Loading DICOM Images and Metadata into the XIP Platform

Alex Shnayder[a], Christina Sillery[b], Rachel Embree[c], Dr. Channin[d], Pat Mongkolwat[d]
[a]Lafayette College, Easton, PA, USA, 18042
[b]Siena College, Loudonville, New York, USA, 12211
[c]University of Michigan, Ann Arbor, MI, USA, 48109
[d]Northwestern University, Chicago, IL, USA 60611

1. Background

The NCI-funded program, caBIG In-Vivo Imaging Work Space, is currently working on a project to develop an extensible, open-source platform for imaging and visualization purposes. The eXtensible Imaging Platform (XIP) is an in-the-works, open source environment that allows simple and rapid medical imaging application development to be used by any number of compatible hosting environments such as medical workstations. It is based on DICOM working group 23. The platform will also aid in increasing the uniformity of medical imaging applications used in clinical settings. XIP contains a rapid-action development interface based on the OpenInventor toolkit that allows easy application development in a simple drag-and-drop environment. This interface, which will be released under the name "XIPBuilder," uses OpenInventor modules to provide developers with a hierarchal pipeline structure for an application being created. These modules take in an input(s) either manually from a user or from another compatible module, perform some operation based on the input, and create an output which can be sent to another module such as a "Text" module to display the output. XIPBuilder allows the creation of intricate pipelines and connections using any number of supported modules. Custom modules can be built and added to the extensible platform, allowing easy translation from a research or development site to a clinical setting.

In its final release version, XIP will contain ITK and VTK library compatibility, but in its current state, lacks generic DICOM file manipulation such as that found in the DCMTK ("DICOM ToolKit"). The DCMTK is a popular, open-source collection of libraries and applications which allow users to, among other things, create, convert, manipulate, read and pass DICOM files. The purpose of this project is to create a generic DICOM import module that will integrate DCMTK with XIP and allow users of the platform to access and pass header information and pixel data throughout an application. Currently, XIP does not contain any modules that provide direct access to and control of DICOM header information. This module will prove helpful in many medical imaging applications that require easy access to DICOM information.

2. Materials and Methods

The module we developed was based on previously existing code already developed in XIP. The original intent of the module was to aid in an Automatic Volumetric Breast Density Assessment application, which was designed to calculate the percentage of dense breast tissue in mammography images using various formulas that require specific DICOM header information. Therefore, only specific header information is currently accessible in our module, including tags such as Breast Thickness and Target Material.

However, the header information retrieval is handled in a generic C++ class, called DicomHandler, which does not require any XIP compatibility. As a result, the class can easily be changed to account for all possible tags that the DCMTK supports.

The first step in creating and adding our module to XIP was to create the DicomHandler class. This simply entailed installing DCMTK and writing simple code to retrieve header and pixel information for a given filename. Afterwards, we created the necessary storage/helper classes: NewImage to store all the information and DicomObject to store a NewImage object. Then, we created a small project class, which would be recognized by the XIPBuilder environment, and upon being recognized, would load all the module classes in the project. In our case, we had one module class for our one module.
This project class was also responsible for creating a .dll (dynamically linked library) file for the entire project. The DLL was then placed in the same directory as XIPBuilder's ".INI" file, responsible for loading module libraries such as the SoITK and standard XIP classes. We specified in the .INI file the name of our project file so that every time XIPBuilder would be opened, our DLL would be recognized and thus our classes would be loaded.

Our module, labeled SoXipLoadDICOM, requires only a single input: the name of the DICOM file being loaded. Using this filename, the necessary DCMTK methods to load pixel data and header information are called within the DicomHandler class, and all this information is stored in a "NewImage" object. This NewImage class is derived from a basic XIP image class called "SbXipImage" that other XIP modules look for when loading pixel data. The only difference between the NewImage class and the basic XIP class from which it is derived from is that the NewImage class also stores actual header information in addition to pixel data. A "DicomObject" class, derived from an XIP image data class called "SoXipDataImage", acts as a container class to simply store a NewImage object. Our DICOM import module will output one object, a predefined XIP single-field image data object called "SoXipSFDataImage", which will further act as a container for a DicomObject object. This complicated level of abstraction is necessary in order for modules of varying types to properly communicate with each other, such as when connecting XIP and ITK-based modules together. Therefore, if a module developer wanted to retrieve DICOM header information from our module, they would take in the single-field data object from our module as their input, retrieve the single field being stored in that object (which would be an instance of DicomObject), and would then have to retrieve the NewImage object being stored in that particular DicomObject. At that point, necessary accessor methods could be used to retrieve both the pixel data and particular header information which our module currently supports from the NewImage object.

3. Conclusion

The DICOM import module is a generic XIP module that can pass both DICOM header information and pixel data, both of which can be used by any other modules accessible in the XIP platform. It provides a simple example of how external libraries such as the DCMTK can be easily integrated into XIP and thusly can be used by modules derived from other libraries (ex. ITK, VTK, etc.). While its current state only allows access to specific header information, the DicomHandler class can easily add more accessor methods to account for the missing tags. DICOM tag verification is also a feature missing in the module, though it too can be added independent of XIP. Regardless, our module can be used for a variety of tasks. The module can aid in future application development, can be used as a stand-alone module to load pixel and header information from a DICOM file, or can be used as a model for future XIP class developers to base their own modules off of, particularly for tasks which deal with retrieving information from external sources and passing such information to other modules within the XIP environment.