Linköping University Post Print

Making the PACS workstation a browser of image processing software: a feasibility study using inter-process communication techniques

Chunliang Wang, Felix Ritter and Orjan Smedby

N.B.: When citing this work, cite the original article.

The original publication is available at www.springerlink.com:

Chunliang Wang, Felix Ritter and Orjan Smedby, Making the PACS workstation a browser of image processing software: a feasibility study using inter-process communication techniques, 2010, International journal of computer assisted radiology and surgery, (5), 4, 411-419.

<http://dx.doi.org/10.1007/s11548-010-0417-8>

Copyright: Springer Science Business Media

<http://www.springerlink.com/>

Postprint available at: Linköping University Electronic Press

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-58771>

Making the PACS Workstation a Browser of Image Processing Software: A feasibility study using inter-process communication techniques

Chunliang Wang, Felix Ritter and Örjan Smedby

Chunliang Wang, MD, Dept of Radiology (IMH), Linköping University, Linköping, Sweden; Center for Medical Image Science and Visualization (CMIV), Linköping University, Linköping, Sweden

Felix Ritter, Dr, Fraunhofer MEVIS - Institute for Medical Image Computing, Bremen, Germany

Örjan Smedby, MD, Dr. Med. Sci., Dept of Radiology (IMH), Linköping University, Linköping, Sweden; Center for Medical Image Science and Visualization (CMIV), Linköping University, Linköping, Sweden

Communicating Author: Chunliang Wang, chunliang.wang@liu.se,
Address: Linköping University Hospital, CMIV, Linköping, SE-58185, Sweden
TEL: +46-13 22 8998, Fax: +46-13 22 7020

Purpose: To enhance the functional expandability of a PACS workstation and to facilitate the integration of third-part image-processing modules, we propose a browser-server style method.

Methods: In the proposed solution, the PACS workstation shows the front-end user interface defined in an XML file while the image processing software is running in the background as a server. Inter-process communication (IPC) techniques allow an efficient exchange of image data, parameters, and user input between the PACS workstation and stand-alone image-processing software. Using a predefined communication protocol, the PACS workstation developer or image processing software developer does not need detailed information about the other system, but will still be able to achieve seamless integration between the two systems and the IPC procedure is totally transparent to the final user

Results: A browser-server style solution was built between OsiriX (PACS workstation software) and MeVisLab (Image-Processing Software). Ten example image-processing modules were easily added to OsiriX by converting existing MeVisLab image processing networks. Image data transfer using shared memory added less than 10ms of processing time while the other IPC methods cost 1-5s in our experiments.

Conclusion: The browser-server style communication based on IPC techniques is an appealing method that allows PACS workstation developers and image processing software developers to cooperate while focusing on different interests.

Key words: PACS workstation; Image Processing; Functional Expandability; client-server communication; Inter-process communication.

Making the PACS Workstation a Browser of Image Processing Software: A feasibility study using inter-process communication techniques

1. Introduction

With the development of imaging techniques and the continually increasing size of multidimensional image data, picture archiving and communication systems (PACS) have become more and more indispensable for clinical work, as they provide a fast and systematic way of storing, searching, and retrieving medical images [1]. In developed countries, most radiological images are currently viewed and diagnosed on a PACS workstation. After passing the initial period when all efforts were put into standardization and early development of basic functions for simple storage and transfer, new challenges have emerged from the growing requirements of advanced image processing functions [1], since 2D images with one orientation can no longer provide sufficient information for diagnosis and evaluation. As diverse clinical requirements become apparent with new clinical discoveries and rapidly developing post-processing techniques, a single PACS supplier can no longer provide all desired functions in the self-centered developing manner. Hospitals are forced to buy different PACS workstations from different vendors and clinicians have to move between different workstations for different analyses. As more and more image analysis is incorporated into clinical routine, the diagnostic workflow is disturbed by the separate location of each image processing module. This fact appeals for an expandable PACS workstation system supporting the integration of third party function modules to provide the flexibility required in today's clinical routine.

On the other hand, the clinical needs of image processing tools have incited a growing number of researchers and companies to develop different types of image processing software [2][3][4]. Despite the sophisticated design and intensive work on focused functions, many of them have failed to integrate their software into PACS, possibly due to complexity of network interfaces and limited resources. Many of these software solutions require users to export images from a PACS workstation and import them manually for each case [2][3][4]. Even if the software provides basic PACS functionality, it requires the user to send images between two software applications and adapt to a new user interface. It would be beneficial for clinical users and researchers if newly developed image processing software could take advantage of existing PACS workstation software and merge into the latter without extensive modifications.

To support clinical adoption and integration of new image processing and visualization tools, we propose a browser-server style method based on inter-process communication (IPC) techniques, in which the PACS workstation shows the front-end user interface defined in an XML file while the image processing software is running in the background as a server. IPC techniques allow efficient exchange of image data, parameters, and user input between the PACS workstation and stand-alone image-processing software. Using a predefined communication protocol, the PACS workstation developer or image processing software developer does not need detailed information about the other system, but will still be able to achieve seamless integration between the two systems, and the IPC procedure is totally transparent to the final user. In this feasibility study, the proposed method is implemented between the open source PACS workstation software OsiriX [5] and the medical image-processing software platform MeVisLab [2].

2.Methods

In this study, we implemented IPC techniques to connect the research PACS workstation software OsiriX with the image processing software platform MeVisLab. OsiriX is an open-source software for medical image viewing [5]. As a typical PACS workstation, it provides a user-friendly graphical user interface (GUI) for image data organization, and has great compatibility for connecting with different types of scanners and PACS systems. OsiriX also provides some simple image processing methods and visualization tools for clinical usage; however, it is not adequate for various research projects. MeVisLab, on the other hand, provides a graphic-based programming and testing environment [6]. This prototyping tool includes thousands of modules allowing a developer to design sophisticated image processing networks without writing a single line of code. A shortcoming of MeVisLab is the lack of a user-friendly data organization system; it requires users to import DICOM images for each exam and convert them into special format files on the hard disk. With an increasing number of studies, it becomes more and more difficult to locate a desired patient dataset.

The motivation for building this “transparent” bridge between OsiriX and MeVisLab is to take advantage of both software systems by using OsiriX for data management and simple image navigation and employing MeVisLab to perform various image processing task. In practice, OsiriX provides a front-end GUI for clinicians, while MeVisLab runs in background invisible to them. A developer can design and test his image processing network inside MeVisLab without knowing how to program OsiriX. The communicating procedure is similar to browsing an interactive webpage, where OsiriX is the “browser” and MeVisLab is the server. But instead of using TCP/IP to transfer the data, more efficient IPC methods such as shared memory are implemented to speed-up the data exchange for single platform cases.

2.1 The Data Flow

The data flow of a typical operation starts in OsiriX, where the image data is loaded and an XML-defined GUI is presented (Figure 1). Both images and

user interactions are then transferred from OsiriX to MeVisLab. After processing, the results – which can be numbers, plots or images – will be transferred back to the GUI, and eventually stored into the OsiriX database in DICOM format. The data-transfer procedure is transparent to the end user who will not notice the involvement of a second application.

Figure 1. Data flow of a typical operation. 1, OsiriX loads all input image and shows them in the GUI defined by an XML file; 2, All image data and user input is collected by OsiriX and sent to MeVisLab; 3 and 4, results are sent back to OsiriX and eventually saved into the database in DICOM format. In 2 and 3, inter-process bridges are used which consist of E: exporter module and I: importer module

2.2 Communication Protocol Between OsiriX and MeVisLab

As shown in Figure 1, an inter-process bridge consists of two parts, the exporter module and the importer module. On the bridge, standardized data structures representing image objects, parameters and user interactions are used to pass the image data between the two modules (Figure 3). These two modules act as format translators between the native data structures of the host applications and the standardized data structures.

The data flow on the bridge is *one-way*, i.e. the data can only go from exporter to importer (e.g. 2 in Figure 1). Also, the data flow is *on demand*, which means that the uppermost part will only request the input image when necessary, in accordance with the MeVisLab image-processing network. However, the lower parts can send a notification when the input image or some parameters are changed or new keyboard/mouse input occurs.

A typical communication between OsiriX and MeVisLab can be described as follows (Figure 2): First, OsiriX sends a notification to the importer module in MeVisLab to inform the latter of a new input image or/and user operation. This notification will pass through the image-processing network and the effected modules will change their status to “need updating”; however, no

computing will take place in this step. As the notification reaches the top of the processing network – the exporter module – it will be passed back to OsiriX. Then OsiriX sends an operation request to the Exporter module. This will in turn cause the activation of the image processing network and the request is passed down through the image processing network to the importer module, which then sends a request back to OsiriX to get the input image (Figure 2). Finally, the image processing is carried out from bottom to top in the network and the final result will be returned to OsiriX.

Figure 2. Communication Protocol. Left: After Input images are loaded or user input events triggered, notifications are sent to MeVisLab and passed on in the processing network and eventually sent back to OsiriX via the exporter module; Right: After being notified, OsiriX sends a request to MeVisLab asking for result images. The request triggers the processing network to perform design operations on the updated input data.

Allowed user interactions are predefined in the XML file. All types of user interaction are classified as changing a parameter and marking an image in our application. The latter is further divided into point, curve, rectangle, oval and polygon (Figure 3).

Figure 3. Standardized data structure representing image objects, parameters and user interactions.

2.3 Implementing the techniques

Two key inter-process communication techniques are used in the software bridge. One is “Distributed Objects Programming” based on Object C 2.0 [7], the other is “POSIX Shared Memory” based on the Unix system (chapter 12 in [8]).

The Distributed Objects technique provides an easy way to send messages between different applications (even if they are on different computers). The procedure, similar to remote procedure calls, offers an abstraction from low-level network communication where one application opens a listening port, the other tries to connect to that port. After the connection is established, one application can invoke one/several pre-defined functions in the listening application (containing the distributed objects). Arguments and returned data of these function are automatically copied between applications by the operating system. This technique can be used to transfer text/numerical parameters and user responses, like mouse clicks, between two processes in real time. Since the data in the message pipe need to be copied between applications, it is not an efficient way for exchanging large amounts of data such as medical images between applications running on the same computer.

Figure 4. Memory Sharing: one piece of memory is mapped to two applications’ memory addresses. They can both read and write into this block of memory and changes will reach the other application immediately.

Shared memory is a technique that maps the same physical memory block to different applications’ memory addresses. Due to the limited physical memory, modern operating systems use [virtual memory](#) techniques to give an application program the impression that it has contiguous working memory, while in fact it is physically fragmented and may even overflow on to disk storage. As shown in Figure 4, when Application 1 loads all the images into the data block, it actually writes this information in one/several blocks of the global physical memory and maps it into the virtual memory of Application 1. Normally, application 2’s virtual memory is mapped to a

different part of the global physical memory; hence the two applications will not interfere with each other. However, the shared memory technique allows Application 2 to map parts of its virtual memory to the same block of physical memory where Application 1 stores the image data, i.e., both applications will read/write to the same part of physical memory. Changes to the image data by either of the applications will immediately reach the other application.

Using these two IPC techniques and the communication protocol described above, a plug-in for OsiriX (named MeVisLabHub, including both exporter and importer module) and two MeVisLab modules (named OsiriXImporter and OsiriXExporter) were built, and distributed as open source packages at <http://mevislabmodules.svn.sourceforge.net>. Using these instruments, MeVisLab users can easily convert their image processing network into an OsiriX functional module by changing the image loading and exporting module to OsiriXImporter and OsiriXExporter and defining a user interface in the MeVisHub. Figure 5 gives an example network that performs level-set segmentation on the input images. The configuration of the user interface template is shown in Figure 6. Figure 7 presents the final user interface and segmentation result. For more details on the image-processing network design readers are encouraged to refer to the MeVisLab user manual [6]. A more detailed description of the steps required to add such an extension function module into MeVisLabHub can be found in the user's manual for MeVisLabHub [9].

Figure 5. Network for Level-set image segmentation in MeVisLab


```

<object type="OPERATIONS" id="z110">
  <attribute name="operationname" type="string">Levelset</attribute>
  <attribute name="instruction" type="string">Instruction For Users</attribute>
  <attribute name="ifsupportseedsplanting" type="bool">1</attribute>
  <attribute name="ifsupportmemorysharing" type="bool">0</attribute>
  <attribute name="ifshowmeviswindow" type="bool">0</attribute>
  <attribute name="ifoperateslicebyslice" type="bool">0</attribute>
  <attribute name="commandlinestr" type="string">/Applications/MeVisLab.app/Contents/MacOS/MeVisLab -quick -nowelcome
Applications/MeVisLab.app/Contents/MacOS/MeVisLab -quick -nowelcome -singleinstance </attribute>
  <relationship name="objectslist" type="0/0" destination="SEGMENTEDOBJECTS" idrefs="z111"></relationship>
  <relationship name="parameters" type="0/0" destination="PARAMETERS" idrefs="z113 z112 z114"></relationship>
</object>
<object type="SEGMENTEDOBJECTS" id="z111">
  <attribute name="updateinrealtime" type="bool">1</attribute>
  <attribute name="toolstomark" type="string">Point</attribute>
  <attribute name="objectname" type="string">AortaSeed</attribute>
  <attribute name="objectcolor" type="string">Blue</attribute>
  <relationship name="operation" type="1/1" destination="OPERATIONS" idrefs="z110"></relationship>
</object>
<object type="PARAMETERS" id="z112">
  <attribute name="updateinrealtime" type="bool">0</attribute>
  <attribute name="type" type="string">float</attribute>
  <attribute name="paraname" type="string">CurvatureScaling</attribute>
  <attribute name="minvalue" type="double">0</attribute>
  <attribute name="maxvalue" type="double">1</attribute>
  <attribute name="defaultvalue" type="double">0.1</attribute>
  <relationship name="operation" type="1/1" destination="OPERATIONS" idrefs="z110"></relationship>
</object>
<object type="PARAMETERS" id="z113">
  <attribute name="updateinrealtime" type="bool">0</attribute>
  <attribute name="type" type="string">float</attribute>
  <attribute name="paraname" type="string">GradientThreshold</attribute>
  <attribute name="minvalue" type="double">0</attribute>
  <attribute name="maxvalue" type="double">100</attribute>
  <attribute name="defaultvalue" type="double">20</attribute>
  <relationship name="operation" type="1/1" destination="OPERATIONS" idrefs="z110"></relationship>
</object>

```

Figure 6. Part of the XML file that defines the GUI. In this part the parameter and the expected user inputs are defined.

Figure 7. The user interface of the level-set module with the segmentation result.

3. Result

Ten example image processing modules, which can be classified into four main categories (image filtering, image segmentation, quantitative

measurement and 3D visualization, as shown in Table 1), were easily added to OsiriX by converting existing MeVisLab image processing networks and provided at <http://mevislabmodules.svn.sourceforge.net>. The time needed for converting such a MeVisLab network into an OsiriX extension module ranged from 5 minutes to 15 minutes.

Although the image processing time varies considerably depending on the size of the image and the chosen algorithm, the image data transfer time is usually less than 10ms using the shared memory technique. There was no noticeable slow-down in processing speed for any of the function modules compared to running the same functions in MeVisLab using the native image loading module. In Table 2, the time spent on data exchange and total operations are compared among different IPC techniques. While the first two columns represent using shared memory on a single PC, the remaining four are using inter-process messaging to copy the image data among applications on a single PC or different PCs. All 4 algorithms are provided by MeVisLab. Anisotropic filter is based on itkCurvatureAnisotropicDiffusionFilter and is implemented as multi-thread algorithm while the other 3 methods are single thread algorithm. The size of the testing data is $355 \times 512 \times 512$ voxel for thresholding and region growing methods, and $200 \times 200 \times 200$ voxel for anisotropic filtering and level-set based segmentation. The single PC tasks are finished on a MacBook with 2.4GHz Intel Core 2 Duo CPU and 4GB (DDR2 667MHz) memory. For remote processing, a Mac Pro with 2 Quad-Core 2.26GHz Intel Xeon CPU and 8GB (DDR3 1066MHz) memory is used as a server.

Table 1. Examples of MeVisLab modules added to OsiriX

Image Filtering	Image Segmentation	Quantitative Measurement	3D Visualization
Gaussian Filter	Fuzzy c-means method	Region Histogram	Iso-Surface Rendering
Anisotropic Filter	Region Growing	Dynamic Enhancement Curve	Volume Rendering
Morphological Operations	Level-set method		

Table 2. Comparison of time efficiency of using different IPC techniques

Image Processing	Shared Memory		Copy Data on Single PC		Remote Processing	
	Transfer Data	Total Task	Transfer Data	Total Task	Transfer Data	Total Task
Thresholding	≈5ms	2s	3s	5s	5s	6s
Region growing	≈5ms	4s	3s	7s	5s	7s
Anisotropic filtering	≈5ms	61s	0.5s	62s	1s	15s
Level-set segmentation	≈5ms	9min	0.5s	9min	1s	5min

4. Discussion

The development of imaging techniques and image processing methods provides clinical physicians with powerful tools that may hopefully lead to reliable diagnosis, quantitative evaluation and intuitive visualization[10][11][12]. In reality, each function is often associated with a particular “advanced” workstation, as one software producer may be strong at one function and weaker at some others [13][14]. This may lead to a waste of resources, since it involves not only great monetary costs for different workstations, but also considerable time for re-training staff and for transferring medical image data between workstations. Although many PACS vendors have realized the importance of integrating image processing functions into PACS, most of them are trying to solve this problem on their own by exclusively updating their own workstations to provide as many image processing functions as possible. With new image analysis algorithms and techniques being introduced incessantly, physicians can never rely on their PACS vendor to provide them with all state-of-art analysis functions. This is especially true when it comes to research.

In this paper, we have proposed to use the IPC techniques and an open communication protocol to enhance the expandability and flexibility of a PACS workstation. The communication model of proposed method resembles the browser and web-server model. The PACS workstation acts as a browser which shows the GUI defined in an XML file, sends input images, parameter and user input to the server – image processing software, and receives the results. On the PACS workstation side, as input and output images in the GUI are shown in their native format using native viewer class/module, users will already know how to interact with them as the operation is the same as anywhere else, and the developer of the PACS workstation should be confident with the robustness of the system as no third part module is loaded. On the other hand, the developer of the image processing software need not have any knowledge of the structure of the PACS workstation as long as an XML file is given that specifies what types of images, parameters and user inputs are expected.

To make the above model work and let the browser and server understand each other, data structure translation has to be performed at both ends of the bridge. In this feasibility study, three image object parameters and users inputs were standardized. Compared to the well-known DICOM format, the proposed communication protocol simplifies the definition of an image object to be transferred. Only essential information, such as size and dimension is kept on the “bridge” (Figure 3). This reduces the complexity of developing the translation module in the image processing software. As the PACS workstation keeps track of which patient and which exam is being processed, the meta-information will automatically be added to the processed results before they are saved into the PACS.

Another advantage of the proposed solution is the ability to include user interactivity in the data exchange. Although the users’ input varies much between different types of tasks, most of them can be classified into two categories: changing a parameter and drawing a shape on the image. The

latter can be further abstracted to several kinds of predefined shapes, such as a dot, a curve, a rectangle etc. By converting these operations into a standard description and sending it through the bridge, the image processing module may understand most user interaction. Thanks to these abstract definitions, the developer of image processing functions (in this case the developer of MeVisLab network) does not need any information on how the host system handles this interaction to be able to integrate their function into the host's user interface. In more complicated cases, the developer can choose to show the native user interface of the image processing software (cf. section 2.2).

Another noticeable advantage is time and memory use efficiency achieved by using the shared memory technique. As the PACS workstation software loads the images from the hard disk, the image processing module need not allocate new physical memory and load the images again. So even if there are several GByte of data to be transported, it will take less than 10ms. More importantly, although the memory usage increases in both applications, physically only one copy is loaded in the real memory, which will reduce the risk of out-of-memory error and avoid swap data between RAM and disk.

A drawback of using shared memory is that it restricts the two software applications to be running on the same workstation. In our study, this restriction was overcome by using socket-based IPC techniques for remote image processing. As shown in Table 2, remote processing can benefit tasks that require intensive computing power. However the performance is lagged down by the transferring data over network if many user interactions are required during the task. As most of the PACS workstations are fast enough to handle additional computations from extension modules, running both programs on a single platform should be preferred in most cases.

Although, in this feasibility study, for convenience reason, Mac OS specific IPC class libraries were used, which limited the solution to Mac platform, it is, in theory, possible to use the same method to connect any pair of PACS workstation and image processing software. As both inter-process message passing and shared memory are also supported by other operating systems, such as Microsoft Windows, this method can be generalized to other platforms as well. In this context, the need for international standards (like DICOM) should be emphasized.

It should be noticed there are already some client-server based PACS workstation on market like syngo workspace of Siemens Medical Solution and Aquarius intuition from TeraRecon Inc., which provide web-based image processing. However their aim is to make their existing function available to more users rather than integrating more third-part functionality. Combining the proposed solution with these client-server based PACS workstation, it can be expected that, in the future, a clinician will be able to perform any image analysis function available on the network from any client station they prefer.

5 Conclusion

With the maturity of PACS techniques and the rapid development of image processing techniques, an inevitable request has emerged that these two should integrate seamlessly. In this feasibility study, it has been proven that with proper IPC techniques and an open defined communication protocol, a PACS workstation software and third-party image processing modules can be seamlessly integrated, allowing a user to perform image processing without being aware of the involvement of several software applications. Furthermore, PACS developers and image processing researchers do not need extensive knowledge of each other's systems.

Acknowledgements.

We want to thank Ola Friman for helpful discussion and Osman Ratib, Antoine Rosset and Joris Heuberger, for providing the OsiriX software. Financial support was given by the Swedish Heart-Lung Foundation and the VINNOVA supported program Agoralink.

References:

1. Ratib, O. (2009) Imaging informatics: from image management to image navigation, *Yearb Med Inform* 167-172.
2. MeVisLab, *Medical image processing and visualization*, <http://www.mevislab.de/>.
3. 3D Slicer, <http://slicer.org/>.
4. Toussaint, N., Souplet, J., and Fillard, P. (2007) MedINRIA: Medical Image Navigation and Research Tool by INRIA. In *Proc. of MICCAI'07 Workshop on Interaction in medical image analysis and visualization*.
5. Rosset, A., Spadola, L., and Ratib, O. (2004) OsiriX: an open-source software for navigating in multidimensional DICOM images, *J Digit Imaging* 17, 205-216.
6. MeVis Medical Solution AG. (2009) Getting Started: First Steps with MeVisLab, <http://www.mevislab.de/developer/documentation/>.
7. Apple Inc. (2007) Distributed Objects Programming, <http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/DistrObjects/DistrObjects.html>.
8. Stevens, W. R. (1998) UNIX Network Programming, Volume 2: Interprocess Communications, 2nd ed., Prentice Hall PTR.
9. Wang, C., and Smedby, Ö. (2009) MeVisHub User Guide, <http://mevislabmodules.svn.sourceforge.net/viewvc/mevislabmodules/trunk/Community/General/Sources/ML/MLOsiriXImporter/>.

10. Wang, C., Frimmel, H., Persson, A., and Smedby, Ö. (2008) An interactive software module for visualizing coronary arteries in CT angiography, *International Journal of Computer Assisted Radiology and Surgery* 3, 11-18.
11. Fisichella, V. A., Jäderling, F., Horvath, S., Stotzer, P., Kilander, A., Båth, M., and Hellström, M. (2009) Computer-aided detection (CAD) as a second reader using perspective file view at CT colonography: effect on performance of inexperienced readers, *Clin Radiol* 64, 972-982.
12. Knöss, N., Hoffmann, B., Fabel, M., Wiese, C., Jochens, A., Bolte, H., Heller, M., and Biederer, J. (2009) Lung nodule assessment in computed tomography: precision of attenuation measurement based on computer-aided volumetry, *Rofo* 181, 1151-1156.
13. Sörstedt, E., Persson, A., Noren, B., Björnlert, U., Malcherek, P., Axelsson, M., Johansson, J., and Smedby, Ö. (2005) Computed tomographic colonography: Comparison of two workstations, *Acta Radiologica* 46, 671-678.
14. Dikkers, R., Willems, T. P., de Jonge, G. J., Marquering, H. A., Greuter, M. J. W., van Ooijen, P. M. A., van der Weide, M. C. J., and Oudkerk, M. (2009) Accuracy of noninvasive coronary stenosis quantification of different commercially available dedicated software packages, *J Comput Assist Tomogr* 33, 505-512.