**A Transparent Bridge Between OsiriX and MeVisLab**

# Motivation:

OsiriX is an ideal software for medical image research, as it provide a user-friendly GUI for medical staffs to organize and view medical images. More important is its compatibility with different types of scanners and PACS system. OsiriX also provides some simple image processing methods and visualization tools for clinical usage, however it is not enough for various research projects. The plugin interface is an important extension to this system, which allow third-part provided algorithm running for special tasks. But developing plugin requires a lot of coding, compiling and testing. It can take very long time.

MeVisLab, on the other side, provides a graphic-based programming and testing environment. This prototyping tool includes hundreds of modules that allow a developer to design a complicate workflow without writing a single code. Another great thing is when you are changing parameters you needed to recompile the software which is very annoying when building a software from C code. A shortcoming of MeVisLab is lack of user-friendly data organization system. When you have more cases saved in the file system, it will become more and more difficult to locate a desired file.

Inspired by Felix Ritter’s OsiriX MeVisLab Bridge, I started to build this “transparent” bridge between OsiriX and MeVisLab. The idea is to use OsiriX as a front-end GUI for doctors and make MeVisLab running in background invisible to them. Meanwhile the developer can design and test they image processing network inside MeVisLab without knowing how to programming inside OsiriX.

# Techniques Involved:

Two key inter-process communication techniques are used in the bridge. One is “[Distributed Objects Programming](http://developer.apple.com/documentation/Cocoa/Conceptual/DistrObjects/DistrObjects.html)” based on Object C 2.0 , the other is “[POSIX Shared Memory](http://www.kohala.com/start/unpv22e/unpv22e.chap12.pdf)” based on Unix system.

Distributed Objects technique provides a way to sending message between different applications. The procedure is very similar to the network communication, one application open a listening port, the other try to connect to that port (figure 1). However, data in the message is copied between applications, so it is not suitable to transport large mount of data between applications.

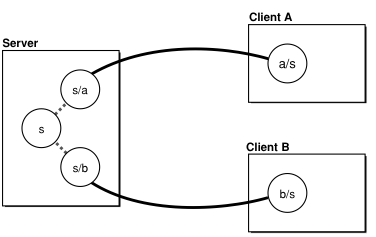


Figure 1, Client application A and B communicate with Server application via distributed object technique.

Shared memory is a technique that maps the same physical memory block to different application’s memory address. Although the memory usage increases in both application, physically only one copy is loaded in the real memory.

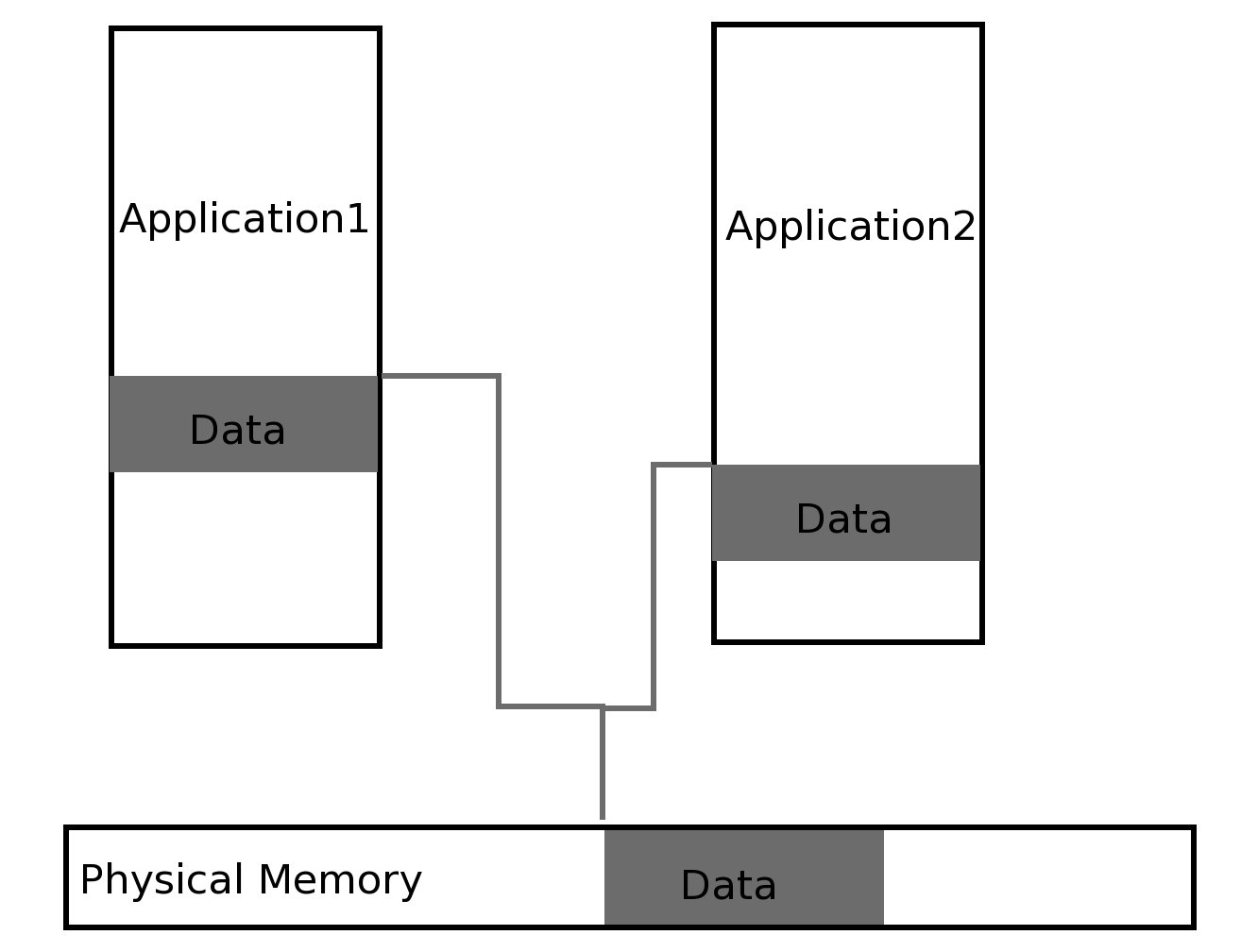


Figure 2. Memory Sharing, one piece of memory is mapped to two applications’ memory address. They can both read and write into this block of memory and changes will effect the other application immediately.

Combining these two techniques, we can easily transport almost any kind of data between OsiriX and MeVisLab very quickly.

# The Data Flow:

In Background

Users Interface

Figure 3. Data flow of a typical operation. Image data and user input is all collected by OsiriX and send to MeVisLab. A multiple-step operation may require the data exchanging for several times.

In Background

Process Network

Process Network

MeVisLab GUI

Users Interface

Figure 4. Another possibility of the data flow. Image data is passed down to MeVisLab, and MeVisLab will take the user input and showing the result. Results can be send back to OsiriX if necessary.

The image data is transferred from OsiriX to MeVisLab and transfered back after processing. The user’s input, on the other side, has two ways to enter the data flow. One is via the plugin’s user interface and transferred to MeVisLab (figure 3), the other way is using the use interface of MeVisLab directly(figure 4).

# Communication Between OsiriX and MeVisLab

A typical operation consists of three steps of communication between OisriX and MeVisLab (Figure 5).

First OsiriX sends out the input image to the Importer module in MeVisLab. The transferred information is a NSArray object which contains at lease 5 NSDictionary objects representing input data(Figure 5 gives more detail definition of a image). The order of those first 5 images is fixed, the first one is the input images in float type, the second is label map in char type associated with the input image(loaded from brush ROIs), the third is a second input image from the fused volume(via OsiriX fusion function), the fourth is the label map associated with this input image, and the fifth is users’ input (clicking points) in ROI form. The five export ports of importer ML are also in this order. If the key work “type” of a image is set to be “null”, this image will be ignored.

Then OsiriX sends an operation request to the Exporter module (definition in Figure 6).

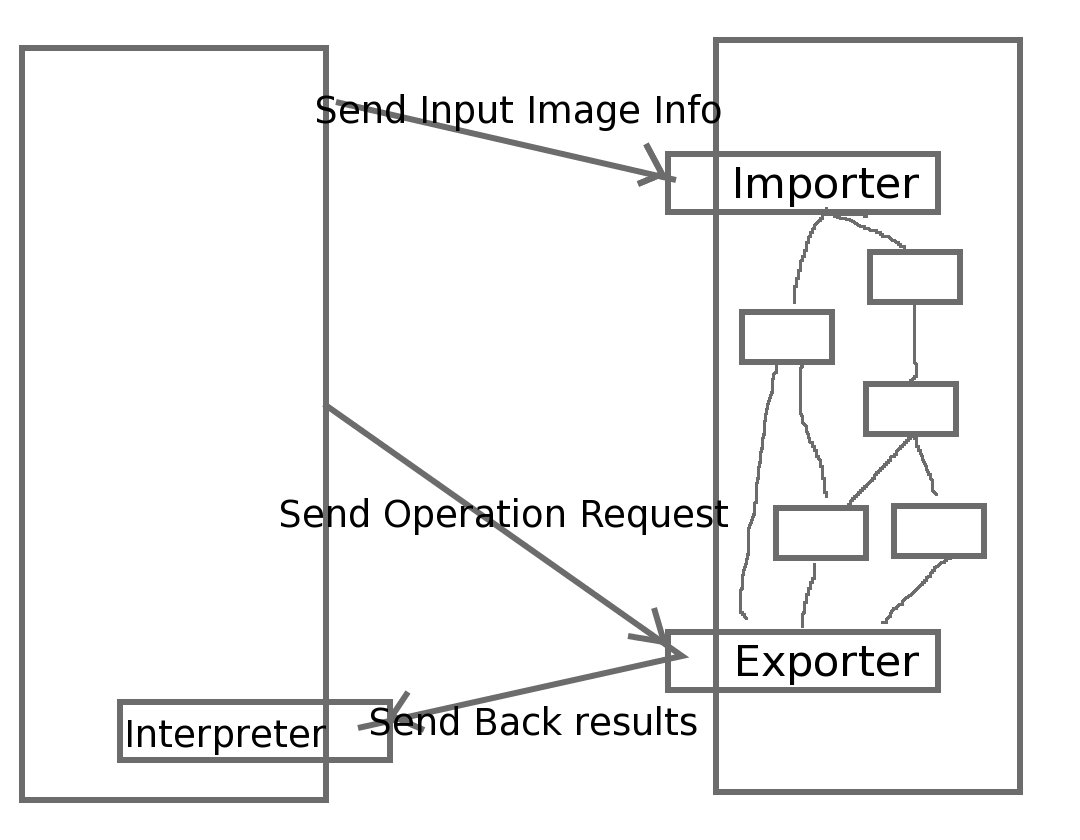


Figure 5. Communication between OsiriX and MeVisLabe.

Exporter module will create the output images corresponding to the operation request and send it back to OsiriX where the result is shown and saved into the database. The transferred information is also a NSArray object which contains the output images. There can be at most one image in float type and one in char type which will be shown in the result window as image and label mask. There can be one or more ROI type output images which represents the contour of an object or a centerline of a shape. The order of those outcome images is not important anymore as OsiriX will show them in one window in different form.

Exporter module can also send an operation request, like “save to database”, to OsiriX after sending the output images.

NSDictionary\* aImage{

Description: NSString\*

Type: NSString\* (“float”, “char”, “overlay”)

SharedMemoryID: NSString\*

MemSize: NSNumber\*

DICOMSeriesUID: NSString\*

Dimension: NSArray\* {NSNumber\*;NSNumber\*;NSNumber\*;NSNumber\*} (x,y,z,t)

Spacing: NSArray\* {NSNumber\*;NSNumber\*;NSNumber\*;NSNumber\*}(x,y,z,t)

MatrixToPatientCo: NSArray\* {NSNumber\*;NSNumber\*;NSNumber\*……} (4\*4=16)

Minimum: NSNumber\*

Maximum: NSNumber\*

Points: NSArray\* {

NSDictionary\* apoint {NSNumber\*;NSNumber\*;NSNumber\*; NSNumber\*;NSNumber\*} (x,y,z,t,value)

NSDictionary \* apoint {NSNumber\*;NSNumber\*;NSNumber\*; NSNumber\*;NSNumber\*} (x,y,z,t,value)

.........

}

OverlayType: NSString\* (“points”, “polygon”, “openpolygon”, “smoothedpolygon”, “smoothedopenpolygon”)

ColorLUT: NSArray\*{NSColor\*, NSColor\*, NSColor\*…}

}

Figure 6. The key valued contained in an image object transferred between OsiriX and MeVisLab.

NSDictionary\* aOperation{

Operation: NSString\*

Parameters: NSDictionary \*{

Key🡪Parameter Name: NSString\*,

Value: 🡪Parameter Value: id

}

}

RelatedImages: NSMutableArray\*{

NSDictionary\* aImage //as shown in Figure 6

}

}

Figure 6 An operation(NSDictionary) object transferred between OsiriX and MeVisLab.

# An Example of Using OsiriX – MeVisLab Transparent Bridge

# Further Improvement:

Memory Saving

Running On Different Machin