GERAÇÃO DE CÓDIGO ANDROID A PARTIR DE MODELOS UML

PARADA, Abilio; BRISOLARA, Lisane

Universidade Federal de Pelotas, Centro de Desenvolvimento Tecnológico
{agparada, lisane}@inf.ufpel.edu.br

1 INTRODUÇÃO

Dispositivos móveis, como os *SmartPhones*, estão se tornando uma opção ao uso de computadores pessoais [Wang, 2011]. Muitos destes dispositivos utilizam sistema operacional Android, o qual vem se destacando dentre os demais, como sendo uma alternativa completa, com ferramentas de apoio ao desenvolvedor e amplo suporte para os mais diversos equipamentos [Android, 2012].

O crescente uso destes dispositivos vem impulsionando o mercado de desenvolvimento de software para este nicho e aumentando a demanda por novos aplicativos, que devem ser desenvolvidos em curto período. Porém, o projeto de aplicações móveis incluem novos desafios, além das restrições impostas pelo projeto de aplicações para sistemas embarcados como, gerenciamento de memória e consumo de energia, o que aumenta a complexidade dos projetos.

Uma das soluções para lidar com projetos complexos é o uso de modelos [Selic, 2003]. A UML se destaca como a linguagem padrão de modelagem e promoveu a Engenharia Orientada a Modelos (MDE), que considera os modelos como principais artefatos do processo de software. Desta forma, a MDE proporciona abstração, permite diferentes visões do sistema, e diminui o tempo de desenvolvimento de aplicativos [Terrier, 2007]. Em [Wang, 2011], o emprego da MDE no desenvolvimento para dispositivos móveis é defendido. O uso deste paradigma pode proporcionar uma redução de até 70% no tempo de desenvolvimento, diminuindo assim os custos do projeto [Weigert, 2011].

No entanto, o uso efetivo de MDE requer ferramentas que capturem modelos, transformem esses modelos, e gerem código. Algumas ferramentas comercias como a IBM Rational Rhapsody [IBM, 2012] e outras livres como AndroMate [Telematica Instituut, 2012], já suportam a modelagem e automatizam o desenvolvimento de aplicativos Android. Porém, muitas vezes estas são meramente ferramentas de programação visual, e não fazem uso dos benefícios da UML padrão.

Este artigo apresenta uma ferramenta que facilita e acelera o desenvolvimento de aplicações Android, através do suporte ao paradigma MDE. A ferramenta permite o uso das notações da UML padrão para a modelagem de aplicações móveis, sem redefinições ou uso de estereótipos, e provê a geração de código automática a partir do modelo UML.

O artigo está organizado da seguinte forma. A Seção 2 apresenta a geração de código proposta, bem como o estudo de caso realizado. Trabalhos relacionados são discutidos na Seção 3, enquanto conclusões são apresentadas na Seção 4.

2 MATERIAL E MÉTODOS

Esta seção apresenta detalhes sobre a ferramenta proposta, bem como os procedimentos adotados para seu desenvolvimento e um estudo de caso usado para demonstrar as facilidades providas pela ferramenta.

2.1 Desenvolvimento da Ferramenta

A nova ferramenta baseou-se em trabalhos anteriores de nosso grupo de pesquisa, onde uma ferramenta chamada GenCode [Parada, 2011] foi desenvolvida, a qual gera código Java, estrutural e comportamental, a partir de diagramas UML. Assim como a GenCode, a nova ferramenta captura as informações do modelo UML e gera do código Java. A parte estrutural (ou estática) do código é gerada a partir de informações do diagrama de classes e a parte comportamental baseia-se em diagramas de sequência, podendo gerar chamadas de métodos, condicionais, laços, dentre outros aspectos dinâmicos do código.

No entanto, algumas especificidades devem ser consideradas na geração de código, a medida que o Java-Android não é totalmente igual ao Java padrão, visto que uma API específica é usada para desenvolvimento de aplicativos Android. Nesta API, conceitos como Atividade (*Activity)* e Serviço (*Service*) são definidos. Uma *Activity* é utilizada quando a aplicação ou trecho de código irá executar uma atividade relacionada com a interface gráfica, enquanto um serviço é responsável pelas atividades executadas em segundo plano. No desenvolvimento de aplicativos estes conceitos podem ser especializados e redefinidos. A ferramenta captura estas redefinições a partir do relacionamento de herança entre a classe da aplicação e as classes *Activity* ou *Service*, gerando assim a declaração de *extends* no código Java.

A ferramenta também gera a declaração de *imports* de acordo com os atributos ou parâmetros das classes, assim como gera chamadas de métodos padrão na geração dos métodos do ciclo de vida da aplicação. Os métodos padrão de uma *Activity* são *onCreate*, *onStart*, *onResume*, *onStop* e *onDestroy*. O método *onCreate* descreve as primeiras atividades quando a aplicação é iniciada. O *onStart é* responsável por colocar a atividade em primeiro plano na tela, enquanto que *onResume* a coloca em segundo plano. Já o *onStop é* invocado quando a atividade não é mais visível e o *onDestroy* para que a aplicação execute as ultimas atividades antes de ser fechada. Em um serviço, o método *onBind* realiza uma conexão persistente com o serviço e *onDestroy* do *Service* é similar ao *onDestroy* da *Activity*.

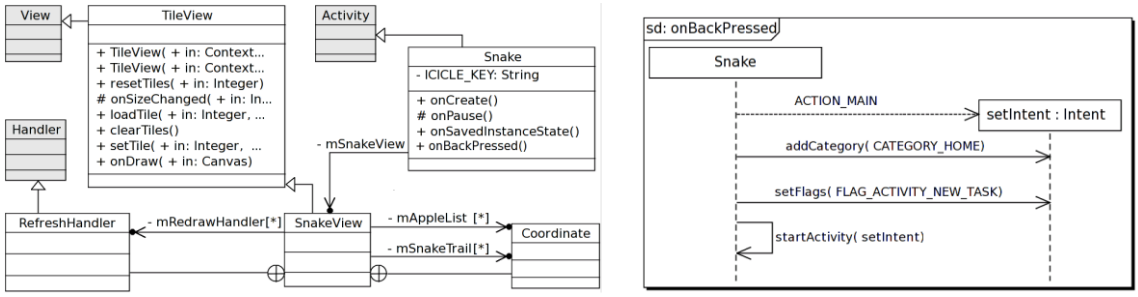
Além de atividade e serviço, aplicações Android também fazem uso de Intenção e Provedor de Conteúdo, componentes específicos para comunicação entre funcionalidades (outra aplicação ou recurso do dispositivo) e dados. O uso de uma intenção requer a criação de um objeto *Intent* e a indicação da atividade requerida, a fim de que o serviço apropriado seja invocado. Em nossa abordagem, o uso deste conceito é descrito em um diagrama de sequência, onde a criação da *Intent* é representada por uma mensagem de criação, contendo como argumento a atividade requerida. Uma solicitação de dados, tratada pelo Provedor de Conteúdo, também pode ser capturada a partir do diagrama de sequência usando uma mensagem contendo como argumento o dado solicitado.

2.2 Estudo de Caso

O estudo de caso demonstra o apoio dado por nossa ferramenta. Para construir o modelo, foi aplicada a engenharia reversa no código da aplicação Snake, disponível em [Android, 2012]. O modelo UML foi construído na Papyrus [Papyrus, 2012] e consiste em um diagrama de classe e vários de sequência. No entanto, pela limitação de páginas, no artigo apenas um diagrama comportamental é ilustrado.

O diagrama de classes ilustrado na Fig 1-a representa a estrutura da aplicação, a qual é composta de cinco classes, sendo duas delas internas à classe *SnakeView.* As classes em cinza representam componentes da API Android, sendo estes reusados para definir classes da aplicação, o que é indicado atráves do relacionamento de herança. Por exemplo, a classe *Snake*, redefine a classe *Activity*. Na classe Snake é importante destacar o uso de métodos que descrevem o comportamento da aplicação como *onCreate*, e também a existência de associação com *SnakeView*, e também atributos com valor padrão.

O comportamento do método *onBackPressed* da Snake é descrito através do diagrama de sequência da Fig 1-b, onde uma *intent* é usada. A sequência é iniciada pela criação do objeto *Intent,* representada pela linha tracejada, e que inclui argumento *ACTION_MAIN* que indica a intenção desejada. Após, caracteristicas são adicionadas à intenção através de *addCategory* e *setFlags*. Por fim, é invocado o serviço *startActivity* tendo como argumento a intenção criada (*setIntent*).



(a) (b)

Figura 1. Diag. de Classes da *Snake* (a); Diag. de sequência do método *onBackPressed* (b)

Para demonstração da ferramenta, foi gerado código a partir dos diagramas que representam a estrutura e comportamento do aplicativo. Os fragmentos de código gerados para a classe *Snake* são ilustrados na Fig. 2. Nas linhas 1-3, observa-se a declaração dos *imports*, na 5 a extensão de *Activity* e nas linhas 7-8, os atributos. A Fig. 2 também inclui trechos gerados a partir do diagrama da Fig.1-b, incluindo a criação e especificação da intenção nomeada *setIntent* na linha 63 e a invocação do serviço na linha 66, geradas automaticamente a partir do modelo.

```
1 import android.app.Activity;
2 import android.content.Intent;
3 import android.os.Bundle;
4
5 public class Snake extends Activity{
6   /**Attributes */
7   private SnakeView mSnakeView;
8   private String ICICLE_KEY = "snake_view";
```

```
61  public void onBackPressed() {
62     /** Specified by sd onBackPressed */
63     Intent setIntent = new Intent(Intent.ACTION_MAIN);
64     setIntent.addCategory(Intent.CATEGORY_HOME);
65     setIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
66     startActivity(setIntent);
67  }
```

Figura 2. Fragmentos do código gerado para a classe *Snake*

3 RESULTADOS E DISCUSSÃO

O trabalho propõe uma ferramenta para suportar MDE no desenvolvimento de aplicações Android. Poucas ferramentas já suportam ao desenvolvimento para Android. Dentre as ferramentas disponíveis, destacam-se a IBM Rational Rhapsody [IBM, 2012] e AndroMate [Telematica Instituut, 2012]. A primeira delas requer o uso de extensões da UML, enquanto a AndroMate baseia-se em programação visual.

A ferramenta proposta neste artigo baseia-se na UML padrão, evitando uso de novas notações que dificultariam o trabalho dos projetistas já habituados ao padrão. Quanto a automação provida pela ferramenta, o estudo de caso demonstra o tipo de linhas de código geradas automaticamente a partir dos modelos.

4 CONCLUSÃO

Este artigo apresenta uma ferramenta de geração de código para aplicativos Android a partir de modelos UML. Esta ferramenta suporta o uso do paradigma MDE no desenvolvimento de aplicações móveis. A ferramenta proposta baseia-se no uso de notações padrão da UML, sem novas definições. Desta forma, metodologias já usadas na modelagem de software tradicional podem ser adotadas também por desenvolvedores de aplicativos móveis. Contudo a geração proposta não faz uso de nenhuma otimização de código. Como trabalho futuro, serão estudadas otimizações que poderiam ser aplicadas automaticamente na geração de código Android.

5 REFERÊNCIAS

ANDROID. Anatomia de uma aplicação Android. Disponível em: <http://code.google.com/android/intro/anatomy.html>. Acesso em: 13 jul. 2012.

IBM. Rational Rhapsody. Disponível em: <http://www.ibm.com/software/awdtools/rhapsody/>. Acesso em: 13 jul. 2012.

PARADA, A.; SIEGERT, E.; BRISOLARA, L. Generating Java code from UML Class and Sequence Diagrams. In: WORKSHOP DE SISTEMAS EMBARCADOS, Florianópolis, 2011.

PAPYRUS. Papyrus, Eclipse Modeling. Disponível em: <http://www.eclipse.org/modeling/mdt/papyrus>. Acesso em: 13 jul. 2012.

SELIC, B. Models, software models, and UML. In: LAVAGNO, L.; MARTIN, G.; SELIC, B. (Ed.) UML for real. Boston: Kluwer Academic Publishers, 2003. p.1-16.

TELEMATICA INSTITUUT. AndroMate. Disponível em: <http://www.lab.telin.nl/~msteen/andromate/>. Acesso em: 13 jul. 2012.

WANG, Z., The study of smart phone development based on UML, In: INT. CONF. ON COMPUTER SCIENCE AND SERVICE SYSTEM, Nanjing, China, 2011.

WEIGERT, T., Practical experiences in using model-driven engineering to develop trustworthy Computing Systems. In: IEEE INT. CONF. ON SENSOR NETWORKS, UBIQUITOUS, AND TRUSTWORTHY COMPUTING, Taichung, Taiwan, 2006.