Methods of Executing Python Script using C#

Project title: 3D Geometry HULL Calculations

SEP Group: HULL2

By: Nivin Jose Kovukunnel (a1749677)

Research Scope

The implementation for Part A is done using the programming language Python 3. An important specification that the client stated is that, a C# wrapper must be created. This allows a python script to be executed from C#.

Basic concept

The outside layer needs to be C# - the program you compile and run, or the function that takes a model and returns a 2D hull. Hence, internally, the function can be implemented using Python, however, for it must be able to integrated with C#.

There are several ways run to a Python script in C#. A method is to call the Python script using a new process with the installed Python interpreter. Another alternative is to, call the Python script using IronPython interpreter, hosted in the .NET application.

Calling Python Script Using New Process Initialization:

The main idea here is to call the script using a newly initialized process and get its standard output. This method needs an external Python interpreter to actually execute the script.

A simple code solution for this method is illustrated in Figure 1.

```
public string run_cmd(string cmd, string args)
        {
            ProcessStartInfo start = new ProcessStartInfo();
            start.FileName = "PATH_TO_PYTHON_EXE";
            start.Arguments = string.Format("\"{0}\" \"{1}\"", cmd, args);
            start.UseShellExecute = false;// Do not use OS shell
            start.CreateNoWindow = true; // We don't need new window
            start.RedirectStandardOutput = true;// Any output, generated by application will be redirected back
            start.RedirectStandardError = true; // Any error in standard output will be redirected back (for example exceptions)
            using (Process process = Process.Start(start))
            {
                using (StreamReader reader = process.StandardOutput)
                {
                    string stderr = process.StandardError.ReadToEnd(); // Here are the exceptions from our Python script
                    string result = reader.ReadToEnd(); // Here is the result of StdOut(for example: print "test")
                    return result;
                }
            }
        }
```

Figure 1. New Process Initialization

It is important to consider permissions of the account under which the main application is running.

Calling the Python Script Using IronPython Interpreter, Hosted in Your .NET Application.

IronPython interpreter can be hosted in the .NET application. Using NuGet, the IronPython package can be downloaded and installed. Then, embed the script execution (actually the IronPython engine) right into the application.

```
public string PatchParameter(string parameter, int serviceid)
        {
            var engine = Python.CreateEngine(); // Extract Python language engine from their grasp
            var scope = engine.CreateScope(); // Introduce Python namespace (scope)
            var d = new Dictionary<string, object>
            {
                { "serviceid", serviceid},
                { "parameter", parameter}
            }; // Add some sample parameters. Notice that there is no need in specifically setting the object type, interpreter will do that part for
            us in the script properly with high probability

            scope.SetVariable("params", d); // This will be the name of the dictionary in python script, initialized with previously created .NET
            Dictionary
            ScriptSource source = engine.CreateScriptSourceFromFile("PATH_TO_PYTHON_SCRIPT_FILE"); // Load the script
            object result = source.Execute(scope);
            parameter = scope.GetVariable<string>("parameter"); // To get the finally set variable 'parameter' from the python script
            return parameter;
        }
```

Figure 2. IronPython Method

The main difference here is that the second method is fully managed code, so you get all of the exceptions and control over the script execution right there in your application. With the first method, you lose control over the started process internals. Sure you can see the exceptions using standard output and control process wait timeouts, but all this is all additional code that needs to be supported.