



Published in final edited form as:

Annu ORNL Biomed Sci Eng Cent Conf. 2010 May 25; 2010: 1–4. doi:10.1109/BSEC.2010.5510850.

Interfaces and Integration of Medical Image Analysis Frameworks: Challenges and Opportunities

Kelsie Covington,

Vanderbilt University, Department of Electrical Engineering, Nashville, TN 37215 USA

Evan S. McCreedy,

National Institutes of Health, Center for Information Technology, Bethesda, MD 20892 U S A

Min Chen,

Johns Hopkins University, Department of Electrical and Computer Engineering, Baltimore, MD 21218. USA

Aaron Carass [Member, IEEE],

Johns Hopkins University, Department of Electrical and Computer Engineering, Baltimore, MD 21218. USA

Nicole Aucoin [Member, IEEE], and

Brigham and Women's Hospital, Department of Radiology, Surgical Planning Laboratory, Boston, MA 02115. USA

Bennett A. Landman [Member, IEEE]

Vanderbilt University, Department of Electrical Engineering, Nashville, TN 37215 USA

Kelsie Covington: covingkj@gmail.com; Evan S. McCreedy: mccreedy@mail.nih.gov; Min Chen: dontminchenit@gmail.com; Aaron Carass: aaron_carass@jhu.edu; Nicole Aucoin: nicole@bwh.harvard.edu

Abstract

Clinical research with medical imaging typically involves large-scale data analysis with interdependent software toolsets tied together in a processing workflow. Numerous, complementary platforms are available, but these are not readily compatible in terms of workflows or data formats. Both image scientists and clinical investigators could benefit from using the framework which is a most natural fit to the specific problem at hand, but pragmatic choices often dictate that a compromise platform is used for collaboration. Manual merging of platforms through carefully tuned scripts has been effective, but exceptionally time consuming and is not feasible for large-scale integration efforts. Hence, the benefits of innovation are constrained by platform dependence. Removing this constraint via integration of algorithms from one framework into another is the focus of this work. We propose and demonstrate a light-weight interface system to expose parameters across platforms and provide seamless integration. In this initial effort, we focus on four platforms Medical Image Analysis and Visualization (MIPAV), Java Image Science Toolkit (JIST), command line tools, and 3D Slicer. We explore three case studies: (1) providing a system for MIPAV to expose internal algorithms and utilize these algorithms within JIST, (2) exposing JIST modules through self-documenting command line interface for inclusion in scripting environments, and (3) detecting and using JIST modules in 3D Slicer. We review the challenges and opportunities for light-weight software integration both within development language (e.g., Java in MIPAV and JIST) and across languages (e.g., C/C++ in 3D Slicer and shell in command line tools).

I. Introduction

Numerous, complementary platforms are available for large-scale medical imaging data analysis. Typically, these platforms are not readily compatible in terms of workflows or data formats, so image scientists and clinical investigators are constrained to collaborate using a single platform. Often, this drives researchers to choose a platform that is less natural for the problem at hand for any investigator, and, instead, favors a compromise platform. Algorithms with a wide variety of applications are implemented in only one platform, and must be reinvented, reimplemented, or accessed via complicated scripting techniques to be taken advantage of in other frameworks. The complications stemming from platform dependence impede innovation and collaboration amongst researchers by impeding dissemination. We propose a compromise between the independent development of frameworks and a single uniform collaboration framework through automated framework integration. This approach enables development and code reuse on a variety of platforms — each of which may lend itself naturally to specific applications for specific developer-scientists. Therefore, algorithms may be easily explored in new contexts and used within other software tools.

Our initial effort focuses on four major platforms Medical Image Analysis and Visualization (MIPAV) [1], Java Image Science Toolkit (JIST) [2], command line tools, and 3D Slicer [3]. In this manuscript, we explore three case studies: (1) a system for MIPAV to fully expose internal algorithms, (2) a method for self-documenting command line programs which provide for inclusion of JIST algorithms in scripting environments, and (3) an eXtensible Markup Language (XML) approach which allows 3D Slicer to detect and utilize the JIST module SPECTRE.

Section II of this paper describes the software analysis platforms involved in this study. Sections III–V describe the case studies and our efforts results with platform integration. Section VI concludes with the benefits and challenges of collaboration through framework integration.

II. Overview of Platforms to be Considered

A. Medical Image Analysis and Visualization (MIPAV)

MIPAV is a comprehensive, extensible, and platform-independent JAVA based medical image processing and visualization application developed at the National Institutes of Health (NIH) [1]. MIPAV provides the foundation and tools needed to assist NIH intramural researchers with the processing of hypothesis-driven research data. Development includes the implementation of known solutions and novel algorithms to quickly and efficiently meet the needs of our collaborators. It is freely available via the MIPAV website (<http://mipav.cit.nih.gov>). MIPAV supports over 70 different industry standard image formats including: DICOM, Analyze, Minc 1.0 & 2.0, NRRD, NIFTI, and TIFF. Many of the image processing algorithms and utilities in MIPAV are accessible through a scripting system, which allows them to be applied to batches of image data.

B. Java Image Science Toolkit (JIST)

Java Image Science Toolkit (JIST) provides a native Java-based imaging processing environment similar to the Insight Segmentation and Registration Toolkit (ITK)/ Visualization Toolkit (VTK) paradigm, distributed under the GNU Lesser General Public License (LGPL) [2]. Initially developed as an extension to MIPAV for visual programming [4], JIST grew to encompass rapid prototyping concepts [5] by providing automated GUI generation for application plugins, graphical layout tools, and command line interfaces.

Image processing tools can be accessed through JIST modules by users of the Pipeline Layout Tool, including the MIPAV scripting system and some algorithms, the Cortical Reconstruction Using Implicit Surface Evolution (CRUISE) [6] tool and Coregistration, Adjustment, and Tensor-solving – a Nicely Automated Program (CATNAP) software [7].

C. 3D Slicer

3D Slicer (also known as Slicer)[3] is a free, open source software package for visualization and image analysis. 3D Slicer is natively designed to be available on multiple platforms, including Windows, Linux and Mac Os X. 3D Slicer makes extensive use of the ITK/VTK. Functionalities include support for image segmentation, registration, multimodal visualization, advanced image modeling (e.g., diffusion, functional MRI) and image-guided radiation therapy. 3D Slicer has been used in both retrospective clinical assessment as well as real-time inter-operative research procedures.

D. SPECTRE

Simple Paradigm for Extra-Cerebral Tissue Removal (SPECTRE)[8], is a brain extraction algorithm that combines elastic registration, tissue segmentation, and morphological techniques, all guided by a novel watershed principle. SPECTRE is specifically designed to retain cortical gray matter so that subsequent processing designed to find the cortex will not be forced into making errors due to skull stripping mistakes. The main evaluation of SPECTRE's performance was against 1200 manually skull stripped brains, with the results showing it is a strong contender to replace a human rater.

III. Case Study I: Exposing Existing Algorithms

MIPAV contains an extensive library of image processing tools both developed internally at the NIH as well as by extramural investigators which are accessed through a plug-in mechanism. Furthermore, MIPAV sports a robust record-playback system for automating series of actions. However, the initial implementation of the framework did not expose the input/output relationships of plugins in a human readable or machine readable manner. The only way for a program to probe a module was to run it (which is especially problematic because the required input data types were not known *a priori*). The first JIST software accessed these features by forcing the user to either record script or to write an adapter class to translate between the MIPAV algorithm and JIST. Both of these options require prior knowledge of the algorithm and the parameters required to execute it. Any change made to the algorithm parameters or the scripting system required alterations to the JIST macro script or adapter code. Together, these factors have limited the extent of MIPAV/JIST integration and required significant resources on an ongoing basis to maintain JIST's access to MIPAV's functionality.

To more fully integrate MIPAV into the JIST framework and reduce the effort required to keep their systems compatible, we propose making a series of limited (optional) additions to internal MIPAV algorithms so that they can self-describe their functionality and enable other program (such as JIST) to make automated use of scripting system already in place. To accomplish this, we define an API (known as interface `ActionDiscovery`, see Table 1) for algorithms and utilities to describe the input and output parameters they use.

Algorithms that implement this interface create two lists of parameters without assigning any values to them. The caller of these methods (JIST) then sets the parameter values within the input table and passes it to the `scriptRun` method. After executing the algorithm, the calling software may use the list of output parameters to retrieve the results.

By using the methods and classes provided by the MIPAV macro scripting system wherever possible, relatively minor changes were required for the “Gaussian Blur” and “Subsample” algorithms to implement the new `ActionDiscovery` interface. Each module now reports a list of the same input parameters that it requires before it is run as part of a script, the only difference is that the value for each parameter is left empty. We then created a test program which directly accesses each of the Gaussian Blur and Subsample algorithms, gets its parameter lists, automatically interrogates the user for the proper input parameter values, runs the algorithm, and shows the resulting blurred image, all without prior knowledge of the specific parameters used by the algorithm.

IV. Case Study II: Self-Documenting Modules

JIST provides an infrastructure for rapid prototyping, validation, and publication of algorithms with self-generating graphical user interfaces within a visual programming environment (Fig. 1). While this is an important capability for end-to-end processing, it is difficult to concatenate graphically oriented programs in an automated manner. Therefore, we developed a command line interface (CLI) encapsulation system so that all methods implemented in the JIST framework would be accessible in a consistent manner from a shell command line.

Methods implemented in the JIST infrastructure make use of automated GUI generation. Methods define their input and output objects in terms of native types (e.g., integer, floating point numbers, arrays, images, strings, files) or collections of native types. These definitions include both programmatically defined types and human readable descriptions. To generate the CLI interface, we created a procedure to report all modules detected in JIST framework. Once the user (or a program) has the name of the method, the executable will self-report the input/output relationships for a POSIX compliant command line call in a human and machine readable format in response to a “`-help`” call (Fig. 2). The user may specify both program and system arguments on the command line and the POSIX parsers will marshal these objects to pass to the algorithm module. Upon completion, results are written to a summary file (for simple data types and file pointers) and to local files (for complex or binary data objects).

V. Case Study III: XML-Based Integration

The JIST CLI provides for human readable and script accessible manipulation of JIST modules. However, programmatic integration requires both the ability to access (e.g., call) and interpret the input/output relationship in a manner that can be sensibly presented in human-readable form. 3D Slicer includes an Execution Model which supports exchange of such information in an XML format (http://www.3DSlicer.org/3DSlicerWiki/index.php/3DSlicer3:Execution_Model_Documentation). Programs that respond to a “`-xml`” query provide 3D Slicer with the relevant input and output information, (i.e. file types, file locations, parameter values, etc.) for 3D Slicer to build a GUI widget and run the algorithm. In this case study, we integrate the advanced brain isolation techniques in SPECTRE (which is implemented for JIST/MIPAV) in the 3D Slicer framework.

In the XML approach, whenever the user selects a program (e.g., SPECTRE), 3D Slicer queries the xml descriptor from the CLI, and creates a SPECTRE module in 3D Slicer. The user then sets the parameters and runs SPECTRE through the 3D Slicer GUI, which then passes the parameters set by the user to the CLI. In turn, the CLI spawns a JIST process. When the execution completes, the results from JIST are collected and loaded back into the 3D Slicer framework.

The major challenges faced during this integration arose from how the data and input/output are represented differently between the 3D Slicer and JIST frameworks. This included differences in format types for the images, different representation of parameter inputs, and incompatible handlers for error reporting and program exiting. In addition, since JIST operates in Java while 3D Slicer uses native compilation, installation complexity issues needed to be resolved. The source code modifications were included in open source for both JIST and 3D Slicer distributions. These changes enabled the systems to generate visually consistent GUIs for both platforms (Fig. 3).

The final SPECTRE result produced through the 3D Slicer integration is identical to that produced through JIST (Fig. 4). However, as part of the 3D Slicer framework, it can be more easily used with a variety of algorithms available in 3D Slicer (e.g., registration, segmentation, and surface modeling) without having to manually move and set files between the two frameworks. Overall, this allows easier pipelining, a less complex learning curve, and higher availability of algorithms for the end user, especially for users of the 3D Slicer system.

VI. Conclusions

The test cases highlight that integration from one framework into another is feasible and possibly straightforward. The most severe difficulties arise when the underlying platform does not capture the input/output relationships in a systematic manner. In the first case study, we demonstrate how a light weight interface can be added to existing modules to provide for automated detection and exploitation of functionality. Implementation of these interfaces during initial development is nearly trivial and is consistent with layout of a custom GUI. However, after the fact implementation is a bit more challenging as new programmer must infer the intent of each parameter from heterogeneously documented code. Efforts ongoing to implement this interface for all 192+ modular algorithms in MIPAV. Once this effort is complete, this method's functionality will be available for use in a variety of different manners.

The second case study illustrated how functionality can be exposed in a systematic way once the input/output relationships are captured. The JIST CLI framework supports a variety of data types, including imaging format, native structures, arbitrary file objects, and collections of simple types. This provides the flexibility to represent nearly any data type and allows algorithm developers to make optional use of the JIST/MIPAV data structures.

Finally, the third case study highlighted the need for agreement on standardized basic data formats so that differing packages can readily exchange data. Formats with optional and vendor specific tags are useful for within a package, however, these enhanced features complicate exchange of data. It is often easier to pass this "intent" information as side channel. Nevertheless, the requirements can be captured in a structured format, such as XML, which is readily accessible for automated parsing as well as human inspection/editing.

To date, these integration efforts have largely focused on packages present on the same physical machine or cluster. Yet, the dissemination of structured input/output relationships can readily be extended to remote network connection. Other fields are already making extensive use of standardized communication syntax, such as the Web Service Definition Language (WSDL). We note that 3D Slicer supports remote input/output, allowing users to specify URIs (Uniform Resource Identifiers) in XML files that point to remote hosts and allows 3D Slicer to load data from a combination of local disk and remote resources.

If the discovery and marshaling functionalities captured in these case studies is embedded in a WSDL context, neuroimaging applications could more readily make use of distributed computation. Authentication, access, and control of these modular jobs could be accomplished with tools that are non-specific to neuroimaging. As such, continued advancement in the encapsulation will allow more sophisticated and large-scale application of software for new and exciting neuroimaging analysis challenges.

Acknowledgments

This research was supported in part by the Intramural Research Program of the NIH, CIT and under extramural grants 1R21NS064534-01A109 and R21-EB009900. Additional support was provided by startup research funds from Vanderbilt University.

References

1. McAuliffe, M., et al. Medical image processing, analysis and visualization in clinical research. Computer-Based Medical Systems; 14th IEEE Symposium; 2001; 2001. p. 381-386.
2. Lucas BC, et al. The Java Image Science Toolkit (JIST) for Rapid Prototyping and Publishing of Neuroimaging Software. Neuroinformatics. 2010 In press.
3. Pieper, S., et al. The NA-MIC Kit: ITK, VTK, pipelines, grids and 3D slicer as an open platform for the medical image computing community. 3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro; 2006. p. 698-701.
4. Lucas, B., et al. MAPS: A Free Medical Image Processing Pipeline. presented at the Organization for Human Brain Mapping; Melbourne, Australia. 2008.
5. Landman, BA., et al. Organization for Human Brain Mapping. San Francisco, CA: 2009. A Rapid Prototyping Environment for NeuroImaging in Java.
6. Han X, et al. CRUISE: cortical reconstruction using implicit surface evolution. Neuroimage Nov; 2004 23:997-1012. [PubMed: 15528100]
7. Landman, BA. CATNAP and JIST: DTI Processing Made Ridiculously Simple. International Society for Magnetic Resonance in Medicine Educational Sessions; Honolulu, HI. 2009.
8. Carass, A., et al. A Joint Registration and Segmentation Approach to Skull Stripping. presented at the 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro; 2007.

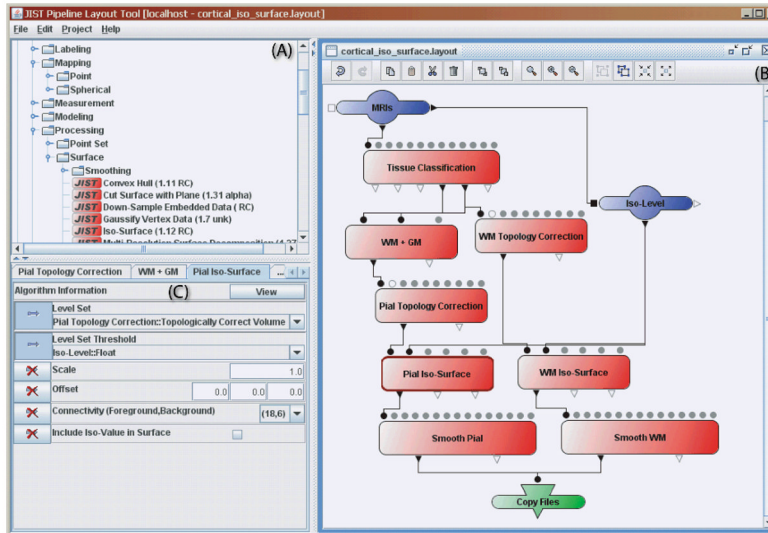


Fig. 1. JIST graphical layout tool. JIST provides a library of tools (A), a visual scripting area (B), and an auto-generated GUI (C) for direct parameter specification (as opposed to connections from other modules).

```

cli usage: edu.jhu.eco.iact.plugins.utilities.volume.MedicAlgorithmCrop [-h]
[-inBorderSize <arg>] [-isThreshold <arg>] [-isVolume <arg>]
[-outCropped <arg>] [-outExecution <arg>] [-outNew <arg>]
[-outOffset <arg>] [-outOriginal <arg>] [-outBboxOf <arg>]
[-setDefault <arg>] [-cpu <arg>] [-file <arg>] [-debug <arg>]
[-saveFromas <arg>] [-xml] [-saveFast <arg>] [-saveRefChanges]
[-saveIDing <arg>]

Crop 1.6 RC
Crops an image. The resulting output "Cropped Image" contains all voxels
whose intensities are greater than the input "Threshold". Also outputs the
dimensions of the original and cropped volumes. The dimensions of the
original images should be used with the "Crop" module
-h, -help Print this message
-inBorderSize <arg> BorderSize [Integer] (default=3)
-isThreshold <arg> Threshold [Integer] (default=0.0)
-isVolume <arg> Volume [file] (required)
-outCropped <arg> Cropped Image [file]
-outExecution <arg> Execution Time [string]
-outNew <arg> New Dimensions [image point: semi-colon separated
list of 3 coordinates]
-outOffset <arg> Offset [image point: semi-colon separated list of
3 coordinates]
-outOriginal <arg> Original Dimensions [Integer point: semi-colon
separated list of 3 coordinates]
--DebugOn <arg> Set Debug output level[0-5] = Error messages
only, 1 = Major events and warnings, ect. 1-5
warning cause performance decrease.
--DefaultNew <arg> Set Set default maximum heap size
--dir <arg> Request Output Processing Directory (default
current) [directory] (optional)
--file <arg> Request Output Results File (default output.txt)
[file] (optional)
--jreLoc <arg> Set location of JRE to use
--MaxProcess <arg> Set default maximum number of processes.
Prints an out that is compatible as a 3D SD Slice
--xml
--saveFast <arg> Module
--saveRefChanges <arg> Set Preferred Output Image Extension
--saveIDing <arg> Save any changes made to the JIST preference
--saveIDing <arg> Sets the use of the Grid Engine[true/false]

Provided by: JIST (Java Image Science Toolkit) Command Line Interface v0.1
http://www.nitrc.org/projects/jist/

```

Fig. 2. Sample JIST POSIX compatible syntax for an image crop module. JIST automatically generates a command line program for each detected module. These programs respond to standard help requests and enable configuration of both mandatory and optional module parameters as well as system settings.

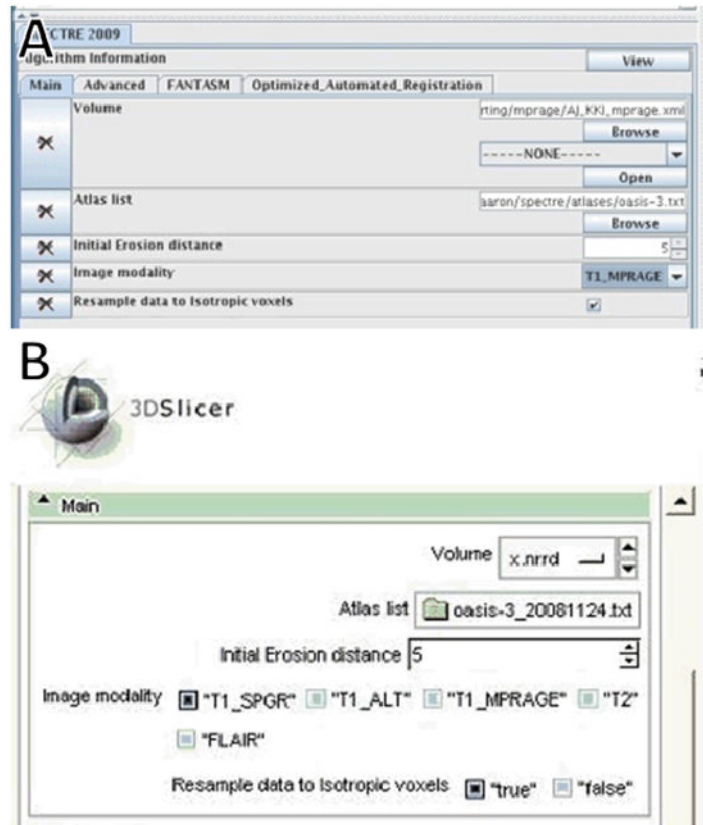


Fig. 3. Consistent cross-framework GUI presentation. XML exchange of parameter information allows JIST (A) and 3D Slicer (B) to construct consistent graphical representation of the input/output parameters for human interaction.

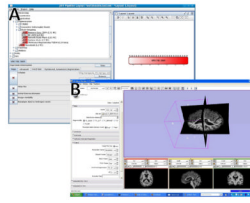


Fig. 4. The same SPECTRE binary program runs within JIST (A), 3D Slicer (B), and MIPAV (not shown).

TABLE I

Action Discovery Interface

Method	Description
get Action Metadata	Returns meta-information for categorization and labeling purposes.
create Input Parameters	Returns a table listing the input parameters (which should match up with the implicitly defined scripting parameters).
create Output Parameters	Returns a table listing the output parameters of this algorithm (usually labels used to obtain output image names later).
script Run	Calls the algorithm.
get Output Image Name	Returns the name of an image output by this algorithm, the image returned depends on the parameter label given.