Automating mobile application development: UML-based code generation for Android and Windows Phone

Abilio G. Parada [1]
Milena R. S. Marques [2]
Lisane B. de Brisolara [2]

Data submissão: 01.06.2015
Data aceitação: 01.11.2015

Abstract: This paper proposes a MDD approach for mobile application development, which includes modeling and code generation strategies for Android and Windows Phone. UML class and sequence diagrams are employed for modeling mobile applications and code is generated from this model. To support the automatic code generation, GenCode was re-structured and extended to meet the particularities of these two platforms. As result, GenCode's current version is able to automatically generate Java-Android and C# codes, according to the specified application model and target platform. Finally, case studies are used to demonstrate the proposed approach, as well as to validate the code generation tool.

1 Introduction

The mobile devices market is very attractive, motivating big companies, such as Google, Microsoft, and Apple, to invest in the development of mobile. Market researches have highlighted Android [5] and Windows Phone [14] as the most promising platforms, mainly due to the increasing number of device manufacturers supporting these platforms [3] [9].

A few years ago communication devices, like cell-phones, were used only to make calls. Nowadays, due to technology advances, these devices are seen as computation devices. Thus, the demand for new and sophisticated applications to run on it has increased, mainly due to Internet connection and resources available at cloud [26].

[1]Instituto de Informática, UFRGS
{agparada@inf.ufrgs.br}
[2]Centro de Desenvolvimento Tecnológico, UFPel
{mrsmarques,lisane}@inf.ufpel.edu.br

Mobile application developers are constrained by time-to-market pressure and more and more developers are requested to develop applications for multiple platforms. Each platform provides specific tools and usually adopts different programming languages and APIs, which turns hard developing for multi-platforms [18] [22].

The increasing sophistication of mobile applications allied to time-to-market pressure motivates the applying of model-driven development (MDD) on this domain. Employing MDD, models are used to guide development and code can be automatically generated from model [21] [25].

However, traditional modeling approach and available tools are considered unsuitable for mobile applications, because do not consider particularities as application life-cycles and concepts from the target language. In order to solve this problem, recently, tools and approaches are been proposed for the mobile domain [12]. Most of these efforts are focused on Android, but few and very initial efforts have proposed with focus on Windows Phone [23]. It is important to highlight that the available approaches usually focus on a specific platform and provide limited code generation support.

This paper proposes a MDD approach for mobile applications development, which includes UML-based modeling and code generation strategies for two of the most promising mobile platforms on the market, Android and Windows Phone. Following our approach, structure and behavior of mobile applications are represented by UML diagrams, using class and sequence diagrams, respectively. The resulting UML-based model is automatically translated to Java-Android or C# codes in order to obtain an implementation.

The remaining of this paper is organized as follows. Section 2 discusses related work. The proposed approach and code generator are presented in Section 3. Section 4 demonstrates our approach through two case studies. Section 5 draws main conclusions and introduces future work.

2 Related Work

Recently, some research groups have proposed specific MDD approaches for mobile application development [19] [12] [23] [16]. Most of these are focused on Android as Arctis [12] and GenCode [19]. In Arctis, a small UML profile is proposed and UML activity diagrams are adopted, which are translated to a state machine to achieve an executable Android application. In turn, GenCode uses UML sequence diagrams to represent application behavior, and adopts the UML standard notation. This work is an extension of GenCode, and thus shares the same principles,

allowing the use of any UML tool for application modeling and not requiring any new knowledge.

For supporting development for Windows Phone (WP), Microsoft provides the Visual Studio [13], which allows build UML models and generate code from them. However, this tool only supports class diagrams and, consequently its generator is limited to structural code. Some academic works addressing Windows Phone applications modeling can be found. For example, Min et al. [16] propose an extended metamodel for WP application modeling based on the Model View Controller framework, which focuses also only on structural aspects. In another effort, Son et al. [23] propose an approach for modeling WP applications based on class and sequence diagrams, similar to our approach. However, the authors propose to firstly model application using new stereotypes and then transform this model in a target specific model. While these works focus on application modeling, our approach also handles automatic code generation for Windows Phone platform.

The diversity of mobile platforms and the need of develop applications for different platforms have motivated the investigation of cross-platform tools. Many of these solutions are based on web technologies such as HTML5, PHP, Ruby, CSS, and JavaScript as RhomobileSuite [17], IBM MobileFirst Platform Foundation [8], Appcelerator [1], Telerik AppBuilder [24], and PhoneGap [20]. In another effort, named Xamarin [27], developers are encouraged to use C# for application code and transform this description on the final target language, offering support for iOS, Android and Windows Phone. However, all these approaches are focused directly on the coding (implementation step) and do not provide any support for analysis and design based on models.

More recently, MD2 [7] and AXIOM [10] were proposed as approaches for model-driven cross-platform development. Following MD2, developers specify an app using a high-level domain-specific language (DSL) and from this specification, native apps for Android and iOS are automatically generated. This specification is textual and not graphical, what can be seen as a limitation since modeling languages are often expected to be graphical languages [2]. Similar to MD2, AXIOM also uses a DSL to define platform-independent models. In this approach, the apps specification is firstly transformed to an application model (platform-independent), which is converted to an implementation model and finally to code. AXIOM uses a model representation, called an Abstract Model Tree, as the basis for all model transformations and code generation. However, the input model is built using Groovy and this model is not a pattern modeling language for software developers. In contrast to the other cross-platform approaches, our approach advocates the use of UML as modeling language in order to avoid any specific language designer

may not be familiar with. Additionally, following our approach, from an UML model composed of class and sequence diagrams, code is generated to Android and Windows Phone.

3 Proposed Approach

In order to facilitate and accelerate mobile apps development, we propose a MDD approach, which supports UML-based code generation for Android and Windows Phone. To automate our approach, we extend our code generator named GenCode, whose first version were able to generate standard Java code from UML diagrams. This code generator is detailed in Section 3.1. Our approach considers particularities of the target platforms, since the modeling until the code, which are detailed in sections 3.2 and 3.3.

3.1 Code Generation Tool

GenCode is an open source initiative proposed by our research group to support MDD. This tool is available in GitHub [4]. Firstly, this tool only generated Java code from UML diagrams. Currently, this tool focuses on mobile applications development and is able to generate Android and Windows Phone code according to an UML model.

The captured model is loaded in a class structure named Model, which is composed of two packages: Structure and Sequence. The Structure package contains classes responsible to load information from the class diagram (cd), while the Sequence one is responsible to load the behavioral view from sequence diagrams (sd).

After model capturing, Model invokes the code generation step supported by the Generator package. To support different code generators, this package defines an interface that should be implemented by each generator, following the Strategy design pattern. Currently, GenCode has two generators AndroidGenerator and CSharpGenerator. Both generators, firstly, create a directory before start the structural code generation and after that, generate behavioral code. GenCode is able to generate code for methods, including its body, if sd is used to describe it.

3.2 Supporting Android apps development

As mentioned before, the development of Android applications has its own features, which should be considered during modeling and code generation. In the

modeling, traditional object-oriented design elements are used, but specific elements from the Android API should be also included. Among them, the most important is the *Activity* class, once it is always present in any Android application. This class handles the graphical user interface (GUI) and also the application life cycle. In our approach, its usage should be specified through the specialization of this Android class by application classes, as illustrated in Fig. 1a, where *MainClass* specializes the *Activity* to define a graphical interface.

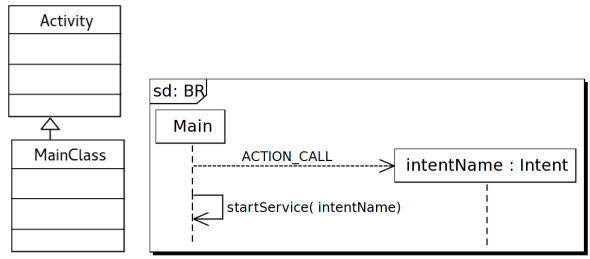

Figura 1: Activity (a). Modeling the use of an Intent (b).

Activity has a set of methods in charge of the Android application life cycle, including methods like *onCreate*, *onStart*, *onStop*, *onResume*, and *onClose*. In our approach, *onCreate* should be always implicitly declared in the model, while other methods should be declared only when required by the application. These redefined methods or new methods should be detailed in a sequence diagram in order to enable GenCode to produce code for them.

Regarding behavior modeling, sequence diagrams are adopted, in which beyond the traditional elements, also are supported Android behavioral elements. *Intent* is one of these elements, which is used to invoke services from applications or from device resources. In this example, from Fig. 1, an Intent object is created by the *Main* object, and the action to be invoked is described by the *ACTION_CALL* message. After that, *Main* invokes the required service (e. g. startService) using the intent as parameter.

As already highlighted, the structural and behavioral model provide information for code generation, the final goal of our approach. Starting by the structural view, the AndroidGenerator creates a ".java" file for each class or interface found into the model. Some special classes from Android API, such as *Activity* and *Service*, are ignored in this process.

During the generation for a given class, firstly, "imports" are all generated, ac-

cording to API elements, attributes, parameters and other elements used in the model. In the sequence the class header is generated, which includes visibility, modifiers, class name, as well as specialization/generalization or realization relationships. In the next step, the attributes should be described. These can be explicitly specified in the model or can be originated from an association. For each attribute a line of code is generated defining its visibility, type and name. Attributes are used in the constructor generation, where these are initialized. Further, attributes are also used to create the access methods, *setters* and *getters*, generated for all private or protected attributes.

For methods whose class specializes *Activity*, all statements of *onCreate* method are generated. For other methods of Android API, a default signature and also some invoked methods are automatically generated. Finally, for methods whose behavior is specified in a sd, the sequence of message exchanges are generated, including conditionals and loops. When the sequence diagram represents a Android particularity, such an *Intent*, GenCode provides a special generation for this element, which will be demonstrated on Section 4.

3.3 Supporting WP apps development

Windows Phone developers, as well as Android developers, should consider platform-dependent aspects when creating new WP apps. One of these particularities is the *PhoneApplicationPage* class, which describes the GUI for a WP application. Similar to Activity, *PhoneApplicationPage* is used in the application structure and also is specified through a specialization relationship. Fig. 2 exemplifies how this class is used to define WP application classes, during the application's structure modeling. The App, another WP class, deals with application life cycle. We propose to omit this class from modeling and consider it only directly in the code generation.

Another particularity of WP applications is the use of *partial* classes. A partial class has a logical part and a view part and requires a special code generation. In our approach, ≪*Partial*≫ label should be used to indicate that a class is partial, as exemplified in Fig. 2. In this example, *MainPage* is a partial class, and its logical side is responsible for application initialization, while its visual side is in charge of splash screen.

All these mentioned particularities are concentrated on application's structure because differently of Android, behavioral modeling of WP applications only uses sd standard notations.

Our code generation for WP applications is very similar to that employed for Android applications, been initialized by the Model class. During classes generation

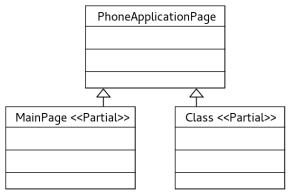

Figura 2: Structural view for a Windows Phone application.

GenCode analyzes class type, once for partial ones should be generated both visual and logical parts. Visual side is described using XAML, which specifies a basic structure for GUI, while logical part is described by C#. Thus, when the class is partial, GenCode creates a ".xaml" and a ".xaml.cs" files for the logical part. Otherwise, only a ".cs" file for is generated.

To generate the logical part of a class, firstly GenCode defines "using" statements, according to class needs, considering attributes and parameters. Following, "namespace" statement are generated for the package declaration. After that, inner classes are specified and then, the class header is generated, including modifiers and generalization/specialization relationship. Thereafter, GenCode generates attributes and methods for this class. In a C# class, attributes can be followed by a directive to specify the access level, such as private and protected. Yet from the cd, GenCode generates the method signatures, considering their configurations included into the model. However, methods bodies are generated from their behavior described using sd without any special consideration, as already proposed in [19] but using C# syntax, instead of Java.

The particularities of WP code generation are summarized by the support for partial classes as, and for the *PhoneApplicationPage* and *App* classes. *PhoneApplicationPage* is used by GenCode just to generate code for their subclasses, since this class is already defined by the Windows Phone API. Finally, following our approach, the *App* class can be omitted from the application model and its code is automatically generated by our tool, once its code follows a standard.

4 Case Studies

To demonstrate our approach, two case studies are presented, which address mobile applications modeling and code generation for the target platforms. For both case studies, the UML models were manually created from source code by reverse engineering.

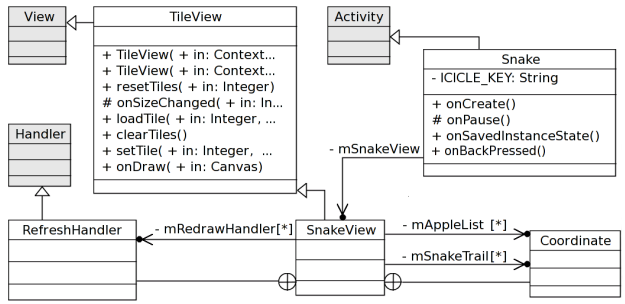

Figura 3: Class diagram - Structural view of Snake.

4.1 Developing for Android: a case study

Snake is an application from the Android developer training and available on the Android web site [6]. This application is a simple example of Android application and allows demonstrate the main features of our approach.

Firstly, the Snake structural view was specified using a cd, as illustrated in Fig. 3. In this diagram, Android classes such as *Activity*, *View*, and *Handler* are highlighted in gray, while the application classes, such as *Snake*, *SnakeView*, and *TileView*, and two *SnakeView* inner classes (*RefreshHandler* and *Coordinate*) are illustrated in white.

Snake is the main application class, managing the GUI, application initialization, and life-cycle, as indicated by the *Activity* specialization. This class has an association with the *SnakeView* class. Moreover, Snake has some operations such *onCreate*, *onPause*, *onSavedInstanceState*, and *onBackPressed*, which are standard *Activity*'s methods. As proposed, their explicit definitions indicate that these will be customized.

TileView, *SnakeView*, as well as the inner classes define and control the application's graphical view. *TileView* describes the basis tile views and specializes the Android *View* class, overriding methods such *onSizeChanged* and *onDraw* to customize it. *SnakeView* and its inner classes, are responsible for the game tile view and game's elements management. *RefreshHandler*, a specialization of *Handler*, performs game animation, while *Coordinate* manages game elements, as *mAppleList* and *mSnakeTrail*.

Fig. 4 illustrates the customization of *onCreate* method through of a sd, describing the behavior when the application is started. During this game initialization, it checks the status of last game and chooses to start a new one or continue from previous state. *Alt* is used to describe the mentioned conditional behavior. Thus,

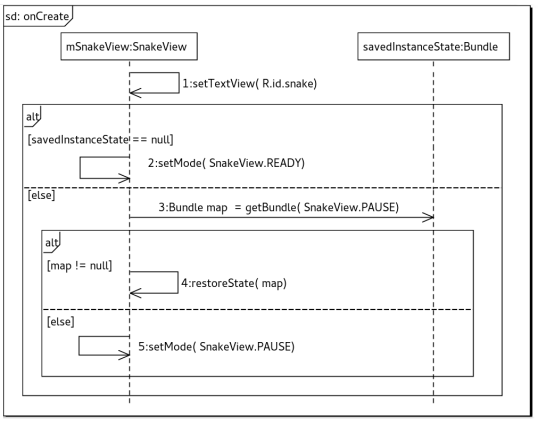

Figura 4: Sequence diagram for the onCreate method.

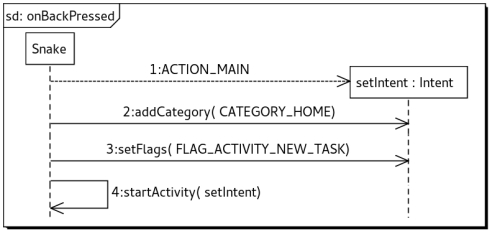

Figura 5: Sequence diagram for the onBackPressed method.

when *savedInstanceState* is null, no previous state is found, and then, a new game is load, otherwise, a previous state is load. The *savedInstanceState* is an instance of Android *Bundle* class, which is used to restore the previous application's state.

Following our approach, other sequence diagrams are used to describe methods customized for the application. The sd from Fig. 5, for example, details the behavior of *onBackPressed* method. This scenario represents the tasks executed when the Back button is pressed, pausing the game and calling the main screen. In this case, the current game state must be saved, enabling its retrieving when the game is called again. To implement this behavior, an intent is used. Following the represented sequence, the intent object (named *setIntent*) is created and the message *ACTION_MAIN*, identifies the required action. After that, the two following messages, *addCategory* and *setFlags*, represent the addition of new features to the intent. In the sequence, the *startActivity* service is invoked using as argument the *setIntent* object.

```
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";
9   ...
```

Listing 1: Generated Code for Snake - Imports and Attributes.

```
1  /** Methods */
2  public void onCreate(Bundle savedInstanceState){
3   super.onCreate(savedInstanceState);
4   setContentView(R.layout.Snake);
5   /** Specified by sd onCreate */
6   mSnakeView.setTextView(R.id.snake);
7   if(savedInstanceState == null){
8    mSnakeView.setMode(SnakeView.READY);
9   } else {
10   Bundle map = savedInstanceState.getBundle(SnakeView.PAUSE);
11   if(map != null){
12    mSnakeView.restoreState(map);
13   } else {
14    mSnakeView.setMode(SnakeView.PAUSE);
15   }
16  ...
```

Listing 2: Generated Code for Snake - onCreate Method.

From this Snake UML model, GenCode returns Java code for the application classes. Listing 1 illustrates the code generated for the *Snake* class, declaring imports for Android API components, the *Activity* extension, and the class attributes.

Besides of structural aspects, behavioral aspects are also handled by our approach, as illustrated in Listing 2, where the code generated to *onCreate* is shown. This code firstly performs the default invocations to *super.onCreate* and *setContentView* methods, followed by the behavior specified by *onCreate* sd from Fig. 4, which includes inlining if statements.

To demonstrate more of our behavioral code generation, we discuss the code generated from the *onBackPressed* sd, which exemplifies an intent usage and is depicted in Listing 3. In this code, the creation of the *setIntent* Intent, as *ACTION_MAIN*,

```
1 public void onBackPressed() {
2  /** Specified by sd onBackPressed */
3  Intent setIntent = new Intent(Intent.ACTION_MAIN);
4  setIntent.addCategory(Intent.CATEGORY_HOME);
5  setIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
6  startActivity(setIntent);
7 }
```

Listing 3: Code generated for Snake methods - onBackPressed Method.

is specified and after that, two methods, *addCategory* and *setFlags*, are invoked to *setIntent*. Finally, the service is invoked through the *startActivity* message using the Intent as argument.

Moreover, GenCode generates of constructors, attributes with default values, setting and getting methods, and inner classes. Listing 4 depicts illustrates some code fragments from *SnakeView*, demonstrating generated code lines, which include initializations of attributes outside or inside the constructor, and setting and getting methods. Code fragments representing the *RefreshHandler* inner class can also be observed in Listing 4, since this inner class is part of *SnakeView*.

Table 1 summarizes the classes automatically generated by GenCode from Snake's UML model. This table presents the total lines of code (LoC) generated for each class, as well as the number of LoCs generated from the class diagram (cd) or from a sequence diagram (sd). From the cd, for the *Snake* and *SnakeView* classes were generated 49 LoC and 117 LoC, respectively. Yet for *Snake*, 12 LoC were generated from the *onCreate* sd and 6 from *onBackPressed* sd. Moreover, to *TileView* were generated 93 LoC from cd and 9 LoC from the *onDraw* sd, totalizing 102 LoC. The total LoC automatically generated for this application were 286.

4.2 Developing for WP: a case study

Puzzle was chosen because it allows demonstrate the main features of our approach. Likewise, it is a simple application example from the WP developers web site [15].

Firstly, the Puzzle structural view is described for the cd depicted in Fig. 6. In this diagram, WP API classes such as *PhoneApplicationPage* and *EventArgs* are empathized, while the applications classes such as *PuzzlePage*, *MainPage*, *PuzzleGame* and its inner classes are in white. Some of these classes such as *MainPage* and *PuzzlePage* have a special label "≪Partial≫", indicating that these classes have logical

```
1  public class SnakeView extends TileView{
2   /**Attributes */
3   public int READY = 1;
4   public int RUNNING = 2;
5   public int LOSE = 3;
6   ...
7   public SnakeView( String TAG, ... , ArrayList<Coordinate> mSnakeTrail){
8    this.TAG = TAG;
9    ...
10   this.mSnakeTrail = new ArrayList<Coordinate>();
11  }
12
13  /** Get */
14  public String getTAG(){
15   return this.TAG;
16  }
17
18  /** Set*/
19  public void setMRedrawHandler(RefreshHandler mRedrawHandler ){
20   this.mRedrawHandler = mRedrawHandler;
21  }
22
23  public class RefreshHandler extends Handler{
24   /** Constructor */
25   public RefreshHandler(){
26    super();
27   }
28   ...
```

Listing 4: Generated Code for SnakeView.

Tabela 1: Obtained results for the Android case study

Class	**Diagram**	**LoC**
Snake.java	cd	49
	sd onCreate	12
	sd onBackPressed	6
		67
SnakeView.java	cd	**117**
TileView.java	cd	93
	sd onDraw	9
		102
	Total:	***286***

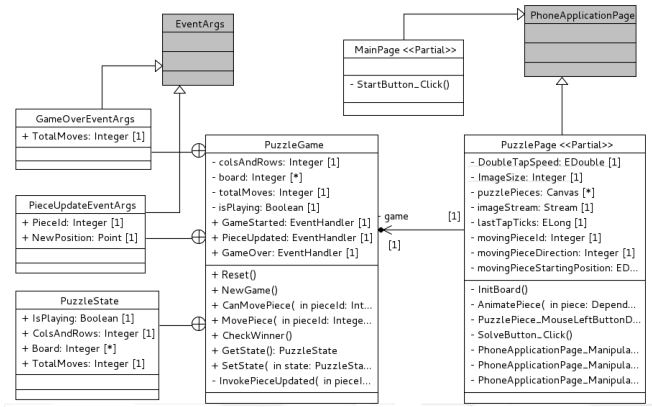

Figura 6: Class diagram - Structural view of Puzzle.

and view components.

This diagram describes the *PuzzlePage* and *MainPage* classes as specializations of the WP *PhoneApplicationPage* class. The class diagram also describes two specialization of the WP class named *EventArgs*, which are *GameOverEventArgs* and *PieceUpdateEventArgs*. Likewise, this diagram also describes an association relationship between *PuzzlePage* and *PuzzleGame*.

PuzzlePage is in charge of the GUI and user interaction. This class extends *PhoneApplicationPage*, which is the default WP class responsible by these aspects. *PuzzlePage* has many attributes such as *puzzlePiece*, *imageStream*, *lastTapTicks*, *movingPieceId*, *movingPieceDirection* and *movingPieceStartingPosition*, and also static attributes such as *DoubleTapSpeed* and *ImageSize*.

PuzzleGame is in charge of game management, providing operations such as *Reset*, *NewGame*, *CanMovePiece*, *MovePiece*, *CheckWinner*, *GetState*, *SetState*, and *invokePieceUpdated*. This class has many attributes such as *colsAndRows*, *board*, *totalMoves*, *isPlaying*, and also event attributes such as *GameStarted*, *PieceUpdated* and *GameOver*.

The sd (from Fig. 7) details the *CheckWinner* method from *PuzzleGame*, which is responsible for checks the user solution. This task is represented using a *loop* to verify the piece position, and an *opt* to verify if it is a complete solution.

From the Puzzle UML model, GenCode provided a WP implementation. The generated code is composed of C# and XAML files for normal and "Partial" classes specified in the project, such as *MainPage.xaml.cs*, *MainPage.xaml*, *PuzzleGame.cs*,

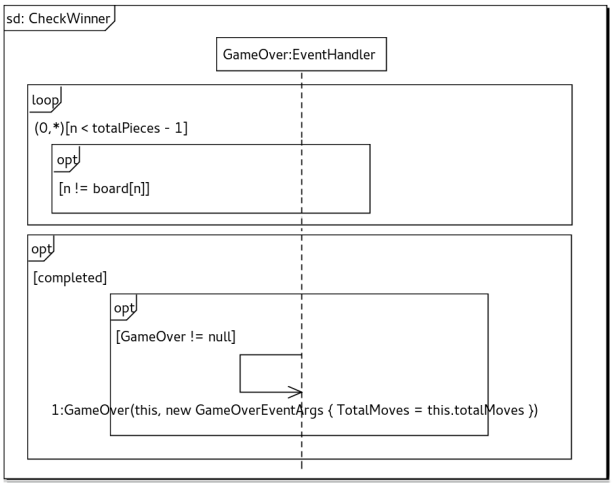

Figura 7: Sequence diagram for the CheckWinner method.

PuzzlePage.xaml.cs, and *PuzzlePage.xaml*. In addition, our tool also generated files for the *App* partial class, creating *App.xaml.cs* and *App.xaml*.

Listing 5 illustrates the code snippet generated for *PuzzleGame* class, which includes specifications of the inner classes such as *PieceUpdateEventArgs* and *GameOverEventArgs*.

Fragments of code generated for *PuzzlePage* are presented in Listing 6 and Listing 7. Listing 6 illustrates the code generated from the cd, focusing on structural aspects like attributes, which were specified explicitly into the model or were derived from relationships. Listing 7 focuses on behavioral aspects generated from the *CheckWinner* sd, which describes the behavior of the correspondent method. This code includes a *for* obtained from the *loop* fragment, and *if* statements obtained from the *opt* fragments.

Table 2 summarizes the quantity of code generated by GenCode from the Puzzle UML model. In this table, for each class is given the total generated LoC as well as partial LoC obtained from each diagram. For the *App* class, 141 LoC were generated, without any modeling effort, being 20 LoC for the visual code (.xaml) and 121 LoC for the logical code (.xaml.cs). Moreover, for *MainPage* 35 LoC were generated for visual part, and 28 LoC for logical code, both from the class diagram. For *PuzzleGame* 120 LoC were generated, from the cd and 14 LoC from *CheckWinner* sd, respectively, totalizing 134 LoC. For *PuzzlePage* 140 LoC were generated, when

```
7  namespace WindowsPhonePuzzle
8  {
9   public class PieceUpdateEventArgs : EventArgs
10  {
11   //Attributes
12   public int PieceId{ get; set; }
13   public Point NewPosition{ get; set; }
14
15   //Constructor
16   public PieceUpdateEventArgs(int PieceId, Point NewPosition)
17   {
18    this.PieceId = PieceId;
19    this.NewPosition = NewPosition;
20   }
21  }
```

Listing 5: Generated Code for PuzzleGame - Inner Classes.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Net;
5  using System.Windows;
6  ...
7
8  namespace WindowsPhonePuzzle
9  {
10  public partial class PuzzlePage : PhoneApplicationPage
11  {
12   //Attributes
13   private PuzzleGame game;
14   private static double DoubleTapSpeed=500;
15   private static int ImageSize = 435;
16   private Canvas[] puzzlePieces;
17   ...
```

Listing 6: Generated Code for PuzzlePage - Attributes.

```
1  public void CheckWinner()
2    {
3   // Specified from Sequence Diagram CheckWinner
4   for(int n = 0; n < totalPieces - 1; n++)
5   {
6    if (n != board[n]){}
7   }
8   if (completed)
9   {
10   if (GameOver != null)
11   {
12    GameOver(this, new GameOverEventArgs { TotalMove = GameOver.this.
          totalMoves });
13   }
14   ...
```

Listing 7: Generated Code for CheckWinner method.

35 are from visual code (.xaml), and 105 for logical code (.xaml.cs). Among these, 82 LoC are from class diagram and 23 were from the sequence diagrams. Considering all LoC generated from our Puzzle model, one can see that using our approach, we obtained automatically 478 LoCs.

5 Conclusion and Future Work

This work presents a MDD approach to automate the mobile application development, which defines modeling and code generation strategies for Android and Windows Phone mobile platforms. Following our approach, mobile developers can model an application and automatically generate code for the target platform. This automation is supported by GenCode, whose current version is enable to generate Java-Android and C# code from UML class and sequence diagrams. Our generation for behavioral code is based on sequence diagrams and for that reason it basically generates lines describing method invocations. These invocations can be inside of a loop or a conditional, and in these cases, for and if statements are also generated.

Two case studies are presented to demonstrate our approach, discussing application model and generated code. We present numbers of generated lines, but we do not compare these numbers with the total LoCs used for the original implementation. This comparison is avoided because it is well-known that automatically generated codes are larger than manual [11]. In addition, as mentioned before, our approach is limited mainly by the behavior abstraction of the UML sequence diagram.

Tabela 2: Obtained results for the WP case study

Class	**Diagram**	**LoC**
App.xaml		20
App.xaml.cs		121
		141
MainPage.xaml		35
MainPage.xaml.cs	cd	28
		63
PuzzleGame.cs	cd	120
	sd CheckWinner	14
		134
PuzzlePage.xaml		35
PuzzlePage.xaml.cs	cd	82
	sd PhoneApplication-Page _ManipulatedStar-ted	16
	sd SolveButton_Click	3
	sd PuzzlePiece _Mouse-LeftButtonDown	4
		105
		140
	Total:	***478***

As future work, we intent to revise the proposed modeling strategies in order to define a platform-independent approach that should be applied for cross-platform mobile development. Moreover, we intent to provide support to iOS, another important mobile platform.

6 Acknowledgment

The authors acknowledge financial support received from CNPq (process 483464/2013-9) and FAPERGS (NESS Project - process 10/0043-0).

Referências

[1] Appcelerator. Appcelerator, master the mobile shift. `http://www.appcelerator.com/`, Apr. 2015.

[2] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan and Claypool, 2012.

[3] Gartner. Gartner technology researcher identifies the top 10 strategic technology trends for 2013. `http://www.gartner.com/newsroom/id/2209615`, Apr. 2015.

[4] GitHub. Repository of gencode tool. `https://github.com/abiliogp/genCode`, Oct. 2015.

[5] Google. Android. `http://www.android.com/`, Oct. 2015.

[6] Google. Android application samples. `http://developer.android.com/tools/samples/index.html`, Oct. 2015.

[7] Henning Heitkötter, Tim A Majchrzak, and Herbert Kuchen. Cross-platform model-driven development of mobile applications with md 2. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 526–533. ACM, 2013.

[8] IBM. Ibm mobile first platform. `http://www-03.ibm.com/software/products/en/mobilefirstplatform`, Apr. 2015.

[9] IDC. Android pushes past 80% market share while windows phone shipments leap 156.0% year over year in the third quarter. `http://http://www.idc.com/getdoc.jsp?containerId=prUS24442013`, Apr. 2015.

[10] Chris Jones and Xiaoping Jia. The axiom model framework: Transforming requirements to native code for cross-platform mobile applications. In *International Conference on Novel Approaches to Software Engineering (ENASE), 2014*, pages 1–12. IEEE, 2014.

[11] John Klein, Harry Levinson, and Jay Marchetti. Model-driven engineering: Automatic code generation and beyond. Technical report, Software Engineering Institute at Carnegie Mellon University, The address of the publisher, 3 2015.

[12] Frank Alexander Kraemer et al. Engineering android applications based on uml activities. *In the 14th international conference on Model driven engineering languages and systems (MODELS), 2011 Jon Whittle, Tony Clark, and Thomas Kühne (Eds.). Springer-Verlag, Berlin, Heidelberg*, pages 183–197, 2011.

[13] Microsoft. The visual studio sdk. `http://www.visualstudio.com/`, Apr. 2015.

[14] Microsoft. Windows phone. `http://www.windowsphone.com/`, Oct. 2015.

[15] Microsoft. Windows phone developer center. `http://developer.windowsphone.com/en-us`, Oct. 2015.

[16] Bup-Ki Min, Minhyuk Ko, Yongjin Seo, Seunghak Kuk, and Hyeon Soo Kim. A uml metamodel for smart device application modeling based on windows phone 7 platform. In *TENCON 2011-2011 IEEE Region 10 Conference*, pages 201–205. IEEE, 2011.

[17] Motorola. Rhomobile suite. `http://rhoMobile.com`, Oct. 2015.

[18] Julian Ohrt and Volker Turau. Cross-platform development tools for smartphone applications. *Computer*, 45(IX):72–79, 2012.

[19] A.G. Parada and L.B. de Brisolara. A model driven approach for android applications development. In *Brazilian Symposium on Computing System Engineering (SBESC)*, pages 192–197, Nov 2012.

[20] PhoneGap. Phonegap. `http://phonegap.com/`, Apr. 2015.

[21] Brain Selic. Uml 2: A model–driven development tool. model–driven software development. *IBM Systems, Riverton*, 45, n. 3:607–620, 2006.

[22] P. Smutny. Mobile development tools and cross-platform solutions. In *13th International Carpathian Control Conference (ICCC), 2012*, pages 653–656, May 2012.

[23] Hyun Seung Son, Woo Yeol Kim, Jae Seung Kim, and Robert YoungChul Kim. Concretization of uml models based on model transformation for windows phone application. *Information Science and Technology (IST)*, pages 288–291, 2012.

[24] Telerik. Appbuilder. `http://www.telerik.com/appbuilder`, Oct. 2015.

[25] Mark GJ van den Brand and Jan Friso Groote. Advances in model driven software engineering. *ERCIM News*, 91:23–24, 2012.

[26] Anthony I Wasserman. Software engineering issues for mobile application development. *Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER)*, pages 397–400, 2010.

[27] Xamarin. Xamarin, cross platform development. `http://xamarin.com/`, Apr. 2015.