NSMuseum – A Case Study for Developing Cross-platform Mobile Applications in Augmented Reality

Mircea-Ioan Dragotă
Technical University of Cluj-Napoca
Str. G. Bariţiu 28, 40027, Cluj-Napoca, România
dragota.mircea@gmail.com

Adrian Sabou
Technical University of Cluj-Napoca
Str. G. Bariţiu 28, 40027, Cluj-Napoca, România
adrian.sabou@cs.utcluj.ro

ABSTRACT
This paper presents a method to streamline the process of developing a mobile application in augmented reality, in order to maximize exposure to a wider audience. In addition, a comparative analysis will be developed between the best-known ways to develop cross-platform applications in augmented reality, ARFoundation and Vuforia. The process described in this paper aims to add value to such an application, integrating Unity with another cross-platform technology, Flutter. As an example of this development process, a cross-platform application was created that offers an appearance as close as possible to the native one and an experience in augmented reality as pleasant as possible, being accompanied by a modern user interface. In addition to this, the example also focused on the integration of augmented reality in the museum visiting experience, enhancing the educational process. Our cross-platform approach has allowed the creation of a single application running on both iOS and Android operating systems, so both the cost, development time and resources have been reduced.

Author Keywords
Augmented Reality; Cross-platform; Flutter; Unity; Mobile Application;

ACM Classification Keywords
H.5.m. Information interfaces and presentation: Human – Computer Interaction

General Terms
Augmented Reality; Cross-platform;

DOI: 10.37789/rochi.2021.1.1.12

INTRODUCTION

Museums have a long history since the third century BC, over time, the culture of the museum has spread to almost all parts of the world, making the concept of a museum, a global one. In addition to research and conservation, the essential purpose of the museum in contemporary society is education.

Nowadays, technology has begun to develop at a fast pace and completely take over our daily lives, proving to us that anything is possible. This accelerated change caused by the digital revolution has a drastic effect on people, transforming the way they access, create and enjoy culture. In our modern society, it has become necessary and indeed urgent for museums to redefine their missions, objectives, functions and strategies to reflect the expectations of a constantly changing world. For museums to retain their relevance and become positive partners in the development of our societies, they should use their unique resources and potential to become more receptive to the dynamics of modern society and urban change. Education is essential for development. One of the fundamental objectives of the museum is to educate, having the ability to share cultural education effectively, because it hosts the tools and materials needed to make this possible.

Augmented reality has caught the general public and offers users a sample of the revolutionary future of mobile devices that awaits us. This technology has the role of interacting with the world around us and can be defined as the technique of adding virtual content to the world, allowing us to observe with additional information superimposed. Thus, it represents the combination of digital data and data from human sensory inputs that are apparently attached to physical space. The first important thing in the development of mobile applications in augmented reality is the choice of a platform for which it is supported. It is usually preferable for the application to reach a wide range of users. The most well-known and widespread are smartphones running iOS and Android operating systems and most AR applications specifically target one of these platforms. This technique is called native development, and for iOS an application will be developed using Swift or Objective-C languages, and for Android, using Kotlin or Java. Native development brings a lot of advantages, such as a natural-looking interface. Each operating system provides certain native graphics that the user is familiar with, and these can be integrated into applications made using this process. The natively developed code runs much faster and the integration with the device's functionalities is much easier, the programming language being already dedicated to the hardware. To develop an augmented reality application, it is necessary to integrate the frameworks dedicated to each platform, ARKit for iOS and ARCore for Android. All the advantages that native development brings, come with a certain price that can be considered a huge disadvantage, the need to develop and maintain two separate projects, which brings a very high cost

for the implementation of the final application, increasing its development time.

This is where cross-platform solutions come in such as Unity, which focuses on developing games and applications in AR (Augmented Reality) and VR (Virtual Reality), but there are also a wide variety of solutions such as Flutter [1], React-Native [2], Apache Cordova [3] or Ionic [4], which allow the imitation of an experience as close as possible to the native one. Cross-platform development means that an application will be delivered on different platforms, while the code will be written in a single programming language. Unity offers two options for developing augmented reality applications. The first and most used option is the integration with the ARFoundation library, developed by Unity [5]. This package covers the native libraries, ARKit [6] and ARCore [7], with an additional layer, creating a single common interface, so we can benefit from the native SDKs of each platform, without having to do two separate native projects. Only one code that will be able to run on both platforms. Another option is to integrate the Vuforia library [8] into Unity, an SDK (Software Development Kit) for creating augmented reality applications. Unity is the most popular tool for creating games and AR and VR applications, so this development engine focuses a lot on the graphics processing and interaction scene. The visual elements offered by Unity for the development of the user interface are very rough and do not offer a pleasant and natural experience. A mobile application must provide a natural experience, regardless of how it is developed, native or cross-platform. This is why this paper describes a different approach, in order to streamline the process of developing applications in augmented reality.

The aim of our work is to develop a cross-platform application in augmented reality that provides a user experience as close to native as possible. Our approach involves combining Unity and Flutter, ensuring bidirectional communication, resulting in a single executable that will run both on iOS and Android systems. Flutter is a tool developed by Google to create cross-platform applications using the Dart programming language. This framework integrates all the native elements of both platforms (iOS and Android) to make beautiful user interfaces. As mentioned above, all the augmented reality side will be developed in Unity resulting in a library. Flutter will have the role of integrating this library, building both the user interface needed for the experience as well as the augmented reality component as a whole application. Due to bidirectional communication, the two technologies will be able to transmit their data easily, so each module will perform its task properly. The library developed using Unity will constantly communicate to the application developed in Flutter all the information gathered from the environment, and this in turn will process the data and update its interface status according to the data received. From an architectural point of view, the distinct separation of the Flutter module from the Unity module will isolate the experience in the augmented reality made in Unity from the rest of the application and will allow for flexible addition of new functionalities in either module. In addition, the library developed in Unity can be easily reused in any other flutter project that wants to integrate functionality into augmented reality.

AN AUGMENTED REALITY EXPERIENCE IN MUSEUMS

The world is constantly evolving in the technological field, and museums must retain their main goal, educating. In order for this to remain possible, museums need to innovate and adopt new technologies. That is why, along with interactivity, this project aims at promoting education. NSMuseum wishes to provide a new learning technique, using augmented reality, and aims to develop young taste for art, history and beautiful, and at the same time familiarize them with current technologies. The application integrates augmented reality in order to provide additional information about paintings on a mobile phone. This digital content will be projected alongside the painting and will offer an overview of the exhibitor and the history behind it. Informative digital content placed next to paintings after scanning them will contain both information about the painter and the painting's history, as well as visual information to help immerse the user in the artist's world and an audio guide.

In what follows we will give a brief overview of other AR applications developed for enhancing the visiting experience in a museum. One of the most popular applications that support augmented reality learning in an art museum is ArtLens 2.0 [9], developed by the Cleveland Art Museum and launched in the summer of 2016 for both iOS and Android. The main purpose of the application is to recognize the exhibits and to present the story behind them, presenting interpretive content on the various pieces of the painting to understand the story behind and what the painter wanted to express. Another innovative feature is "ArtLens Wall", a concept that allows users to interact with exhibits, they are projected on a screen, and with the help of the application can save their moments right on their phone, connecting via Bluetooth smartphone to the museum screen. "ArtLens Wall" takes the interaction with the user to another level.

Another good example is Stanford University's collaboration with Sid Lee to launch an AR mobile app for "The Anderson Collection" [10] in 2014, to give visitors a new experience.

However, due to various reasons, a lot of museums cannot afford to develop an AR application for gallery spaces from ground up. For this reason, using free AR applications developed by technology companies is a good choice, such as "Layar" [11], an application that allows you to enter content that is then scanned along with the necessary information that should be displayed after recognition with goal success.

The "Skins & Bones" app, designed by the Smithsonian National Museum of Natural History, serves as an excellent

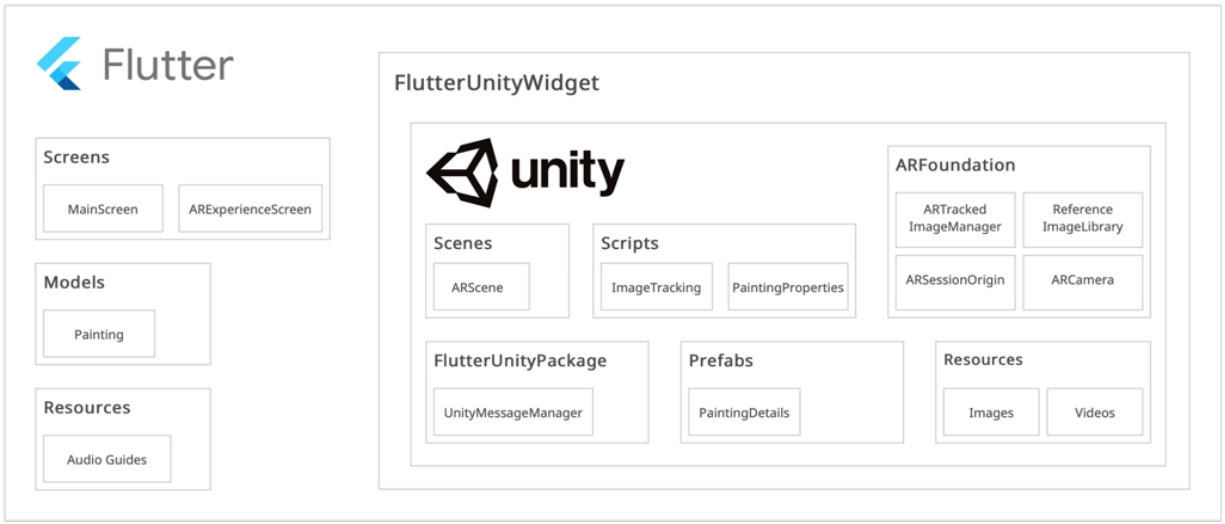

Figure 12 - Project Architecture

example of using the AR app as an educational tool. Launched in January 2015, the app was developed for an exhibition called "Hall of Bone", which was installed in the 1960s and had almost no improvement over the next 50 years. [12] The app aims to share inanimate stories behind the museum's most iconic collections. By using the phone's camera to scan the presented specimens, visitors can see the skeleton of the specimens coming to life.

NSMUSEUM – A CROSS-PLATFORM AR SOLUTION FOR ART MUSEUMS

Project Architecture

The main architecture of the project is layered, being illustrated in Figure 12. As can be observed, the base of the project is Flutter, which deals with all the logic of application navigation and user experience, displaying information in a style as natural as possible. The most important components that make this project possible are FlutterUnityWidget and FlutterUnityPackage [13], two open-source libraries, which have the role of actively maintaining a two-way communication channel between Flutter and Unity, so the necessary information can be transmitted from a lower layer to one superior and vice versa.

The skeleton of the entire application is made in Flutter and is organized in three packages: Screens, Models and Resources. In the Screens package, there are the main scenes of the application, the main screen and the augmented reality screen, this scene being responsible for the presentation of FlutterUnityWidget, a custom widget offered by an open-source library of the same name. This widget encapsulates and displays the scene developed with Unity and maintains a two-way communication channel. In the Models package is the Painting model class, used to encapsulate the details of each painting. The audio package for each panel is stored in the Resources package. As mentioned in previous sections, one of the advantages of this cross-platform development process is that you can add new features very easily using only Flutter, without modifying the library made with Unity, this will be exemplified by the functionality of the audio guide.

The library developed in Unity is organized in the packages: Scenes, Scripts, ARFoundation, Flutter UnityPackage, Prefabs and Resources. As described in previous sections, ARFoundation is the SDK that deals with the realization of augmented reality experience, which is chosen in favor of the Vuforia SDK, this decision will be motivated in more detail in the following sections. FlutterUnityPackge is a package used to create a communication channel with the module made in Flutter. Scenes is the package in which all the scenes are organized, in our case we have only one scene, the one in augmented reality. The Scripts package contains all the scripts written in the C# language, which are attached as a component to the objects in the scene to give them the desired behavior. The following sections will describe the behavior that these scriptures expose. In the Prefabs package are all the 3D objects used, in our case it will be a mask that will have a video player attached and will be positioned over the detected panel. The Resources package contains all the media resources, the videos played in augmented reality, including the markers, the reference images used for the recognition of the paintings.

Painting recognition module

This module, implemented in Unity, has a single scene, which presents the augmented reality experience using the device's camera. This scene, ARScene, which is also found in the system architecture presented above has three main components. These are found in the hierarchy of objects present in the scene which is illustrated in **Error! Reference source not found.**. These components have been featured in,

and they are the basis of any application developed with this SDK.

ARTrackedImageManager is the most important component of this library with the role of recognizing and tracking images in the environment using some known markers stored in the reference image library. This script provides certain methods that are called when an image has been detected, is still detected, or is no longer detected. These methods are accessed from the ImageTracking script developed for this project, which will also use the UnityMessageManger script to communicate to the top layer, Flutter, information regarding which array has been detected.

The image library component, which was mentioned above, has the role of storing the entire collection of paintings that need to be detected, providing a reference to each image. Figure 13 illustrates the configuration of this component, offering the possibility to specify the physical size of the image to increase the accuracy of its detection in the environment. This component had the biggest impact in the decision to choose ARFoundation over Vuforia. It is very suitable for the requirements of this project, in that this object can handle the entire collection of detected images, unlike Vuforia where to recognize a single image requires a separate GameObject positioned in the scene for each picture, which is highly inefficient.

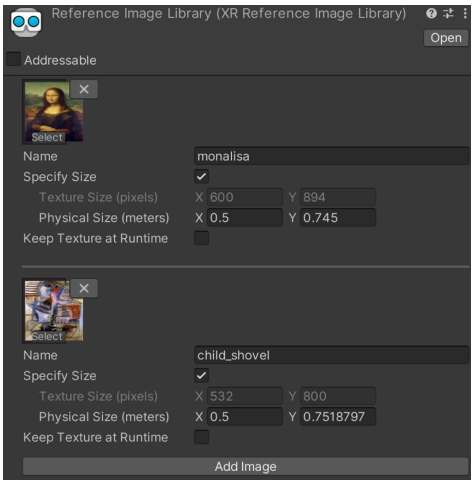

Figure 13 - Reference Image Library

ARSessionOrigin's role is to keep virtual objects in the correct position in AR environments. This component receives continuous updates from the subsystem and changes the position of virtual objects to stay aligned with the real environment. AR objects must be placed under the session origin in the object hierarchy of the scene. Otherwise, the positions of the objects will not be updated. In order to receive these updates from the environment, it will need access to the device's camera, so an ARCamera object will be attached in order to retrieve the data and pass it to the parent object, the session origin, to process it.

As can be observed in Figure 14, the main objects of this prefab are a plane, which is positioned to the right of the detected array and a text, which is positioned below the detected array. These items are populated with informative data about the painting, such as the date it was made, as well as a video presentation of it. Being a template, this object will be populated dynamically according to the detected array, with its informative data.

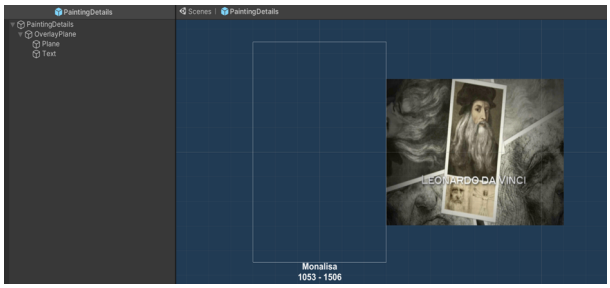

Figure 14 - Painting Details Prefab

The PaintingProperties script is attached to the Painting Details prefab itself and serves to populate the graphics with the detected array data, so it will populate the text element with the date it was created. In addition, it allows access to the Video Player component, attached to the prefab plan, in order to set the video source and play it. When the picture is out the camera's field of view, the video player will be paused.

The ImageTracking script is attached to session origin and, together with the tracking image manager, handles everything from detecting paintings, displaying PaintingDetails prefabs in the augmented environment and communicating with the application developed in Flutter. The method provided by the manager, which reflects the status of detected and tracked objects is called OnImageChanged. This method is called continuously by the subsystem and returns an object which contains three lists of type ARTrackedImage, these are called added, updated, removed. As the name suggests, these lists will group the images that should be detected and tracked, associating them with three states: added, the image has been detected, updated, the image has been detected and tracked, its position being changed and removed, the image is not followed at all. Using these states the system will update the interface accordingly, showing or hiding the information about the focused painting.

Figure 15 illustrates the execution of the Unity module, so we have a cross-platform application in augmented reality running on both iOS and Android devices. When the painting "Monalisa" is detected, a video presentation about the author will be projected next to it, also some information about the painting will be displayed just below the masterpiece.

There is one last step to complete the implementation of this library, namely the export of the project as a native iOS and

Android library, in order to be integrated into the project developed in Flutter. The FlutterUnityPackage is the one that offers an option to generate and export the native libraries required.

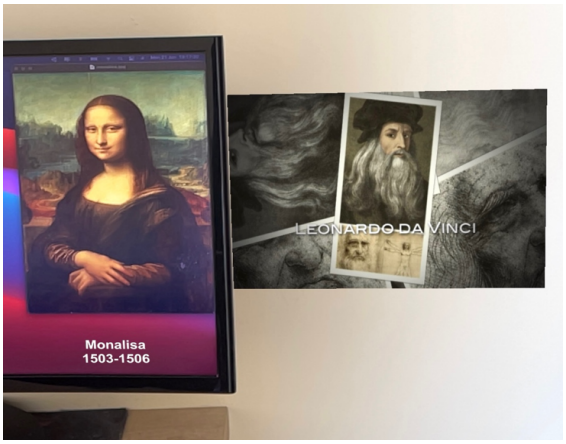

Figure 15 - Unity module execution

Flutter Module

At this point we already have a fully functional module in augmented reality implemented in Unity. This section will present the implementation of the final application. There will be a single project that can be delivered on both iOS and Android operating systems, in order to maximize exposure to a wider audience. In essence, the key goal of cross-platform technology is actually to provide applications as close as possible to the native ones. Due to technologies such as Flutter, this is possible, giving us a wide range of native visual elements. These elements are found in the Cupertino packages, dedicated to the iOS platform and Material Design, dedicated to the Android platform. As previously specified, this architectural model allows the easy addition of new functionalities. This will be exemplified by adding an audio guide using only Flutter, without changing the functionality of the library implemented in Unity. Thus, we will make the most of the bidirectional communication between the two technologies, allowing the incorporation of new functionalities outlined around the panel detection module. In this case, when detecting a picture, an audio guide will be presented that can be listened to in the background. Flutter combines ease of development with native-like performance, while maintaining visual consistency across platforms, with Flutter's programming language being Dart. The entire skeleton of the final application is developed in Flutter, which consists of navigation logic and other features in addition to augmented reality logic, that will be offered exclusively by the module developed in Unity. The final application consists of two scenes, the first will be the main screen, MyHomePage providing some information about how to use this application. Figure 16 illustrates this scene and shows the graphic interface that depicts the main functionality of the application and instructions on how to use it.

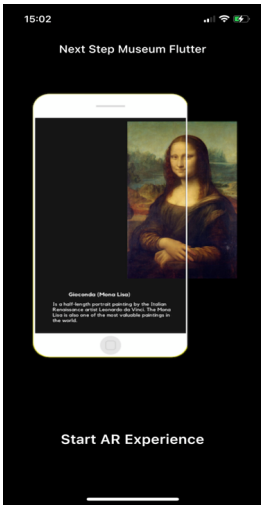

Figure 16 - My Home Page

All the navigation logic of the application is based on routes, the scene presented above being the initial route, so when pressing the "Start AR Experience" button the method Navigator.pushNamed will be called the desired route name, in this case being "/ar". This will present the new scene, the one that integrates the library developed in Unity and offers the experience in augmented reality. The animation of presenting a new scene is very natural, being pushed from right to left, like a native application.

ARExperienceScreen is the scene that offers the experience in augmented reality, adds an additional layer on the library developed in Unity, as mentioned in previous sections, offers a modern user interface and as native as possible. In order for visitors to fully enjoy the exhibit and focus only on it, the application also offers an audio guide after scanning it. A new feature that this screen introduces is the audio guide and additional information about the detected picture, so the information will appear with a beautiful animation from the bottom up. The visual elements are dressed in a card, which contains information about the masterpiece, as well as a media player. The user is able to: play the associated audio guide, control the player using a slider, play the audio file from the beginning or to pause. Audio content can be played without the need to focus the painting with the device's camera. The user can look at the painting in detail, enjoying the relevant and interesting information presented in the background.

UnityWidget is the main component found in this scene and it has the role of actively maintaining the bidirectional communication channel between the two platforms. This component has two important methods: onUnityCreated, onUnityMessage. As its name suggests, onUnityCreated has the role of announcing when the Unity scene has been

successfully uploaded and returns an instance of an UnityWidgetController object with which messages can be sent from Flutter to Unity using the postMessage method. The onUnityMessage method is implemented for retrieving messages sent from Unity to Flutter, such as the identifier of the array that is detected. The onUnityCreated method accesses the controller instance to send a message to Unity, for instance to set the augmented reality videos to mute mode so that only the sound associated with the audio guide can be heard. The arguments that this method receives are the object name in Unity, the name of the called method, and the value of the argument.

Implementation of this method represents the interception and processing of messages transmitted by Unity. Thus, with the help of this method, the animation of the information card is controlled, which appears only if a scan of a painting has been performed. If a valid message has been received, the model associated with the detected panel will be accessed, the audio player will be reset and the setState method call will update the user interface, presenting an animated information card from the bottom up, which will also have a player attached. After displaying the information card, the user will be able to play the audio guide by pressing the dedicated buttons and will be able to control the audio guide by moving the slider. These actions will call the play, pause, and seek methods on the AudioPlayer object. These methods are used to play, pause, or scroll the audio file. When playing the audio player buttons, the playAudioGuide method will be called.

Figure 17 illustrates the final result, a new layer that is implemented in Flutter and is added over the library made in Unity. We can observer the modern, natural and intuitive design and all the graphic elements that are known to mobile device users. All icons are official and used by Google, being from the Material Design package integrated directly into Flutter. Using this combination of frameworks, a clear and intuitive interface can be created. As already mentioned, this architectural model allows for easy addition of new features, exemplified by the addition of the audio guide using only Flutter, without changing the functionality of the library implemented in Unity. Thus, it took full advantage of the bidirectional communication between the two technologies, allowing the realization of new functionalities outlined around the panel detection module.

COMPARATIVE ANALYSIS BETWEEN AR SDKS

This section contains a comparison of the ARFoundation framework and the Vuforia framework, emphasizing their key differences and the main advantages that one has upon the other. This comparison is meant as a helper towards mobile AR developers in the process of choosing the best candidate based on their requirements.

For starters, an advantage that Vuforia has and should be highlighted is the fact that it can also address older devices that do not support ARKit or ARCore, so it can address devices running at least iOS 9.0 or Android 6.0. In addition, it can dynamically determine if it can benefit from ARKit or ARCore, thus, if the device supports these capabilities, Vuforia will take advantage of this and use them in its favor. As for ARFoundation, it can only run on devices that support native SDKs for augmented reality support, so it only addresses devices running at least iOS 11.0 or Android 7.0.

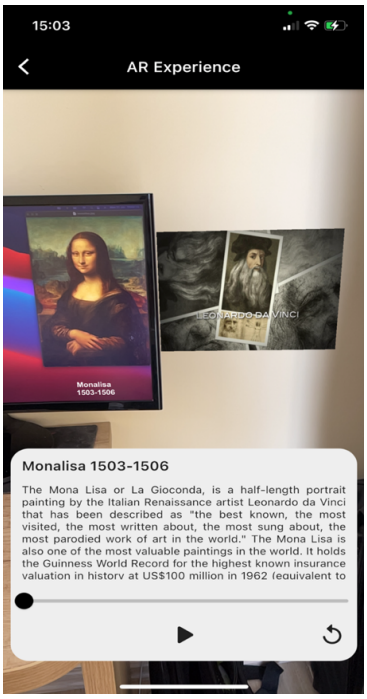

Figure 17 - AR Experience

Regarding this project both frameworks are known for their main objective, image recognition and tracking. Both did very well in recognizing the paintings as can be seen in Figure 18, but the big difference is the implementation technique for this feature. On the visual side you can see only the watermark "Vuforia" on the left of the screen that cannot be removed because the use of this SDK requires a license, unlike Unity, which offers free software.

As mentioned above the major difference is the implementation. Vuforia is noted for the simplicity of its integration, such as the implementation not requiring any line of code. The implementation based on ARFoundation described in the previous sections could be done in Vuforia without writing any line of code. But this advantage comes with a price: for each image to be detected, Vuforia imposes a 1:1 ratio with a GameObject in the scene. This means that for each image in the collection of markers in the scene, we must add the object to be attached to the detected picture, in our case the prefab PaintingDetails, and for this there is a limitation of maximum 100 instances. Each recognized array must be added to the scene as in Figure 19, which is a tedious process.

Vuforia ARFoundation

Figure 18 - Comparison between Vuforia and ARFoundation

Vuforia ARFoundation

Figure 19 - Vuforia scene vs ARFoundation scene

With ARFoundation you can use a single prefab that can dynamically change the code with the desired values, such as the information associated with the detected array. This is possible because there is only one object that deals with the entire collection of markers that need to be recognized. This object is ARTrackedImageManager and together with the ReferenceImageLibrary component, it is able to recognize any image in its collection. The 1:1 ratio imposed by Vuforia is no longer necessary, so the scene is not loaded with unused objects. The process of adding a new painting to the collection is a simple one, without the need to create a new GameObject associated with the scene. It is much more efficient that only one component has to deal with the entire image collection and the image reference of each associated GameObject no longer needs to be added.

Taking all this into account, and due to the project's requirements, as well as maintenance and future development, ARFoundation is much better suited than Vuforia, because it offers a much more efficient process in updating the reference images. Also, a series of manual tests performed on a mobile device using two versions of the same application, one implemented with ARFoundation, and another implemented with Vuforia, revealed a much smoother experience, without stuttering, in the case of the former, as well as a sharper and clearer camera feed.

COMPARATIVE ANALYSIS BETWEEN THE FLUTTER UI AND UNITY

This section contains a comparison of the Flutter framework and Unity with regard to UI development for AR cross-platform applications. Figure 20 contains a visual comparison of two versions of the same application, one created using UI elements provided by Flutter and one created using UI elements provided by Unity. As we can see, the graphics UI elements provided by Unity display a certain roughness, unlike Flutter which has a much friendlier and more natural user interface.

Flutter Unity

Figure 20 - Flutter UI vs Unity UI

Visual elements offered by Flutter are more natural and friendlier, similar to native mobile UI elements that the user is already used to. Flutter provides support for button icons using the official ones from Google, found in the Material Design library. Another advantage of the Flutter framework is that the implementation of user interfaces is extremely fast and much easier than in Unity. A very clear example is that each Flutter widget has a property for changing the background color. With Unity, which is a game development engine, we would have to create a Material type object with this color property and only then we would be able to change the color of another object in the scene through this material object. Another example of Flutter's advantages is the fact that you can easily round the corners of visual elements, an indispensable operation when creating user interfaces. In Unity, this can only be done by using already edited images with a transparent background.

Performance-wise, Unity, similar to all game development engines, runs the application in an infinite loop, generating intensive processor and graphics card usage, thus increasing the battery consumption. By using Flutter, only the screen that integrates the scene from Unity will have this effect, any other parts of the application having a reduced impact on device components. Moreover, Flutter has the Hot Reload function, which helps save interface development time, leaving the developer to see the changes applied in real time, without the need to recompile the modified code. Thus, we can conclude that Flutter is the best option in creating the

user interface for a cross-platform mobile application in augmented reality.

CONCLUSION

This paper focused on describing a more efficient approach for the development process of a mobile application in augmented reality, in order to maximize exposure to a wider audience. This approach has allowed the creation of a single application running on both iOS and Android operating systems, so both the cost, development time and resources have been reduced.

To exemplify this development process, an application was created that offers an appearance as close as possible to the native one and an experience in augmented reality as pleasant as possible, being accompanied by a modern user interface. In addition, it focused on integrating augmented reality into learning in a museum. This technology has the role of interacting with the world around us and can be defined as the technique of adding virtual content to the world, allowing us to observe the world around us with additional information superimposed on it. Augmented reality is a portable tool for discovery-based learning, enhancing the information available to customers when they visit gallery spaces and interact with real-world objects.

Unity is the main technology in the development of the augmented reality experience, but being an engine dedicated to creating games, it focuses mostly on the interaction on stage and not on the application itself, offering a robust and minimalist user interface. Being a game development engine, the resulting applications run in an infinite loop, using intensively the processor and the graphics card, thus increasing the battery consumption. By integrating with Flutter, only the screen that integrates the scene from Unity will have this effect, so any other new functionalities of the application implemented in Flutter will not have the same impact on the device components, such as Unity, this being a huge advantage. Flutter can deliver the same application to five operating systems: iOS, Android, Windows, macOS and Linux, and in addition to these it can also produce a web application, so further developments have a wide spectrum, presenting a wide range of users. In addition, the need to use the Flutter framework in the development of the user interface was motivated.

In addition to all the advantages of using Flutter technology together with Unity, we not only offer a more enjoyable experience, but we also easily extended the functionality of the application. From an architectural point of view, Flutter is the basis of the system, thus isolating the experience in augmented reality made with Unity from the rest of the application, allowing the addition of new UI functionalities as easily as possible. In addition, the library developed in Unity can be easily reused in any other Flutter project that focuses on integrating any other functionalities into augmented reality, increasing the reusability of the code.

REFERENCES

1. Flutter, "Flutter Documentation", 2021. https://flutter.dev/docs/resources/architectural-overview
2. React Native, "React Native", 2021. https://reactnative.dev
3. Apache Cordova, "Apache Cordova", 2021. https://cordova.apache.org
4. Ionic, "Ionic Framework", 2021. https://ionicframework.com
5. Unity, "AR Foundation", 2021. https://unity.com/unity/features/arfoundation
6. Apple, "Apple Developer", 2021. https://developer.apple.com/augmented-reality/arkit
7. Google, "ARCore", 2021. https://developers.google.com/ar
8. Vuforia, "Vuforia Developer Library", 2021. https://library.vuforia.com/articles/Training/vuforia-fusion-article.html
9. M. Ding, Augmented Reality in Museums, Pittsburgh, 2017.
10. S. University, "The Anderson Collection", 2021. https://anderson.stanford.edu/
11. Blippar Group, "LayAR", 2021. https://www.layar.com/
12. "Statista", 2021. https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/
13. R. Raphael, "Github - Flutter Unity View Widget", 2021. https://github.com/juicycleff/flutter-unity-view-widget
14. K. Lee, "AR Foundation in Unity: Getting Started", 2021. https://www.raywenderlich.com/14808876-ar-foundation-in-unity-getting-started.