Geração Automática de Código para Aplicações Móveis Centradas em Dados Suportando Diferentes Infraestruturas de Nuvem

Kellerson Kurtz[1], **Adenauer Correa Yamin**[1], **Lisane B. Brisolara**[1]

[1]Centro de Desenvolvimento Tecnológico – Universidade Federal de Pelotas (UFPEL)

{kkurtz,adenauer,lisane}@inf.ufpel.edu.br

Abstract. *This work proposes a code generation model-driven approach for applications centered on data stored in the cloud. Our approach is based on the application modeling using UML diagrams, and defines rules to model the interaction application to cloud, aiming to turn it independent of the adopted cloud infrastructure. Thus, the proposed approach allows the employment of different infrastructures without requiring specific knowledge of its APIs, besides support easy migration among different infrastructure. Moreover, this paper discusses the integration of the proposed approach into a code generation tool. Both approach and its automation are validated through a case study.*

Resumo. *Este trabalho propõe uma abordagem orientada a modelos para a geração de código para aplicações centrada em dados na nuvem. A mesma baseia-se na modelagem da aplicação usando diagramas UML e define regras para modelar a interação da aplicação com a nuvem, visando torná-la independente da infraestrutura de nuvem adotada. Desta forma, a abordagem proposta permite o emprego de diferentes infraestruturas de nuvem sem exigir conhecimentos específicos de suas APIs, além de permitir fácil migração entre diferentes infraestruturas. Além disso, este artigo discute a integração da abordagem proposta em uma ferramenta de geração de código. Abordagem e automatização propostas são validadas através de um estudo de caso.*

1. Introdução

Hoje em dia, *tablets* e *smartphones* representam a maior parte dos acessos a internet, serviços e sistemas de informação [CISCO 2016]. No entanto, desenvolver aplicações móveis possui uma série de dificuldades adicionais, dentre as quais destacam-se: limitações de capacidade de armazenamento, processamento e restrições de consumo energético. Estas limitações aumentam significativamente o tempo de desenvolvimento.

Para contornar as limitações de processamento, armazenamento e consumo de energia, a computação móvel na nuvem [Dinh et al. 2013] vem sendo empregada. Usando recursos na nuvem, aplicações móveis não ficam limitadas aos recursos dos dispositivos, podendo armazenar um grande volume de dados, obter maior poder de processamento e, também, reduzir o consumo energético. Um grande número de aplicativos emprega a nuvem para armazenamento de dados, nestes casos usualmente os aplicativos realizam operações de criação, leitura, edição e remoção de dados (CRUD), podendo ser classificados como aplicativos orientados a dados [Xanthopoulos and Xinogalos 2013].

Embora os benefícios da utilização de computação na nuvem sejam evidentes, o seu uso também acarreta em um aumento no período de desenvolvimento, pois *APIs* e *frameworks* específicos devem ser empregados o que implica em desafios adicionais aos desenvolvedores. Cada infraestrutura de nuvem oferece uma interface distinta de comunicação aplicação-nuvem, o que dificulta a portabilidade do aplicativo de uma infraestrutura para outra, além de exigir conhecimentos específicos sobre a plataforma a ser empregada na implementação do aplicativo [Gonidis et al. 2013].

Um paradigma bastante utilizado no desenvolvimento de *software* é o desenvolvimento orientado a modelos. Neste paradigma, modelos são construídos desde a análise e projeto do *software* e estes modelos guiam o seu desenvolvimento [Papotti 2013]. Estes modelos normalmente são descritos usando a linguagem padrão UML e são compostos de uma série de diagramas que constituem visões diferentes do *software*. Estes diagramas são então utilizados como entrada em ferramentas de geração automática de código, visando acelerar a obtenção de uma implementação do *software*.

A *GenCode* [Parada et al. 2013] [Parada et al. 2015] é um exemplo de ferramenta disponível para a geração automática de código a partir de modelos UML. Esta utiliza diagramas de classe e de sequência para automatizar a geração de código para aplicações móveis para os sistemas operacionais *Android* e *Windows Phone*. No entanto, a ferramenta *GenCode* não gera trechos de código responsável pela interação do aplicativo móvel com uma dada infraestrutura de nuvem. Esta limitação está relacionada ao fato de notações e metodologias padrão empregadas para a modelagem do *software* não oferecerem suporte a representação desta interação.

Para resolver esta limitação, este trabalho define uma abordagem padronizada para modelagem de aplicativos orientado a dados, visando suportar a geração de código para diferentes infraestruturas de nuvem. Para demonstrar nossa abordagem, este trabalho propõe uma extensão da ferramenta *GenCode* – denominada *GenCodeCloud* – com foco na geração automática de código *Android* responsável pela interação aplicação-nuvem. A geração de código proposta baseia-se na padronização da modelagem desta interação, que torna a modelagem independente da infraestrutura empregada, e assim facilita o suporte a diferentes infraestruturas. Através de um estudo de caso que faz emprego da abordagem proposta, os principais pontos da mesma são discutidos.

Este artigo está organizado da seguinte maneira. Na Seção 2 são discutidos trabalhos relacionados. Na Seção 3 é discutido o suporte a operações de manipulação de dados armazenados nas plataformas de nuvem, mencionando duas plataformas de nuvem distintas. A abordagem proposta é detalhada na Seção 4 e um estudo de caso é apresentado na Seção 5. Na Seção 6, conclusões e trabalhos futuros são apresentados.

2. Trabalhos Relacionados

Aplicações móveis dirigidas a dados ou centrada em dados estão cada vez mais sendo empregadas, no entanto seu desenvolvimento exige uma série de preocupações [Agrawal et al. 2013]. Recentemente, *frameworks*, *APIS* e infraestruturas têm sido propostas para facilitar o desenvolvimento destas aplicações, muitos destes esforços focam na sincronização dos dados do dispositivo e com a nuvem, como *Mobius* [Chun et al. 2012] e *Simba* [Agrawal et al. 2013]. Ambos baseiam-se em *APIs* e *frameworks* no nível de código e não na abstração e transparência provida por modelos.

Abordagens orientadas a modelos têm sido propostas para lidar com os desafios do desenvolvimento de aplicações móveis tais como *GenCode*, *XIS-Mobile* [Ribeiro et al. 2014], e *MobiCloud* [Ranabahu et al. 2011]. *XIS-Mobile* define um perfil UML para descrever modelos independentes de plataforma e suporta a geração de código para as plataformas *iOs*, *Android* e *Windows Phone*. Similar ao *XIS-Mobile*, *GenCode* também foca na geração de código para múltiplas plataformas a partir de modelos UML. Porém, dentre as citadas, apenas *MobiCloud* aborda aplicações de computação na nuvem e o suporte às operações CRUD. Embora, um esforço interessante por gerar também o *back-end* das aplicações para as plataformas *Google App Engine* e *Amazon EC2*, o *MobiCloud* usa uma linguagem textual de domínio específico para representação da aplicação.

O problema da migração de uma aplicação de uma plataforma de nuvem para outra é discutido em [Gonidis et al. 2013] através de um estudo de caso com uma aplicação simples de CRUD. No entanto, este é um trabalho inicial, que discute desafios, mas não propõe soluções nem no nível de código e nem no nível de modelos.

Nossa abordagem visa tratar aplicações centradas em dados desde a modelagem e assim permitir a geração automática de código referente a estas operações, sem exigir do desenvolvedor conhecimentos específicos de uma dada infraestrutura de nuvem. Além disso, nossa abordagem é baseada em modelos UML, sem propor extensões da linguagem ou empregar notações não padronizadas ou ainda linguagens textuais como a *MobiCloud*.

3. Suporte ao CRUD nas Plataformas de Nuvem

Devido a sua importância em aplicações centradas em dados, operações CRUD são suportadas pelas diferentes plataformas de nuvem disponíveis no mercado. Embora as operações suportadas pelas plataformas possuam a mesma função semântica, cada plataforma de nuvem possui uma interface de comunicação distinta das demais, dificultando o trabalho de desenvolvedores. Nesta seção, as plataformas *Google App Engine*[1] e *Bluemix*[2] são revisadas, visando detalhar aspectos relacionados a como estas plataformas tratam as operações CRUD, permitindo a compreensão da abordagem proposta e do estudo de caso.

A *Google App Engine*, por exemplo, oferece um recurso adicional para armazenamento de dados, chamado *Firebase*, e organiza os dados de forma hierárquica em um sistema de diretórios. Para acessar os dados, a aplicação utiliza referências para diretórios ou arquivos específicos, a partir das quais as operações CRUD são realizadas. Com o armazenamento organizado por diretórios, a aplicação necessita gerenciar o caminho de diretórios que precisam ser acessados para a manipulação de cada dado hospedado. Um meio usual para gerenciamento deste sistema de armazenamento, é nomear cada diretório com o atributo identificador do objeto nele armazenado. Nesse sistema, cada objeto possui uma chave de acesso - nome de diretório - para referência na infraestrutura de nuvem.

A plataforma *Bluemix*, diferentemente da *Google App Engine*, não organiza seus dados em diretórios e apenas exige que objetos a serem hospedados na nuvem sejam derivados da classe *IBMDataObject*. A partir desta relação, são herdados métodos que realizam as operações de criar, atualizar e remover o objeto na nuvem. Ao solicitar a operação de leitura de um tipo de objeto na nuvem, é retornado uma lista com todos os

[1]https://cloud.google.com/appengine/

[2]http://www.ibm.com/cloud-computing/bluemix/br-pt/

objetos deste tipo hospedados na infraestrutura, diferentemente da plataforma *Google App Engine* que permite a leitura de objetos específicos.

A plataforma *Google App Engine* oferece um conjunto de *listeners* que monitoram alterações nos dados hospedados na nuvem e mantém a aplicação com os dados atualizados. Dessa forma, a sincronização dos dados é garantida, sem a necessidade de implementação de eventos periódicos para sincronização ou preocupação com a desatualização dos dados no momento de operações locais sobre os dados.

A plataforma *Bluemix* já necessita que, periodicamente, a aplicação exclua suas listas de objetos locais e faça o *download* de todos estes objetos novamente para garantir a sincronização dos dados com a infraestrutura de nuvem. Com esta limitação pode ocorrer atraso na sincronia dos dados - quando a frequência da atualização dos dados é baixa - ou custo computacional elevado - quando a atualização é feita excessivamente.

4. Modelagem e Geração de Código: Abordagem Proposta

A abordagem proposta contempla aspectos desde a modelagem da aplicação até a geração automática de código. Nesta seção, estes dois aspectos da abordagem são discutidos, bem como a implementação de seu suporte junto a uma ferramenta de geração de código.

4.1. Modelando Aplicações CRUD Empregando a Nuvem

Tradicionalmente, diagramas UML são empregados para modelagem de *software*, representando diferentes perspectivas do *software*. Em [Parada et al. 2015], diagramas de classes e de sequência são empregados para descrever estrutura e comportamento visando a geração de código. Nossa abordagem complementa estes modelos de forma a padronizar e explicitar a interação da aplicação com a infraestrutura de nuvem.

Para a modelagem da comunicação de uma aplicação centrada em dados com a nuvem, três pontos precisam ser modelados: quais tipos de objetos serão hospedados na nuvem, quais operações CRUD serão realizadas sobre cada tipo de objeto e, também, quais classes serão responsáveis por realizar estas operações em cada tipo de objeto.

Na modelagem proposta, estes três pontos são modelados facilmente a partir, unicamente, do diagrama de classes da aplicação e de um pequeno conjunto de regras para sua construção. A identificação da classe ou classes de objetos que serão armazenados na nuvem é realizada através de uma relação de herança com uma superclasse chamada *CLOUD*. As classes derivadas de *CLOUD*, também precisam possuir um atributo chamado *key* do tipo *String*. Este atributo servirá de identificador dos dados na nuvem, sendo empregado nas buscas. As operações CRUD a serem implementadas para cada tipo de objeto devem ser representadas por métodos presentes nas classes que serão responsáveis por realizar as operações e devem conter o seguinte formato de nomenclatura: ”<operação><tipo_de_objeto>”, no qual ”<operação>”deve ser substituído por *create*, *read*, *update* ou *delete*; e ”<tipo_de_objeto>”pelo tipo a ser manipulado na operação.

As informações capturadas no diagrama de classes construído permitem a implementação automática das operações CRUD, comandos de configuração e linhas para tratamento de exceções da comunicação entre a aplicação e a infraestrutura de nuvem. Quando este modelo de classes é complementado com diagramas de sequência, a ferramenta consegue aumentar a sua capacidade de geração automática de código, visto que

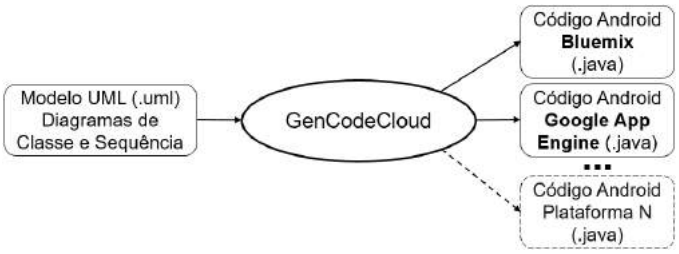

Figura 1. Fluxo de geração proposto

interações de outros objetos da aplicação com as operações CRUD poderão ser descritas, assim como mais detalhes do comportamento da aplicação pode ser capturado.

4.2. Fluxo de Geração Proposto

A Figura 1 apresenta o fluxo proposto para geração automática de código para a interação entre uma aplicação *Android* e múltiplas plataformas de nuvem. O modelo UML serve de entrada para a ferramenta GenCodeCloud capaz de gerar código para diferentes plataformas de nuvem.

Os diagramas UML são construídos na ferramenta *Papyrus*[3], que gera um único arquivo XML contendo todas as informações presentes no modelo. Primeiramente, a ferramenta GenCodeCloud realiza a leitura de todas as informações contidas nos diagramas, capturando-as em uma estrutura de classes (estilo metamodelo). Após, a ferramenta analisa as classes e métodos descritos no diagrama de classes, conforme funcionalidade herdada da ferramenta *GenCode*.

Posteriormente, a GenCodeCloud identifica os pontos de comunicação com a plataforma de nuvem, considerando as regras de modelagem propostas, para que a geração de código apropriada seja realizada. A partir de um mesmo arquivo XML de entrada, como o modelo é independente de plataforma, a ferramenta pode gerar código compatível com diferentes plataformas de nuvem. O código gerado a cada execução da ferramenta é adaptado à interface de comunicação da infraestrutura de nuvem selecionada pelo usuário.

4.3. Implementação da GenCodeCloud

A ferramenta *GenCodeCloud* é uma extensão da ferramenta *GenCode*, a qual inclui o suporte ao padrão de modelagem proposto neste trabalho e, também, à geração de código compatível com as plataformas *Google App Engine* e *Bluemix*. Esta emprega a mesma estruturação de classes da *GenCode*, porém estende algumas de suas classes visando o suporte a captura e geração da interação da aplicação com a nuvem, conforme ilustrado na Figura 2.

No pacote *Utilities* encontra-se a classe *Parser*, responsável pela captação dos dados presentes nos diagramas UML. Na implementação da *GenCodeCloud*, a classe *CloudParser* foi implementada como derivada da classe *Parser* original. A *CloudParser* sobrescreve métodos do parser para incluir a captura e interpretação da interação com a nuvem segundo a modelagem proposta neste trabalho. A classe *CloudParser* é utilizada na geração de todos os projetos que utilizam armazenamento na nuvem e seus métodos são independentes da plataforma de nuvem selecionada pelo usuário.

[3]https://eclipse.org/papyrus/

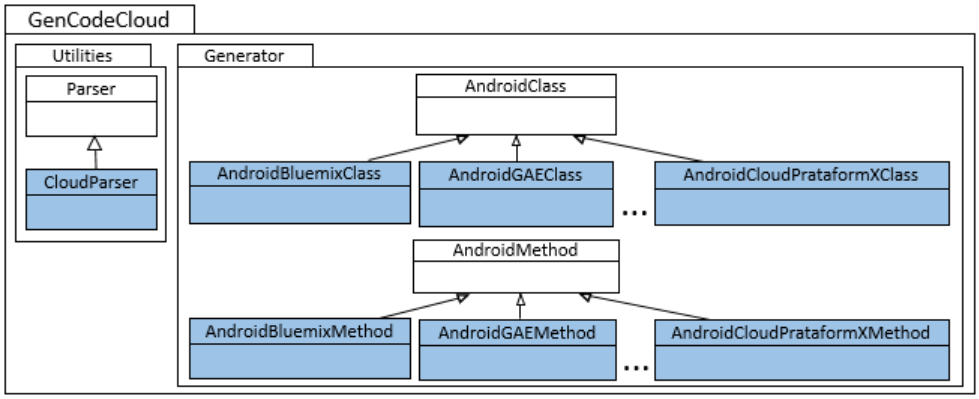

Figura 2. Estrutura da GenCodeCloud - extensão da *GenCode*

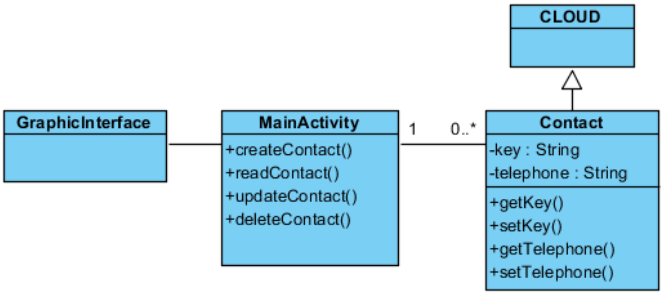

Figura 3. Diagrama de classes simplificado da aplicação Lista de Contatos

No pacote *Generator*, a classe *AndroidClass* realiza a geração de código nativo *Android* para as classes da aplicação. Para a versão *GenCodeCloud*, as classes *AndroidBluemixClass* e *AndroidGAEClass* foram criadas como derivadas de *AndroidClass* e sobrescrevem os principais métodos da superclasse visando implementar as classes da aplicação compatíveis com *Bluemix* e *Google App Engine*, respectivamente. Quando nenhuma plataforma de nuvem é selecionada pelo usuário, a classe *AndroidClass* padrão é utilizada, gerando código da mesma forma como feito na versão *GenCode* original.

Ainda no pacote *Generator*, a classe *AndroidMethod* é responsável pela geração de código para os métodos da aplicação. Para a versão *GenCodeCloud*, as classes *AndroidBluemixMethod* e *AndroidGAEMethod* foram criadas como derivadas da *AndroidMethod*, sobrescrevendo os seus principais métodos de forma a implementar os métodos da aplicação compatíveis com as plataformas *Bluemix* e *Google App Engine*, respectivamente. Objetos da classe *AndroidMethod* ou de suas subclasses são criados pela classe *AndroidClass* ou suas derivadas, dependendo do tipo de geração de código solicitada. No caso da geração de código para a plataforma BlueMix, por exemplo, a classe *AndroidBluemixClass* instancia objetos *AndroidBluemixMethod* para gerar métodos da aplicação.

5. Estudo de Caso

Para ilustrar o uso da abordagem padronizada para modelagem de aplicações CRUD proposta neste trabalho e o seu emprego para geração automática de código, uma aplicação *Android* chamada *Lista de Contatos* foi empregada como estudo de caso. Esta aplicação hospeda na nuvem uma lista de contatos com informações de nome e telefone; e permite

```
51      public void createContact(){
52          String key = "SET KEY";
53          String telephone = "SET TELEPHONE";
54          if (!key.equals("")) {
55              Contact contact = new Contact();
56              contact.setKey(key);
57              contact.setTelephone(telephone);
58              createContactCloud(contact);
59           }
60      }
```

Figura 4. Código independente de plataforma para a criação de um novo contato.

```
95  private void createContactCloud(Contact contact){
96          contact.save().continueWith(new Continuation<IBMDataObject, Void>() {
97              @Override
98              public Void then(Task<IBMDataObject> task) throws Exception {
99                  if (task.isCancelled()){
100                     Log.e(CLASS_NAME,"Exception : Task "+task.toString()+" was cancelled.");
101                 }
102                 else if (task.isFaulted())
103                     Log.e(CLASS_NAME, "Exception : " + task.getError().getMessage());
104                 else listContacts();
105                 return null;
106             }
107         });
108 }
```

Figura 5. Código para *Bluemix* para armazenamento de um contato.

```
90      private void createContactCloud(Contact contact){
91              mFirebaseRef.child(contact.getKey()).setValue(contact);
92      }
```

Figura 6. Código para *Google App Engine* para armazenamento de um contato.

ao usuário visualizar, cadastrar, pesquisar, editar e excluir estes contatos.

A Figura 3 apresenta o diagrama de classes simplificado da aplicação Lista de Contatos construído seguindo a abordagem de modelagem proposta. Os objetos do tipo *Contact* serão hospedados na nuvem, pois foram indicados como subclasses de CLOUD e a classe *MainActivity* realizará as operações de criar, ler, atualizar e remover objetos do tipo *Contact* da infraestrutura de nuvem. Objetos do tipo *Contact* possuem o atributo *key* que será utilizado para armazenar o nome de cada contato e este será utilizado para referenciar o objeto nas buscas na infraestrutura de nuvem.

Usando como entrada o diagrama de classes da Figura 3, a ferramenta *GenCodeCloud* interpreta os elementos presentes neste modelo, identificando classes, métodos e atributos. Na sequência, caso selecionado o uso de alguma plataforma de nuvem, a ferramenta identifica os elementos que modelam a comunicação entre a aplicação e a plataforma de nuvem e realiza a implementação das operações CRUD, bem como, comandos de configuração e tratamento de exceção específicos para a plataforma selecionada.

Cada operação CRUD implementada é dividida em dois métodos distintos, o primeiro é independente de plataforma de nuvem e é responsável pela manipulação dos objetos antes da interação com a nuvem, o segundo é totalmente dependente de plataforma de nuvem e é responsável pela comunicação direta com a nuvem. A Figura 4 apresenta o método independente de plataforma *createContact* para a operação de criar um novo contato na nuvem para a aplicação Lista de Contatos. Neste método, os atributos do novo objeto *Contact* são inicializados nas linhas 52 e 53. Na sequência, o objeto é criado nati-

```
135 public void listContacts() {
136     try {
137         IBMQuery<Contact> query = IBMQuery.queryForClass(Contact.class);
138         query.find().continueWith(new Continuation<ArrayList<Contact>, Void>() {
139         @Override
140             public Void then(Task<List<Contact>> task) throws Exception {
141                 final ArrayList<Contact> objects = task.getResult();
142                 if (task.isCancelled())
143                     Log.e(CLASS_NAME,"Exception:Task "+task.toString()+" was cancelled.");
144                 else if (task.isFaulted())
145                     Log.e(CLASS_NAME, "Exception : " + task.getError().getMessage());
146                 else {
147                     contactList.clear();
148                     for(IBMDataObject contact:objects)contactList.add((Contact) contact);
149                 }
150                 return null;
151             }
152         },Task.UI_THREAD_EXECUTOR);
153     }  catch (IBMDataException error) {
154         Log.e(CLASS_NAME, "Exception : " + error.getMessage());
155     }
156 }
```

Figura 7. Código para *Bluemix* para busca de modificações de dados na nuvem.

```
29 mFirebaseRef.addChildEventListener(new ChildEventListener() {
30     public void onChildAdded(DataSnapshot snapshot, String previousChildKey) {
31         Contact newContact = snapshot.getValue(Contact.class);
32         contactList.add(newContact);
33     }
34     public void onChildRemoved(DataSnapshot snapshot) {
35         Contact contactRemoved = (Contact) snapshot.getValue();
36         for(Contact contact : contactList){
37             if(contact.getKey().equals(contactRemoved.getKey()))
38                 contactList.remove(contact);
39         }
40 });
```

Figura 8. Código para *Google App Engine* busca de modificações de dados na nuvem.

vamente e é invocado o método *createContactCloud*, na linha 58, para o armazenamento deste novo objeto na nuvem. A implementação deste método é dependente de plataforma.

Quando selecionada a plataforma *Bluemix*, o método *createContact* invoca a implementação do método *createContactCloud* ilustrada na Figura 5, específica para a referida plataforma. Na operação de criar um novo contato na nuvem, o comando *save*, localizado na linha 96 do trecho de código da Figura 5, é responsável por esta operação. Após a invocação do *save*, são empregados comandos para tratamento de exceções no trecho entre as linhas 98 e 107.

Quando selecionada a plataforma *Google App Engine*, o método *createContact* invoca a implementação do método *createContactCloud* ilustrada na Figura 6. No comando da linha 91 deste método, o novo objeto do tipo *Contact* é hospedado na nuvem empregando o serviço *Firebase*. O novo objeto contato é armazenado em um novo diretório, o qual terá o atributo *key* do objeto como chave de acesso.

Para a aplicação Lista de Contatos, a ferramenta *GenCodeCloud*, além de implementar as operações CRUD, também adiciona um atributo do tipo *ArrayList* chamado *ContactList* à classe *MainActivity*, por esta ser a classe responsável pela manipulação dos objetos do tipo *Contact* na nuvem. Este atributo é responsável pelo armazenamento local - e temporário - da lista de contatos. Os comandos de atualização desta lista também são gerados automaticamente pela ferramenta.

A Figura 7 apresenta o método *listContacts* responsável por preencher o *ArrayList contactList* com a lista de contatos mais recente hospedada na nuvem quando selecionada a plataforma *Bluemix*. No comando da linha 141, os objetos hospedados em nuvem são copiados para a memória local do dispositivo e adicionados para a *listContacts* na linha 148. O método *listContacts* é invocado na inicialização da aplicação e, também, após cada operação CRUD, de modo a manter a lista local de contatos atualizada.

A Figura 8 apresenta a implementação do método *listContacts* quando selecionada a plataforma *Google App Engine*. Diferentemente da implementação para a *Bluemix*, a implementação para a plataforma da *Google* emprega um conjunto de *listeners*, invocados automaticamente toda vez que há alguma alteração nos dados presentes na nuvem. Estes *listeners* são implementados no método *onCreate* ou método construtor da classe responsável pelas operações CRUD. O *listener* ilustrado na linha 30 adiciona novos contatos hospedados na nuvem e o *listener* da linha 34 remove objetos da lista de contato local quando estes forem removidos da infraestrutura de nuvem.

A partir, unicamente, do diagrama de classes da Figura 3, a *GenCodeCloud* faz a geração de 150 linhas para a classe *MainActivity* quando selecionada a plataforma *Bluemix* e 92 linhas quando a *Google App Engine* é empregada. Na classe *Contact* são geradas 33 linhas para *Google App Engine* e 34 para a plataforma *Bluemix*. Por não possuir interação direta com a infraestrutura de nuvem, a geração da classe *GraficInterface* é feita da mesma forma como realizada na versão *GenCode* original, gerando 6 linhas de código, independentes da plataforma de nuvem empregada.

Vale destacar que quando, juntamente ao diagrama de classes, são empregados diagramas de sequência no modelo de entrada, a ferramenta *GenCodeCloud* é capaz de interpretar a modelagem comportamental da aplicação e ampliar a sua geração de código. Dessa forma, por exemplo, a ferramenta pode gerar também trechos de código onde os métodos responsáveis pelas operações CRUD independentes de plataforma são invocados por outros métodos presentes em outras classes da aplicação.

6. Conclusão

Este trabalho propôs uma abordagem para modelagem e geração automática de código para aplicações móveis centradas em dados armazenados em infraestruturas de nuvem. Nossa abordagem de modelagem suporta a abstração da interação do aplicativo com a nuvem, através de regras que servem de padronização.

O objetivo é suportar a geração de código para diferentes plataformas de nuvem a partir de um mesmo modelo de alto nível, e assim simplificar tanto o desenvolvimento de aplicativos quanto a migração entre plataformas. A geração de código proposta também é discutida em detalhes neste artigo, bem como discute-se a implementação deste suporte em uma ferramenta de geração de código *Android* a partir de modelos UML. Esta implementação é realizada através da extensão da ferramenta *GenCode*, criando assim a *GenCodeCloud*. Através de um estudo de caso foi demonstrado o emprego da abordagem proposta e são discutidas linhas de código geradas automaticamente pela *GenCodeCloud*. As linhas de código geradas incluem suporte às operações CRUD nas plataformas *Google App Engine* e *Bluemix*.

Como trabalhos futuros, novos e mais complexos estudos de caso serão realizados,

incluindo validações dos códigos gerados pela ferramenta. Além disso, o suporte a outras plataformas de nuvem está previsto como trabalho futuro.

Referências

Agrawal, N., Aranya, A., and Ungureanu, C. (2013). Mobile data sync in a blink. In *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*, San Jose, CA. USENIX.

Chun, B.-G., Curino, C., Sears, R., Shraer, A., Madden, S., and Ramakrishnan, R. (2012). Mobius: Unified messaging and data serving for mobile apps. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 141–154, New York, NY, USA. ACM.

CISCO (2016). Cisco visual networking index: Global mobile data traffic forecast update, 2015-2020 white paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611.

Gonidis, F., Simons, A. J. H., Paraskakis, I., and Kourtesis, D. (2013). Cloud application portability: An initial view. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 275–282, New York, NY, USA. ACM.

Papotti, P. (2013). Um processo dirigido a modelos para geração de código. In *Dissertação de Pós-Graduação em Ciência da Computação – PPGCC*. UFSCar.

Parada, A., Marques, M., and de Brisolara, L. B. (2015). Automating mobile application development: Uml-based code generation for android and windows phone. *Revista de Informática Teórica e Aplicada*, 22(2):31–50.

Parada, A., Tonini, A., and Brisolara, L. (2013). Geração automática de código android eficiente a partir de modelos uml. In *Proceedings of the 16th Conferencia Iberoamericana en Software Engineering*, CIbSE '13, pages 71–84.

Ranabahu, A. H., Maximilien, E. M., Sheth, A. P., and Thirunarayan, K. (2011). A domain specific language for enterprise grade cloud-mobile hybrid applications. In *Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11*, SPLASH '11 Workshops, pages 77–84, New York, NY, USA. ACM.

Ribeiro, A., da Silva, A. R., et al. (2014). Evaluation of xis-mobile, a domain specific language for mobile application development. *Journal of Software Engineering and Applications*, 7(11):906–919.

Xanthopoulos, S. and Xinogalos, S. (2013). A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 213–220, New York, NY, USA. ACM.