. Ela é baseada em Cloud Foundry (IBM, 2014), plataforma como serviço (PaaS) de código aberto que permite criar e implementar aplicativos rapidamente na nuvem. A Figura 3 ilustra a arquitetura do Bluemix.

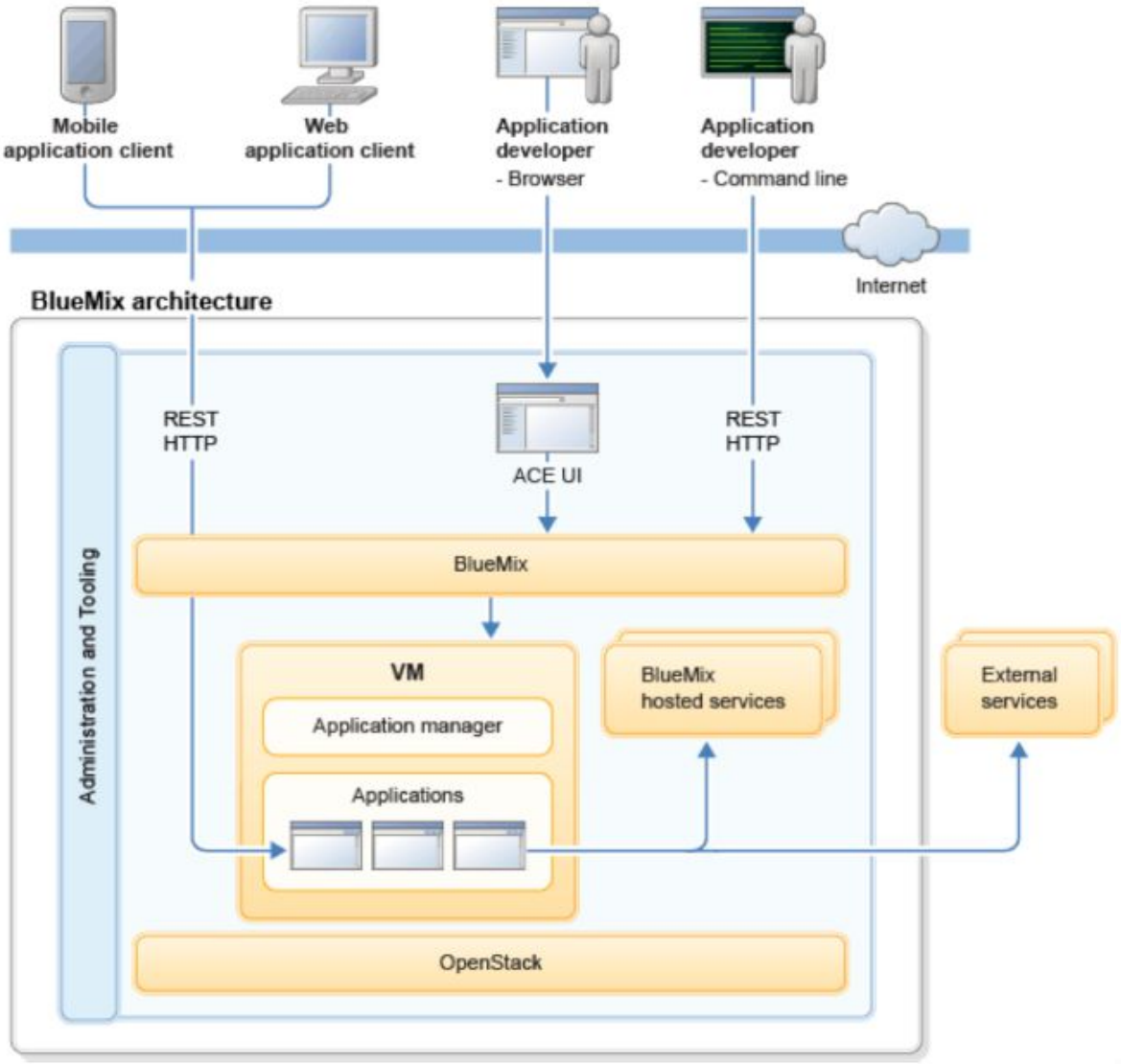

Figura 3 Arquitetura Bluemix
Fonte: IBM (2014)

Ao criar um aplicativo e implementá-lo no Bluemix, o ambiente determina uma máquina virtual (MV) apropriada para a qual o aplicativo é enviado. Depois que a MV é escolhida, um gerente de aplicativos em cada MV instala a estrutura e o *runtime* adequados para o aplicativo. Em seguida, o aplicativo pode ser implementado nessa estrutura. Para fazer uso dos serviços da plataforma da IBM, é necessária a implementação de métodos que realizam a inicialização dos serviços. Esses métodos possuem parâmetros que são necessários para que os serviços de nuvem sejam utilizados.

2.2.2 Plataforma de Nuvem Google App Engine

Google App Engine (GAE) é uma plataforma como uma oferta de serviço (PaaS), que lhe permite construir e executar aplicativos na infraestrutura computacional do Google. Esta plataforma oferece suporte a linguagens abertas de mercado, como o Java e o Phyton, modelos de aplicação, APIs para integração aos recursos do Google e fornecimento de um ambiente com ajustes e balanceamentos de carga automáticos (NETO; FREITAS, 2011).

Para desenvolvedores, a Google App Engine disponibiliza SDKs (*Software Development Kit*), que são pacotes de desenvolvimento de *software* que incluem todas as APIs e bibliotecas disponíveis e ferramentas de implantação que permitem fazer upload do seu aplicativo para a nuvem através da ferramenta de linha de comando APPcfg.py (fornecida pelo SDK) ou através da interface gráfica, clicando no botão *deploy* (GOOGLE DEVELOPERS, 2014).

A plataforma faz uso de bancos de dados, entre eles o Firebase, o qual é uma plataforma para construção de aplicativos que oferece armazenamento de dados em tempo real e serviços de sincronização, autenticação, entre outros.

Pode-se visualizar o funcionamento da arquitetura do GAE na Figura 4, onde a aplicação envia os dados para o firebase, o qual é o banco de dados que o GAE utiliza nessa aplicação, e o firebase por sua vez, conversa com o app engine para fazer algumas tarefas, podendo ser processamento de informação, entre outros. Ao contrário da plataforma da IBM, nesta plataforma não se faz necessária a invocação de métodos para a inicialização de serviços.

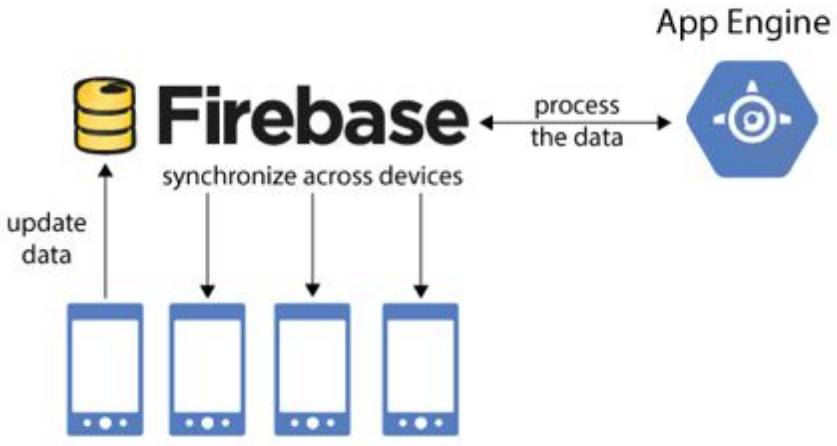

Figura 4 Arquitetura Google App Engine
Fonte: Google Developers (2014)

2.3 MDA

A arquitetura MDA (do inglês *Model Driven Architeture*) (OMG, 2014) proposta pela OMG (*Object Management Group*) reconhece a importância de ter modelos durante todo o processo de desenvolvimento, tornando-os o ponto chave. A arquitetura define que o processo de desenvolvimento de *software* deve ser direcionado para a modelagem do sistema, independente de qualquer plataforma ou implementação, e através de transformações realizadas sobre esse modelo conceitual, novos modelos, com níveis de abstração cada vez mais específicos e ligados à implementação sejam produzidos, de forma que o sistema final seja gerado automaticamente, a partir da especificação definida no modelo conceitual (SOUZA, 2015).

A arquitetura MDA está ilustrada na Figura 5 e compreende alguns conceitos e passos que são descritos a seguir. O primeiro passo é a geração de modelo CIM (*Computation Independent Model*), o qual é uma visão do sistema independente de computação. O CIM representa apenas requisitos do sistema e não inclui detalhes de sua estrutura. O segundo passo é uma geração, a partir do CIM, do modelo PIM (*Platform Independent Model*), que é definido com um alto nível de abstração, independente de qualquer tecnologia. Descreve o sistema de *software* de uma perspectiva que melhor represente o que está sendo modelado.

A geração de um ou mais modelos específicos de plataforma, PSM (*Platform Specific Models*) é o terceiro passo, onde cada PSM é gerado levando em conta detalhes específicos de uma determinada tecnologia a ser utilizada na implementação. Para cada PIM, vários PSMs podem ser gerados, e o quarto passo é a geração de código a partir de cada PSM. Esta geração não constrói apenas estruturas básicas e *templates*. O código gerado no MDA deve ser o mais próximo possível da solução de *software* definitiva.

Figura 5 Arquitetura MDA
Fonte: Souza (2015)

Segundo OMG (2014), os principais benefícios que MDA oferece são:

- Portabilidade: uma vez que o PIM é, por definição, independente de plataforma, o mesmo PIM pode ser transformado automaticamente para vários PSMs, específicos para diferentes plataformas.
- Interoperabilidade: a interoperabilidade em MDA é alcançada através de *bridges*, que realizam a comunicação entre PSMs e códigos de diferentes plataformas.
- Manutenção e Documentação: todo trabalho de manutenção não é mais feito no código, mas sim no nível mais alto (o modelo PIM). Desta maneira, a manutenção é facilitada e a documentação constantemente atualizada.

2.4 SoaML

SoaML (*Service Oriented Architecture Modeling Language*) é uma extensão da UML2 para suportar um serviço de modelagem explícita em ambientes distribuídos. Esta extensão tem como objetivo apoiar os diferentes cenários de modelagem de serviços, tais como descrição única de serviços, arquitetura orientada a serviços, ou a definição de contrato de serviços (OMG, 2012).

A UML é uma linguagem de modelagem de uso geral para visualizar, especificar, construir e documentar os artefatos de sistemas intensivos de *software* (OMG, 2015). Um perfil UML customiza a linguagem para um domínio ou objetivo

específico, usando mecanismos de extensão, tais como estereótipos e metaclasses. A Figura 6 ilustra os principais estereótipos definidos no perfil UML para SoaML, por exemplo, o estereótipo <<ServiceInterface>> estende a metaclasse Class da UML.

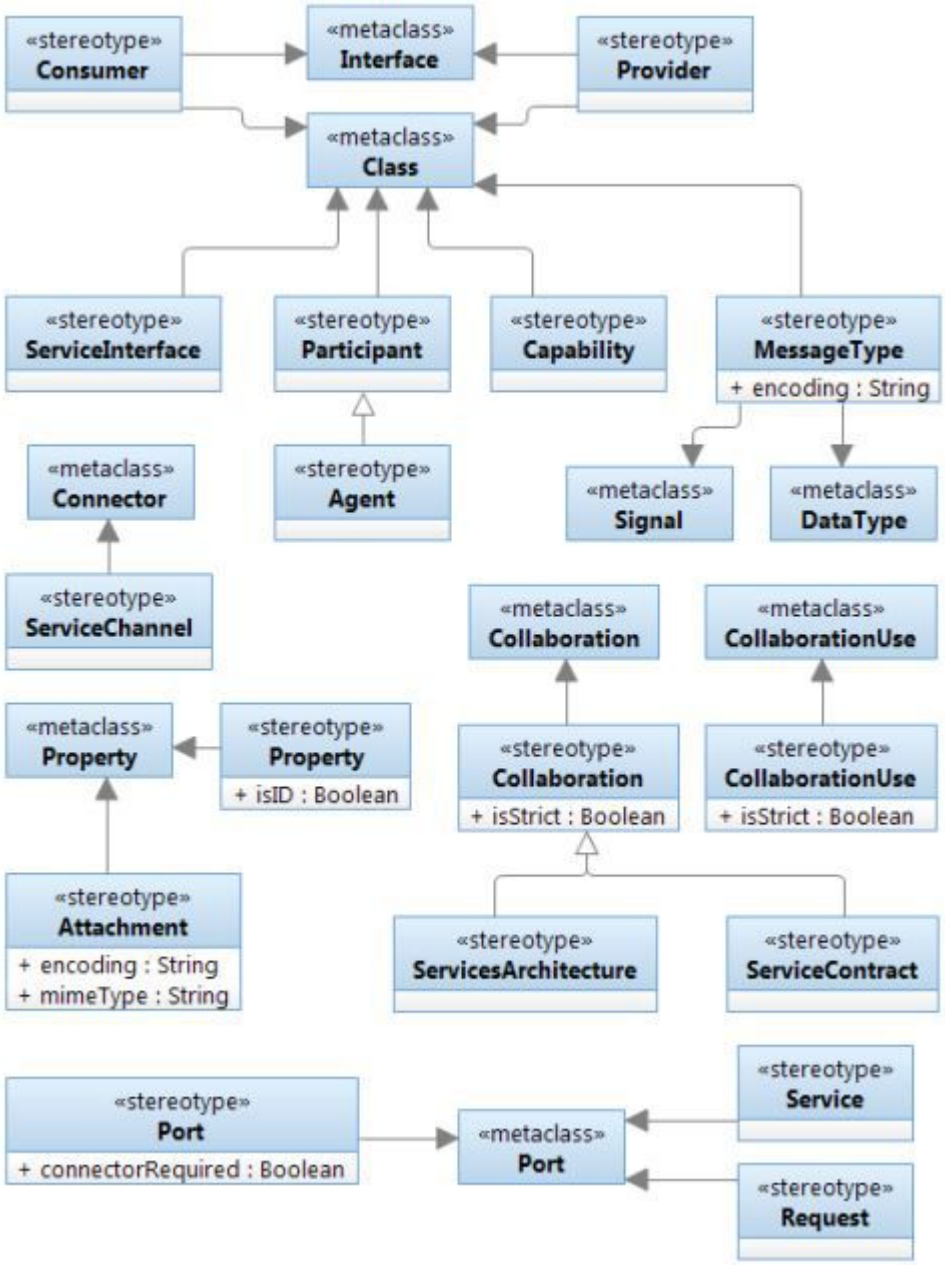

Figura 6 Principais Estereótipos SoaML
Fonte: OMG (2012)

SoaML estende UML em seis áreas principais: Participantes, Interfaces de serviços, Contratos de serviços, Arquiteturas de serviços, Dados de serviço e Capacidades. Todos estes conceitos representam a base das classes definidas pela SoaML e são revisados nesta seção (OMG, 2012).

Participants (participantes) são usados para definir os prestadores de serviços e consumidores em um sistema. Um participante pode desempenhar o papel de prestador de serviços, ou consumidor ou ambos. Quando um participante atua como um provedor contém portas de serviço, e quando um participante atua como um consumidor este contém portas da solicitação.

Service interfaces (interface de serviços) são usadas para descrever as operações previstas e necessárias para concluir a funcionalidade de um serviço. Uma interface de serviço pode ser utilizada como o protocolo para uma porta de serviço ou de uma porta de pedidos.

Service contracts (contrato de serviço) são usados para descrever padrões de interação entre as entidades de serviços. Um contrato de serviço é usado para modelar um acordo entre duas ou mais partes. Cada papel de serviço em um contrato de serviço tem uma interface que geralmente representa um provedor ou um consumidor.

Services architectures (arquiteturas de serviço) são utilizadas para definir como um conjunto de participantes trabalha em conjunto com algum propósito, fornecendo a utilização de serviços. Os serviços são expressos em contratos de serviços providos em uma arquitetura de serviços.

Service data (serviço de dados) são usados para descrever as mensagens de serviço e anexos de mensagens. O tipo de mensagem é usado para especificar as informações trocadas entre os consumidores e os prestadores de serviços.

Capabilities (capacidades) representam uma abstração da capacidade de afetar a mudança. Capacidades de identificar ou especificar um conjunto coeso de funções ou recursos que um serviço prestado por um ou mais participantes podem oferecer.

O acesso ao serviço é feito através de um contrato que define as políticas e restrições que devem ser obedecidas. Um serviço é fornecido por um participante que atua como o provedor para ser utilizado por outros participantes ditos consumidores.

Participantes oferecem serviços através de portas via o estereótipo <<Service>> e requisitam serviços através de portas com o estereótipo <<Request>>. A Figura 7 mostra o participante "*Productions*" fornecendo o serviço "*scheduling*". A porta de serviço é a interface UML "*Scheduling*" que tem duas operações: "*requestProductionScheduling*" e "*sendShippingSchedule*" (OMG, 2012).

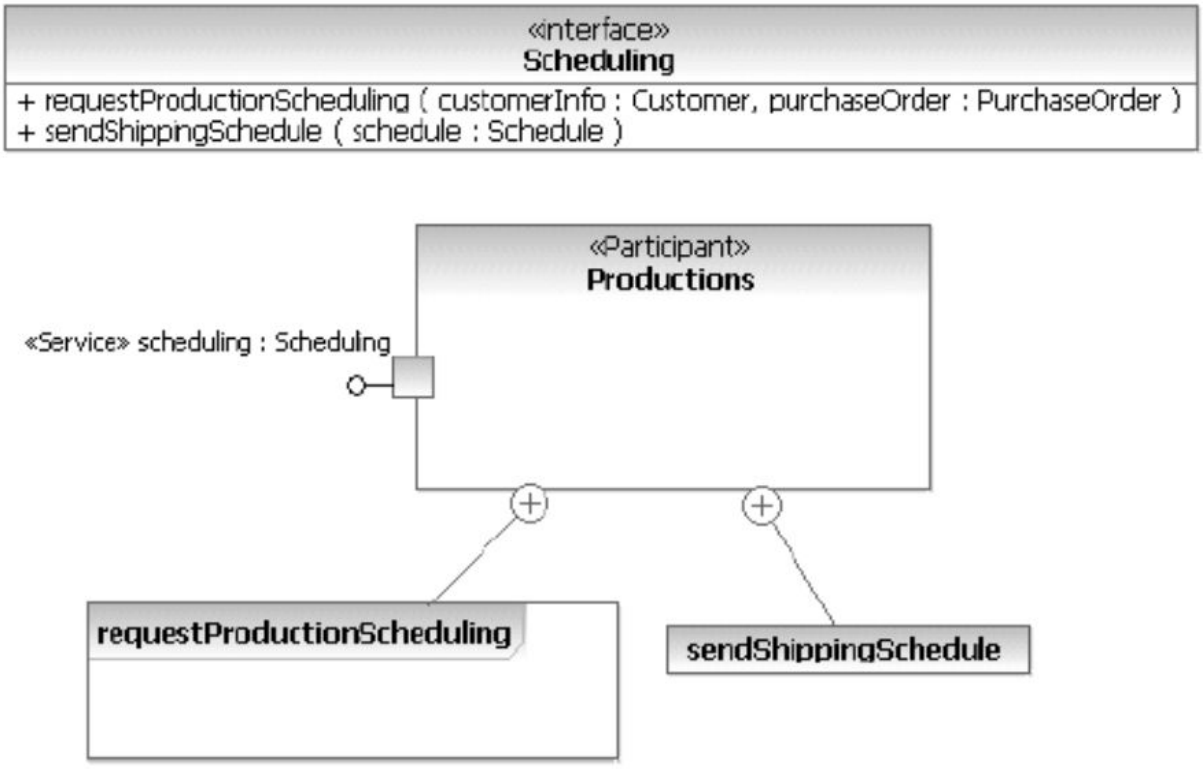

Figura 7 Diagrama de Interface de Serviço
Fonte: OMG (2012)

A porta de serviço tem um tipo que descreve como usar esse serviço. Esse tipo pode ser uma interface UML (para serviços muito simples – *Simple Interface*) ou uma *Service Interface*. Em ambos os casos o tipo da porta de serviço especifica tudo que é necessário para interagir com esse serviço, ou seja, representa o contrato entre os provedores e consumidores desse serviço (BRAGA, 2011).

O diagrama de Participantes define os prestadores de serviços e consumidores em um sistema. Participantes proveem capacidades e consomem serviços através de portas de serviços devidamente identificadas pelos estereótipos *ServicePoints* e *RequestPoints*, que tem seus tipos definidos pelas Interfaces de Serviços, ou interfaces UML. Um serviço usa o conceito UML de porta e indica o ponto de interação onde um participante interage com outros participantes.

O diagrama de Interface de Serviços descreve as operações previstas e necessárias para concluir uma funcionalidade de um serviço. São as formas para especificar como um participante interage como provedor ou consumidor. Uma Interface de Serviço é modelada como uma classe UML.

Uma Interface de Serviço especifica as seguintes informações:

- Nome do serviço indicando o que ele faz ou sobre o que é;

- Capacidades do serviço através das interfaces;
- Necessidades ou capacidades exigidas pelo consumidor para utilizar o serviço através das interfaces;
- Uma especificação detalhada de cada capacidade usando uma operação, incluindo seu nome, pré-condições, pós-condições, entradas e saídas e qualquer exceção que possa surgir;
- Qualquer protocolo ou regras para o uso de capacidade ou como se espera que consumidores respondam e quando através de um comportamento próprio;
- Regras para como os serviços devem ser implementados pelos provedores;
- Restrições que podem determinar se o serviço atingiu com sucesso o seu propósito.

2.5 Padrões de Projeto

Padrões de projeto expressam uma solução reutilizável para um problema que ocorre com frequência. Gamma et al. (1995) afirma que um padrão possui quatro elementos essenciais:

- Nome do padrão: é uma referência usada para descrever um problema de projeto, suas soluções e consequências.
- Problema: descreve em que situação o padrão pode ser aplicado, explicando o problema e seu contexto. Algumas vezes o problema incluirá uma lista de condições que devem ser satisfeitas para que faça sentido aplicar o padrão.
- Solução: descreve os elementos que compõem o padrão de projeto, seus relacionamentos, suas responsabilidades e colaborações. A solução não descreve um projeto concreto ou uma implementação em particular porque um padrão é como um gabarito que pode ser aplicado em muitas situações diferentes. Em vez disso, o padrão fornece uma descrição abstrata de um problema de projeto e de como um arranjo geral de elementos o resolve.
- Consequências: são resultados e análises das vantagens e desvantagens da aplicação do padrão. As consequências para o *software* frequentemente envolvem balanceamento entre espaço e tempo. Elas também podem abordar aspectos sobre linguagens e implementação.

Conforme propõe Gamma et al. (1995), os padrões de projeto são classificados por dois critérios: propósito e escopo. O primeiro divide os padrões em três grupos, criação, estrutural e comportamental. Enquanto que o segundo especifica se o padrão

se aplica a classes ou objetos. A Tabela 1 demonstra essa divisão proposta por Gamma et al. (1995).

Tabela 1 Divisão dos Padrões de Projeto
Fonte: Gamma et al. (1995)

		Propósito		
		De criação	Estrutural	Comportamental
Escopo	Classe	Factory Method	Adapter (Class)	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter (Object) Bridge Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strate gy Visitor

Segundo Brizeno (2011) os padrões de criação têm como intenção principal abstrair o processo de criação de objetos. Desta maneira, o sistema não precisa se preocupar com questões sobre, como o objeto é criado, como é composto, qual a sua representação real. Se ocorrer alguma mudança na instanciação do objeto, o sistema em geral não será afetado. Isso é a famosa flexibilidade que os padrões de projeto buscam prover.

Padrões de criação, voltados a classes, repassam alguns passos da criação dos objetos para suas subclasses, como por exemplo o padrão de projeto Factory Method, que pode criar várias subclasses para criar o produto, enquanto os padrões criacionais, voltados a objetos, delegam esse processo de criação para um outro objeto, como por exemplo o padrão Prototype.

Os padrões estruturais se preocupam em como as classes e objetos são compostos, ou seja, como é sua estrutura. Esses padrões têm por objetivo facilitar o design do sistema, identificando maneiras de realizar o relacionamento entre entidades, deixando o desenvolvedor livre dessa preocupação.

Segundo Gamma et al. (1995), os padrões estruturais podem ser aplicados a classes ou objetos. Os padrões estruturais de objetos demonstram maneiras de complementar objetos para adquirir novas funcionalidades, como por exemplo o padrão de projeto Adapter, que define uma nova interface para adaptar uma interface

a outras. Já os padrões estruturais de classe usam herança para definir implementações ou interfaces, como por exemplo, o padrão Composite.

Os padrões de projeto comportamentais facilitam a comunicação entre objetos. Conforme Gamma et al. (1995) os padrões comportamentais descrevem a interação e atribuição de responsabilidades entre objetos e classes. Para determinar o comportamento entre classes, estes padrões utilizam herança, como por exemplo o padrão Template Method, que fornece um algoritmo padrão e deixa as subclasses definirem alguns pontos da execução para este algoritmo. Para comportamento entre objetos, estes utilizam composição de objetos, como por exemplo no padrão Mediator, que define um objeto que realiza a comunicação muitos para muitos (BRIZENO, 2011).

Padrões de projeto formam um vocabulário comum que permitem uma melhor comunicação entre os desenvolvedores, uma documentação mais completa e uma melhor exploração das alternativas de projeto. Reduz a complexidade do sistema através da definição de abstrações que estão acima das classes e instâncias.

Segundo Thomazini (2006), padrões de projeto constituem uma base de experiências reutilizáveis para a construção de *software*. Eles funcionam como peças na construção de projetos de *software* mais complexos; podem ser considerados como microarquiteturas que contribuem para arquitetura geral do sistema. Além disso, os padrões de projeto reduzem o tempo de aprendizado de uma determinada biblioteca de classes.

3 TRABALHOS RELACIONADOS

O projeto e a modelagem de aplicativos MCC tem sido investigado por diferentes grupos de pesquisa, alguns destes enfocam o emprego de linguagens de modelagem, enquanto outros focam na discussão de padrões de projeto dedicados para estas aplicações. Neste capítulo, a seção 3.1 revisa os trabalhos relacionados à modelagem de aplicações móveis e a seção 3.2 os trabalhos relacionados a padrões de projeto para aplicações móveis e/ou nuvem.

3.1 Modelagem de Aplicações Móveis Envolvendo a Nuvem

A linguagem SoaML vem sendo investigada para emprego na modelagem de soluções baseadas em serviços (ELVESÆTER ET AL., 2011) (DIIRR ET AL., 2013). Elvesæter et al. (2011) realizaram um estudo preliminar de como usar SoaML para modelagem de serviços na nuvem, onde apontam a limitação do SoaML no que se refere à modelagem do comportamento. Considerando esta limitação, neste trabalho, SoaML é combinado com UML para assim, obter uma modelagem mais completa da aplicação, facilitando a modelagem e a abstração da nuvem.

Os autores observaram também que SoaML poderia auxiliar na migração de sistemas existentes para uma plataforma de nuvem. Para este propósito, uma extensão de SoaML é proposta para auxiliar nessa migração. Porém, não há mais relatos dos autores sobre essa possível extensão para auxiliar na migração. Quando se fala em plataformas de nuvem, associa-se também o fato de uma possível migração. Por esse motivo, ter uma abordagem que possa ser utilizada em múltiplas plataformas facilita o desenvolvimento das aplicações, já que pode-se ter uma modelagem da aplicação que possa ser utilizada independente da plataforma adotada. Por esse motivo, a abordagem proposta neste trabalho suporta essa modelagem independente de plataforma, facilitando uma possível migração de plataformas.

Em Diirr et al. (2013) o emprego de SoaML na especificação de informações de serviços, seus provedores e consumidores tem sido proposto. Neste trabalho, o modelo produzido é usado na geração de código para os serviços, gerando descrições WSDL (*Web Service Description Language*). WSDL é uma linguagem de descrição de serviços que permite descrever os serviços e as trocas de mensagens. Através da descrição o provedor de serviços publica as especificações necessárias para o cliente invocar o serviço (W3C, 2001).

Para isso, eles utilizaram o *software* Modelio Case Tool Enterprise Edition para a modelagem da aplicação e geração automática do código de serviço. O trabalho é uma apresentação do perfil SoaML e uma demonstração do mesmo sendo utilizado para geração de código de serviço, o que não ajudaria na modelagem do comportamento das aplicações. A abordagem proposta neste trabalho não permite uma geração de código automática através da modelagem, porém auxilia no entendimento da aplicação para futuramente automatizar esse processo, visando uma geração de código da aplicação através da modelagem, e não de código de serviço.

Parada, A. et al. (2015) propõe uma abordagem MDA para modelagem de aplicações móveis com foco em geração automática de código a partir de modelos. A abordagem proposta é integrada a uma ferramenta de geração de código que suporta a geração de código para as plataformas Android e WindowsPhone. No entanto, esta abordagem foca em aplicações móveis em geral, não prevendo nenhuma notação para especificar uso de serviços.

3.2 Padrões de Projeto para Aplicações Móveis e/ou Nuvem

Erl (2007) define que um padrão de projeto SOA fornece uma solução para problemas que normalmente surgem quando um serviço é construído. Nessa área de SOA, encontram-se diversos padrões de projeto, os quais são divididos em cinco grupos: Padrões de serviço, padrões de inventário de serviços, padrões de composições de serviços, padrões ESB (*Enterprive Service Bus*) e padrões de orquestração. Porém todos os padrões de projeto para SOA tem como foco o serviço e não o uso do serviço. A abordagem proposta neste trabalho visa definir um padrão de projeto com foco direcionado a utilização do serviço, o qual tem por objetivo modelagem de aplicações desenvolvidas baseadas em serviços existentes.

Adewojo, A. et al. (2014) realizaram uma pesquisa, na qual constataram que padrões de projeto podem auxiliar na migração de sistemas existentes para a nuvem. Porém, o artigo não especifica como fazer essa migração de forma mais simples através de padrões. O padrão proposto neste trabalho tem por objetivo ser independente de plataforma, o que auxiliaria um processo de migração, permitindo o reuso da modelagem, sem ter que construí-la novamente quando a plataforma de nuvem for trocada.

Assim como existem padrões de projeto para *software*, padrões para SOA, existem também alguns padrões de projeto para nuvem. Alguns padrões inclusive são

específicos para uma determinada arquitetura de nuvem. Como por exemplo a arquitetura de nuvem AWS, que possui padrões de projeto que propõem um conjunto de soluções para o uso de tecnologia de nuvem AWS (ARCITURA™, 2015).

No entanto, esses diversos padrões que vem sendo propostos para SOA são definidos de forma muito abstrata, não empregando nenhuma linguagem de modelagem padrão. Embora, estes sejam definidos seguindo a metodologia dos padrões de Gamma et al. (1995), como nome do padrão, problema, solução, porém cada padrão possui um modelo diferente, empregando notações que não são padronizadas, nem mesmo pelos padrões propostos pelo mesmo grupo ou organização (ARCITURA™, 2015). A abordagem proposta neste trabalho emprega linguagens de modelagem padrão, facilitando o uso do padrão de projeto proposto e simplificando a modelagem da aplicação, por utilizar diagramas e definições já conhecidas à quem trabalha na área de desenvolvimento de *software*.

Embora seja possível encontrar guias de boas práticas para desenvolvimento de aplicações para dispositivos móveis, não há disponível na literatura uma coleção de padrões de projeto para este domínio. Em (MCCORMICK; SCHMIDT, 2012), são definidos de padrões de projeto para aplicações móveis que envolvam sincronização de dados entre dispositivo e um sistema remoto (serviço de nuvem, por exemplo). Embora este trabalho seja um esforço interessante, pois discute diferentes formas e mecanismos para sincronização de dados, os modelos propostos não explicitam a interação com o serviço na nuvem.

4 ABORDAGEM PROPOSTA

Neste capítulo, na seção 4.1 será apresentada a visão geral da abordagem proposta para aplicações de computação móvel na nuvem. Na seção 4.2, o emprego desta abordagem é discutido, considerando aplicações do tipo CRUD.

4.1 MIMAC: Visão Geral

A abordagem proposta, denominada MIMAC (*Modeling Interaction between Mobile Apps and Cloud*) foca em aplicações de computação móvel na nuvem, e tem por objetivo auxiliar os projetistas para abstrair a complexidade da nuvem, facilitando o desenvolvimento dessas aplicações através do emprego de modelos. A MIMAC baseia-se na arquitetura MDA da OMG, propondo modelos que podem ser traduzidos para outros modelos mais detalhados, até que uma implementação da aplicação possa ser alcançada. A abordagem MIMAC define diagramas da UML e SoaML para modelar as aplicações MCC, auxiliando no desenvolvimento das mesmas.

O objetivo principal desta abordagem é auxiliar projetistas na modelagem das aplicações alvo, definindo modelos que possam ser utilizados mais vezes, e que utilizem um padrão na modelagem, facilitando no entendimento e na definição dos serviços das aplicações. Por ser baseada na arquitetura MDA, é possível realizar a modelagem de aplicações em dois níveis, tanto direcionada a uma plataforma de nuvem específica (PSM) como independente de plataforma (PIM).

A abordagem foca em modelar o comportamento das aplicações, porém também utiliza diagramas estruturais para representar aspectos estruturais de projeto. O modelo da aplicação proposto na abordagem deve auxiliar os projetistas tanto a entender melhor a aplicação, detalhando a arquitetura e comportamento, bem como pode ser empregado para fins de geração de código.

A Figura 8 ilustra a abordagem MIMAC, detalhando os diagramas empregados e definindo em que camada estes modelos se encontram. Como usual em várias metodologias, o diagrama de casos de uso da UML será empregado para definir as funcionalidades da aplicação. Este diagrama servirá de base para a construção do modelo PIM, o qual é independente de plataforma e tecnologias. Para alcançar uma implementação, no entanto, este modelo deverá ser transformado em um modelo dependente de plataforma, o PSM.

Na camada PIM, encontra-se o modelo independente de plataformas e tecnologias, onde serão utilizados o diagrama de participantes do SoaML para definir quem fornece o serviço de nuvem e quem o utiliza; o diagrama de classes da UML, para definir os métodos e atributos da aplicação; e, por fim, o diagrama de sequência da UML, o qual é utilizado para modelar a parte comportamental da aplicação, definindo a ordem temporal em que as mensagens são trocadas entre os participantes. Todos estes aspectos devem ser descritos nesta camada, independentes da plataforma de nuvem que será empregada.

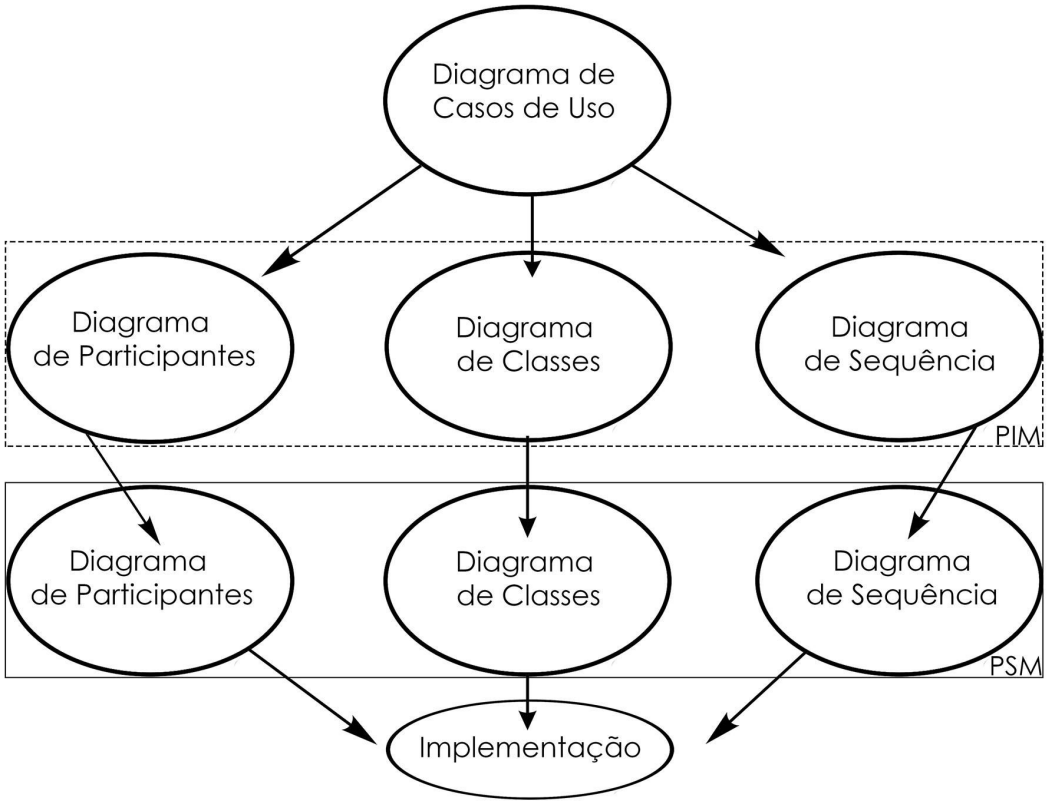

Figura 8 Abordagem MIMAC: Diagramas Empregados

A segunda camada é a camada PSM, já voltada para uma plataforma ou tecnologia específica. Nesta camada também emprega-se: um diagrama de participantes do SoaML para definir os participantes da aplicação e qual participante provê o serviço e qual requer o serviço; um diagrama de classes da UML; e, por fim, o diagrama de sequência da UML, o qual será utilizado para modelar a ordem temporal da troca de mensagens entre os participantes, utilizando métodos específicos e protocolo da plataforma de nuvem a ser empregada.

Nesta camada, as características específicas das plataformas devem ser explicitadas. Algumas plataformas, por exemplo, requerem que o serviço seja inicializado antes de seu uso. Desta forma, esta interação de inicialização também deve ser descrita para que a implementação adequada possa ser obtida. Por esse motivo, dependendo da plataforma e do protocolo empregado por esta, o número de diagramas de sequência a serem definidos pode variar.

Esses passos podem ser seguidos na construção de uma aplicação, onde modelos podem ser construídos para definir as funcionalidades da aplicação, podendo utilizar tanto a camada PIM como a camada PSM, dependendo do que se tem para o desenvolvimento da aplicação. Além disso, esses mesmos passos podem ser utilizados em uma engenharia reversa, onde já se tem o código da aplicação, e assim se pode aplicar direto a camada PSM.

4.2 Emprego da Abordagem MIMAC em Aplicações do Tipo CRUD

Para avaliar o emprego da abordagem proposta, foram utilizadas aplicações do tipo CRUD (*Create, Read, Update, Delete*), cujas características são encontradas em boa parte das aplicações de MCC. Essas características se repetem entre as diversas plataformas de nuvem, o que facilita a definição de um padrão de projeto, o qual poderá ser especificado e utilizado sempre que a aplicação possuir essas características.

O objetivo deste padrão de projeto é padronizar a modelagem de aplicações do tipo CRUD, abstraindo as especificações de uso de cada plataforma e permitindo modelar o comportamento das aplicações independente da plataforma de nuvem adotada e assim construindo um modelo PSM. A Figura 9 ilustra a definição deste padrão de projeto, onde são utilizados o diagrama de participantes do SoaML e quatro diagramas de sequência da UML, um para cada operação do CRUD.

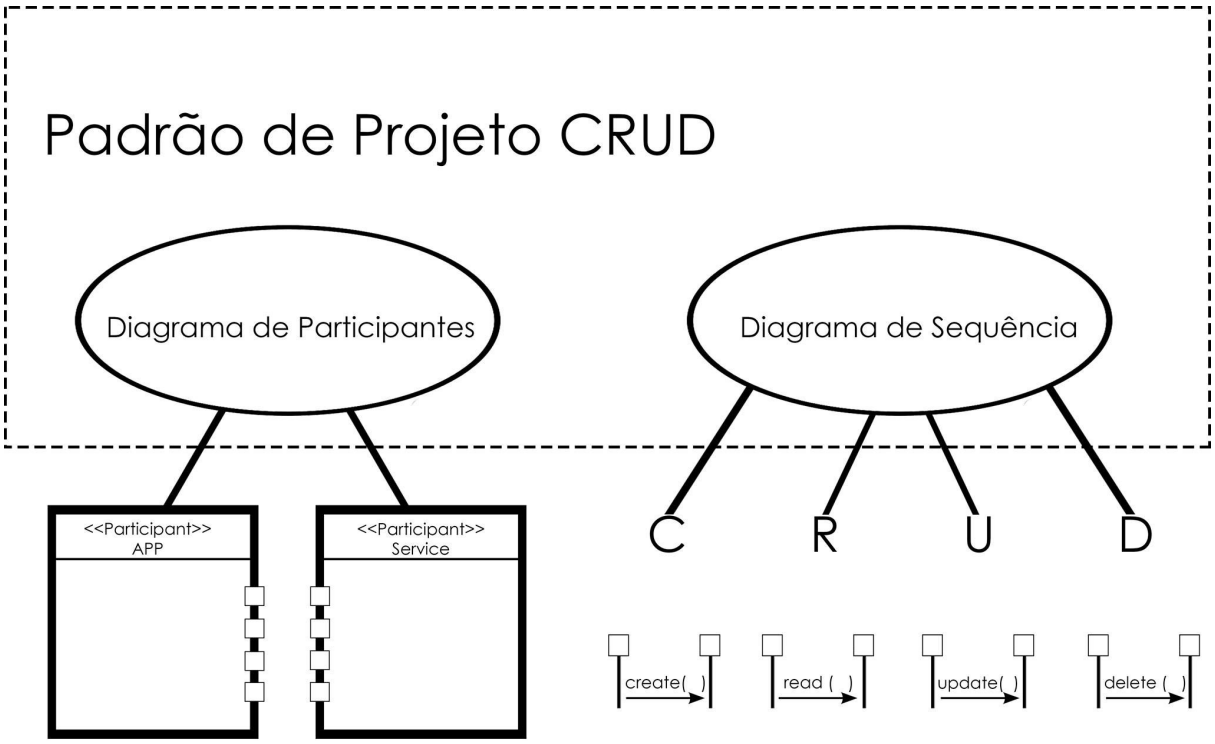

Figura 9 MIMAC: Padronização da Modelagem de Aplicações do Tipo CRUD

O diagrama de participantes servirá para definir qual participante provê os serviços e quais serão esses serviços, e o participante que usará esses serviços. Os diagramas de sequência representam o comportamento das aplicações do tipo CRUD, onde serão utilizados quatro diagramas de sequência, um para cada operação do CRUD (*create, read, update, delete*). No padrão proposto serão utilizadas as nomenclaturas APP e *Service* para representar a aplicação em si e uma plataforma de nuvem genérica, respectivamente.

A Figura 10 ilustra a representação dos participantes da aplicação com as operações do CRUD. O Participante APP, que representa o aplicativo em si, requer os seguintes serviços: *create*, *read*, *update*, *delete*. Ele solicita esses serviços à plataforma de serviço da nuvem para executar essas quatro operações na nuvem. O Participante *Service* representa uma plataforma de nuvem em geral, a qual proverá os serviços solicitados pelo participante que representa a aplicação.

Figura 10 MIMAC: Diagrama de Participantes de uma Aplicação do Tipo CRUD

O diagrama de sequência da operação *create* do CRUD, ilustrado na Figura 11, mostra a ordem em que as mensagens são trocadas entre os participantes da aplicação para criar um dado na nuvem.

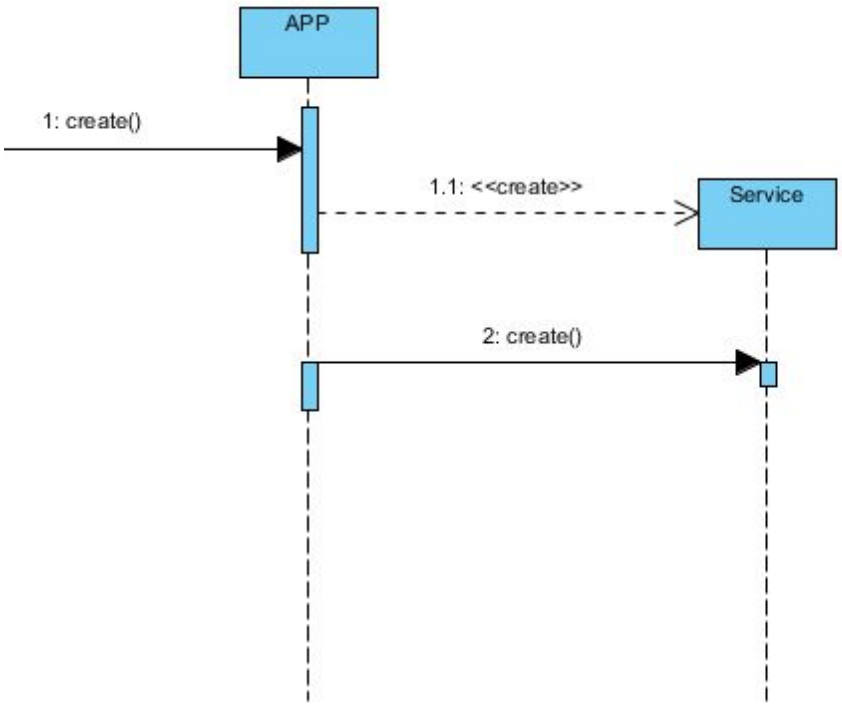

Figura 11 MIMAC: Diagrama de Sequência da Operação *Create* de uma Aplicação do Tipo CRUD

Quando a aplicação for inserir alguma informação na nuvem, o participante APP se comunica com o participante *Service*, solicitando que a operação de inserção seja realizada. Esta solicitação é representada por uma chamada de método. Nesta abordagem, serão usados métodos genéricos, usando os termos comumente

empregados no CRUD de forma a padronizar e tornar independente de implementações de plataformas específicas.

Na Figura 12 se pode observar a realização da operação listar do CRUD. Assim como ocorre na operação de inserção, para listar os dados da nuvem, é invocado o método *read*() da plataforma. Esses passos serão realizados para todas as operações do CRUD, pois nesta abordagem cada operação do CRUD será representada com um diagrama de sequência, utilizando métodos genéricos, o que facilitará uma geração de código.

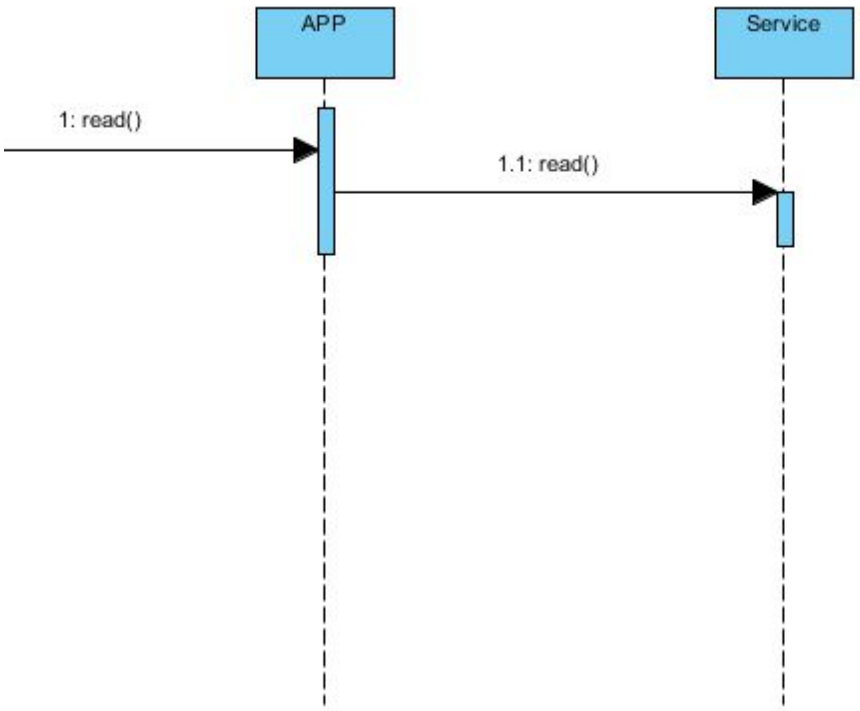

Figura 12 MIMAC: Diagrama de Sequência da Operação *Read* de uma Aplicação do Tipo CRUD

As Figuras 13 e 14 mostram a representação das operações editar e excluir do CRUD, respectivamente.

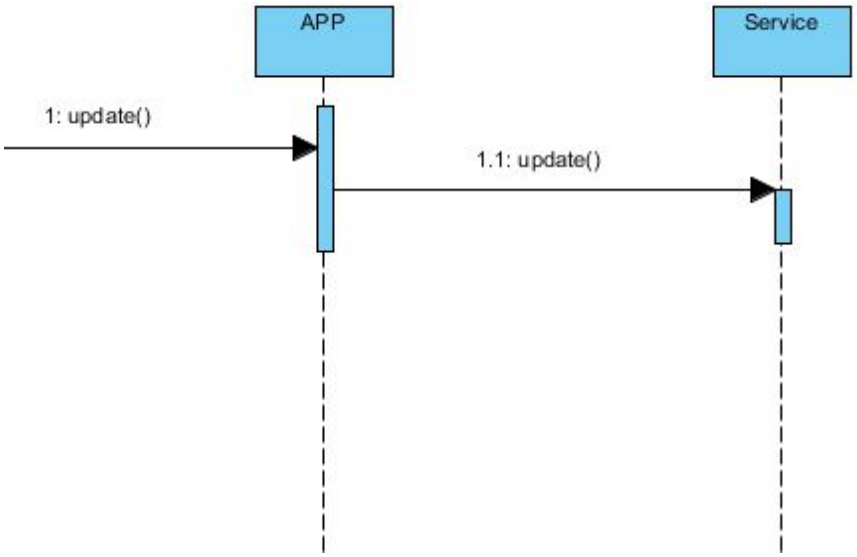

Figura 13 MIMAC: Diagrama de Sequência da Operação *Update* de uma Aplicação do Tipo CRUD

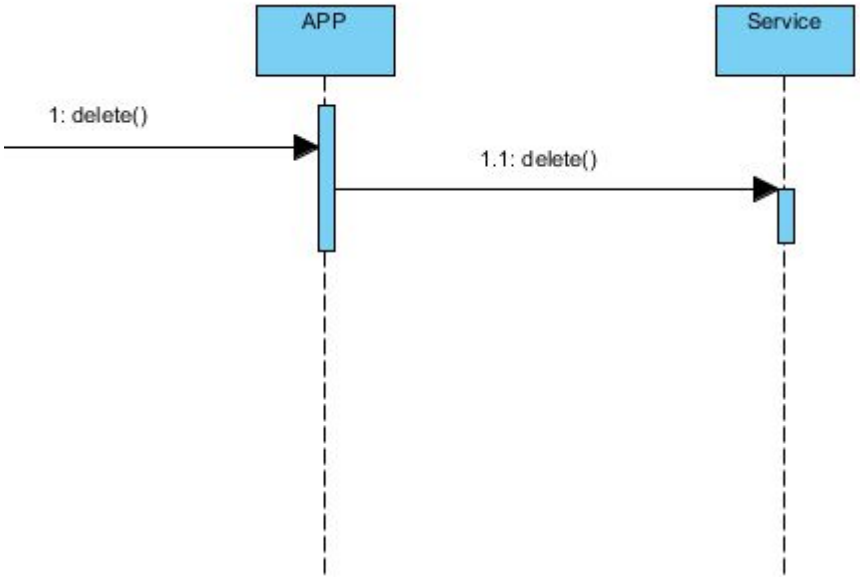

Figura 14 MIMAC: Diagrama de Sequência da Operação *Delete* de uma Aplicação do Tipo CRUD

O nome dos métodos invocados para a realização das operações varia entre as plataformas de nuvem. Porém a ordem temporal em que ocorrem as mensagens das operações do CRUD é algo que se repete independente da plataforma, como será caracterizado nos estudos de casos.

5 ESTUDOS DE CASO

Para demonstrar a abordagem proposta e, principalmente, sua capacidade de lidar com as diversidades das plataformas, foram realizados dois estudos de caso, um primeiro utilizando a plataforma de nuvem BlueMix da IBM, e o segundo utilizando a plataforma Google App Engine. Ambos utilizam a mesma aplicação, a qual consiste na gerência de uma Lista de compras armazenada na nuvem, permitindo que seus usuários realizem as seguintes operações: Inserir, Listar, Editar e Excluir (CRUD) dados da nuvem.

A Figura 15 ilustra as funcionalidades da aplicação empregada nos estudos de caso. Como se pode observar, o usuário poderá realizar as quatro operações do CRUD, podendo inserir um ou vários itens na lista de compras, listar esses itens, editar a lista e, por fim, excluir algum item da lista de compras.

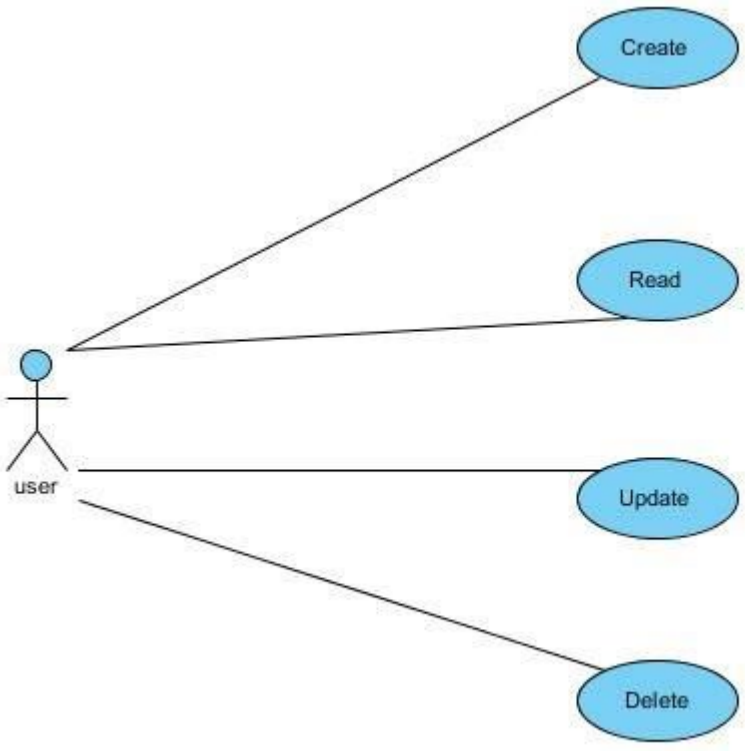

Figura 15 Diagrama de Casos de Uso da Aplicação Lista de Compras

Ambos os estudos de caso utilizaram a mesma aplicação, a lista de compras, porém com implementações diferentes que consideram as especificidades de cada plataforma. No primeiro estudo de caso, uma implementação da aplicação para a

plataforma de nuvem BlueMix, disponível em (IBM, 2015), foi analisada através de engenharia reversa. A partir deste processo, o diagrama de classes da aplicação foi gerado automaticamente pela ferramenta Astah. Esta análise permitiu estudar a aplicação e a comunicação da aplicação com a plataforma da IBM. Além disso, este processo permitiu construir um modelo empregando a abordagem proposta. Este modelo pode ser considerado um modelo PSM, pois este contém aspectos específicos da plataforma Bluemix. Visando permitir a comparação entre diferentes implementações da mesma aplicação, uma segunda implementação foi desenvolvida empregando a plataforma da Google e a engenharia reversa desta implementação também foi realizada de forma a extrair seu modelo PSM a partir do código.

Para construir os diagramas de sequência dos modelos PSM de ambas implementações, foi utilizada a ferramenta Visual Paradigm Enterprise Edition. Esta ferramenta dá suporte tanto para diagramas da UML como diagramas do SoaML, permitindo que uma única ferramenta seja empregada para a construção dos modelos empregados pela MIMAC.

As seções 5.1 e 5.2 detalham as modelagens construídas em cada estudo de caso. A seção 5.1 apresenta o estudo de caso utilizando a plataforma de nuvem Bluemix da IBM e a seção 5.2 o estudo de caso utilizando a plataforma do Google. Por fim, a seção 5.3 apresenta uma discussão sobre esses dois estudos de caso.

5.1 Modelagem de uma Aplicação Baseada na Plataforma Bluemix da IBM

A implementação da Lista de Compras para a plataforma Bluemix possui 4 classes, BlueListApplication, EditActivity, MainActivity e Item. Para a modelagem dessa aplicação, foram utilizados os diagramas previstos na MIMAC. Primeiramente, é construído o diagrama de classes que representa a estrutura da aplicação e pode ser adotado tanto em um modelo independente de plataforma (PIM), quanto um dependente (PSM). A Figura 16 mostra o diagrama de classes desta implementação da aplicação gerado automaticamente pela ferramenta Astah.

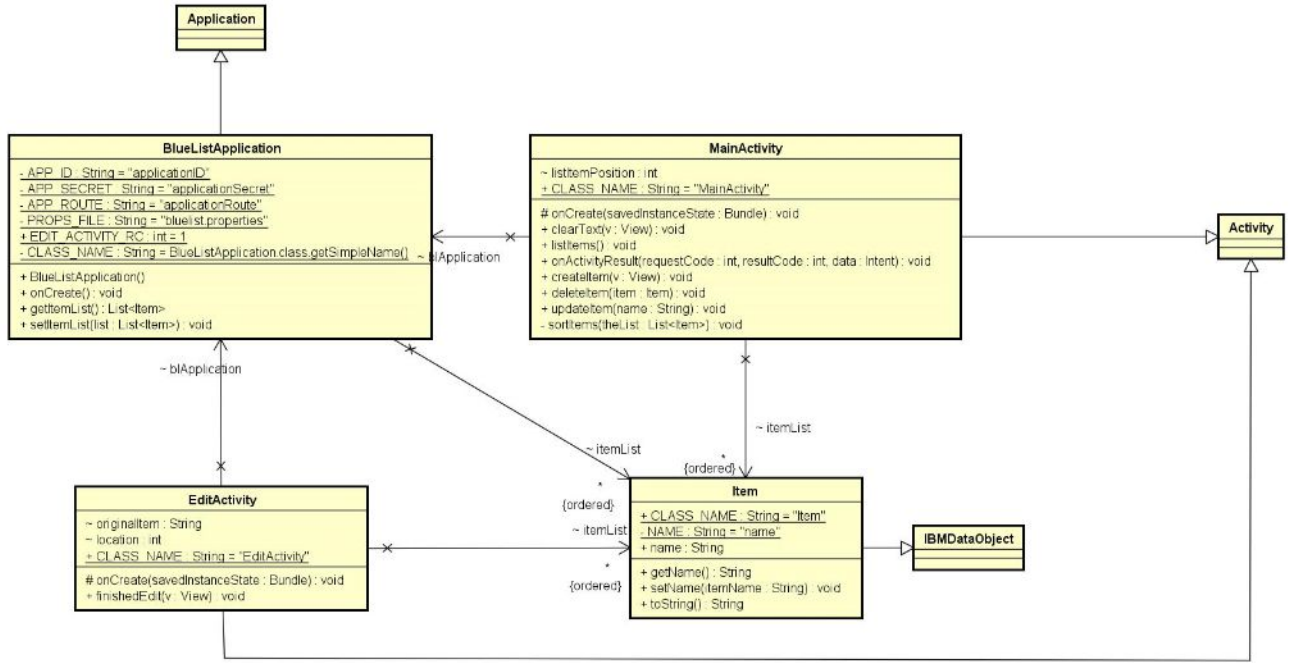

Figura 16 Diagrama de Classes da Lista de Compras - Plataforma Bluemix
Gerado pela ferramenta Astah

A classe BlueListApplication representa o aplicativo em si. Esta contém algumas constantes necessárias para a inicialização do serviço, são elas: ApplicationID, ApplicationRoute e ApplicationSecret. ApplicationID é a chave exclusiva designada ao aplicativo Mobile Cloud criada no Bluemix. Quando se faz uso dos serviços de nuvem da IBM, é necessário um cadastro para poder fazer uso do armazenamento na nuvem.

A propriedade ApplicationID é essa chave cadastrada antes de implementar a aplicação. ApplicationRoute é a rota designada ao Mobile Cloud criada na Bluemix e ApplicationSecret é a chave de segurança, todas cadastradas antes da implementação da aplicação.

A classe Item é utilizada para se comunicar com a nuvem. A classe MainActivity usa os serviços para criar, ler e excluir dados na nuvem. Nela estão as interações com a nuvem e os métodos para criar, listar e deletar itens da lista de compras. Por fim, a classe EditActivity implementa métodos que atualizam os dados na nuvem.

Em um segundo momento, os diagramas de sequência são construídos de forma a detalhar o comportamento da aplicação, explicitando as interações entre o aplicativo e a nuvem. Um diagrama de sequência foi construído para cada uma das operações do aplicativo, seguindo a MIMAC. A Figura 17 mostra o trecho de código

da aplicação que insere um item na lista e a Figura 18 ilustra a modelagem desse trecho do código da aplicação, na classe MainActivity, onde é criado um item na lista de compras. Este diagrama descreve, no nível de modelo, a interação entre a aplicação, representada pelo objeto *BlueList* e a plataforma de nuvem. O objeto *Item* representa, neste caso, a plataforma Bluemix, através da especialização da classe *IBMDataObjects* provida pela plataforma da IBM. Neste cenário modelado pelo diagrama de sequência, a invocação do método *save()* representa a interação propriamente dita entre o aplicativo e o serviço de nuvem, a qual realizará o salvamento dos dados do novo item na nuvem.

```
189     public void createItem(View v) {
190         EditText itemToAdd = (EditText) findViewById(R.id.itemToAdd);
191         String toAdd = itemToAdd.getText().toString();
192         Item item = new Item();
193         if (!toAdd.equals("")) {
194             item.setName(toAdd);
195             // Use the IBMDataObject to create and persist the Item object.
196             item.save().continueWith(new Continuation<IBMDataObject, Void>() {
```

Figura 17 Trecho de código que insere item na Lista de Compras – Bluemix

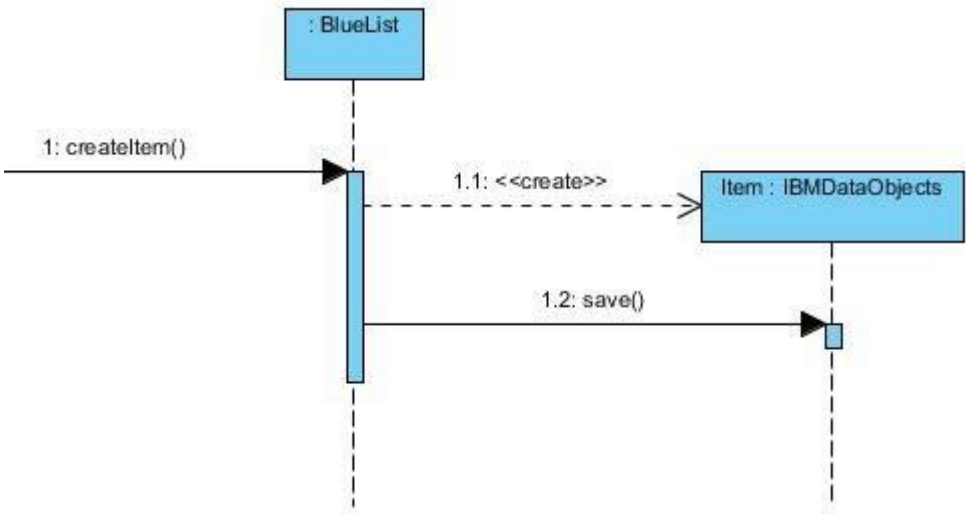

Figura 18 Diagrama de Sequência do método createItem – Bluemix

A Figura 19 mostra o trecho de código da aplicação que remove um item da lista de compras e a Figura 20 demonstra a modelagem desse trecho do código da aplicação, na classe MainActivity, onde um item na lista de compras é excluído.

```
225   public void deleteItem(Item item) {
226           itemList.remove(listItemPosition);
227
228           // tenta excluir o item do servidor.
229           //o método delede da IMBDataObject retorna um Bolts "Task", usado para verificar se o item foi
excluido com sucesso
230           item.delete().continueWith(new Continuation<IBMDataObject, Void>() {
```

Figura 19 Trecho de código que exclui um item na Lista de Compras – Bluemix

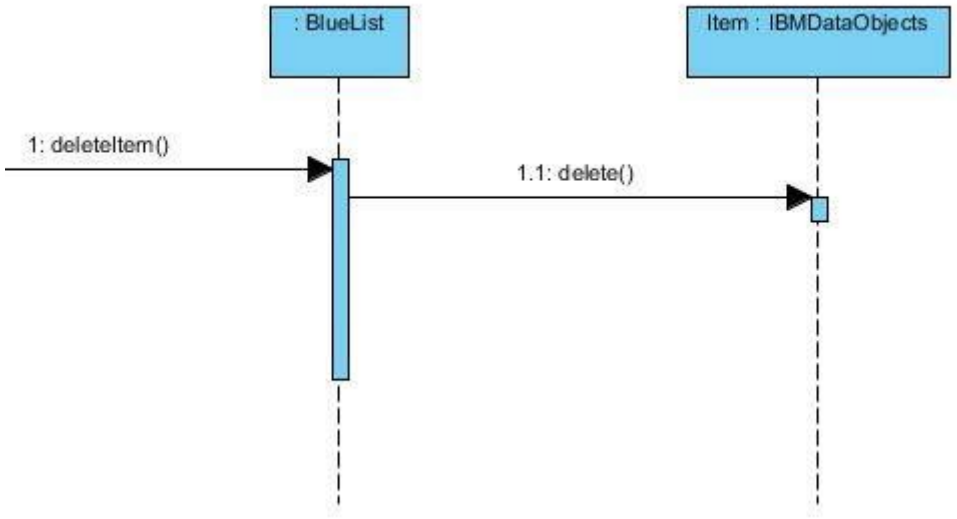

Figura 20 Diagrama de Sequência do método deleteItem – Bluemix

Algumas plataformas de nuvem requerem que os serviços sejam inicializados antes do uso. Esta interação de inicialização adicional precisa estar descrita desde a modelagem para que seja considerada na implementação. A Figura 21 mostra o trecho de código na classe BlueListApplication e a Figura 22 ilustra a modelagem dessa inicialização.

```
83      public void onCreate() {
84          super.onCreate();
85          itemList = new ArrayList<Item>(); //array de compras
86           // lê o arquivo bluelist.properties.
87          Properties props = new java.util.Properties();
88          Context context = getApplicationContext();
89          try {
90              AssetManager assetManager = context.getAssets();
91              props.load(assetManager.open(PROPS_FILE));//carrega o arquivo bluelist.properties, que contem
as informações para acesso da nuvem
92              Log.i(CLASS_NAME, "Found configuration file: " + PROPS_FILE);
93              } catch (FileNotFoundException e) {
94              Log.e(CLASS_NAME, "The bluelist.properties file was not found.", e);
95              } catch (IOException e) {
96              Log.e(CLASS_NAME, "The bluelist.properties file could not be read properly.", e);
97          }
98          // initialize the IBM core backend-as-a-service
99          IBMBluemix.initialize(this, props.getProperty(APP_ID),
100         props.getProperty(APP_SECRET), props.getProperty(APP_ROUTE));
101         // initialize the IBM Data Service
102         IBMData.initializeService();
103         // register the Item Specialization
104         Item.registerSpecialization(Item.class);
105
106
107     }
```

Figura 21 Trecho de código que inicializa os serviços da nuvem – Bluemix

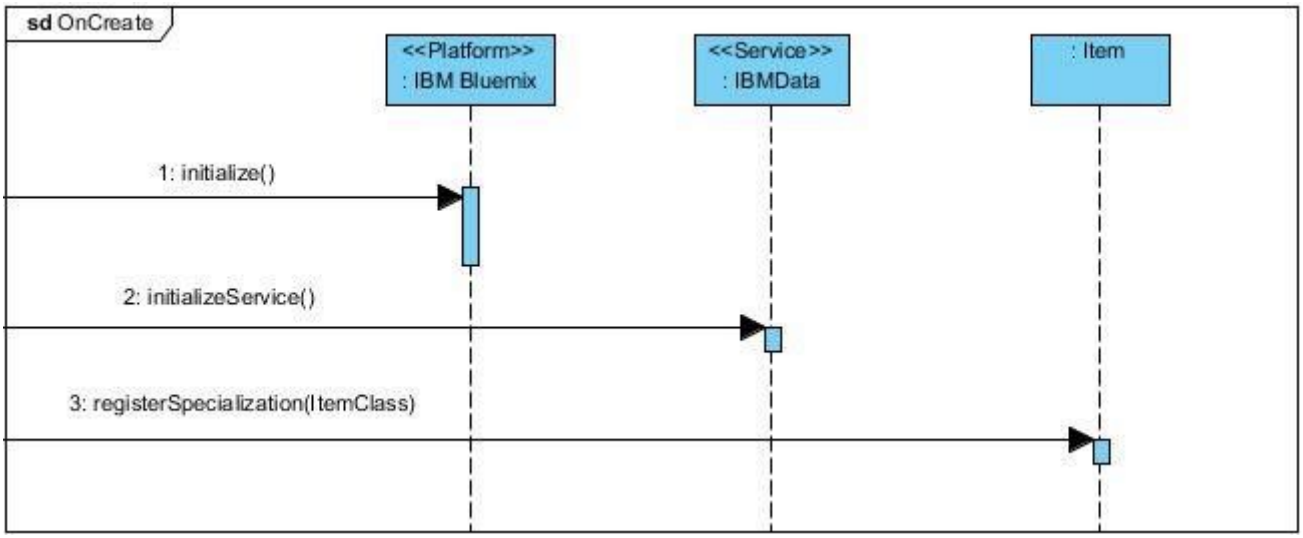

Figura 22 Diagrama de Sequência do método OnCreate – Bluemix

Observa-se que do trecho de código apresentado na Figura 21, três métodos estão representados na modelagem, initialize(), initializeService() e registerSpecialization(ItemClass), apresentada na Figura 22. Esses três métodos são os métodos necessários para inicializar e utilizar os serviços de nuvem fornecidos pela plataforma Bluemix. Neste diagrama, a plataforma e o serviço de armazenamento são representados pelos objetos *IBMBluemix* e *IBMData*, respectivamente. Além disso, a especialização da classe *IBMDataObjects,* realizada pela classe *Item,* deve ser

também registrada para que métodos de acesso a dados possam ser invocados diretamente a partir de objetos do tipo *Item*.

5.2 Modelagem de uma Aplicação Baseada na Plataforma Google App Engine

Para modelagem do estudo de caso da aplicação baseada na plataforma Google App Engine (GAE) foram realizados os mesmos passos do estudo de caso baseado na plataforma Bluemix. Para cada operação do CRUD, foi construído um diagrama de sequência, os quais representam modelos PSM propostos pela MIMAC. Para o desenvolvimento da lista de compras para o GAE, foi utilizada apenas uma classe, a MainActivity, a qual pode ser visualizada no diagrama de classes apresentado na Figura 23, o qual foi gerado automaticamente pela ferramenta Astah. Isso por que esta implementação utiliza Listeners, o qual é um componente que fica escutando/esperando que o usuário faça alguma tarefa (LANHELLAS, 2015). Nesta aplicação, cada operação do CRUD foi implementada utilizando um Listener. Nesse caso, cada item fica escutando/esperando o usuário realizar alguma ação, por exemplo, se o usuário pressionar um item, ele pode editar o mesmo.

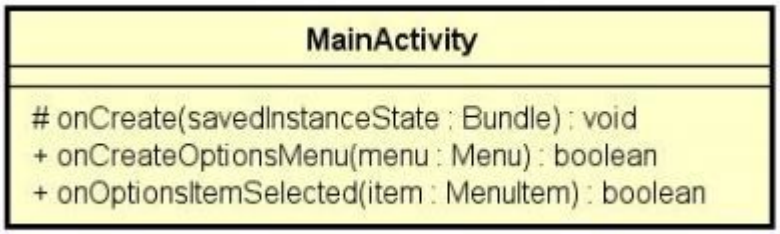

Figure 23 Diagrama de Classes da Lista de Compras com a Plataforma GAE
Gerado automaticamente pela ferramenta Astah

A Figura 23 apresenta o trecho do código da aplicação, responsável pela inserção ou criação de um item na lista de compras do aplicativo baseado na GAE e a Figura 24 ilustra o diagrama de sequência correspondente. Neste modelo, o objeto *MainActivity* representa o aplicativo Android e o objeto Firebase representa a plataforma de nuvem, e a invocação do método *push()* representa a interação do aplicativo com o serviço de nuvem.

```
81          final EditText text = (EditText) findViewById(R.id.itemToAdd);
82          final Button button = (Button) findViewById(R.id.addButton);
83          button.setOnClickListener(new View.OnClickListener() {
84              public void onClick(View v) {
85                  new Firebase("https://flickering-torch-1272.firebaseio.com/groceryItems")
86                          .push()
87                          .child("text")
88                          .setValue(text.getText().toString());
89              }
90          });
```

Figura 24 Trecho de código que inclui um item na Lista de Compras – GAE

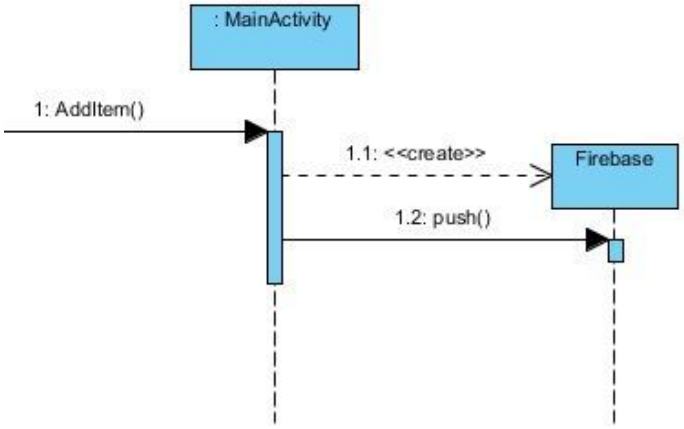

Figura 25 Diagrama de Sequência do método AddItem – GAE

Na Figura 25 pode-se visualizar o trecho de código que exclui um item na lista de compras e a Figura 26 ilustra a modelagem desse trecho de código da aplicação.

```
95          public void onItemClick(AdapterView<?> parent, View view,
96                                  int position, long id) {
97              new Firebase("https://flickering-torch-1272.firebaseio.com/groceryItems")
98                      .orderByChild("text")
99                      .equalTo((String) listView.getItemAtPosition(position))
100                     .addListenerForSingleValueEvent(new ValueEventListener() {
101                         public void onDataChange(DataSnapshot dataSnapshot) {
102                             if (dataSnapshot.hasChildren()) {
103                                 DataSnapshot firstChild = dataSnapshot.getChildren().iterator().next();
104                                 firstChild.getRef().removeValue();
105                             }
106                         }
107
108                         public void onCancelled(FirebaseError firebaseError) {
109                         }
110                     });
111         }
```

Figura 26 Trecho de código que exclui um item na Lista de Compras – GAE

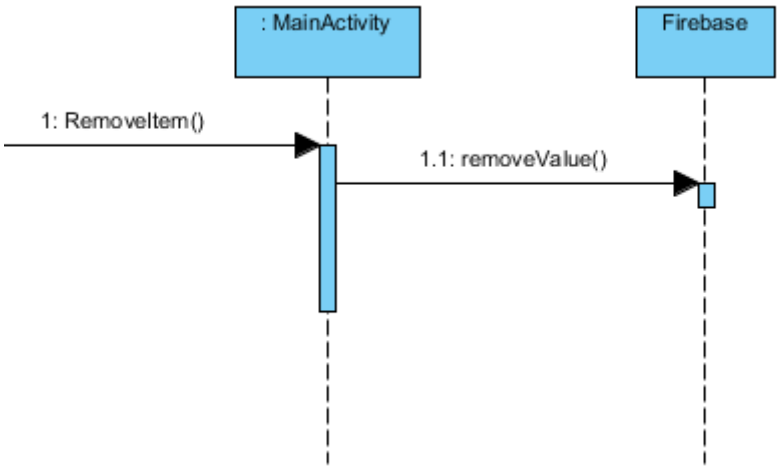

Figura 27 Diagrama de Sequência do método RemoveItem – GAE

No cenário de remoção de um item, o método *removeValue*() é chamado para excluir um dado da lista de compras.

5.3 Discussão

Como demonstrado nos estudos de caso, em ambas implementações pode ser observado um padrão de comunicação entre App e plataforma, o que difere uma da outra, além da invocação dos métodos, são alguns detalhes de comunicação. Por exemplo, a plataforma da IBM necessita a invocação de um método de inicialização de serviço. Esta inicialização faz com que a modelagem desta implementação inclua um diagrama a mais que a modelagem da plataforma do Google, que dispensa essa inicialização de serviço.

Os estudos de caso, embora limitados a duas plataformas de nuvem, a Bluemix e a Google App Engine, já permitiram observar um padrão de comportamento entre as interações do aplicativo com as plataformas. Este padrão de comportamento reforça a adoção do padrão de projeto proposto para aplicativos CRUD.

Através dos estudos de caso a abordagem MIMAC foi demonstrada. No entanto, para uma avaliação mais completa, este estudo deve ser realizado também para outras plataformas de nuvem, como por exemplo a Windows Azure. É importante destacar que a abordagem auxilia projetistas no desenvolvimento de aplicativos para diferentes plataformas de nuvem, permitindo explicitar a interação aplicativo-nuvem. O modelo construído pela adoção da abordagem pode ser empregado futuramente por ferramentas de geração de código automáticas e, assim, estas especificidades das plataformas podem ser tratadas apenas no nível do código. Neste caso, o

diagrama de sequência adicional requerido pela plataforma Bluemix pode ser no futuro suprimido do modelo e a geração das invocações de inicialização podem ser incluídas pela ferramenta de geração de código.

6 CONCLUSÃO E TRABALHOS FUTUROS

Aplicações de computação móvel na nuvem (MCC) estão cada vez mais frequentes, visto que estas conseguem contornar os problemas relativos à limitação dos dispositivos móveis, já que boa parte do processamento e armazenamento pode ser realizado na nuvem.

No contexto da MCC, têm aumentado o número de empresas que vêm desenvolvido plataformas de nuvem, oferecendo serviços de diferentes naturezas. Assim, além dos diferentes sistemas operacionais, linguagens de programação e serviços, os desenvolvedores também devem se preocupar com detalhes para o uso das diferentes plataformas de nuvem, cada uma com suas infraestrutura e API específicas. Para auxiliar os profissionais no tratamento desta complexidade, modelos podem ser empregados para tratar os detalhes de implementação.

Este trabalho propôs MIMAC como uma abordagem para modelagem de aplicações MCC, a qual permite abstrair detalhes das diferentes plataformas ao representar a interação com a nuvem. A abordagem proposta emprega diagramas da UML e da SoaML, combinando as duas linguagens. Embora a abordagem proposta também empregue diagramas estruturais, a mesma tem como principal contribuição modelar o comportamento dessas aplicações, explicitando o procedimento de comunicação com a nuvem.

A perspectiva é que o modelo da aplicação proposto na abordagem possa auxiliar os projetistas tanto a entender melhor a aplicação, detalhando a sua estrutura e comportamento, bem como pode ser empregado para fins de geração de código.

MIMAC baseia-se na arquitetura MDA da OMG, propondo modelos que podem ser traduzidos para modelos mais detalhados até que uma implementação da aplicação possa ser alcançada. Por ser baseada na arquitetura MDA, é possível realizar a modelagem de aplicações em dois níveis, tanto focada a uma plataforma de nuvem específica (PSM) como independente de plataforma (PIM).

Os procedimentos de interação com a nuvem costumam se repetir em diferentes aplicações. Por esta razão, este trabalho também propõe um padrão de projeto para aplicações do tipo CRUD, que são recorrentes em MCC. O objetivo do padrão proposto é modelar aplicações do tipo CRUD, abstraindo os procedimentos

específicos de cada plataforma de nuvem adotada e assim construindo um modelo PSM. Para tanto, o padrão define a utilização de um diagrama de participantes do SoaML, para representar os participantes e suas atribuições, e quatro diagramas de sequência da UML, um para cada operação do CRUD.

Foram utilizados dois estudos de caso para demonstrar a abordagem proposta, um primeiro utilizando a plataforma de nuvem Bluemix da IBM, e um segundo utilizando a plataforma de nuvem do Google, Google App Engine. Os estudos de caso empregam uma aplicação CRUD típica no cenário da MCC, a qual consiste de uma lista de compras armazenada na nuvem, permitindo que seus usuários realizem as quatro operações do CRUD (*create, read, update, delete*) a partir do dispositivo móvel. Porém, cada estudo de caso emprega uma implementação diferente, que considera as especificidades de cada plataforma.

A partir dos estudos de caso, pode ser observado um padrão de comunicação entre a aplicação e a nuvem, diferindo as implementações específicas para cada plataforma de nuvem em aspectos de invocação de métodos e de comunicação. Particularmente, a plataforma da IBM necessita a invocação de um método de inicialização de serviço. Esta inicialização faz com que a modelagem desta implementação inclua um diagrama a mais que a modelagem da plataforma do Google, que dispensa essa inicialização de serviço.

O padrão de comportamento observado nos estudos de caso reforça a adoção do padrão de projeto proposto para aplicativos CRUD. É importante destacar que a abordagem demonstrada pelos estudos de caso, também auxilia projetistas no desenvolvimento de aplicativos para diferentes plataformas de nuvem, permitindo explicitar a interação aplicativo-nuvem.

Como trabalhos futuros, novas plataformas devem ser estudadas, como por exemplo a Windows Azure, para uma avaliação mais completa da abordagem proposta. Além disso, o modelo construído pela adoção da abordagem pode ser empregado futuramente por ferramentas de geração de código automáticas e assim estas especificidades das plataformas podem ser tratadas apenas no nível do código. Neste caso, o diagrama de sequência adicional requerido pela plataforma Bluemix pode ser no futuro suprimido do modelo e a geração desta invocação do método pode ser incluída automaticamente pela ferramenta de geração de código.

REFERÊNCIAS

ADEWOJO, A.; BASS, M.; HUI, K. **Cloud Deployment Patterns: Migrating a Database Driven Application to the Cloud using Design Patterns** (2014). Disponível em: <https://www.researchgate.net/publication/283451720_Cloud_Deployment_Patterns_Migrating_a_Database_Driven_Application_to_the_Cloud_using_Design_Patterns>. Acessado em Novembro de 2015.

AEPONA. White Paper. **Mobile Cloud Computing Solution Brief**. 2010. Disponível em: <http://firstcloudexperience.site/technology-8/white-paper-mobile-cloud-computing-solution-brief-aepona-november-2010.html>. Acessado em Janeiro de 2014.

AMAZON WEB SERVICES Inc. **Amazon EC2** (2010). Disponívem em: <https://aws.amazon.com/pt/?nc2=h_lg>. Acessado em Fevereiro de 2014.

Arcitura™ Education Inc. **Cloud Patterns** (2015). Disponível em: <http://www.cloudpatterns.org>. Acessado em Novembro de 2015.

BRAGA, V. **Um Processo para Projeto Arquitetural de *Software* Dirigido a Modelos e Orientado a Serviços** (2011). Disponível em:

<http://www.cin.ufpe.br/~if718/referencias/vtb_msc_novo_v002-Banca.pdf>. Acessa-do em Fevereiro de 2014.

BRAMBILLA, M., CABOT, J., WIMMER, M. (2012) **Model-Driven Software Engineering in Practice.** Morgan & Claypool Publishers.

BRIZENO, M. **Padrões de Projeto** (2011). Disponível em: <https://brizeno.wordpress.com/padroes/>. Acessado em Novembro de 2015.

DINH, H. T.; LEE, C.; NIYATO, D.; WANG, P. A survey of mobile cloud computing: architecture, APPlications, and APProaches. **Wireless Communications and Mobile Computing**, 2013; vol. 13, n. 18, pp. 1587-1611.

DIIRR, T., AZEVEDO, L. G., FARIA, F., SANTORO, F., & BAIÃO, F. (2013). **Utilizando SoaML para Modelagem e Geração de Código de Serviços em uma Abordagem SOA**. Cadernos do IMESérie Informática, 30, 38-49.

ELVESÆTER, B.; BERRE, A.; SADOVYKH, A. **Specifying Services using the Services oriented architecture Modeling Language (SoaML): A baseline for**

Specification. International Conference on Cloud Computing and Service Science (CLOSER), 2011.

ERL, T. **SOA: Principles if Service Desing**. (2007).1st Edition.

FERNANDO N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing: A survey. **Future Generation Computer Systems**, vol. 29, n.1, pp. 84-106, 2013.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: elements of reusable object-oriented software** (1995).

GLOBAL WEB INDEX. Disponível em: <http://insight.globalwebindex.net/device>. Acessado em Abril de 2015.

GOOGLE DEVELOPERS. **Google APP Engine** (2014). Disponível em: <https://developers.google.com/APPengine/docs/whatisgoogleAPPengine?hl=pt-BR>. Acessado em Fevereiro de 2014.

GUAN, L.; KE, X.; SONG, M.; SONG, J. **A Survey of Research on Mobile Cloud Computing**. IEEE/ACIS International Conference on Computer and Information Science, 2011.

IBM. **Cloud Computing** (2014). Disponível em: <http://www.ibm.com/cloud-computing/br/pt/?lnk=bucl#community>. Acessado em Fevereiro de 2014.

IBM. **Build an Android app using the IBM Mobile Data for Bluemix cloud service**, 2015. Disponível em: <http://www.ibm.com/developerworks/library/mo-android-mobiledata-app/>. Acessado em Maio de 2015.

KHAN, R.; OTHMAN, M.; MADANI, S. A.; KHAN, S. U. **A Survey of Mobile Cloud Computing APPlication Models**. IEEE Communications Surveys & Tutorials, 2013, vol. 16, n. 1, pp. 393-413.

LANHELLAS, R. **Java Listeners: Trabalhando com ActionListener e KeyListener em Java**. Disponível em: <http://www.devmedia.com.br/javalisteners-trabalhando-com-actionlistener-e-keylistener-em-java/31850> Acessado em Janeiro de 2016.

MANJUNATHA, A., RANABAHU, A., SHETH, A., THIRUNARAYAN, K. (2010). **Power of Clouds In Your Pocket: An Efficient Approach for Cloud Mobile**. IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom).

MCCORMICK, Z. and SCHMIDT, D. C. (2012). **Data synchronization patterns in mobile application design.** 19th Conference on Pattern Languages of Programs (PLoP).

MICROSOFT. **O que é o Windows Azure?** (2011). Disponível em: <http://www.windowsazure.com/pt-br/overview/what-is-windows-azure/>. Acessado em Fevereiro de 2014.

NETO, O.; FREITAS, R. **Computação em Nuvens, Visão Comparativa entre as Principais Plataformas de Mercado**. Disponível em: http://olavooneto.files.wordpress.com/2011/01/computacao_em_nuvens_visao_olavo_neto.pdf, acessado em fevereiro de 2014.

NIST. **The NIST Definition of Cloud Computing**. National Institute Of Standards and Technology (2011).

OMG. **Service oriented architecture Modeling Language (SoaML) Specification**. (2012). Disponível em: <http://www.omg.org/spec/SoaML/1.0.1/PDF>. Acessado em Fevereiro de 2014.

OMG. **MDA Guide rev 2.0.** (2014) Disponível em: <http://www.omg.org/mda/>. Acessado em Janeiro de 2015.

OMG. **Unified Modeling Language 2.5.** (2015) Disponível em: <http://www.omg.org/spec/UML/2.5/PDF/>. Acessado em Junho de 2015.

PARADA, A., MARQUES, M., BRISOLARA, L. (2015). **Automating mobile application development: UML-based code generation for Android and Windows Phone**. Revista de Informática Teórica e Aplicada, vol. 22, n. 2, pp. 31-50.

SOUZA, T. **Model Driven Architecture – Conceitos Fundamentais** (2015). Disponível em: <http://www.linhadecodigo.com.br/artigo/1953/model-driven-architecture-conceitos-fundamentais.aspx>. Acessado em Janeiro de 2016.

THOMAZINI, L. F. **Padrões de Projeto**. (2006). Disponível em: <http://bibdig.poliseducacional.com.br/document/?down=92>. Acessado em Novembro de 2015.

VISUAL PARADIGM. **Modelagem SoaML**. Disponível em: <http://www.visual-paradigm.com/product/vpuml/features/soamlmodeling.jsp>, Acessado em Fevereiro de 2014.

W3C. **Web Services Description Language (WSDL) 1.1** (2001) Disponível em: <https://www.w3.org/TR/wsdl>. Acessado em Novembro de 2015.