v. 3, n. 1, 2022

Journal of
Engineering
Research

DIFFERENT APPROACHES TO INTEGRATING APIS INTO THE MOST POPULAR ANDROID FRAMEWORKS – A CROSS-PLATFORM MIDDLEWARE FOR BUSINESS AUTOMATION DEVICES

Rodrigo Choji de Freitas
Universidade do Estado do Amazonas – UEA / Escola Superior de Tecnologia
Manaus-AM
http://lattes.cnpq.br/3915653613278294

Neide Ferreira Alves
Universidade do Estado do Amazonas – UEA / Escola Superior de Tecnologia
Manaus-AM
http://lattes.cnpq.br/4068095206484923

Ramayana Assunção Menezes Júnior
Universidade do Estado do Amazonas – UEA / Escola Superior de Tecnologia
Manaus-AM
http://lattes.cnpq.br/5941666476087346

Eduardo de Souza Nogueira
Universidade do Estado do Amazonas – UEA / Escola Superior de Tecnologia
Manaus-AM
http://lattes.cnpq.br/9747117584343324

Luiz Felipe Duarte Alves
Universidade do Estado do Amazonas – UEA / Escola Superior de Tecnologia
Manaus-AM
http://lattes.cnpq.br/1974711758279794

Douglas Silva de Melo
Universidade do Estado do Amazonas – UEA / Escola Superior de Tecnologia
Manaus-AM
http://lattes.cnpq.br/4559258194990884

Abstract: This work presents the research and development process carried out for the creation of a software layer that behaves as a driver between the commercial automation application developed in the market and commercial automation hardware. This middleware aims to provide an environment where the developer does not have to stick to specific details of commercial automation equipment to create their applications, thus accelerating the software development process. This framework eliminates or mitigates programming difficulties with hardware, leaving software companies only to worry about the business rule.

1. INTRODUCTION

Software development aimed at Commercial Automation equipment is migrating from Windows and Linux operating systems to Android, consequently, applications are changing from *desktop* to *mobile*. Considering that the Android operating system is aimed at the universe of mobile devices, such as *tablets* and *smartphones*. This means that the seven thousand commercial automation companies existing in Brazil will have to adapt their systems to the Android platform.

Manufacturers of equipment aimed at this market have been investing in the Android platform, as there are not many commercial automation manufacturers in Brazil working with this operating system in their products, that is, the vast majority still work with the Windows platform, and in this environment there are about 80% of the commercial automation market, so the main motivation for migrating from desktop to mobile is the strategic look at changing this market, since the American market is dominated by the Android platform, as well as the European one, so it will be a matter of time and breaking some technological barriers for the Brazilian market to also migrate to Android.

Therefore, there is a segment of products aimed at the point of sale, the so-called POS (Point of Sale or Point of Service), these allow applications to be executed directly on the POS machine, remembering that they run the Android operating system, therefore, it allows software programmers to develop their own solutions for this type of equipment, but there are a multitude of languages or even frameworks aimed at the mobile environment, making development time difficult and increasing, because if a programmer uses a language other than Java, that is the native language of Android, your codes will have to align with this language and it is not always possible or in some cases it is very laborious or difficult.

The difficulty arises from the lack of domain and intellectual capital to develop without libraries and tools that help development on the Android platform.

Thus, the objective of this research is to accelerate the development process of commercial automation applications, delivering to the Brazilian market a low-level communication *framework* on the *hardware* side, but very high-level on the integration side with the *software* developer, that is, encapsulating the POS machine requests, such as printing, camera and configuration, as these are activities performed by the operating system, through the native Java language Android.

THEORETICAL FRAMEWORK AND RELATED WORK

Given that, in general, the ecosystem of applications for commercial automation is in a process of transition from desktop to mobile devices, so there is a lack of research focused on this topic. To the best of our knowledge, the results of this research constitute the first Brazilian publication of its kind.

Fincotto's work (2014) presents a case study on the use of mobile applications for commercial automation using the Android platform in the corporate environment. The article shows that Android can be used as a mobile operating system for such applications, exposing the main features of its architecture.

The work by Santos (2018) highlights the diversity of platforms and the difficulty for companies to develop applications for different operating systems, so the work compared the performance and usability of some components in relation to native development for iOS with Swift and with React Native, this generates a unique code for Android and iOS. In the proposal of this article, several frameworks were also worked in order to generate a single platform for the same application.

The development of mobile applications for business automation is increasingly integrated into software development companies. Currently, *software houses* expect applications with good performance in less time and at a lower cost. In the work by Brito et.al (2019), a comparative analysis of the development time of the main functionalities of an application for Java (Android), Swift (iOS) and React Native (Android and iOS) is made.

AUTOMATION FRAMEWORK-CONCEPT, OPERATION AND IMPLEMENTATION

As a way to provide a mobile application strategy or model that will serve as an intermediate software layer between the application focused on commercial automation and the equipment that uses Android, a framework was developed with some of the most used technologies available in the development market. applications for the Android operating system.

The purpose of this framework is to provide an environment where developers do not have to stick to specific details of the equipments *Androids* to create your applications, thus accelerating the development process of your software aimed at commercial automation. With that, the developed framework also has the objective of carrying out a work that can be considered more complex of communication with the hardware of the equipment, allowing software companies and their developers to be only concerned with their business rules.

This way, the methodology adopted for the implementation and development of this framework was first to separate which technologies would be used, where a total of 9 were selected, they are: i - *Java Android*, ii – *Delphi FireMonkey*, iii – *Flutter*, iv – *React Native*, v – *Ionic*, vi – *Xamarin Forms*, vii – *Xamarin Android*, viii – *App Inventor* e ix – B4A (Figure 1).

All selected technologies are aimed at developing mobile applications, and in some of them it is possible with the same source code to generate executables that can be used not only on ANDROID platforms, but also on iOS and DESKTOP, as is the case with Flutter, React Native and Ionic. However, considering that the equipment only uses Android, and the library that was used to communicate with the Hardware of these equipment is only possible to communicate with the native Java language, it was necessary to create strategies that also meet the needs of technologies that have other source code languages. To facilitate the development of the framework with other technologies, one of the strategies was to create the settings and classes in Java Android and use the application and source code developed as the basis of the framework. Thus, to continue with the creation of the framework, the first step was to develop one of the Features with

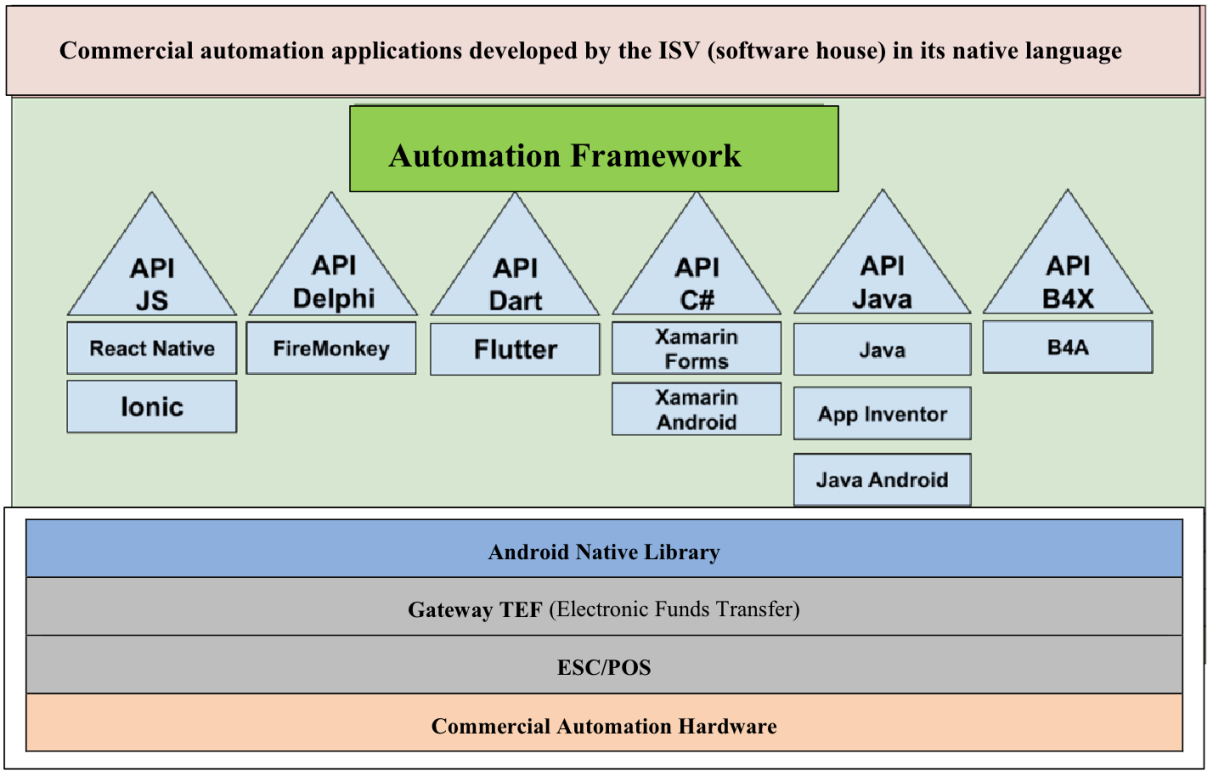

Figure 1. Automation Framework proposed in this work.

this native technology, and then carry out a study of which alternatives were made available by the other technologies for use and adaptation to the implemented classes.

During some of these studies, some alternatives were found that can be divided as Communication Channels and Plugins. More information about the technologies and the alternatives used by each one can be found in the next sections.

COMMUNICATION CHANNELS

Technologies such as Flutter and React Native do not support libraries that have an.aar extension, libraries developed with Java, however, offer the possibility of direct communication with Java classes, even if their own source code is a different language. Thus, it is possible to make and receive calls via "Communication Channels".

Flutter, um *framework Multiplatform* which has characteristics of being fast, productive, free and open source. Its launch took place in December 2018 by Google, and its source code language is Dart (FLUTTER, 2022). To develop and adapt the *features* of the framework developed with this technology, a native technology communication channel known as ***Method Channel***, This strategy allows the *Flutter* communicate with native Android by making function calls from classes implemented in *Java* and receiving your feedback. Figure 2 illustrates the representation of how this method serves as a bridge for sending and receiving calls made. In detail, the solution presented uses the common parameters of each commercial automation device, passing the parameters and the type of action to be performed. When the function returns, the output is mapped to select similar attributes. Therefore, with this strategy, an application completely validated in Flutter itself was obtained, making the specific communication APIs.

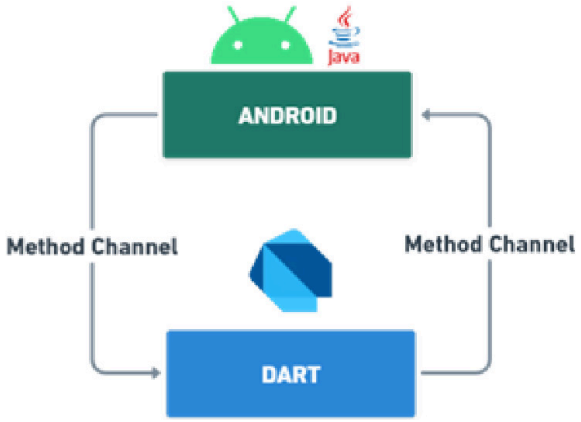

Figure 2. Representation of the operation of *Method Channel*.

React Native, a *framework* for creating native applications *mobile*, it's a project *Open Source* launched in the year 2015 by Facebook where it is continuously maintained and updated. Its source code language is *JavaScript,* considered an easy-to-learn language. This technology also provides a communication channel, known as ***Native Modules,*** with it, it is possible to send and receive information from methods and functions on the native (Java) side of the application (REACT, 2022). Figure 3 is a representation of the way native modules serve as a bridge between different languages.

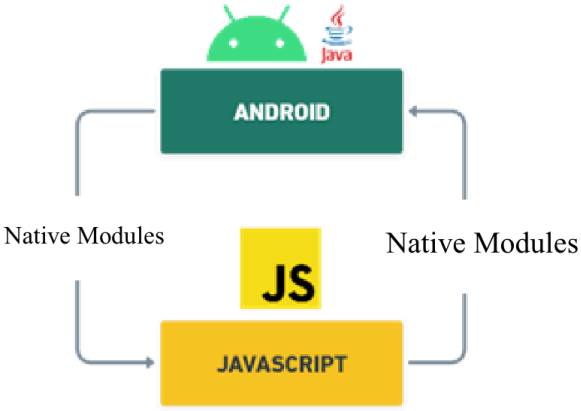

Figure 3. *Native Modules* flow representation.

This way, the use and availability of both these communication channels within the frameworks developed in each of these two

technologies, managed to achieve its objective of delivering a *software* intermediary that facilitated the use and development of software companies that use these technologies. The functionality of the Java API (React Native) has similarities with the Dart API (Flutter), but the communication between Dart-Java and JavaScript-Java are very different. For React-Native, a module was used to make the communication, that is, classes in Android (Java) created in the project itself with an extension of a context in React and a base in Java modules were used.

The framework ***FireMonkey*** is a tool that allows the development of cross-platform applications using the Delphi IDE, using the Object Pascal language, considered a programming language already consolidated in the market. FireMonkey allows the same source code to be compiled for different devices, generating native code for different platforms, such as Windows, Mac, Android and iOS2. As Delphi does not support Android libraries in Android Archive (.aar) format.; it is necessary to unpack the library. Right after extraction, a file is generated that will make the bridge (native bridge file) between Object Pascal and the Java library files (.jar), thus enabling the use of frameworks developed by the project (FIREMONKEY, 2022).

In the context of the multiplatform framework presented in this work, a study was carried out on FireMonkey technology and the source language used for development, with the aim of having a better adaptation of the project. After being able to communicate with commercial automation devices (Figure 4), specific functions were developed for the integration of printing and NFC resources.

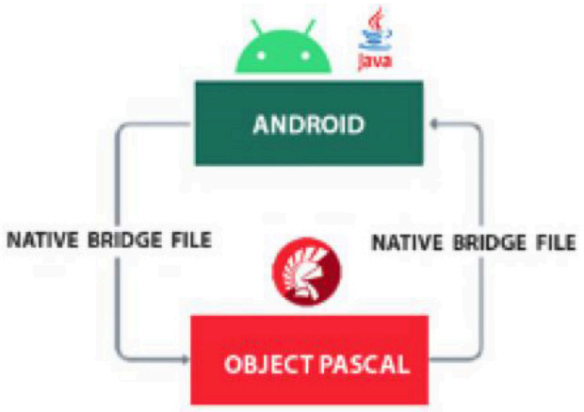

Figure 4. Representation of the FireMonkey-Android communication flow.

The ***Xamarin***, currently owned by the company Microsoft, it is a free and open source platform that allows building and embedding iOS and Android applications using the C# language. As the use of Android Archive is also not supported in Xamarin, it is necessary to create a Java Binding Library (Figure 5), which generates a Dynamic Link Library (DLL) to be imported into the Xamarin project, generating the bridge between the C# language and the Java library. In the context of the Framework, three functions were developed: access to Barcode V2, Printing and NFC Reading and Writing (XAMARIN, 2022).

Figure 5. Java flow representation *Binding Files*.

App Inventor, is an open source application that allows the creation of applications for Android systems in an easy and fast way using the concept of dragging blocks, originally created by Google in 2010. This application is widely used to introduce the concepts of application creation and logic of programming. The logic behind the blocks is carried out through the Java Android programming language (APP INVENTOR, 2022).

In this technology, it is possible to communicate with the native side to use aar and jar library, necessary for communication with the native automation library, but this does not occur directly in the App Inventor editor, to carry out the communication it is necessary to go directly to the open source of the App Inventor and create an extension (".aix") using the Java language directly (Figure 6). There are some examples of how to create extensions in App Inventor, but there are few that create extensions communicating with aar and jar, which makes the activity complex.

However, by researching technology communities and exploring open source, it was possible to develop an extension that can be used in the App Inventor editor and that calls native *Android* functions.

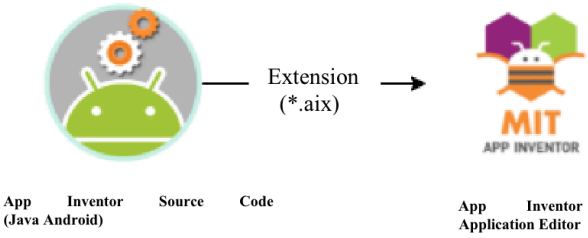

Figure 6. Integração App Inventor – Java Android.

Ionic, It is an open source cross-platform framework capable of producing high quality mobile, desktop and web apps using technologies such as HTML, CSS and Javascript (IONIC, 2022).

In order to achieve the native functionalities of Android by the application in Ionic, **Cordova** was used, an open source mobile development framework that allowed the operation of modules that need to communicate with Android Native through the creation of a plugin (Figure 7).

Cordova includes plugins that have the code referring to native calls and bridges the gap between it and Javascript, offering this support to various frameworks such as Ionic.

Figure 7. Access native functionality through *Cordova.*

PLUGIN

Basic4Android, or as it is known B4A, is a tool that has a source code language similar to Visual Basic and Visual Basic.NET, however it is adapted to run native Android environments. Its proposal is to develop in a short period of time applications for mobile devices known as *Rapid Application Development* (RAD). Its communication with the native side of the applications takes place through the development of Java classes that are later converted to '.jar' and '.xml' files and included in the B4A project, enabling communication. Figure 8 represents how this flow occurs (B4A, 2022).

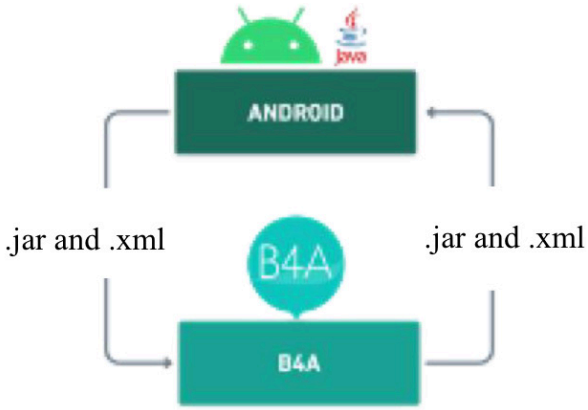

Figure 8. Representation of the Communication flow via.jar and.xml files.

RESULTS AND ASSESSMENTS

Considering the ecosystem and the need for commercial automation on Android and also the vast number of frameworks that arise with the objective of delivering applications for the Android operating system, it was necessary to research and develop techniques that promoted the integration of firmware resources from POS machines to the different Android Frameworks.

To achieve this integration, it was necessary to design and implement some "bridge resources" that would interconnect "low-level" services (programming closer to machine understanding, direct to hardware or communication through bytes) to the different "high-level" Android frameworks. level" (programming closest to human understanding). More specifically, the following scenarios of difficulties encountered and overcome can be highlighted:

1. Technologies such as Flutter and React Native do not natively support libraries with the ". aar" extension. Because the library that communicates with the equipment's printer has exactly this ". aar" extension, it was necessary to understand the operation of some communication protocols in order to then carry out research and development of mechanisms that would enable their use. In the end, a communication bridge was created between the Java language, native Android code, and the source code languages of two other technologies, Dart for Flutter and JavaScript for React Native.
2. Some technologies do not have their own libraries for functionalities inherent to POS equipment, such as, for example, NFC and barcode reader. So, to fill such gaps, it was necessary to carry out research and implement specific features so that they could work natively in the technology.
3. Some technologies like App Inventor and B4A have a limitation on what can be natively implemented in the technology. So, a technique based on intents combined with the development of an APK (Android Package) was developed exclusively to meet the need of the technology.

FINAL CONSIDERATIONS

The commercial automation framework presented in this work provides resources necessary for accessing the following resources available in commercial automation equipment:

- Access module to *gateways* and credit card payment – enables the developer to perform credit card transactions with EFT (Electronic Funds Transaction) operators. There are commands in this module that allow access to payment gateways available in the commercial automation market and their use for the transaction of a purchase;
- Native class printer access module – the main functionality of this module is the printing of formatted text, barcode and QRCode;
- Display access module – this module has as the main functionality the display of text on the screen (display) and the

display of QRCode for payments with electronic wallet;

- Magnetic Card Access Module (Magnetic Stripe) – the main functionality of this module is the reading of the three tracks (or one of them) on the card that is passed through the device;
- Chip card access module (Smartcard) – this functionality works in private mode and not in open mode. These are commands for reading Smartcard trails and password digits typed (even if encrypted) on the device's keyboard;
- Access module to the equipment's NFC device – its purpose is to read an NFC card (approximation) when approached to the device;
- Consumer electronic invoice printing module – this module receives an XML file. Upon receiving the XML, it is read and interpreted, identifying the mandatory fields for printing an electronic consumer invoice.

REFERENCES

H. Brito, Á. Santos, J. Bernardino and A. Gomes, "Mobile development in Swift, Java and React Native: an experimental evaluation in audioguides," *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, 2019, pp. 1-6, doi: 10.23919/CISTI.2019.8760864.

Marcos Apolinário Fincotto, Marilde Terezinha Prado Santos. Automação Comercial utilizando Aplicativos Móveis - Um Foco na Plataforma Android Tecnologias, Infraestrutura e Software, vol 3, n2,. 151-161, 2014.

Tutoriais App Inventor. Acessado em 05/12/2022. Disponível em: https://appinventor.mit.edu/explore/ai2/tutorials

Tutoriais FireMonkey. Acessado em 01/12/2022. Disponível em: https://www.devmedia.com.br/firemonkey-introducao-ao-desenvolvimento-com-banco-de-dados-revista-clube-delphi-136/23066

Tutoriais Flutter. Acessado em 05/12/2022. Disponível em: https://docs.flutter.dev/reference/tutorials.

Tutoriais Ionic. Acessado em 05/12/2022. Disponível em: https://www.b4x.com/android/documentation.html

Tutoriais Ionic. Acessado em 29/11/2022. Disponível em: https://ionicframework.com/docs

Tutoriais React-Native. Acessado em 04/12/2022. Disponível em: https://reactnative.dev/docs/getting-started

Tutoriais Xamarin. Acessado em 06/12/2022. Disponível em: https://dotnet.microsoft.com/en-us/apps/xamarin