Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

Implement the SDR representation in the MAUI application

Anh Tong Ngoc Minh
anh.tongngocminh@stud.fra-uas.de

Son Pham Tien
son.phamtien@stud.fra-uas.de

***Abstract*— Due to the drastic demand of users in problems involving real-time solutions, Software developers have options for using sufficient frameworks. In the specification of the app user, a cross-platform framework, .NET Multi-platform App User Interface (.NET MAUI), is a good choice for developers to implement their models compatibly on multiple devices with less work. Besides, a separation in MAUI structure of visualization and functions provides better keeping up with the app development. This paper shows the creation of a .NET MAUI app to interact with users via User Interface (UI). The structure of vital iterations in generating the MAUI app is specified. The purpose is to replace the current Sparse Distributed Representation (SDR) visualization with a simpler and more efficient tool, the new library for SDR drawing implemented using Maui.Graphics.**

Keywords—MAUI, Maui.Graphics, User Interface (UI), cross-platform APIs, Data Binding, XAML, Model-View-ViewModel (MVVM), Sparse Distributed Representation (SDR)

I. Introduction

Visualization plays a vital role in checking and monitoring the project. Developers could point out any error in the system and correct it. Sparse Distributed Representation (SDR) visualization in this project is used to keep track of active cells, which is fundamental to how the Hierarchical Temporal Memory (HTM) model represents information [1]. SDR is essential for gaining insights of the underlying features and characteristics of the processed data.

SDR is a foundational concept in machine learning and artificial intelligence. An SDR is a binary representation of data characterized by its sparsity, where only an insignificant percentage of its bits are active at any given time. Inspired by neuroscience, SDRs mimic encoding information in the human brain, offering a powerful framework for representing and processing complex data patterns. The concept of active columns further enriches the SDR framework, as it comprises an array containing the information of active column indicators. These active columns play a crucial role in encoding and interpreting data, providing insights into relevant features and patterns within the dataset [2].

The concept of Active Columns within SDR is understood as how SDRs function. SDR consists of numerous bits, in which an insignificant fraction are active (set to 1) while the majority remain inactive (set to 0) at any given time. This activation pattern mirrors the behavior of neurons in the human brain, where a 1 represents an active neuron and a 0 signifies an inactive one. Crucially, each bit within an SDR holds semantic meaning, and the set of active bits in a representation collectively encode the semantic properties of the information being represented [1].

The flexibility provided by .NET MAUI allows developers to design intuitive interfaces that seamlessly integrate with underlying data structures. With features such as data binding, Extensible Application Markup Language (XAML), and Model-View-ViewModel (MVVM) architecture, developers can create interactive interfaces that enable users to work with active cell columns.

The construction of the MAUI app comes from layers of User Interface (UI) pages. The system can custom this properly by adding navigation among pages, so that content is split independently depending on the user's arrangement, and still able to be displayed for the user's purpose. The INavigation interface is used to create directional links between two pages [3]. This interface allows different methods of navigation, such as generating the new content of navigated page or keeping existing parameters.

The paper describes the implementation of SDR representation visualization with equivalent required functions, using the Maui.Graphic graphics canvas. This approach offers users an intuitive and efficient means of working with SDR visualization, enhancing their experience and interaction with the MAUI application.

II. Methods

This section describes the vital interfaces used in the .NET MAUI applications, involving the UI design, model structure for logic implementation, and SDR representation function with *Maui.GraphicsView*, and local accessing for file picking.

A. User Interface design

UI design is used to take input SDR parameters from the user. Utilizing the XAML crafts the UI by featuring input fields, representing images, sliders, and buttons. This design strategy allows users to define SDR parameters directly without the complexities of traditional command-line interfaces using Python.

The advantage of the design lies in the ability to create a responsive and easily navigable interface across various devices and orientations. XAML's structure facilitates a more organized and visually coherent UI development process, aligning well with the MVVM architecture. This ensures a correlated connection between the UI and the underlying data models through data bindings [4].

Moreover, XAML's syntax is inherently more concise and readable compared to traditional coding methods. It mirrors the parent-child hierarchy of UI objects, enhancing the visual clarity of the interface structure.

B. Using MVVM as structure for logic implementation

The properties of UI elements come with behind-logic functionality. As applications evolve, they may encounter challenges related to maintenance, such as intricate connections between UI controls and underlying logic, which hinder UI modifications and complicate unit testing.

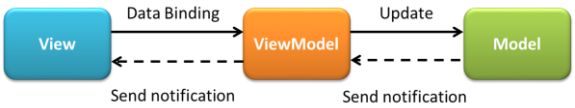

Figure 1: General structure of MVVM model

The MVVM pattern offers a solution by separating an application's business and presentation logic from its UI elements. This separation enhances development by facilitating easier testing, and maintenance. Additionally, it promotes code reusability and fosters collaboration between developers and UI designers.

The three key components of MVVM are the model managing data, the view rendering the UI, and the view model serving as an intermediary between the model and the view. Fig.1 shows the relationships between the three components.

The properties of UI elements come with behind-logic functionality. The **View** holds the outline of the arrangement, design, and visual presentation of the content visible to the user on the screen. Each View is created using XAML, accompanied by minimal code-behind that refrains from integrating business logic.

The **ViewModel** sets up properties and commands where View can connect, allowing data binding. It informs the View about any changes in state using notification events. While the View Model determines what functionality the UI will offer based on the properties and commands it provides. The View decides how to display this functionality. Additionally, the view model manages the interactions between the View and any necessary Model class.

The **Model** is a class, having no visual representation, but is used to hold the data. As a result, the Model is seen as a reflection of the application's domain Model, which typically encompasses the data Model, and various business rules and validation logic.

C. SDR reprentation function with Maui.GraphicsView

The GraphicsView, serves as a canvas for rendering 2D graphics, utilizing Microsoft.Maui.Graphics namespace types. Defined within this framework is the Drawable property, using an IDrawable interface, which delineates the rendered content [5]. This particular property is supported by a BindableProperty, enabling it to become a focal point for data binding and styling processes. The general implementation of *GraphicsView()* in an XAML file is shown in Listing 1. *HeightRequest* and *WidthRequest* in Listing 1 are defined with constant values, while those are auto-adjusted in this project.

```
<ContentPage.Resources>
   <drawable:GraphicsDrawable x:Key="drawable" />
</ContentPage.Resources>
<VerticalStackLayout>
   <GraphicsView Drawable="{StaticResource drawable}"
                 HeightRequest="300"
                 WidthRequest="400" />
</VerticalStackLayout>
```

Listing 1: The XAML code sniper of GraphicsView() in the UI [5]

ICanvas is the graphical tool's interface, part of Maui.Graphics, used for drawing purposes [6]. The interface offers a wide selection of high-quality canvas prints, ranging from classic to contemporary styles, making it easy for customers to find the perfect artwork to adorn their spaces. The drawing library is based on three methods: DrawString(), DrawRectangle(), and DrawRoundedRectangle().

D. Accessing files in local device

The FilePicker() class in the .NET MAUI IFilePicker interface represents a significant leap forward in enhancing user experience by simplifying access and selection of files from the device's storage. The class bridges the gap between the application and the device's native file management capabilities, offering an intuitive interface for users across all supported platforms, including iOS, Android, macOS, and Windows [7].

Because MAUI-based applications in this project focus on data representation, such as those visualizing SDRs, FilePicker() enables users to import the necessary data files for visualization. Therefore, it appears as a more interactive experience for users. The ease with which users can select and upload files directly impacts the effectiveness of the data visualization process.

III. IMPLEMENTATION

The app, named AppSDR, is conducted on specific steps of implementation. Fig. 2 shows the overview structure of the MAUI app, with the flow of input data across the system. Inputs are taken from the Main Page and Text Editor Page, while Page 1 represents an output manufacturer with input streams from the other two and the drawing library. This section provides the implementation of UI, drawing library and logic to build AppSDR, the .NET MAUI app.

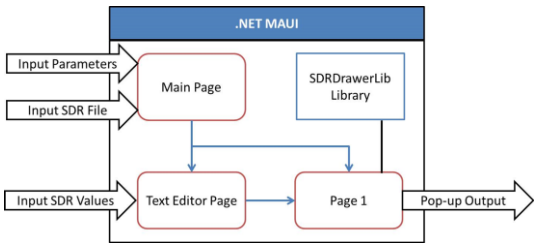

Figure 2: General structure of MAUI implementation

A. UI implementation

UI implementation can be illustrated in Fig. 3, showing the primary components of the three displayed pages. The

correlation of some elements is shown and discussed in the next paragraphs.

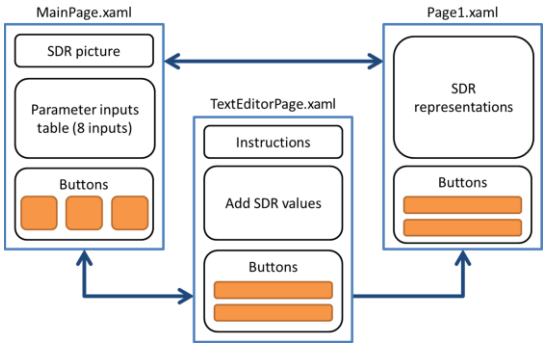

Figure 3: General UI structure implementation

The app has three pages interacting with users: Main Page, Text Editor Page, Page 1. The primary page, loaded when running AppSDR, is Main Page. In .NET MAUI, the user can define the primary page, via AppShell.xaml. The App Shell is the first accessed when running the MAUI app, holding the defined primary page and presenting its content.

There is an SDR descriptive image, a table having eight entry cells to take user's parameters, and three buttons, in Main Page. The Text Editor Page has one block for entered SDR values, instructions, and two buttons. Page 1 has a big block for SDR representations and two buttons.

Arrows present the navigation among pages. As can be seen, Main Page can navigate to Page 1 and vice versa, same for Main Page and Text Editor Page. However, navigation between Text Editor Page and Page 1 is one-direction. When users are on Page 1, they cannot go back to Text Editor Page. One option is to go back to Main Page and start again.

The important property of Page 1 is that it has two visualization modes, fit and expand mode. Fit mode exists when the number of SDR columns, or SDR representations is less than 30, ensuring the visualization for the displayed screen. Extend mode happens when the columns are more than 30 but less than 5100. This mode enables the horizontal ScrollView to expand the screen, leaving space for the plot to cover the whole data range. Specific characteristics of two modes are shown in library implementation and testing cases.

B. Drawing library implementation

The drawing library has one c# file defining class SdrDrawable(), the SDR drawing c# class. For drawing properties, this class uses two libraries as in the following figure. System.ComponentModel is used for any changing variables, and Microsoft.Maui.Graphics is used for specifying the Font internal variable, and allows the ICanvas drawing interface.

Seven parameters need to be initialized when the class is first called in the project. They are GraphName, MaxCycles, HighlightTouch, XAxisTitle, YAxisTitle, MinRange, and MaxRange. These are not normally changed. In addition, there are four public variables, assigned later when needed. They can be changed multiple times.

The Draw(canvas, dirtyRect) is the primary function, must-generated when assigning SdrDrawable() with the IDrawable interface. This function includes primary functions, depending only on the initial parameters. When this function is called, all functions defined are executed.

After parameters initialization comes the distinguished drawing functions. Necessary parameters are defined on the method signature. The components include specifications as: the availability of the function, the function name, and used-parameter names along with their types.

The implementation of the library is specified in Listing 3. Before using the library, it must be included in the logic page, with *using* the library namespace syntax. The SdrDrawable() acts as an object, so a drawing object is initialized and parsed with compulsory parameters.

```
using AppSDR.SdrDrawerLib;
…
        SdrDrawable drawable = new
SdrDrawable(graphName,maxCycles,highlightTouch,x
AxisTitle,yAxisTitle,minRange,maxRange);
        drawable.Draw(canvas, dirtyRect)
```

Listing 3: Code snippet of implementing SdrDrawerLib library

The drawing library is successfully implemented is ready to be used as the user's demand.

C. Logic implementation

Logic implementation is constructed as in Fig. 4. Besides the default logic c# files of each UI page, ViewModel is used as the reason described in MVVM in the introduction and method. In short, there are default logic files and two ViewModel files.

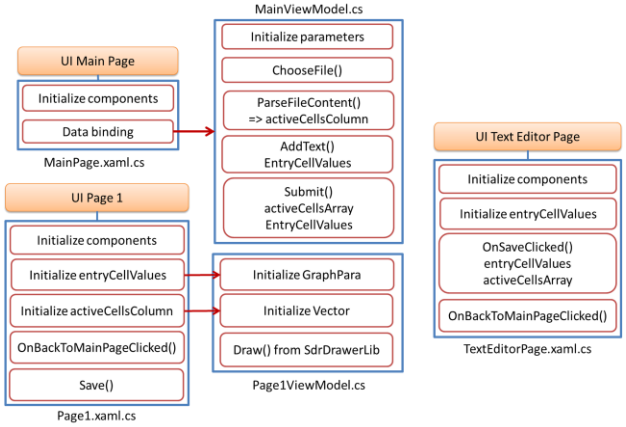

Figure 4: General logic implementation of AppSDR

The logic configuration is described in MainPage(), and MainModelView().

- MainPage() has the responsibilty of initializing all the components in corresponding UI, and parsing all binding context to MainViewMode(). To access the ViewModel pattern, it must be included.
- MainViewModel() works on the binding parsed parameters; therefore, interface INotifyPropertyChanged is implemented within the class. These parameters must be initialized and then

processed. The participating functions involve file selection, form input, and submission.

- The ChooseFile() method enables users to select a text file from their device using FilePicker API. It gracefully handles any exceptions during file picking and alerts users of errors. Properties like graph names, axis titles, and ranges store user input for visualization parameters. They automatically update the UI through the OnPropertyChanged event.
- AddText() creates an array of entry cell values and navigates to a Text Editor Page for text editing when users want to input text data. This function is valid only when the first 7 parameters are entered.
- Submit() processes form submission by reading the selected file's content, parsing it into a 2D array of integers, and preparing data and navigate to Page 1. It alerts users if no file is selected. This function is valid only when the first 7 parameters are entered.
- ParseFileContent() parses the file content into a 2D array of integers, validating rows to include only those with at least one non-zero value. It handles parsing errors by throwing exceptions, ensuring reliable data handling.

Next, Page1(), and Page1ViewModel() classes are discussed.

- Width adjustment in the Page1 class is calculated according to the number of SDR columns present in the dataset. Different configurations are applied for smaller and larger datasets to optimize the visualization's display.
- Save() allows users to capture and save a screenshot of the visualization. Robust error handling is integrated to manage potential failures during the saving process, ensuring a seamless user experience.
- The BackToMainPageButton_Clicked() handles navigation back to the main page of the application, providing users with an intuitive way to transition between different sections of the app.
- The Page1ViewModel() implements the IDrawable interface and inherits from BindableObject and INotifyPropertyChanged to facilitate data binding and property change notifications.
- The Draw() method is responsible for rendering the visualization on the canvas. It retrieves the graph parameters and SDR vectors, calculates the dimensions of the drawing area, and then utilizes the SdrDrawable class to draw the visualization elements.
- The method iterates through each SDR column, determining the height and position of each rectangle based on the cell values. It also handles highlighting of specific columns and draws tick marks and axis titles for better visualization.
- Conditional logic is applied to adjust the visualization based on the size of the canvas and the provided data parameters, ensuring optimal display regardless of the dataset's characteristics.

Third, TextEditorPage() class is discussed.

- The OnSaveClicked function orchestrates actions when the save button is clicked. It first checks for empty content and prompts the user if necessary. Then, it parses the content into a 2D integer array, ensuring non-zero rows. After successful parsing, it presents a success alert and navigates to the next page. Exception handling is in place to promptly notify users of any encountered errors.
- The OnBackToMainPageClicked() function responds to the back button click event by initiating navigation back to the MainPage of the application. Its primary role is to facilitate seamless navigation, allowing users to return to the main interface effortlessly.

IV. Results and Evaluation

For the evaluation of the app's efficiency, several tests are considered. There are three test cases as the representatives of the app properties. Table 1 shows the differences in the configured participating parameters. All three cases take configured parameters from the table on the Main Page.

Case 1 and case 2 use input SDR values from a text file. While case 1 has a total column of less than 30, as defined, the plot will fit the UI screen, case 2 has more than 1000 columns, requiring the screen expanding, and adapting with the horizontal ScrollView. Case 3 allows users to input SDR values from the keyboard. Test cases in Table 1 are also run on the Python version, for comparison purposes.

TABLE I. Comparison characteristic within 3 cases

Variables	**Case 1**	**Case 2**	**Case 3**
Input parameters	Table entry	Table entry	Table entry
Input SDR values	File	File	Keyboard
SDR columns	<30	>1000	8
Screen fit	Fit	Scroll view	Fit
Saved figure	Good quality	Bad quality	Good quality

Fig 3 is included as the example of case 1 of the Python model, and App SDR test results. The tests' results obtain the following points about the AppSDR's performance.

- In Fig. 3, some of the defined parameters are missed, compared to the AppSDR.
- The data range is not plotted accurately in the Python model, while the AppSDR plots the requirements about the limit range.
- When saving the plot, it took less time using App SDR.
- Case 2 has similar advantages when comparing AppSDR's results to the Python model's results. The differences lie in the narrow covering data plot and the saving time.
- Case 3 happens with AppSDR only, as the new feature of the visualization model.

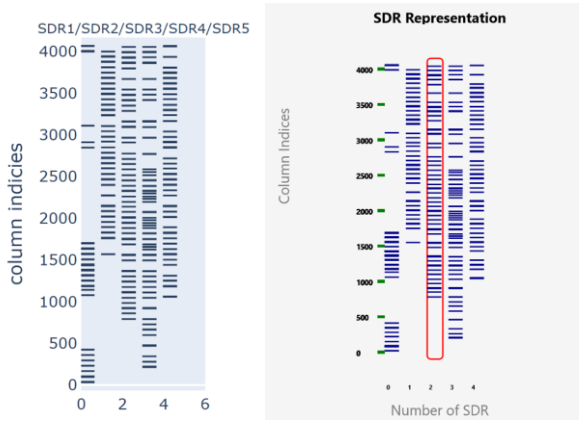

Figure 3: SDR representations of Python model and MAUI Application

In this study, the potential of .NET MAUI in enhancing visualization and interaction capabilities of SDR applications is provided. Inspired by neuroscience principles, SDRs offer a powerful framework for representing complex data patterns, with active columns playing a crucial role in encoding and interpreting data. Leveraging the flexibility and cross-platform capabilities of .NET MAUI, user interaction is upgraded and facilitated ease of use in SDR applications.

Through the project, a user-friendly interface is successfully implemented for inputting SDR parameters, mirroring the functionality of traditional command-line interfaces and eliminating the need for manual input of file paths via command-line arguments. The .NET MAUI and XAML integration allowed us to design responsive and easily navigable interfaces across various devices and orientations, aligning well with the MVVM architecture.

Introducing features such as the File Picker System in .NET MAUI has significantly enhanced the user experience by simplifying the process of accessing and selecting files, facilitating a more interactive user experience. The additional Text Edit feature implementation empowers users to customize and input SDR values directly, enhancing flexibility and control.

Finally, users can save the output of SDR representations directly to their desktop as image files, enhancing the utility of the application and enabling further analysis or dissemination of visualized SDR representations.

V. Conclusion

In conclusion, .NET MAUI is an integrated tool in app generation. With a combination of supported interfaces in .NET MAUI, a friendly user application is generated and implemented within the project. The app allows users to have one more option in inputting the SDR values and configuring the visual-defined parameters. The input local content is dynamic, so it is possible to implement the app on any device. The app is concerned about the user's input parameters, files, or entered SDR values, not the directory or dependent connection to any project; therefore, it can be integrated with distinguished projects having various demand.

The integration of .NET MAUI in SDR applications holds tremendous potential for enhancing user interaction, accessibility, and usability, and enhancing user satisfaction in software applications. Further research and development in this project promise to unlock even greater possibilities for leveraging SDR visualization.

Acknowledgment

We would like to thank Dr. Dobric for his active support and assistance in the direction and model configurations. We are thankful to our tutor for helping us solve encountered problems.

References

[1] S. Ahmad and J. Hawkins, ‘Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory’. arXiv, Mar. 25, 2015. doi: 10.48550/arXiv.1503.07469.

[2] ‘Sparse Distributed Representation and Hierarchy: Keys to Scalable Machine Intelligence’. Accessed: Mar. 29, 2024. [Online]. Available: https://apps.dtic.mil/sti/citations/tr/AD1006958

[3] dotnet-bot, ‘INavigation Interface (Microsoft.Maui.Controls)’. Accessed: Mar. 29, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/microsoft.maui.controls.inavigation?view=net-maui-8.0

[4] davidbritch, ‘XAML - .NET MAUI’. Accessed: Mar. 29, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/xaml/?view=net-maui-8.0

[5] davidbritch, ‘Graphics - .NET MAUI’. Accessed: Mar. 29, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/graphics/?view=net-maui-8.0

[6] dotnet-bot, ‘ICanvas Interface (Microsoft.Maui.Graphics)’. Accessed: Mar. 29, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/microsoft.maui.graphics.icanvas?view=net-maui-8.0

[7] davidbritch, ‘File picker - .NET MAUI’. Accessed: Mar. 29, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/storage/file-picker?view=net-maui-8.0