Creating a state machine to handle speech recognition feature

This article explains how to handle the speech recognition feature using a state machine.

Introduction

This article explains how to use a state machine to manage behaviour of a speech recognition application - it does not cover speech recognition itself, which is already covered in a number of articles on the wiki (see the References section). Speech recognition is really just the domain here - the pattern could be used in many similar applications.

Speech recognition is a very nice feature for Windows Phone 8 applications. Voice commands are recognised quickly and a developer can improve their user interface considerably by the inclusion of simple voice commands. Moreover, the user interaction can be changed completely using voice commands.

I've started to use speech recognition feature in some of my projects and it was cool. But when my projects became more complex, the speech recognition became much more harder to handle. The main problem I've faced is that as long as the number of voice commands increase, the `if`-clause to handle them became larger and I had to manage in this `if`-clause when a specific grammar is acceptable or not.

Note: Special thanks to rudyhuyn for his useful information on how to perform speech recognition using a SGRS file.

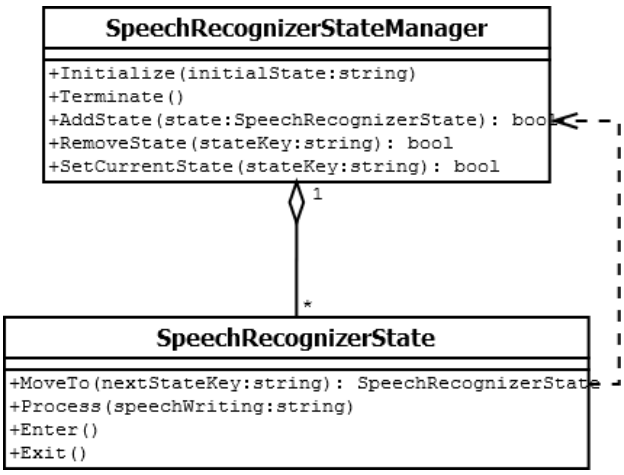
Entering the State design pattern

For this kind of a problem, a good solution is the state design pattern. This design pattern is able to split larger conditional clauses into objects; at runtime. Moreover, this pattern also allows to execute a method when the application change from a state to another. This is exactly what I wanted to do.

This post is about a Windows Phone class library that implements the state design pattern to control the speech recognition feature. It's helping me in some speech-recognition based projects and I hope that it can help other developers too.

Architecture

The class library is composed by only two classes as shown below:



The `SpeechRecognizerStateManager` class manages the available states and send the speech writing to the current state; The `SpeechRecognizerState` abstract class has three abstract methods: `Process(string speechWriting)`; `Enter()`; and `Exit()` and a constructor with two parameters.

The `Enter()` method is called after a state is activated; The `Exit()` method is called before a state is inactivated; The `Process(string speechWriting)` is called every time a string is *listen* by the application and the user must call the `MoveTo(string stateKey)` to set a different current state; Also, the `SpeechRecognizerState` constructor has two parameters: a state key, that is a unique string in the state machine and a grammar source, that can be a `IEnumerable<string>` or a `Uri` representing the recognized grammar.

How to use (A small example)

This section provides a small example of how to use the class library. The example illustrates how to listen for user's questions and answer them. It handles only two subjects: `Math` and `English`. If the current subject is `Math`, the application is able to listen for both questions **1 + 1** and **2 + 1**; if the current subject is `English`, the application is able to listen for both words **car** and **run** and tell the user whether the word is a noun or a verb.

(C) Copyright Nokia 2013. All rights reserved.

If the user tries to ask **1 + 1** while in `English` subject, the application will do nothing. The same happens if the user says **car** while `Math` subject is activated. To control this feature in an If-clause, the user would have to create a control variable in order to know what is the current state, and handle any command in an unique big If-clause method.

I'm going to map each subject into a `SpeechRecognizerState` subclass which allows me to define a different grammar for each subject and split the logic between the two new states. Moreover, I'm able to activate only one subject at a time.

EnglishState class

It inherits from the `SpeechRecognizerState` class and implement the three methods shown in the previous class diagram.

```
class EnglishState : SpeechRecognizerState
    {
        ///The MainPage instance used to update UI.
        private MainPage currentPage;

        /// <summary>
        /// Creates the EnglishState instance. It sends the stateKey and the recognized commands to the base class.
        /// </summary>
        /// <param name="page">The main page used to update the UI.</param>
        public EnglishState(MainPage page)
            : base("english", new string[] {"car", "run", "math"})
        {
            currentPage = page;
        }

        /// <summary>
        /// Process the 'car' and 'run'' voice commands and change to the MathState if the 'math' voice command is received.
        /// </summary>
        /// <param name="speechWriting">The speech wirting.</param>
        public override void Process(string speechWriting)
        {
            if (speechWriting.Equals("car"))
            {
                currentPage.Dispatcher.BeginInvoke(() =>
                {
                    currentPage.CurrentAnswer.Text = "noum";
                });
            }
            else if (speechWriting.Equals("run"))
            {
                currentPage.Dispatcher.BeginInvoke(() =>
                {
                    currentPage.CurrentAnswer.Text = "verb";
                });
            } else if(speechWriting.Equals("math"))
            {
                MoveTo("math");
            }
        }

        /// <summary>
        /// Update Ui after entering the state.
        /// </summary>
        public override void Enter()
        {
            currentPage.Dispatcher.BeginInvoke(() =>
            {
                currentPage.CurrentSubject.Text = "English";
            });
        }

        /// <summary>
        /// Update Ui before exiting the state.
        /// </summary>
        public override void Exit()
        {
            currentPage.Dispatcher.BeginInvoke(() =>
            {
                currentPage.CurrentSubject.Text = "";
                currentPage.CurrentAnswer.Text = "";
            });
        }

    }
```

MathState class

It inherits from the `SpeechRecognizerState` class and implements three methods shown in the previous class diagram.

```
    class MathState : SpeechRecognizerState
    {

        private MainPage currentPage;

        /// <summary>
        /// Creates the MathState instance. It sends the stateKey and the recognized commands to the base class.
        /// </summary>
        /// <param name="page">The main page used to update the UI.</param>
        public MathState(MainPage page)
            : base("math", new string[] {"1 plus 1", "1 plus 2", "english"})
        {
            currentPage = page;
        }

        /// <summary>
        /// Process the '1 plus 1' and '1 plus 2' voice commands and change to the EnglishState if the 'english' voice command is re
        /// </summary>
        /// <param name="speechWriting">The speech wirting.</param>
        public override void Process(string speechWriting)
        {
            if (speechWriting.Equals("1 plus 1"))
            {
                currentPage.Dispatcher.BeginInvoke(() =>
                {
                    currentPage.CurrentAnswer.Text = "2";
```


(C) Copyright Nokia 2013. All rights reserved.

```
            });
        }
        else if (speechWriting.Equals("1 plus 2"))
        {
            currentPage.Dispatcher.BeginInvoke(() =>
            {
                currentPage.CurrentAnswer.Text = "3";
            });
        } else if(speechWriting.Equals("english"))
        {
            MoveTo("english");
        }
    }

    /// <summary>
    /// Update Ui after entering the state.
    /// </summary>
    public override void Enter()
    {
        currentPage.Dispatcher.BeginInvoke( () =>
        {
            currentPage.CurrentSubject.Text = "Math";
        });
    }

    /// <summary>
    /// Update Ui before exiting the state.
    /// </summary>
    public override void Exit()
    {
        currentPage.Dispatcher.BeginInvoke(() =>
        {
            currentPage.CurrentSubject.Text = "";
            currentPage.CurrentAnswer.Text = "";
        });
    }

}
```

Main Page class

In this example the `SpeechRecognizerStateManager` instance is created within the `MainPage` constructor besides the `MathState` and `EnglishState` instances.

The `MainPage` class is shown below.

```
public partial class MainPage : PhoneApplicationPage
{
    private SpeechRecognizerStateManager speechRecognizerManager;

    // Constructor
    public MainPage()
    {
        InitializeComponent();
        /// Creates the SpeechRecognizerManager instance.
        speechRecognizerManager = new SpeechRecognizerStateManager();

        // Add the created states in the SpeechRecognizerManager instance.
        speechRecognizerManager.AddState(new MathState(this));
        speechRecognizerManager.AddState(new EnglishState(this));
    }

    protected override void OnNavigatedTo(NavigationEventArgs e)
    {
        // Initialize the SpeechRecognizerManager instance.
        // The attribute is the initial state.
        speechRecognizerManager.Initialize("math");
    }

}
```

I haven't shown any library code in the post because I want to show only the code that the developer has to implement to use the library. If you want to see how the library is implemented, the code is fully commented. You can download the class library File:SpeechRecognizerManagerProject.zip which also has the example project included.

References

- Speech recognition using a SGRS file
- Using SSML for advanced text-to-speech on Windows Phone 8
- Writing a 3D DirectX maze that's navigable with speech commands
- Speech Enabled Calculator For Windows Phone 8
- Voice Commands in Windows Phone 8
- Map with directions in Windows Phone 8
- State design pattern
- Working with grammars on Windows Phone 8
- Managing loaded grammars to optimize recognition for Windows Phone 8

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.