OpenITGLinkMUSiiC: A Standard Communications Protocol for Advanced Ultrasound Research

Release 1.00

Hyun-Jae Kang[1], Philipp J Stolka[1] and Emad Boctor[1,2]

July 15, 2011

[1]Computer Science, Johns Hopkins University, Baltimore, MD/USA
[2]Department of Radiology, DMIP, Johns Hopkins Medical Institutions, Baltimore, MD/USA

Abstract

Ultrasound imaging is the most popular and safe diagnostic medical imaging modality. Over the past years, a considerable number of studies have been conducted on new medical ultrasound imaging modalities such as ultrasound-based elasticity imaging, photoacoustic imaging, thermal imaging, etc. Moreover, results of such advanced ultrasound research are readily integrated with Image Guided Therapy (IGT) systems due to the advantages of ultrasound imaging such as mobility and real-time data acquisition. For integrating two systems, a communication method between systems has to be devised. OpenIGTLink is a standardized TCP/IP-based protocol for the integration of medical imaging and IGT systems. However, OpenIGTLink is not sufficient for communicating between IGT and advanced ultrasound research systems, because OpenIGTLink does not define any message type for general ultrasound data and some results of ultrasound researching.
In this paper, we propose an extension to OpenIGTLink, called OpenIGTLinkMUSiiC, by adding new message types and new network classes. Our new message types are designed for containing raw ultrasound data, advanced ultrasound modalities, and text-based control parameters. The network classes of OpenIGTLinkMUSiiC improve upon the performance of existing network classes by implementing a multithread architecture. Finally, we present two ultrasound research applications using OpenIGTLink-MUSiiC.

Contents

1 Introduction

Clinical ultrasound imaging is the most popular and safe diagnostic medical imaging modality. Moreover, ultrasound is frequently used as the imaging basis of Image Guided Therapy (IGT) systems [1] due to its features such as real-time data acquisition, mobility, and harmless nature for patient and operator.

Over the past years, a multitude of research directions have sprung up around new medical ultrasound imaging modalities: ultrasound elasticity imaging [1-3], ultrasound photo-acoustic imaging [4], ultrasound thermal imaging [5], and etc. The results of the advanced ultrasound research are also integrated with IGT system by providing new ultrasound image modality.

But, for integrating the results of advanced ultrasound research with existing IGT systems, we have to consider two things. One concern is the collection of low-level ultrasound data, Radio-Frequency (RF) data or pre-beamformed RF data, in real-time. Such low-level ultrasound data is a necessary part for computing advanced ultrasound image modalities such as ultrasound elastography or ultrasound photo-acoustic imaging. Another aspect is data communication with existing IGT systems. For solving the former issue, we developed an ultrasound research platform, called MUSiiC Toolkit, in our previous research [6]. In this paper, we will follow up on this and discuss a standard communication protocol for integrating our application modules of the MUSiiC Toolkit with existing IGT systems via TCP/IP networking.

In 2008 a standard TCP/IP network protocol for IGT systems named OpenIGTLink was announced. This protocol enabled the communication of position, transformation, and image data, as well as commands or system status messages between two processes of distributed IGT systems [7-10]. Thanks to its simple and extensible nature, several research groups and commercial companies has taken up OpenIGTLink in their development [8]. Also, our group chose OpenIGTLink for communicating ultrasound data, the outputs of our advanced ultrasound modules, and localization data of tracker devices between modules of our MUSiiC Toolkit and existing research IGT systems such as 3DSlicer.

However, the existing OpenIGTLink turned out not to be sufficient for this application scenario, as it included no specific messages that can contain extra information of low-level ultrasound data or output values of advanced ultrasound computation. So, in this paper, we introduce our extension version of OpenIGTLink, named OpenIGTLinkMUSiiC, for advanced ultrasound research in real-time, and some applications using it.

This paper is organized as following: Section 2 describes new message types and new network classes of OpenIGTLinkMUSiiC. In Section 3, we introduce several applications of advanced ultrasound research with OpenIGTLinkMUSiiC. We conclude with a discussion in Section 4.

2 OpenIGTLinkMUSiiC

The OpenIGTLink protocol has been suggested by the NAMIC group for providing a standardized TCP/IP network mechanism for image-guided therapy (IGT) systems [9, 10]. For its features – open, simple, platform-independent, and ready-to-use – several IGT research groups have used this tool for their own applications and research.

Recently, OpenIGTLink has experienced upgrades by the addition of new message types, supporting several data types of IGT systems: tracking data, image-meta data, points of fiducials, trajectory data etc. [10]. However, there is still no message type for containing general ultrasound data or advanced ultrasound imaging modalities on even the new versions of OpenIGTLink.

Our MUSiiC Toolkit is composed of several specialized executable modules. Each module has its own task, and they are communicating with each other via TCP/IP networking. That is, each module has its own TCP/IP server socket, client socket, or both. Furthermore, communication in the MUSiiC Toolkit is unidirectional. In this case, a server socket provides data to other client modules. On the other side, the client socket of the client module receives the data on behalf of the module. Frequently, a server socket will provide the same data to multiple client modules, e.g. for navigated laparoscope ultrasound systems [6]. For this reason, multiple-clients connection functionality of the server socket is necessary. Although the existing OpenIGTLink has its own server and client socket implementations, these do not support multiple-client connections.

Because of these reasons, our group built our own extension version of OpenIGTLink by adding several new message types as well as network classes to support multiple clients. In this section, we introduce our new message types and describe of our network classes for multiple-client connection.

2.1 New message types in OpenIGTLinkMUSiiC

In OpenIGTLinkMUSiiC, we add five new message types. Two message types are for transmission of general ultrasound data and the results of our advanced ultrasound research modules. The remaining three messages are for communicating text-based control arguments and files.

2.1.1 New ultrasound messages in OpenIGTLinkMUSiiC

USMessage and *EIMessage* message types in OpenIGTLinkMUSiiC are extensions of the *ImageMessage* in OpenIGTLink, containing general ultrasound data with its extra information and elastography data with the results of its computation, respectively. Figure 1 represents the class diagram and the structure of these two messages with *ImageMessage* of OpenIGTLink. As seen from the figure, the two message types have same structure: each message is a subclass of *ImageMessage*, and each carries its own tag after the *ImageMessage* data block.

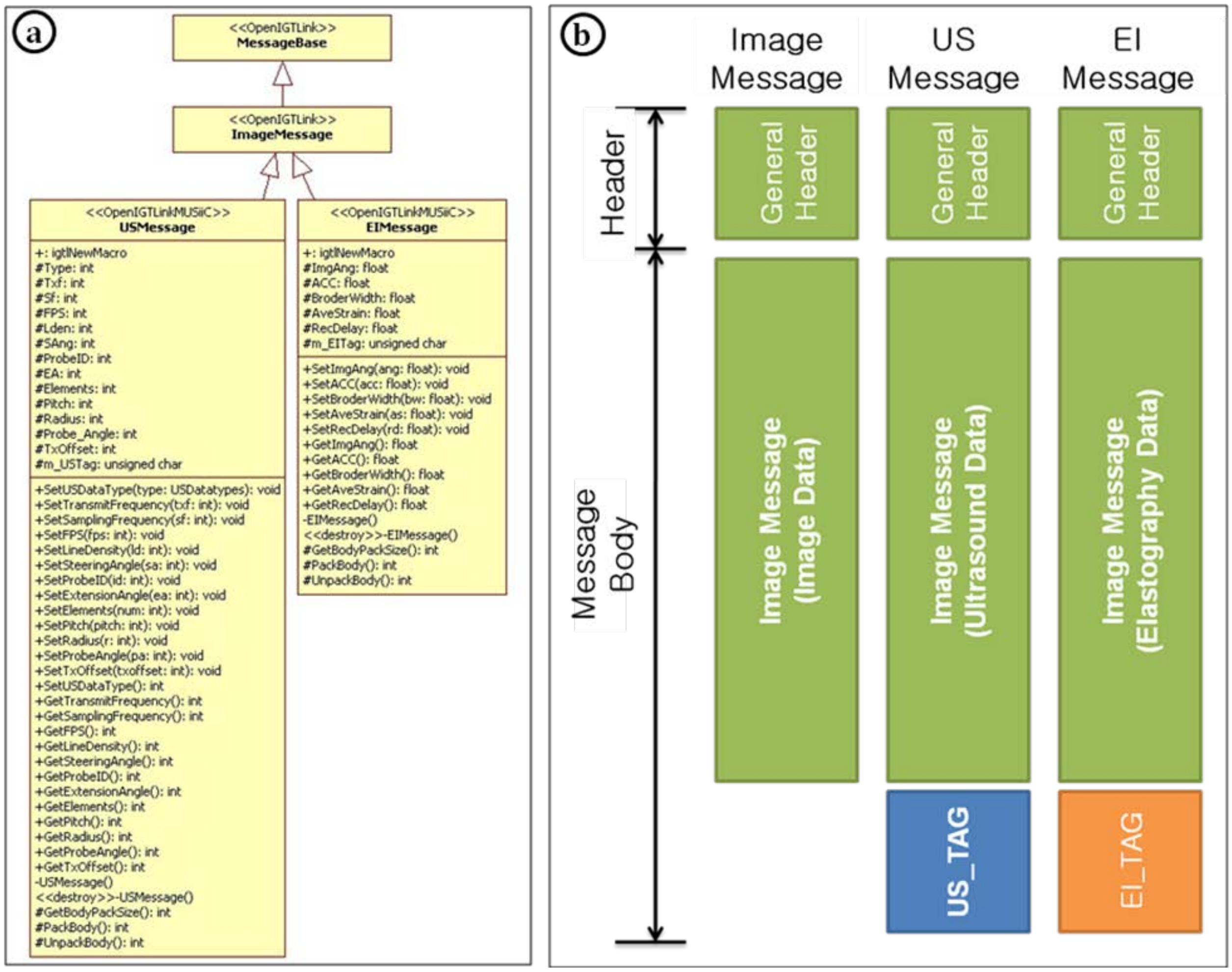

Figure 1 Two new message types – *USMessage* and *EIMessage*: (a) class diagram, (b) general structure.

More specifically, general ultrasound data will be contained in the *ImageMessage* part of *USMessage* with its metric information, data matrix size, position, and orientation. Additionally, ultrasound-specific data is stored in the *US Tag* (Table 1) of *USMessage*. *EIMessage* has same way to store the data of elastography; the *ImageMessage* part of *EIMessage* contains the elastography image and the *EI Tag* (Table 2) has extra information of EI data. As seen in Figure 1, each message has its own serialization and deserialization functions for its own tag.

Because these two message types are subclasses of *ImageMessage*, they can contain 2D/3D ultrasound data or elastography data. In the case of 3D data transfers, sub-volume data can be partially updated using these two messages. Moreover, these messages are backwards-compatible with *ImageMessage* in deserializing procedures.

Data	Type	Description
txf	32bit int	transmit frequency of ultrasound beam
sf	32bit int	sampling frequency of ultrasound data
dr	32bit int	frame rate
ls	32bit int	line density
sa	32bit int	steering angle
probe	32bit int	ID of ultrasound probe
ea	32bit int	extension angle
elements	32bit int	the number of elements in the probe
pitch	32bit int	the spacing between elements
radius	32bit int	the curvature of the probe
probe_angle	32bit int	the field of view of the probe
tx_offset	32bit int	the offset in the steering image (phased array)

Table 1: *US_Tag* of *USMessage*

Data	Type	Description
ImagAng	32bit float	extension angle
ACC	32bit float	Average Cross-correlation
BorderWidth	32bit float	Border-Width of Elasfography
AveStrain	32bit float	Average Strain value
RecDelay	32bit float	The delay of Receiving

Table 2: *EI_Tag* of *EIMessage*

2.1.2 New control messages in OpenIGTLinkMUSiiC

From our experience, we identified the need to communicate a general user-defined message such as “any number of arguments” or “any kind of file”. Although there is a *StatusMessage* type in OpenIGTLink, the message has an already defined set of status parameters. For more universal data transfer and communication, we built new three message types in OpenIGTLinkMUSiiC: *GenMessage*, *ArgMessage*, and *FileMessage*. *GenMessage* has the general header of OpenIGTLink for compatibility with *Pack()* and *UnPack()* functions of the existing OpenIGTLink protocol. There is no definition of any protocol inside the message body part of *GenMessage*. This means that we can put and get any user-defined contents.

ArgMessage has the specific goal of communicating text-based argument message or control parameters. *ArgMessage* is a subclass of the *GenMessage* class (see Figure 2). That is, the arguments of *ArgMessage* will be in the message-body part of *GenMessage*. *ArgMessage* has only serialization and deserialization methods for its text-based arguments or control parameters. A relevant code snippet using *ArgMessage* might look like the following:

ArgMessage* Packing** – Simply add arguments using the function ***AddArgument():

```
        igtl::ArgMessage::Pointer argMsg = igtl::ArgMessage::New();
        argMsg->AddArgument(char* argument1);
        argMsg->AddArgument(char* argument2);
        argMsg->AddArgument(char* argument3);
        argMsg->AllocateArguments();
        argMsg->Pack();
```

ArgMessage* Unpacking** – Extract arguments using the function ***GetArgument():

```
        argMsg->Unpack(1);
        int sz = argMsg->GetNumOfArguments();
        char* arg1 = argMsg->GetArgument(0);
        char* arg2 = argMsg->GetArgument(1);
        char* arg3 = argMsg->GetArgument(2);
```

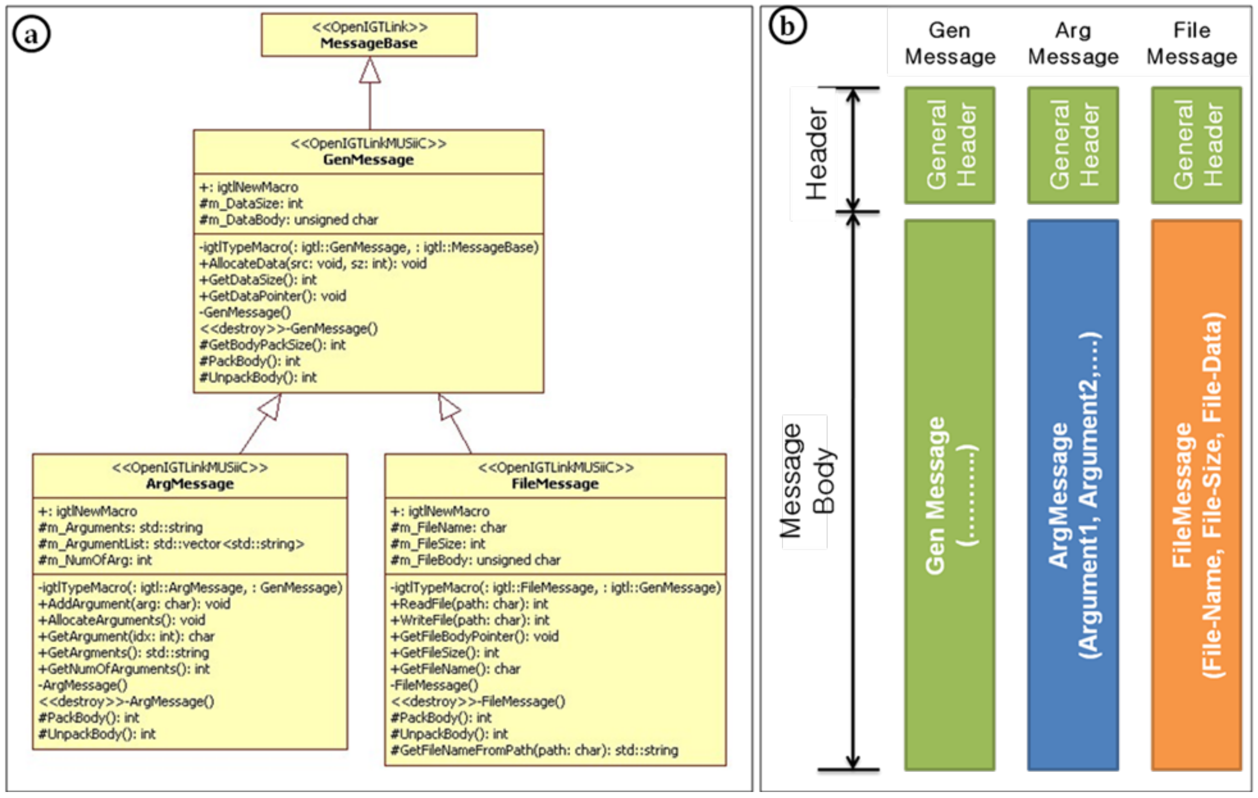
a
<<OpenIGTLink>>
MessageBase
<<OpenIGTLinkMUSiiC>>
GenMessage
+: igtlNewMacro
#m_DataSize: int
#m_DataBody: unsigned char
-igtlTypeMacro(: igtl::GenMessage, : igtl::MessageBase)
+AllocateData(src: void, sz: int): void
+GetDataSize(): int
+GetDataPointer(): void
-GenMessage()
<<destroy>>-GenMessage()
#GetBodyPackSize(): int
#PackBody(): int
#UnpackBody(): int
<<OpenIGTLinkMUSiiC>>
ArgMessage
+: igtlNewMacro
#m_Arguments: std::string
#m_ArgumentList: std::vector<std::string>
#m_NumOfArg: int
-igtlTypeMacro(: igtl::ArgMessage, : GenMessage)
+AddArgument(arg: char): void
+AllocateArguments(): void
+GetArgument(idx: int): char
+GetArgments(): std::string
+GetNumOfArguments(): int
-ArgMessage()
<<destroy>>-ArgMessage()
#PackBody(): int
#UnpackBody(): int
<<OpenIGTLinkMUSiiC>>
FileMessage
+: igtlNewMacro
#m_FileName: char
#m_FileSize: int
#m_FileBody: unsigned char
-igtlTypeMacro(: igtl::FileMessage, : igtl::GenMessage)
+ReadFile(path: char): int
+WriteFile(path: char): int
+GetFileBodyPointer(): void
+GetFileSize(): int
+GetFileName(): char
-FileMessage()
<<destroy>>-FileMessage()
#PackBody(): int
#UnpackBody(): int
#GetFileNameFromPath(path: char): std::string
b
Gen Message
Arg Message
File Message
Header
Message Body
General Header
General Header
General Header
Gen Message (………)
ArgMessage (Argument1, Argument2,….)
FileMessage (File-Name, File-Size, File-Data)

Figure 2 New message types *GenMessage*, *ArgMessage* and *FileMessage*: (a) class diagram, (b) message structure.

FileMessage is used for communicating any file "as itself" in binary form between modules. Apart from *FileMessage*'s information about file name and file size, it is similar to *ArgMessage*; this message is also derived from *GenMessage* and has its own serialization and deserialization functions for files (see Figure 2).

Example codes of packing and unpacking for *FileMessage* might look as follows:

FileMessage* Packing** – Simply add a file using the function ***ReadFile():

```
        igtl::FileMessage::Pointer fMsg = igtl::FileMessage::New();
        fMsg->ReadFile(const char* path);
        fMsg->Pack();
```

FileMessage* Unpacking** – Extract a file using the functions ***WriteFile() and ***GetFileName()***:

```
        fMsg->Unpack(1);
        fMsg->WriteFile(fMsg->GetFileName());
```

2.2 Network classes of OpenIGTLinkMUSiiC

OpenIGTLink provides two network sockets classes (*Serversocket* and *Clientsocket*) for TCP/IP networking. However, there are no independent network threads and buffers for communication in the network classes. Because of this, the TCP/IP networking functionality of the existing OpenIGTLink can be affected by the main thread of a module, or itself affect the performance of a module program.

To improve the networking performance, we implemented our network classes using a multithreaded architecture. That is, our server and client classes have their own independent network thread and data buffer for TCP/IP networking.

2.2.1 Concurrent queue for multithreading program: MUSiiCQueue

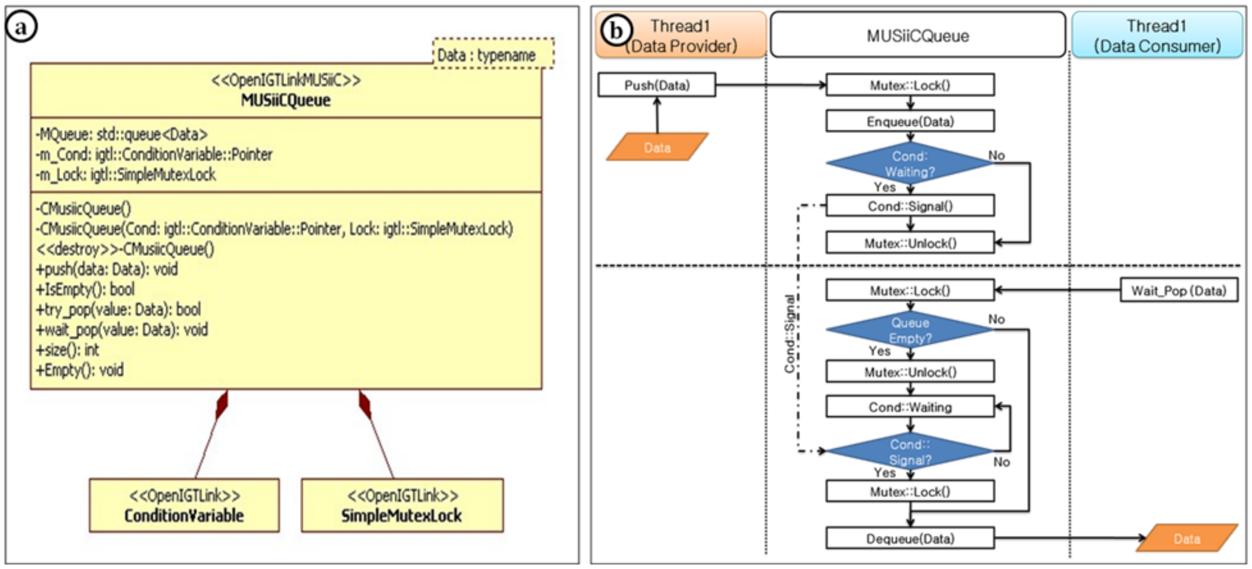

Figure 3 *MUSiiCQueue*: (a) class diagram, (b) flowchart of its *Push()* and *Wait_Pop()* functions

In communication between threads in multithread architectures, possible data corruption should be considered. The traditional method for this job is to use mutexes, critical sections, and locks. To ensure data integrity, we built the concurrent queue class MUSiiCQueue using simple mutex and condition variables of OpenIGTLink.

Figure 3 represents the class diagram of the MUSiiCQueue and the flowchart of its *Push()* and *Wait_Pop()* functions. Our MUSiiCQueue has its own mutex and condition variables of OpenIGTLink as shown. MUSiiCQueue can protect its data from access by another thread when a thread reads or writes data to the queue. Also, if the queue is empty, the condition variable of the MUSiiCQueue holds a thread that requested data from the queue. The condition variable sends a signal to wake up the holding thread when new data arrives in the queue.

2.2.2 Client class of OpenIGTLinkMUSiiC: MUSiiCClientT

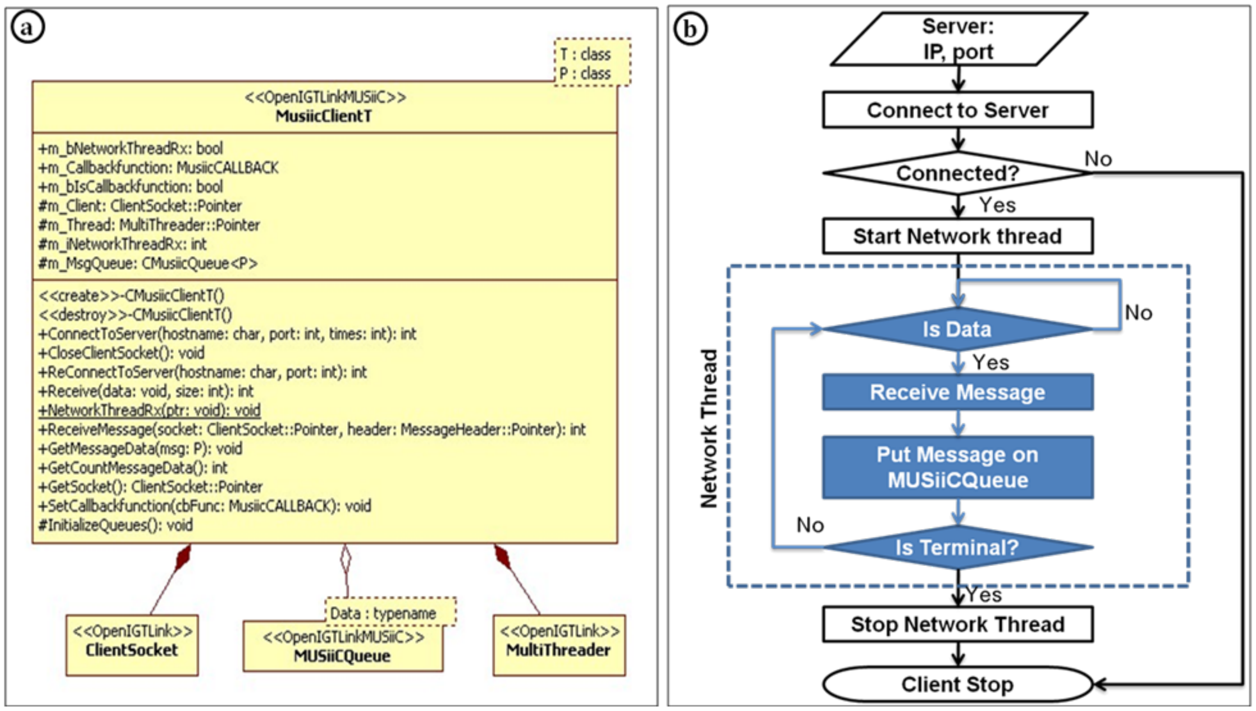
a
T : class
P : class
<<OpenIGTLinkMUSiiC>>
MusiicClientT
+m_bNetworkThreadRx: bool
+m_Callbackfunction: MusiicCALLBACK
+m_bIsCallbackfunction: bool
#m_Client: ClientSocket::Pointer
#m_Thread: MultiThreader::Pointer
#m_iNetworkThreadRx: int
#m_MsgQueue: CMusiicQueue<P>
<<create>>-CMusiicClientT()
<<destroy>>-CMusiicClientT()
+ConnectToServer(hostname: char, port: int, times: int): int
+CloseClientSocket(): void
+ReConnectToServer(hostname: char, port: int): int
+Receive(data: void, size: int): int
+NetworkThreadRx(ptr: void): void
+ReceiveMessage(socket: ClientSocket::Pointer, header: MessageHeader::Pointer): int
+GetMessageData(msg: P): void
+GetCountMessageData(): int
+GetSocket(): ClientSocket::Pointer
+SetCallbackfunction(cbFunc: MusiicCALLBACK): void
#InitializeQueues(): void
<<OpenIGTLink>>
ClientSocket
Data : typename
<<OpenIGTLinkMUSiiC>>
MUSiiCQueue
<<OpenIGTLink>>
MultiThreader
b
Server:
IP, port
Connect to Server
Connected?
No
Yes
Start Network thread
Network Thread
Is Data
No
Yes
Receive Message
Put Message on
MUSiiCQueue
No
Is Terminal?
Yes
Stop Network Thread
Client Stop

Figure 4 *MUSiiCClientT*: (a) class diagram, (b) flowchart

The class diagram and a flowchart of the client class *MUSiiCClientT* are shown in Figure 4. The *MUSiiCClientT* class has an instance of the *ClientSocket* class and a pointer to an instance of the *MultiThreader* class of OpenIGTLink for TCP/IP networking and for its networking thread, respectively. Also, *MUSiiCClientT* has an interface of *MUSiiCQueue* for its own network data buffer and data sharing with other threads. As shown in the class diagram (Figure 4 (a)), *MUSiiCClientT* is implemented as a template class. Thus, any message type of OpenIGTLinkMUSiiC can be received via TCP/IP network using this class.

The network thread checks independently whether there is an OpenIGTLinkMUSiiC message in the TCP/IP buffer or not. If there is a message, *MUSiiCClient* gets the data and puts it in its data buffer to provide the data to other threads. To simplify its usage, only one message type is available using an instance of this class.

2.2.3 Server class of OpenIGTLinkMUSiiC: MUSiiCServerT

Although the existing OpenIGTLink protocol has its own server socket implementation (*ServerSocket* class) for TCP/IP communication, the server socket cannot support multi-client connections by itself. So we built a network server class, named *MUSiiCServerT*, implementing a multithreaded architecture. Figure 5 shows the class diagram and flowchart of our *MUSiiCServerT* class.

In the same way as the *MUSiiCClientT* class, *MUSiiCServerT* has its own independent networking thread for listening to requests of client modules and carries its own network buffer. Moreover, *MUSiiCServerT* has a container of pointers to *MUSiiCClientT* to allow sending data to each client. As seen in Figure 5, *MUSiiCServerT* copies data to the network buffer of its instances of *MUSiiCClientT* when new data arrives in its data buffer. As mentioned in the previous section, *MUSiiCClientT* has its own network thread and data buffer. Due to this feature, each instance of *MUSiiCClienT* which is stored in the container of *MUSiiCServerT* will send the data to the connected client module independently.

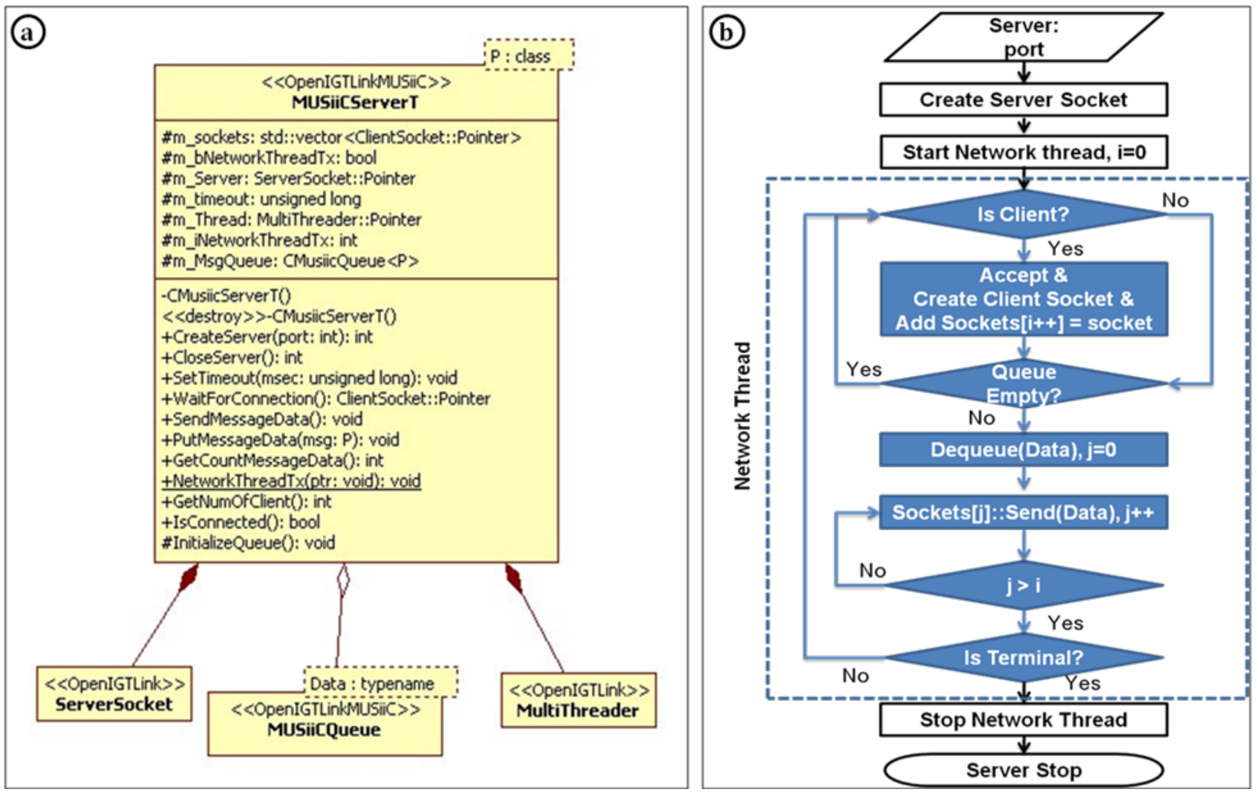
a
P : class
<<OpenIGTLinkMUSiiC>>
MUSiiCServerT
#m_sockets: std::vector<ClientSocket::Pointer>
#m_bNetworkThreadTx: bool
#m_Server: ServerSocket::Pointer
#m_timeout: unsigned long
#m_Thread: MultiThreader::Pointer
#m_iNetworkThreadTx: int
#m_MsgQueue: CMusiicQueue<P>
-CMusiicServerT()
<<destroy>>-CMusiicServerT()
+CreateServer(port: int): int
+CloseServer(): int
+SetTimeout(msec: unsigned long): void
+WaitForConnection(): ClientSocket::Pointer
+SendMessageData(): void
+PutMessageData(msg: P): void
+GetCountMessageData(): int
+NetworkThreadTx(ptr: void): void
+GetNumOfClient(): int
+IsConnected(): bool
#InitializeQueue(): void
<<OpenIGTLink>>
ServerSocket
Data : typename
<<OpenIGTLinkMUSiiC>>
MUSiiCQueue
<<OpenIGTLink>>
MultiThreader
b
Server:
port
Create Server Socket
Start Network thread, i=0
Network Thread
Is Client?
No
Yes
Accept &
Create Client Socket &
Add Sockets[i++] = socket
Yes
Queue
Empty?
No
Dequeue(Data), j=0
Sockets[j]::Send(Data), j++
No
j > i
Yes
Is Terminal?
No
Yes
Stop Network Thread
Server Stop

Figure 5 *MUSiiCServerT*: (a) class diagram, (b) flowchart

The *MUSiiCServerT* class is also based on the template class technique, so any kind of message type of OpenIGTLinkMUSiiC can be sent.

3 Applications for advanced ultrasound research using OpenIGTLinkMUSiiC

For our advanced ultrasound research, we built two applications using the OpenIGTLinkMUSiiC protocol. One is for ultrasound thermal monitoring in real time; another is related to the ultrasound photoacoustic imaging technique.

3.1 Ultrasound thermal monitoring system with real-time

Thermal monitoring imaging provides information about the temperature distribution in the tissue treated by ablative therapy. Due to the advantages of ultrasound imaging systems such as mobility and real-time data acquisition, ultrasound thermal monitoring systems can be utilized and applied intra-operatively.

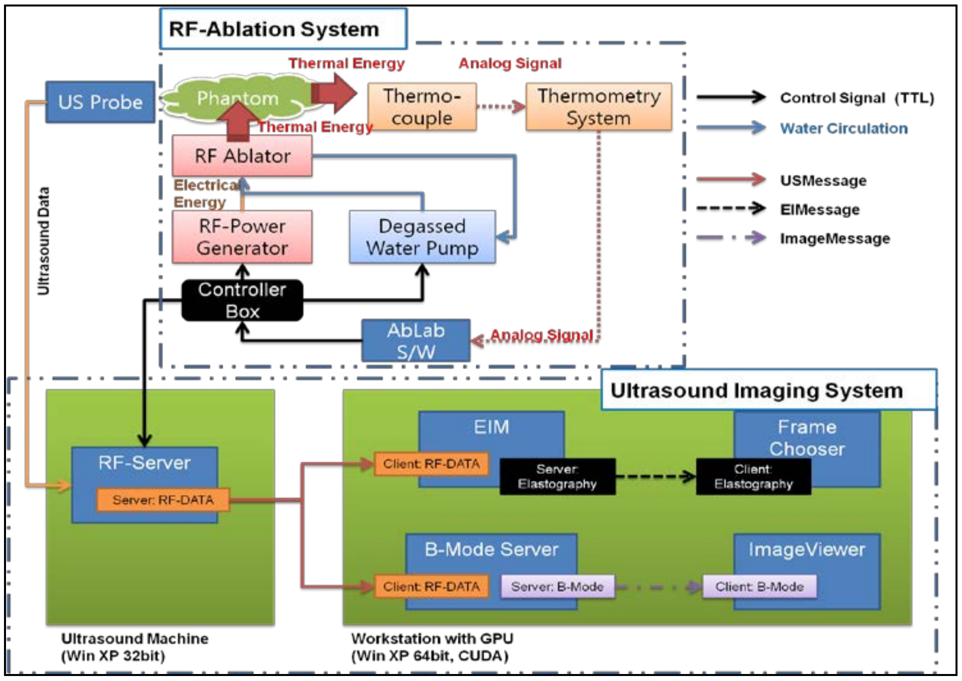

Figure 6 Block diagram of an ultrasound thermal monitoring system

Figure 6 shows the block diagram of our ultrasound thermal monitoring system. Our system can be divided into two groups: the Ultrasound Imaging and the RF-Ablation part. These two groups are synchronized by a Transistor–Transistor Logic (TTL) control signal of Controller box. As far as the purpose of this paper is concerned, the RF-Ablation part is outside the scope of discussion.

In the Ultrasound Imaging part, RF-Server [6] collects ultrasound radio-frequency (RF) data from the ultrasound machine (Sonix CEP, Ultrasonix Co.) and then provides a stream of *USMessage* that contains RF data to client modules, the Elasticity Imaging Module (EIM) and the B-Mode computation module (BMM) [6], via the *MUSiiCServerT* class. The EIM that receives the *USMessage*s computes an elasticity image from sequential *USMessage*s, and then sends the elasticity image to a FrameChooser module [6], encapsulated in an *EIMessage*. Simultaneously, the BMM module generates a B-Mode image from a *USMessage*, and transfers the image to the ImageViewer module using an *ImageMessage* of OpenIGTLink. Because the result of BMM module is an *ImageMessage*, the ImageViewer module can be exchanged with e.g. the 3DSlicer [11] program.

Regarding the performance of our *MUSiiCServerT* and *MUSiiCClientT* classes implementations, the following behavior could be observed between the RF-Server and BMM modules. Real-time ultrasound RF data was acquired from the RF-Server module on the Sonix CEP (Windows XP 32bit, Intel Core 2 Quad) and transmitted via TCP/IP network into the BMM module on the workstation (Windows XP 64bit, Intel i7) using *USMessages* in real time, resulting in a frame rate of around 22 fps. For our test, each *USMessage* contained 1024x256x16bit data for raw RF data, *US_Tag*, and its transformation information.

3.2 Ultrasound photoacoustic imaging system with real-time

Research in ultrasound photoacoustic imaging techniques for prostate brachytherapy has been growing [12, 13]. This imaging technique is based on the principle of the photoacoustic effect, the physical phenomenon of conversion of light waves to sound waves. Figure 7 represents the block diagram of our application for such an imaging system. As seen in the figure, there are two parts in our application: the Ultrasound Imaging and the Laser System component.

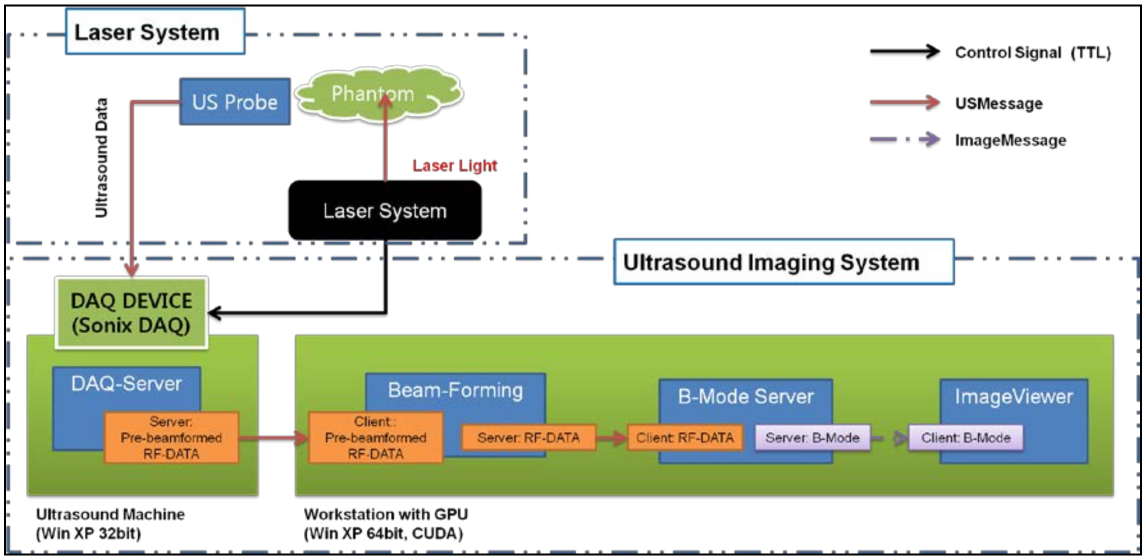

Figure 6 Block diagram of an ultrasound photoacoustic imaging system

The overall system configuration is similar to our previous work [12], except that the ultrasound imaging part supports real-time photoacoustic imaging.

Pre-beamformed RF-data from the ultrasound transducer is necessary for photoacoustic imaging. The DAQ-Server acquires pre-beamformed RF-data form Sonix-DAQ (Ultrasonix Co.) and then sends the data to the Beam-Forming module using *USMessage*. The Beam-Forming module converts pre-beamformed to beamformed RF-data, and then transfers its result to the BMM. A photoacoustic image is generated in the BMM module and displayed by the ImageViewer module. As the output of the BMM module is of type *ImageMessage*, 3D-Slicer can be used instead of the more specific ImageViewer.

4 Conclusions

In this paper, we presented our extension version of OpenIGTLink, OpenIGTLinkMUSiiC, for advanced ultrasound research. Our *USMessage* and *EIMessage* data types are useful to communicate ultrasound data or a result of advanced ultrasound imaging such like elasticity images over TCP/IP networks. Moreover, these two message types backwards-compatible with the *ImageMessage* from OpenIGTLink. Our other new message types – *GenMessage*, *ArgMessage*, *FileMessage* – enhance the flexibility of OpenIGTLink by allowing sending and receiving user-defined memory contents. To improve the performance of network classes of OpenIGTLink, we developed two network classes – *MUSiiCServerT* and *MUSiiCClientT* – by implementing an independent network thread and associated network buffers. In particular, the *MUSiiCServerT* class is designed for multiple-client connections.

Using OpenIGTLinkMUSiiC, we built two applications for our advanced ultrasound research. For real-time ultrasound thermal monitoring, ultrasound RF-data, B-Mode images, and elasticity images are communicated by *USMessage*, *ImageMessage*, and *EIMessage*, respectively. In another application, using an ultrasound photoacoustic imaging system, pre-beamformed RF-data and RF-data are sent and received through *USMessage* data.

Our OpenIGTLinkMUSiiC communications protocol is compatible with the existing OpenIGTLink. We are looking forward to integrate the OpenIGTLinkMUSiiC with a wider variety of Image Guided Therapy systems. In the near future, we will make an open-source implementation of the protocol and the socket classes publicly available.

Acknowledgment

The authors would like to thank Nathanael Kuo and Nishikant Deshmukh for their valuable input and support. Equipment support was generously provided by Ultrasonix.

Reference

[1] N. Deshmukh, H. Rivaz, and E. Boctor, "GPU-Based Elasticity Imaging Algorithms." pp. 45-54.

[2] N. P. Deshmukh, H. Rivaz, P. J. Stolka *et al.*, "Real-time GPU-based Analytic Minimization/ Dynamic Programming Elastography."

[3] P. J. Stolka, M. Keil, G. Sakas *et al.*, "A 3D-elastography-guided system for laparoscopic partial nephrectomies." pp. 76251I-1.

[4] J. L. S. Su, B. Wang, and S. Y. Emelianov, "Photoacoustic imaging of coronary artery stents," *Optics Expr ess*, vol. 17, no. 22, pp. 19894-19901, 2009.

[5] E. M. Boctor, N. Deshmukh, M. S. Ayad *et al.*, "Three-dimensional heat-induced echo-strain imaging for monitoring high-intensity acoustic ablation." p. 24.

[6] P. J. Stolka, H.-J. Kang, and M. B. Emad, "The MUSiiC toolkit: Modular Real-Time Toolkit for Advanced Ultrasound Research," *MIDAS Journal*, 2010.

[7] J. Boisvert, D. Gobbi, S. Vikal *et al.*, "An Open-Source Solution for Interactive Acquisition, Processing and Transfer of Interventional Ultrasound Images."

[8] K. Chinzei, and J. Tokuda, "Extension to OpenIGTLink; Smart Socket Connection, XML as Message, Logging, and One-to-multi Relaying," *The MIDAS Journal*, no. Systems and Architectures for Computer Assisted Interventions, Aug 15, 2009, 2009.

[9] J. Tokuda, G. S. Fischer, X. Papademetris *et al.*, "OpenIGTLink: an open network protocol for image guided therapy environment," *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 5, no. 4, pp. 423-434, 2009.

[10] "OpenIGTLink," http://www.na-mic.org/Wiki/index.php/OpenIGTLink.

[11] "3D Slicer," http://www.slicer.org/.

[12] N. Kuo, H. J. Kang, T. DeJournett *et al.*, "Photoacoustic imaging of prostate brachytherapy seeds in ex vivo prostate (Proceedings Paper)," 2011.

[13] M. Xu, and L. V. Wang, "Photoacoustic imaging in biomedicine," *Review of scientific instruments*, vol. 77, pp. 041101, 2006.