Augmentiqs

SEE THE SMART PATH

Augmentiqs integration with 3rd party algorithms (v.2.3)

The purpose of this document is to outline tool functionality as well as steps for integration with responsibilities of the parties.

Augmentiqs software suite overview

Augmentiqs consists of hardware optical module that is installed in the microscope and a software client that runs on dedicated PC connected to the optical module via USB interface.

Optical module contains various devices (multiple cameras, controllers as well as projection system), that are controlled by the software application.

Augmentiqs application provides core functionality such as morphometrics analysis (selection and measurement via rectangle, ellipse, polygonal shapes that user may freely define), telepathology and more.

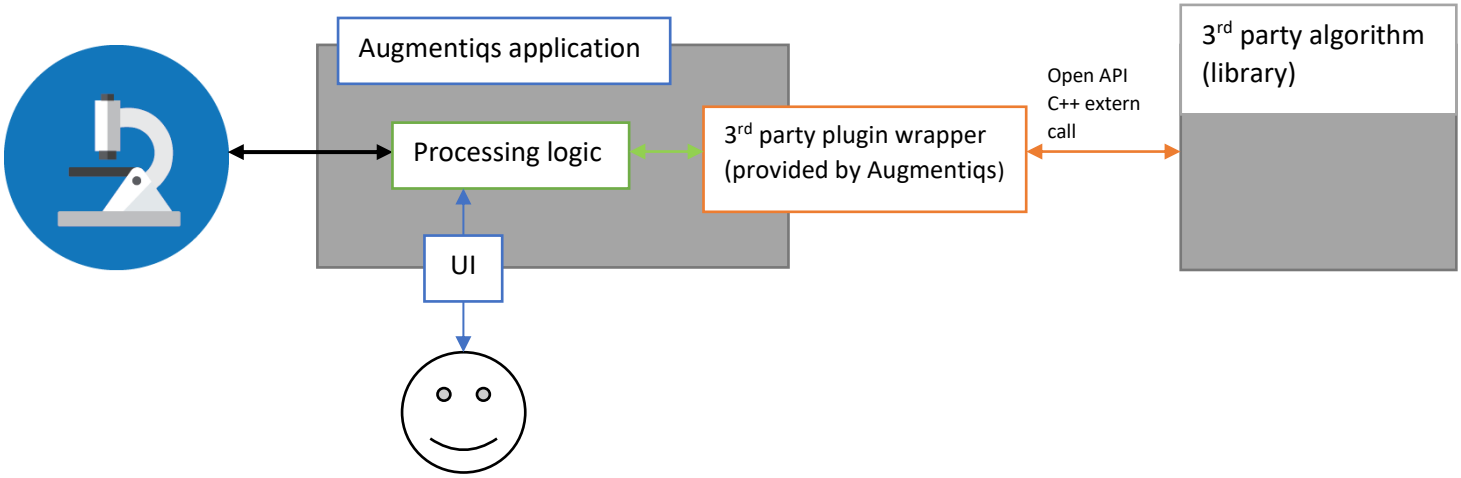
Beside that functionality, Augmentiqs application interacts with 3rd party plugins via standard interface in order to extend its functionality and provide solution for specific diagnostic needs.

The main idea of the interface is that any image processing algorithm, be it classical or ML-based, needs to obtain raster image with some additional metadata that can be conveniently encoded in JSON format. The output of the algorithm may vary, but ultimately can be presented as a whole image overlay in raster format as well as textual vector data that may encode some regions needed to be highlighted, statistical data etc. The flexibility of JSON format enables to encode any sort of data and exact format of JSON structure needs to be negotiated per each plugin separately.

Augmentiqs software takes responsibility to obtain input from the user via its UI, pass the input along with capture still image from the main camera to the 3rd party algorithm, wait till its processing is complete and present the output to the user. Consider the supported flow below:

1. User is shown an unmodified image of the tissue
2. User selects specific 3rd party tool via dedicated button
3. OPTIONAL: User picks desired color for future segments (if applicable)
4. OPTIONAL: User moves mouse to outline ROI over the desired tissue segments
   or provide additional paths via standard morphometrics tooling
5. 3rd party tool analyzes the tissue samples under the ROI/path and gives out some output in form of raster/vector data (e.g. full image overlay and/or highlighted segments of the image colored by previously selected color)
6. OPTIONAL: The cycle may repeat (additively) while user may pick more colors/paths

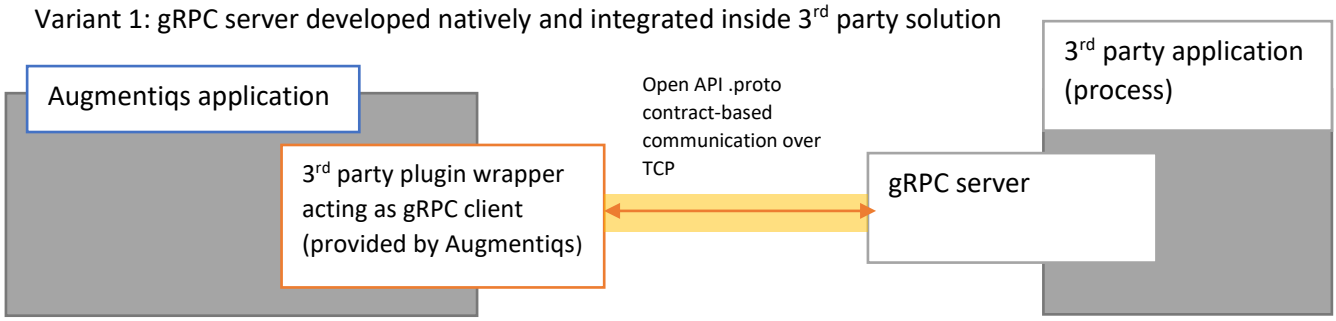
The diagram below shows a typical integration topology with the 3rd party algorithm provided as a dynamically linked library.



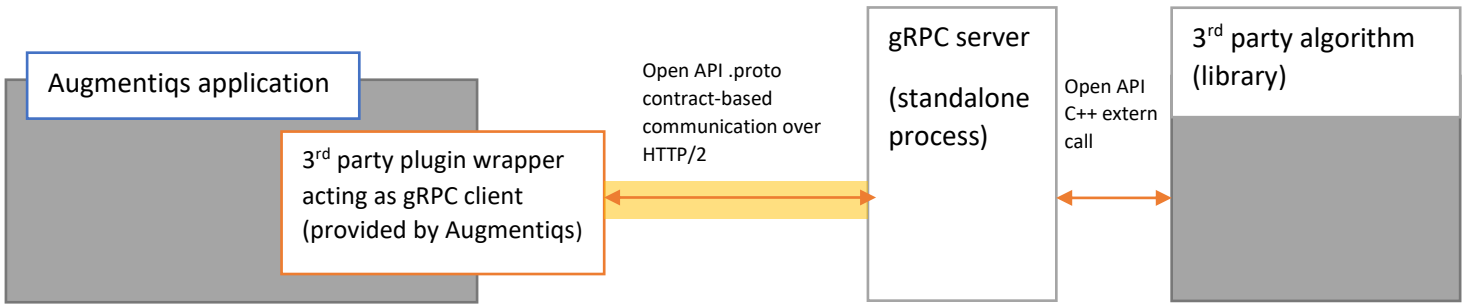
In case this is not possible, and 3rd party solution is a separate process, which may even work on the remote host or different OS, a standard solution for IPC is present, which is based on widely popular gRPC protocol from Google. It is based on HTTP/2 protocol which is fast and secure, while serialization and marshalling is done via Google Protocol Buffers (protobuf) library. There is a plethora of gRPC clients available for any platforms and languages.

Consider variants below:

Variant 1: gRPC server developed natively and integrated inside 3rd party solution



Variant 2: gRPC is a thin standalone process that is developed externally



C++ extern call signature:

```
#define EXPORT_API __declspec(dllexport)

extern "C" int EXPORT_API __stdcall GetProcessedResult(char* inputImage,
                                                       int imageWidth,
                                                       int imageHeight,
                                                       const wchar_t* inputMeta,
                                                       char* outputImage,
                                                       char* outputMeta);
```

P/Invoke call signature:

```
[DllImport("<plugin>.dll", CallingConvention = CallingConvention.StdCall, CharSet = CharSet.Unicode)]
private static extern int GetProcessedResult([In] byte[] inputImage,
                                             [In] int imageWidth,
                                             [In] int imageHeight,
                                             [In] string inputMeta,
                                             [Out] byte[] outputImage,
                                             [Out] byte[] outputMeta);
```

where:

`inputImage` – PNG-encoded bitmap serialized to byte array
`imageWidth` – input and output image width in pixels
`imageHeight` – input and output image height in pixels
`inputMeta` – JSON-encoded string with number of additional parameters
`outputImage` – PNG-encoded bitmap serialized to byte array (transparent background with colored mask of segmentation areas)
`outputMeta` – JSON-encoded string with output data, or alternatively any type of encoded information passed in unstructured byte array
return value is typically used for error codes, where 0 means OK and non-zero values meaning varies per publisher/plugin

inputMeta JSON format (just an example, real format should be negotiated between the parties):

```
{
  "Mag": "4",
  "PixelSize": 1.23456789,
  "Paths": [
    {
      "Points": [
        "96, 563",
        "29, 542",
          ...
      ],
      "Color": "#FF00A2E8"
    },
  ]
}
```

where:

`Mag` – current magnification factor
`PixelSize` – input image pixel size (in microns)
`Paths` – array of `Path` structure
`Path` structure contains the following fields:
`Points` – array of mouse selection path points (pixels X, Y coordinates inside input image, origin is top-left)
`Color` -- selection path color (in web format aka: #AARRGGBB)

gRPC contract

```
syntax = "proto3";

option csharp_namespace = "AugmentIQs";

package plugin;

// The plugin interop service definition.
service PluginInterop {
  // Sends a plugin request
  rpc GetProcessedResult (PluginRequest) returns (PluginResponse);
}

// The request message containing the necessary image data and metadata.
message PluginRequest {
  bytes input_image = 1;
  sint32 image_width = 2;
  sint32 image_height = 3;
  string input_meta = 4;
}

// The response message containing the return code, output image and metadata.
message PluginResponse {
  sint32 return_code = 1;
  bytes output_image = 2;
  string output_meta = 3;
}
```

The definition of this contract is straightforward and repeats the definition of aforementioned extern C++ call.

For obtaining GRPC interop sample code (C++ & C#) please address Augmentiqs support.