Swift UI and Their Integration to MapKit Technology as a Framework for Representing Spatial Information in Mobile Applications

Eduardo Eloy Loza Pacheco[1](✉), Mayra Lorena Díaz Sosa[1], Christian Carlos Delgado Elizondo[1], and Miguel Jesús Torres Ruiz[2]

[1] Universidad Nacional Autónoma de México, Acatlán Edomex, 08544 Naucalpan de Juárez, Mexico
{eduardo.loza,mlds}@acatlan.unam.mx, 805849@pcpuma.acatlan.unam.mx

[2] Instituto Politecnico Nacional, Ciudad de México, CDMX, Mexico
mtorres@ipn.mx

Abstract. Mobile applications are becoming as complex as the kind of problems we need to solve. A normal GIS application needs to incorporate elements such as artificial intelligence more specifically, pattern recognition or machine learning, relational or non-relational databases, spatial representation, and reasoning. In addition to that, it is necessary to design and develop a mobile application to integrate all these elements. After analysis, planning, and design a company will require to develop a full team of engineers to accomplish an application. On the other hand, universities and other organizations, have additional interests. It is necessary to develop a mobile application to test a hypothesis before a large development project. The necessity is to use a technology that permits the integration of advanced technologies and programming tools in a manageable sense. Companies such as Google and Apple are reducing the gap of learning knowledge. They are developing new technologies. For example, Apple presented in 2019 at WWDC2019 and WWDC2020 a novelty technology called SwiftUI, which aims to reduce the complexity of developing a mobile application and allowing us to integrate technology such as Mapkit to represent spatial information. This work presents the advantages of using SwiftUI to integrate Mapkit as a spatial representation framework to ease the development of GIS mobile development. And focus on the problem solution, such as spatial representation and reasoning, robot planning, content image retrieval, etc.

Keywords: Swift UI · MapKit · GIS · Spatial representation

1 Introduction

Computer Science has a large variety of applications in different fields of science. For example, artificial intelligence and machine learning are topics and technologies widely used in several areas of knowledge [1]. The aim is to develop programs that can perform representation and calculations of the information according to the necessity of the study. And to develop new way of representation [2] for the large problems we have today, for example data science. For example, in social sciences, we can create a model that study people activities [3]. In meteorology, we can represent the predictions of the weather [4]. In order to achieve computer science collaborations with all sciences is necessary to use a scientific computing approach. The definition of scientific computing is the intersection of Numerical Analysis, Modeling, and Computer Science, as we can see in Fig. 1 [5]. After we have selected a mathematical model and analyze the computational complexity to reduce error and increase performance. We can design an Algorithm so we can implement it on a computer.

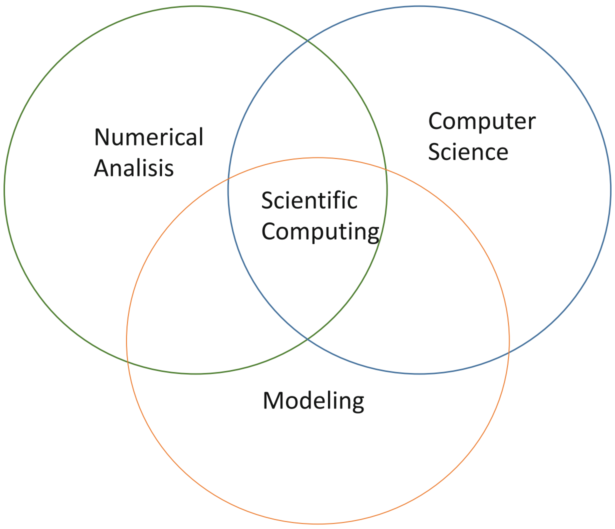


Fig. 1. Definition of scientific computing from [2].

After that, according to [6] the understanding of a computational system has three levels or layers: The concepts of computational theory we need to consider. The representation of the information or knowledge, and the design and implementation of the algorithm. And finally, the Hardware implementation. We can also add software implementation in a computer language and the use of a technological platform. In the last one, we need to consider the technical elements to improve the spatial and temporal performance of the algorithm, reduce the error, scheduling the transactions of blocks of data, access to a shared or distributed memory, etc. [7, 8]. So, we can see these as an architecture that we can use as a framework. As we can see in Fig. 2.

One of the purposes of a layered architecture is the possibility of divide activities, so we can reduce the complexity of the modules, specialize, and reuse. The communication between layers should be simple and easy. As an example, we have the architecture of the Open System Interconnection, where every layer is defined to provide one service. For example, the 5-layer hybrid model. Layers provide the service of packing, routing over the network, synchronization and codifying frames [7]. Finally, the communication

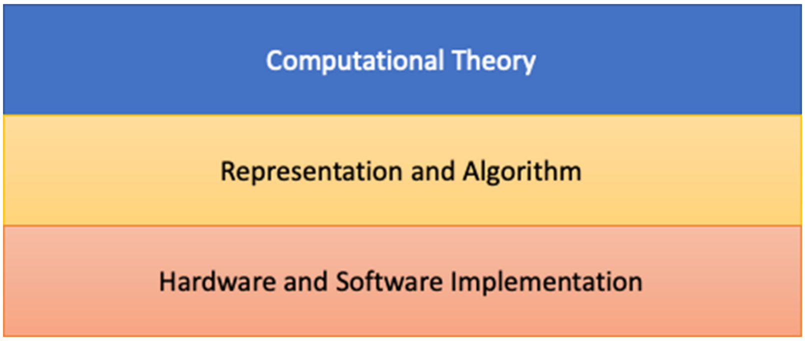

Computational Theory
Representation and Algorithm
Hardware and Software Implementation

Fig. 2. Architecture of a computational system implementation according to [3].

between layers is made by an interface. In Fig. 3 we can see a general model a layered architecture.

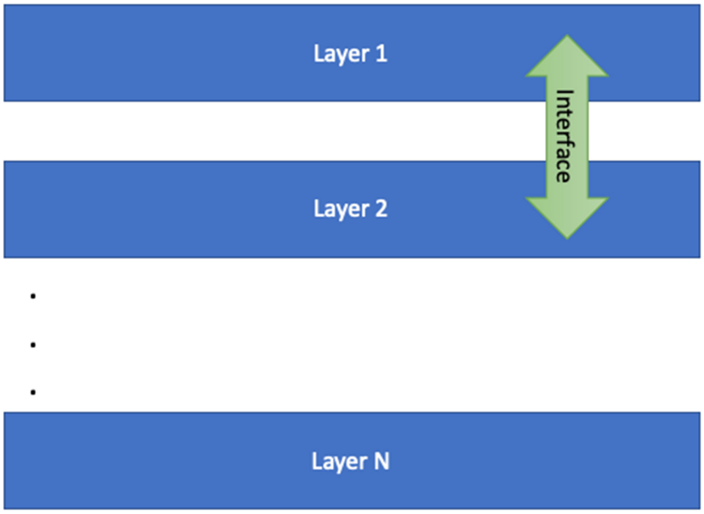


Fig. 3. Layer of an architecture image base on [7].

1.1 Brief State of the Art of User Interfaces

As mentioned before, every layer performs a function. These functions must be transparent for the other layers. So, the interfaces are the key to their communication. In the case of development platforms such as Android, Eclipse, and XCode. All of them have a Graphic User Interface (GUI) that eases the programming [9] and allows to focus in the programmer [10] and increase the productivity [11]. By making the process of development more intuitive. In the case of Xcode is integrating a new Technology called SwiftUI. Arrived in 2019 which is a declarative form of development applications [12, 13]. The declarative model is now used in major technologies such as Xamarin, UWP, and WPF with XAML [14] with the intention to simplify tasks [15]. This 2020 Apple announced new novelties at WWDC 2020 [16], with new features for the stack mode. These interfaces provide a way to progressively add functionalities. Allow us to develop algorithms as a Lego blocks. One of the interesting features are the possibility to avoid and reduce the need to manage technical details. The advantage is that we can develop advance mobile applications with high-end technology.

2 Methodology

The proposed methodology is agile. The intention is to focus on the development of the three parts which are design, implementation, and feedback. The advantages of this approach are. that we can integrate modules gradually to the architecture such as reconfigure the presentation or add another functionality to the applications. As it seems in Sect. 4.1. The methodology for example also allows reconfiguring the applications in a short period of time, whether it is needed to add elements to the GIS functionality or change the technology from Mapkit to ArcGis without affecting other elements in the project (Fig. 4).

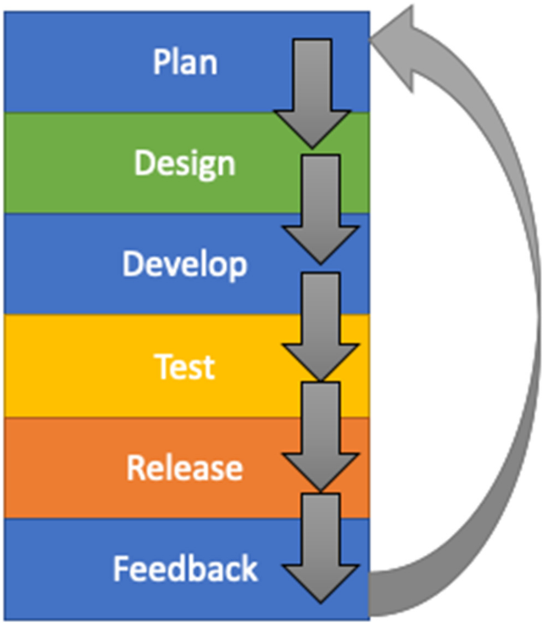


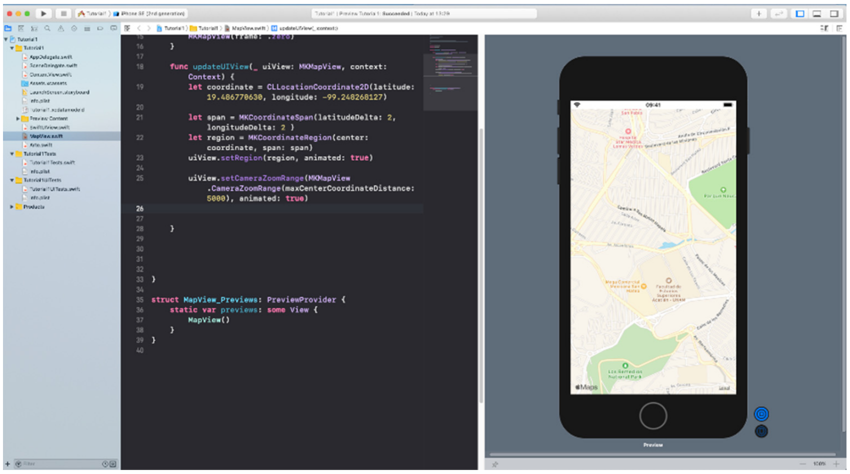
Fig. 4. AGILE development methodology

3 Standard Mobile Implementation of MapKit on XCode

XCode allows the development by using a mechanism called a storyboard. This is made by a file called Main.storyboard. Which represents the structure of the objects and their distribution in the mobile device. As it seems in Xcode the working area of Xcode on the left have file hierarchy section where we can select any file of the project. There's also an edit section where the code can be modified. The file ViewController.swift controls the behavior of the application. Finally, Xcode shows a graphic representation of the Main.storyboard. Where we can insert objects such as buttons, images, and maps and place them where they are suitable. The project needs to be compiled to run in the simulator or an external device. This program is called Simulator. Xcode allows selecting the specific device where the application is designed to be executed. For example, iPad, iPad Pro, iPhone X, iPhone 8, etc. It is important to mention that every time we need to see the result of output. It has to be executed the simulator or the external device. The problem with compiling every time is that if we need to see a minor change, is to spend more time in compiling. As a result, it loses the sense of being programming in real-time.

4 Implementation Using SwiftUI on Xcode

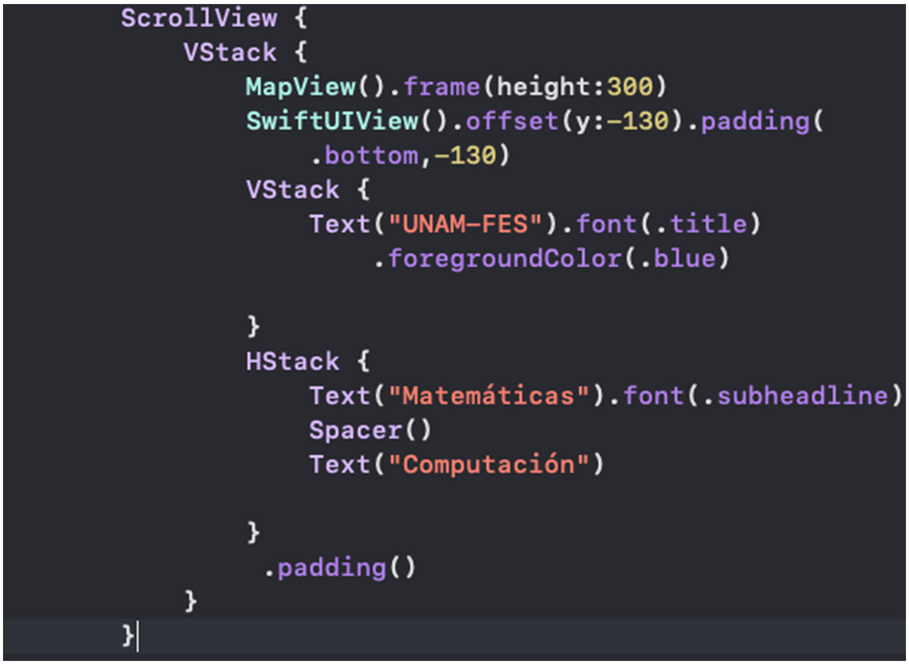
On the other hand, we have a new technology of GUI called SwiftUI. Similar to Main.storyboard. But with new characteristics that permits the development be more dynamically. In Fig. 5 on the left are the working files and edit code (left and center respectively). On the right there is a novelty. A simulation of a mobile phone where every time a change is done in the left a change is reflected on the simulated phone. As was previously described in the last section this model allows also to run the Simulator program.



```
        }

    func updateUIView(_ uiView: MKMapView, context:
        Context) {
        let coordinate = CLLocationCoordinate2D(latitude:
            19.486770630, longitude: -99.248268127)

        let span = MKCoordinateSpan(latitudeDelta: 2,
            longitudeDelta: 2 )
        let region = MKCoordinateRegion(center:
            coordinate, span: span)
        uiView.setRegion(region, animated: true)

        uiView.setCameraZoomRange(MKMapView
            .CameraZoomRange(maxCenterCoordinateDistance:
            5000), animated: true)


    }





}

struct MapView_Previews: PreviewProvider {
    static var previews: some View {
        MapView()
    }
}
```

Fig. 5. SwiftUI graphic interface on Xcode.

One of the features of this technology is that it is possible to build all the components the application needs independently. Which eases the modularity. In addition to that, it allows to create systems and algorithms more robust [8]. Increase cohesion and reduce the coupling of the modules. These are valuable to detect errors easily [17]. That is because the constructions of the elements in SwiftUI use verticals and horizontals stacks structures (Vstack and HStack). In the WWDC2020 introduce an enhance of these structures. As it is shown in Fig. 6 the structures can be nested. In addition to that SwiftUI manages almost automatically the alignment of the elements. So, there is less need to program restrictions. In Fig. 7 we can see the automatic align in a vertical and horizontal position.



```
        ScrollView {
            VStack {
                MapView().frame(height:300)
                SwiftUIView().offset(y:-130).padding(
                    .bottom,-130)
                VStack {
                    Text("UNAM-FES").font(.title)
                        .foregroundColor(.blue)

                }
                HStack {
                    Text("Matemáticas").font(.subheadline)
                    Spacer()
                    Text("Computación")

                }
                 .padding()
            }
        }
```

Fig. 6. Use of Stacks and ScrollView.

4.1 Geographical Implementation Using MapKit and SwiftUI

Suppose our application must add a geographical representation. So, we need to use MapKit. We can divide the application in four parts. One part can describe the distribution of the elements in the phone, similar to HTML. Then we add a functionality or some other elements to the application. We have a module that manages the geographical representation and receive geographical data, longitude and latitude. Finally, a layer that processes the data. In the Fig. 8, we can see the architecture that can be used interchangeably to add the functionality we need. We can see that at the top of the stack we find the Application representation.

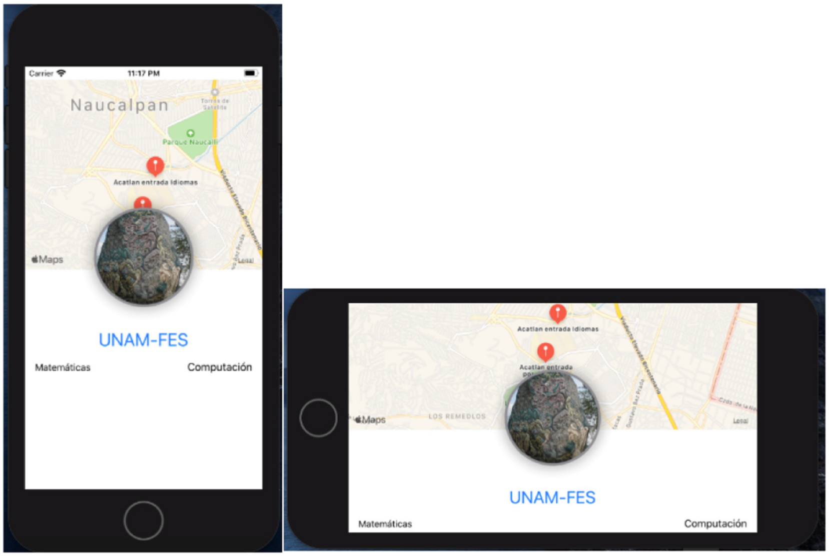


Fig. 7. A vertical and horizontal simulation of an mobile phone.

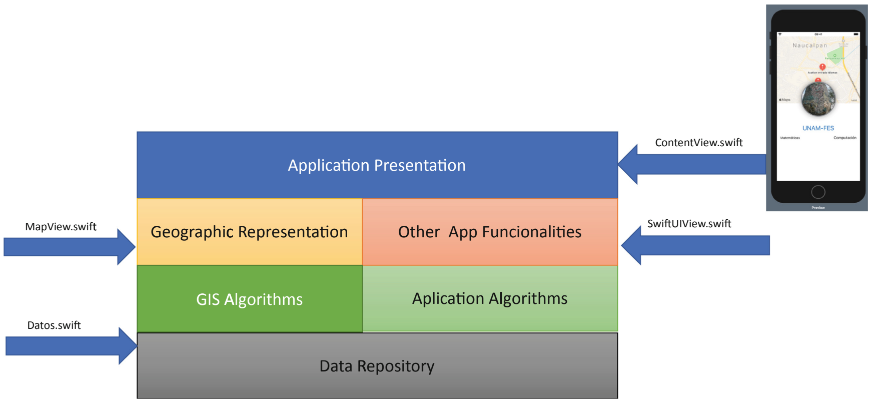
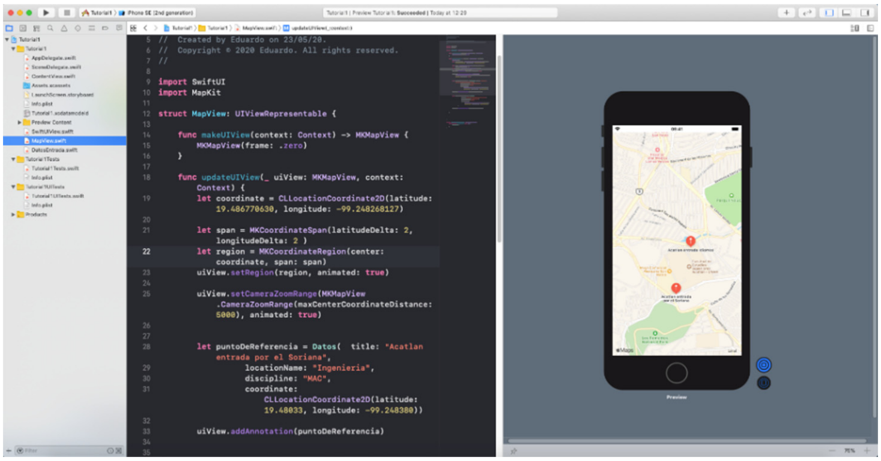


Fig. 8. An architecture for a general geographic application.

4.2 Data Structures Such as GEO JASON or SHAPEFILES

For the manipulation of the Map we have a file that contains all code related to the Maps management. In this case, it is called MapView. And it is important to import the libraries SwiftUI and MapKit and extend the class UIViewRepresentable. The file allows us to represent the geographic locations as it seems in Fig. 9. In SwiftUI, we can add new locations and we will see the change on the right. (The location Added is UNAM-FES Acatlán.).

Then we can add another swift file (In this case SwiftUIView.swift), to add a functionality. In this case we only are interested in adding an image. The result is observed



```
//  Created by Eduardo on 23/05/20.
//  Copyright © 2020 Eduardo. All rights reserved.
//

import SwiftUI
import MapKit

struct MapView: UIViewRepresentable {

    func makeUIView(context: Context) -> MKMapView {
        MKMapView(frame: .zero)
    }

    func updateUIView(_ uiView: MKMapView, context:
        Context) {
        let coordinate = CLLocationCoordinate2D(latitude:
            19.486770630, longitude: -99.248268127)

        let span = MKCoordinateSpan(latitudeDelta: 2,
            longitudeDelta: 2 )
        let region = MKCoordinateRegion(center:
            coordinate, span: span)
        uiView.setRegion(region, animated: true)

        uiView.setCameraZoomRange(MKMapView
            .CameraZoomRange(maxCenterCoordinateDistance:
            5000), animated: true)


        let puntoDeReferencia = Datos(  title: "Acatlan
            entrada por el Soriana",
                locationName: "Ingenieria",
                discipline: "MAC",
                coordinate:
                    CLLocationCoordinate2D(latitude:
                    19.48033, longitude: -99.248380))

        uiView.addAnnotation(puntoDeReferencia)
```

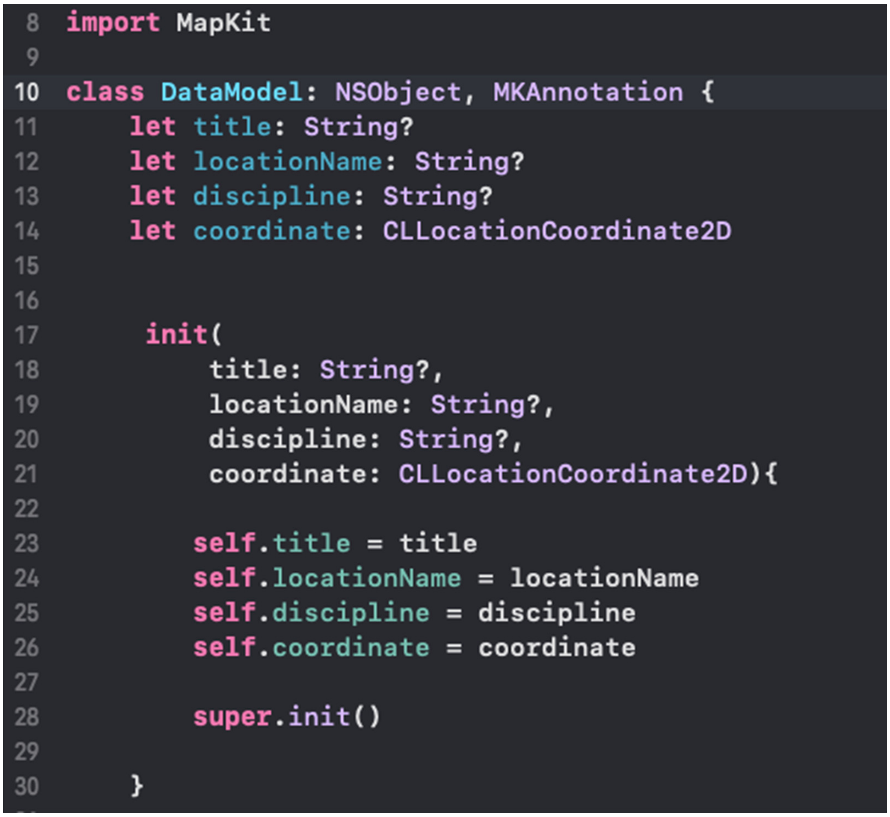
Fig. 9. Adding a map and showing the results on the right.

on the right in the Fig. 9. For this example, the processing of the data is static. But for other applications it is possible to use a database, a GEOJson file (Fig. 10), storing data on the device, etc. The structure uses a class model called DataModel.swift to store the receiving information. The file needs to import the MapKit library so we can use the objects of CLLocationCoordinate and MKAnnotation [18–20]. We also have the option to manipulate shape files. Because as it seems in their technical description [21]. It can be processed as a structure to use it as a Polygon or Multipoint. Now there are also work in ArCGIS to connect it to iOS such as [22] and use it in the ArcGIS implementation in our proposed architecture.

The final result is showed in Fig. 11. Where the ContentView integrates all the elements.

4.3 Similarities Between Main.Storyboard and SwiftUI

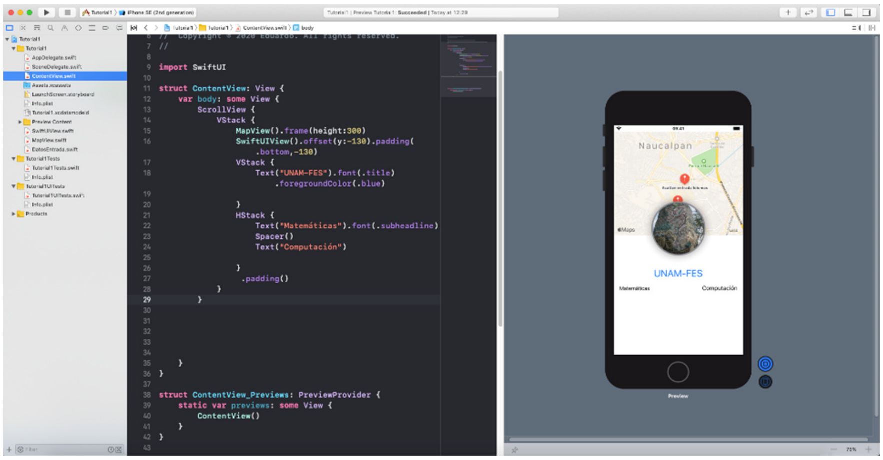
However, it is not necessary to learn new concepts. Whether you are familiarized with Swift and XCode environment. For SwiftUI the structure MKMapView use the function updateUIView(). The function processes the map. And it is possible to set and get elements from MKMapView (such as longitude, latitude, region, zoom). See Fig. 12a).



```swift
import MapKit

class DataModel: NSObject, MKAnnotation {
    let title: String?
    let locationName: String?
    let discipline: String?
    let coordinate: CLLocationCoordinate2D


     init(
         title: String?,
         locationName: String?,
         discipline: String?,
         coordinate: CLLocationCoordinate2D){

        self.title = title
        self.locationName = locationName
        self.discipline = discipline
        self.coordinate = coordinate

        super.init()

    }
```

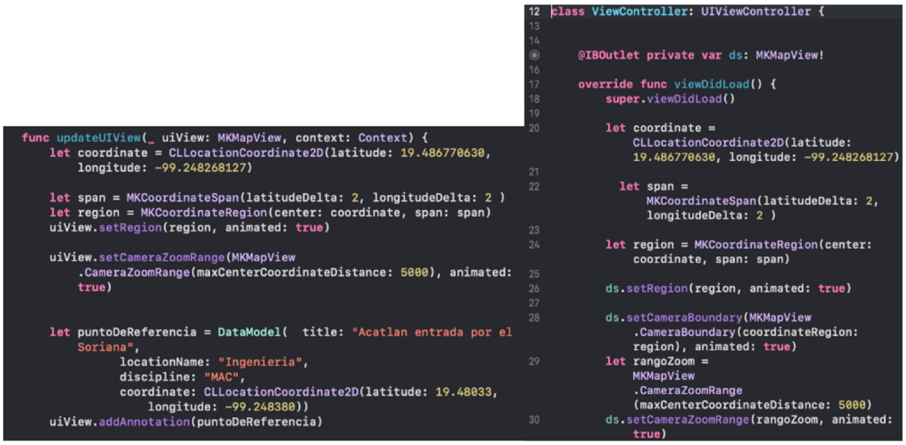
Fig. 10. Class DataModel.

In the case of the Main.storyboard the same methods and properties are used (region, location, span). In this case, there were written in the function viewDidLoad(). To load the map after the application is executed. The difference is that we have to add manually an *outlet* to communicate the Main.storyboard to the ViewController.swift. As it seems on line 15 in the following Fig. 12b).



```swift
//  Copyright © 2020 Eduardo. All rights reserved.
//

import SwiftUI

struct ContentView: View {
    var body: some View {
        ScrollView {
            VStack {
                MapView().frame(height:300)
                SwiftUIView().offset(y:-130).padding(
                    .bottom,-130)
                VStack {
                    Text("UNAM-FES").font(.title)
                        .foregroundColor(.blue)

                }
                HStack {
                    Text("Matemáticas").font(.subheadline)
                    Spacer()
                    Text("Computación")

                }
                 .padding()
            }
        }





    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Fig. 11. The integration of all elements on the ContentView.swift.



```swift
func updateUIView(_ uiView: MKMapView, context: Context) {
    let coordinate = CLLocationCoordinate2D(latitude: 19.486770630,
        longitude: -99.248268127)

    let span = MKCoordinateSpan(latitudeDelta: 2, longitudeDelta: 2 )
    let region = MKCoordinateRegion(center: coordinate, span: span)
    uiView.setRegion(region, animated: true)

    uiView.setCameraZoomRange(MKMapView
        .CameraZoomRange(maxCenterCoordinateDistance: 5000), animated:
        true)


    let puntoDeReferencia = DataModel(  title: "Acatlan entrada por el
        Soriana",
               locationName: "Ingenieria",
               discipline: "MAC",
               coordinate: CLLocationCoordinate2D(latitude: 19.48033,
                   longitude: -99.248380))
    uiView.addAnnotation(puntoDeReferencia)
```

```swift
class ViewController: UIViewController {


    @IBOutlet private var ds: MKMapView!

    override func viewDidLoad() {
        super.viewDidLoad()

        let coordinate =
            CLLocationCoordinate2D(latitude:
            19.486770630, longitude: -99.248268127)

          let span =
            MKCoordinateSpan(latitudeDelta: 2,
            longitudeDelta: 2 )

        let region = MKCoordinateRegion(center:
            coordinate, span: span)

        ds.setRegion(region, animated: true)

        ds.setCameraBoundary(MKMapView
            .CameraBoundary(coordinateRegion:
            region), animated: true)
        let rangoZoom =
            MKMapView
            .CameraZoomRange
            (maxCenterCoordinateDistance: 5000)
        ds.setCameraZoomRange(rangoZoom, animated:
            true)
```

(a) (b)

Fig. 12. a) updateUIView function, b) viewDidLoad function

5 Conclusions

We can see that a simple implementation can be developed with SwiftUI. SwiftUI offers us a high level of characteristics like WYSIWYG. Because as we are implementing a section of code, we can see the result in real-time. This is very helpful when we are searching for accelerated results or testing a functionality. On the other hand, is clear that an application more extensive will require more degree of expressiveness that Main.storyboard can offer. But these two forms of development are not exclusive of any implementation. All the code made in SwiftUI is functional and easily adapted to traditional modality (Main.storyboard) and vice versa. Once you have an algorithm in one version, it can be translated into another. The intention of using this framework is to use SwiftUI as a laboratory. And then integrate the functionalities with a major application or system, isolating possible errors. In addition to that as it was mention in the introduction, now major technologies are used declarative language to ease the development. Summarizing we can see that SwiftUI offers a functional user experience. The Architecture divided into layers is useful to trace errors and add functionalities without affect other modules.

Acknowledgements. The authors would like to thank to the Universidad Nacional Autónoma de México and the Department of Applied Mathematics and Computer Science of FES Acatlán, The "Consejo Nacional de Ciencia y Tecnología" of México. and DGAPA-UNAM (PAPIME PE 304717) for the funds to support this work.

References

1. Theobald, O.: Machine Learning for Absolute Beginners (2017)
2. Bernstein, G.L., Kjolstad, F.: Perspectives: why new programming languages for simulation? ACM Trans. Graph. (TOG) **35**(2), 1–3 (2016)
3. Sarker, I.: Exploiting data-centric social context in phone call prediction: a machine learning based study. EAI Endorsed Trans. Scalable Inf. Syst. **6**(20) (2019)
4. Aybar-Ruiz, A., Jiménez-Fernández, S., et al.: A novel grouping genetic algorithm–extreme learning machine approach for global solar radiation prediction from numerical weather models inputs. Solar Energy **132**, 129–142 (2016)
5. Karniadakis, G., Robert, M., Kirby II, M.: Parallel Scientific Computing in C ++ and MPI. Cambridge University Press (2003)
6. Alpaydin, E.: Machine Learning, pp. 48–49. MIT Press (2016)
7. Tanenbaum, A., Wetheall, D.: Computer Networks. Pearson (2013)
8. Levitin, A.: Introduction to the Design & Analysis of Algorithms, Boston (2012)
9. Inie, N., Dalsgaard, P.: How interaction designers use tools to manage ideas. ACM Trans. Comput.-Hum. Interact. (TOCHI) **27**(2), 1–26 (2020)
10. Dix, A.: Human–computer interaction, foundations and new paradigms. J. Vis. Lang. Comput. **42**, 122–134 (2017)
11. Mcgill, M., Kehoe, A., Freeman, E., Brewster, S.: Expanding the bounds of seated virtual workspaces. ACM Trans. Comput.-Hum. Interact. (TOCHI) **27**(3), 1–40 (2020)
12. Apple. Inc.: Xcode - SwiftUI- Apple Developer. https://developer.apple.com/xcode/swiftui/. Accessed 28 June 2020
13. Apple. Inc.: Introducing Swift UI, *v*. https://developer.apple.com/tutorials/swiftui. Accessed 28 June 2020
14. Microsoft 2020. *Xamarin.Forms | .NET*. https://dotnet.microsoft.com/apps/xamarin/xamarin-forms. Accessed 28 June 2020
15. Heer, J., Bostock, M.: Declarative language design for interactive visualization. IEEE Trans. Visual Comput. Graphics **16**(6), 1149–1156 (2010)
16. Apple WWDC 2020. https://developer.apple.com/videos/play/wwdc2020/10031. Accessed 27 June 2020
17. Singh, Y., Malhotra, R.: Object Oriented Software Engineering. PHH, India (2012)
18. Raywenderlich. https://www.raywenderlich.com/3715234-swiftui-getting-started. Accessed 27 June 2020
19. Hacking with swift. https://www.hackingwithswift.com/books/ios-swiftui/integrating-map kit-with-swiftui. Accessed 27 June 2020

20. Apple Inc. https://developer.apple.com/documentation/mapkit/mkannotation. Accessed 27 June 2020
21. ESRI. ESRI Shapefile Technical Description (1998)
22. ArcGIS Runtime SDK Samples. App Store (2020). https://apps.apple.com/us/app/arcgis-runtime-sdk-samples/id1180714771?mt=8. Accessed 24 July 2020