Cancer Imaging Phenomics Toolkit (CaPTk): Technical Overview

Sarthak Pati

Lead Software Developer

Center for Biomedical Image Computing & Analytics (CBICA)

CaPTk ISBI Page

GitHub

Facebook

Twitter

Penn
University of Pennsylvania

SBIA
Section of Biomedical Image Analysis

Introduction

- A brief history of CaPTk
- Exploring the dependencies + guts of CaPTk
- Integrating “standalone” applications
  - MATLAB
  - Python
- Integrating native C++ applications

What does one get out of it?

- Knowledge of integrating their application into CaPTk
- Leverage existing bodies of validated APIs/algorithms
- Improved visibility and increased citations of your application
- Easier integration with clinical workflow
- Become part of a larger scientific community

A Brief History

- The **C**enter for **B**iomedical **I**mage **C**omputing and **A**nalytics (www.med.upenn.edu/cbica) consists of multiple Principal Investigators, Post Docs, Data Analysts and developers; all producing state-of-the-art algorithms published in high impact publications.

- With such a diverse group, development skills vary considerably and finding a common mechanism for distribution is difficult.

- Imperative to find a unified distribution pipeline that would enable non-computational researchers to leverage our research.

A Brief History – Continued

- Needed a way to decrease the learning curve for everyone to use our computational algorithms

- Existing packages (FSLView, MIPAV, 3DSlicer) had some deficiencies which we wanted to address with our package:
  - FSLView and MIPAV were not extensible with custom algorithms
  - 3DSlicer had a steep learning curve for our collaborators (clinicians and researchers alike)
  - 3DSlicer also did not provide an easy mechanism to extend the basic UI controls which were needed for our algorithms; for instance initializing a tumor point (coordinate + radius information).

A Brief History – Continued

- With the U24 Grant (funded by NIH ITCR), it became possible to design a common framework for distribution.

- With these points in mind, I would like to present the technical details of the **Ca**ncer Imaging **P**henomics **T**ool**k**it (CaPTk) with the idea to contribute the work we have done to the greater scientific community.

C++

- Efficient
- Truly cross-platform
- Natively supported by **ALL** major operating systems
- Code can be distributed as a stand-alone package
- Community and industry driven libraries/toolkits/packages for medical imaging

CMake

- Configuration tool for C++ projects
- Cross-platform
- Integrated Testing and Packaging environment
- Supported by virtually all IDEs, making development easier

Insight Toolkit (ITK)

- Community and industry driven package designed specifically for medical imaging
- Open source
- Well documented and validated
- Provides solid building blocks for developers via native C++ and Python wrapped calls
- Gives low and high-level access for developers, thereby providing a good starting point for developers with all skill levels

Qt

- Industry-leading open source user interface design toolkit
- Cross-platform
- Support ranging from IOT to server class machines
- Flexible and can be used from native C++ and Python wrappings

Visualization Toolkit (VTK)

- Industry-leading visualization framework
- Cross-platform
- Supports a wide variety of visualization algorithms as well as advanced modeling techniques via easy-to-use abstract classes
- Provides methods to perform 2D and 3D visualization

Open Source Computer Vision Library (OpenCV)

- Industry-leading library for computer vision and machine learning
- Cross-platform
- Designed for computational efficiency and with a strong focus on real-time applications
- Wide ranging applications from geospatial image analysis, robotics, interactive art, medical imaging

CBICA Toolkit

- Higher level functions/classes to take care of basic software design needs
  - Command Line Interface design + parsing
  - Logging with auto-generated time-stamp
  - CSV Parsing
  - System-level file/folder checks (file extensions, deleting folder, etc.)
  - Basic Statistics (ROC, etc.)

- Cross-platform and continuous validation

CBICA Toolkit - Continued

- Functions wrapped around ITK and OpenCV data structures to provide common image processing tasks
  - Single line image reading/writing for ITK with sanity checks
  - DTI scalar computation
  - Image structure manipulation (vectorize, etc.)

- Available via GitHub as a separate module!

Internals of the Code

```
${CaPTk_Source}
CMakeLists.txt
/src
/data
/docs
   …
```

- The root CMake configuration file
- Handles the project setup, looks for dependencies, structures the package overall

Internals of the Code – Continued

```
${CaPTk_Source}
CMakeLists.txt
/src
    CAPTk.h
    CAPTk.cpp
```

- Sets up the main executable and the path(s)
- Creates the main executable
- Location of all the source code, which is broken down to the following: `applications`, `cbica_toolkit` and `gui`

Internals of the Code – Continued

```
${CaPTk_Source}
CMakeLists.txt
/src
   /applications
   /cbica_toolkt
   /gui
```

- `/applications`: all the code for the native and stand-alone applications that are part of CaPTk
- `/cbica_toolkit`: CBICA Toolkit
- `/gui`: user interface files that handle the visualizer, dialogs, etc.

Internals of the Code – Continued

```
${CaPTk_Source}
CMakeLists.txt
/src
   /gui

fMainWindow.h
fMainWindow.cpp
ui_fMainWindow.h
```

- fMainWindow constructs the main UI of CaPTk
- Construction of all the menus, image viewer, image and mask loading/saving
- Handles communication between user and applications
- Communication from the main command line (`CAPTk.cpp`) uses file names

Internals of the Code – Continued

```
${CaPTk_Source}
CMakeLists.txt
/src
   /gui

fMainWindow.h
fMainWindow.cpp
ui_fMainWindow.h
```

- fMainWindow components (h, cpp and ui) are to be edited for adding any application

- UI file: declaring the menu items

```
ui_fMainWindow::setupUi( )
```

- Header file: declaring the functions and slots

- Implementation file: defining the functions

Internals of the Code – Continued

```
${CaPTk_Source}
CMakeLists.txt
/src
   /gui


SlicerManager.h
SlicerManager.cpp
```

- SlicerManager stores the image data as `itk::Image` `vtk::Image` data structures and keep both in sync

- In addition, it stores all the other metadata of the image (file path, seed points, etc.) that can be called from fMainWindow using the member variable mSlicerManagers.

Integrating Standalone Applications – Continued

- What are Standalone Applications?

  These are independent binaries that can “exec-ed” from the command line or terminal

- Their dependencies can either be part of the distribution or can be downloaded from an external site

- The simplest integration of an application into CaPTk

Integrating Standalone Applications – Continued

Converting a MATLAB script (or a collection of scripts) into a single binary using MCC (using LIBRA as example):

```
${CaPTk_Source}
  /applications
    /individualApps
      /libra_source
        /Code
        /Model


libra_startup.m
```

```
[sourceDir,~,~] = fileparts( mfilename('fullpath') );
addpath(sourceDir);
addpath(fullfile(sourceDir,'Code'));
addpath(fullfile(sourceDir,'Model'));
modelDir=fullfile(sourceDir,'Model');
```

Integrating Standalone Applications – Continued

Converting a MATLAB script (or a collection of scripts) into a single binary using MCC (using LIBRA as example):

```
${CaPTk_Source}
  /applications
    /individualApps
      /libra_source
        /Code
        /Model


libra_startup.m
libra_compile.m
```

```
[sourceDir,~,~] = fileparts( mfilename('fullpath') );
addpath(sourceDir);
addpath(fullfile(sourceDir,'Code'));
addpath(fullfile(sourceDir,'Model'));
modelDir=fullfile(sourceDir,'Model');

eval(['mcc -m -R -singleCompThread -R -nosplash -a ''',
modelDir, ''' -d ''', install_prefix, ''' -o libra libra.m']);

outputBinary=fullfile(install_prefix,'libra');
if isunix
        % Change file permission (chmod 755)
        fileattrib(outputBinary,'+x','a');
        fileattrib(outputBinary,'+w','u');
end
```

Integrating Standalone Applications – Continued

Since LIBRA uses MCR (CaPTk ships with MCRv2014), a setup file is needed to ensure CaPTk picks up MCR at runtime.

```
${CaPTk_Source}
  /applications
    /individualApps
      /libra


libra.bat
```

```
set inputimage = %1
set outputdirectory = %2
set curdrive = %CD:~0,2%
set curdir = %~dp0
%curdrive%
cd %curdir%


PATH=%PATH%;%~dp0\Mathworks\MCR\v81\runtime\win64
.\libra.exe %1 %2 %3 %4
```

Integrating Standalone Applications – Continued

Converting a Python script (or a collection of scripts) into a single binary using PyInstaller (using Confetti as example):

```
${CaPTk_Source}
  /applications
    /individualApps
      /confetti_source
        /src
          /pyGUI

runPyInstaller.py
```

```
import os
import PyInstaller
print "Creating exe"
```

Integrating Standalone Applications – Continued

Converting a Python script (or a collection of scripts) into a single binary using PyInstaller (using Confetti as example):

```
${CaPTk_Source}
  /applications
    /individualApps
      /confetti_source
        /src
          /pyGUI

runPyInstaller.py
```

```
import os
import PyInstaller
print "Creating exe"

os.system("pyinstaller.exe --onefile --windowed \
--icon=icons/confetti.ico \
--exclude-module=matplotlib \
--exclude-module=tkinter \
--exclude-module=zmq \
--exclude-module=twisted \
ConfettiGUI.py")

print "All Done"
```

Integrating Standalone Applications – Continued

Converting a Python script (or a collection of scripts) into a single binary using PyInstaller (using Confetti as example):

```
${CaPTk_Source}
/applications
 /individualApps
  /confetti_source
    /src
     /pyGUI
      runPyInstaller.py
```

```
import os
import PyInstaller
print "Creating exe"

os.system("pyinstaller.exe --onefile --windowed \
--icon=icons/confetti.ico \
--exclude-module=matplotlib \
--exclude-module=tkinter \
--exclude-module=zmq \
--exclude-module=twisted \
ConfettiGUI.py")

print "All Done"
```

Integrating Standalone Applications – Continued

- Docker
  - This is another mechanism to provide standalone executables that do not have any UI
  - While this is a perfectly valid way to package an application, we have not yet found an application which requires a Docker container, and cannot provide an example
  - If you have a package that is available as a Docker-*ized* container and are interested in integrating it with CaPTk, please let us know how we can help!

Integrating Standalone Applications – Continued

To set up the package, place your compiled application (let's use the example of LIBRA again) in the following location of ${CaPTk_source}

```
${CaPTk_Source}
/src
 /applications
  /individualApps
   /libra
```

Integrating Standalone Applications – Continued

Edit the `applications/CMakeLists.txt` file which sets up the installation of individual applications

```
${CaPTk_Source}
/src
 /applications
  CMakeLists.txt
```

```
FOREACH(subdir ${SUBDIRECTORIES})

  IF (${subdir} STREQUAL "libra")

    INSTALL(FILES ${SUBDIRPATH}/libra.exe
        DESTINATION bin # this location is important
       )
    INSTALL(FILES ${SUBDIRPATH}/libra.bat
        DESTINATION bin # this location is important
       )

  ENDIF()

ENDFOREACH()
```

Integrating Standalone Applications – Continued

Edit the `applications/CMakeLists.txt` file which sets up the installation of individual applications

```
${CaPTk_Source}
/src
 /applications
  CMakeLists.txt
```

```
FOREACH(subdir ${SUBDIRECTORIES})

  IF (${subdir} STREQUAL "libra")

    INSTALL(FILES ${SUBDIRPATH}/libra.exe
        DESTINATION bin # this location is important
       )
    INSTALL(FILES ${SUBDIRPATH}/libra.bat
        DESTINATION bin # this location is important
       )

  ENDIF()

ENDFOREACH()
```

Integrating Standalone Applications – Continued

The source-level functions to add/edit/copy are in `fMainWindow.h`, `fMainWindow.cpp` and `ui_fMainWindow.h`

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

Integrating Standalone Applications – Continued

Edit the relevant Application List (the applications are delineated by a single space).

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
auto brainAppList = " WhiteStripe PopulationAtlases confetti
EGFRvIIISurrogateIndex RecurrenceEstimator SurvivalPredictor
MolecularSubtypePredictor";
auto breastAppList = " librasingle librabatch";
auto lungAppList = " SBRT_Segment";
auto miscAppList = " itksnap GeodesicSegmentation
DirectionalityEstimate PerfusionDerivatives PerfusionPCA
DiffusionDerivatives";
```

Integrating Standalone Applications – Continued

Add the Qt “slots” that connects with the application

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
public slots:

  void ApplicationLIBRASingle(); // single image mode of LIBRA
  void ApplicationLIBRABatch(); // batch mode of LIBRA
```

Integrating Standalone Applications – Continued

Customize the menu text

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::fMainWindow()
{
…
 for (size_t i = 0; i < vectorOfBreastApps.size(); i++) {
    if (vectorOfBreastApps[i].name.find("librasingle") !=
std::string::npos) {
      vectorOfBreastApps[i].action->setText("  Breast Density
Estimator (LIBRA) SingleImage");
      connect(vectorOfBreastApps[i].action, SIGNAL(triggered()),
this, SLOT(ApplicationLIBRASingle()));
    }
  …
  }
…
}
```

Integrating Standalone Applications – Continued

Connect the menu item with the function slot

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::fMainWindow()
{
…
 for (size_t i = 0; i < vectorOfBreastApps.size(); i++) {
    if (vectorOfBreastApps[i].name.find("librasingle") !=
std::string::npos) {
      vectorOfBreastApps[i].action->setText("  Breast Density
Estimator (LIBRA) SingleImage");
      connect(vectorOfBreastApps[i].action, SIGNAL(triggered()),
this, SLOT(ApplicationLIBRASingle()));
    }
  …
  }
…
}
```

Integrating Standalone Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationLIBRASingle()
{
  auto items = m_imagesTable->selectedItems();
  if (items.empty())
  {
    ShowErrorMessage("At least 1 supported image needs to be
loaded and selected");
    return;
  }

…
}
```

Integrating Standalone Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationLIBRASingle()
{
  auto items = m_imagesTable->selectedItems();
  if (items.empty())
  {
    ShowErrorMessage("At least 1 supported image needs to be
loaded and selected");
    return;
  }

…
}
```

Integrating Standalone Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationLIBRASingle()
{
…
 auto scriptToCall = m_allNonNativeApps["libra"];

 cbica::replaceString( scriptToCall, ".bat", ".exe" );

…
}
```

Integrating Standalone Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationLIBRASingle()
{
…
 auto scriptToCall = m_allNonNativeApps["libra"];

 cbica::replaceString( scriptToCall, ".bat", ".exe" );

…
}
```

Integrating Standalone Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationLIBRASingle()
{
…
  QStringList args;
  args << dicomfilename.c_str() << tempFolderLocation.c_str();

 if (startExternalProcess(scriptToCall.c_str(), args) != 0)
  {
    ShowErrorMessage("LIBRA failed to execute. Please check
installation requirements and retry.");
  }
}
```

Integrating Standalone Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationLIBRASingle()
{
…
  QStringList args;
  args << dicomfilename.c_str() << tempFolderLocation.c_str();

 if (startExternalProcess(scriptToCall.c_str(), args) != 0)
  {
    ShowErrorMessage("LIBRA failed to execute. Please check
installation requirements and retry.");
    return;
  }
 readMaskFile(tempFolderLocation +
"/Result_Images/totalmask/totalmask.dcm");
}
```

Integrating Standalone Applications – Continued

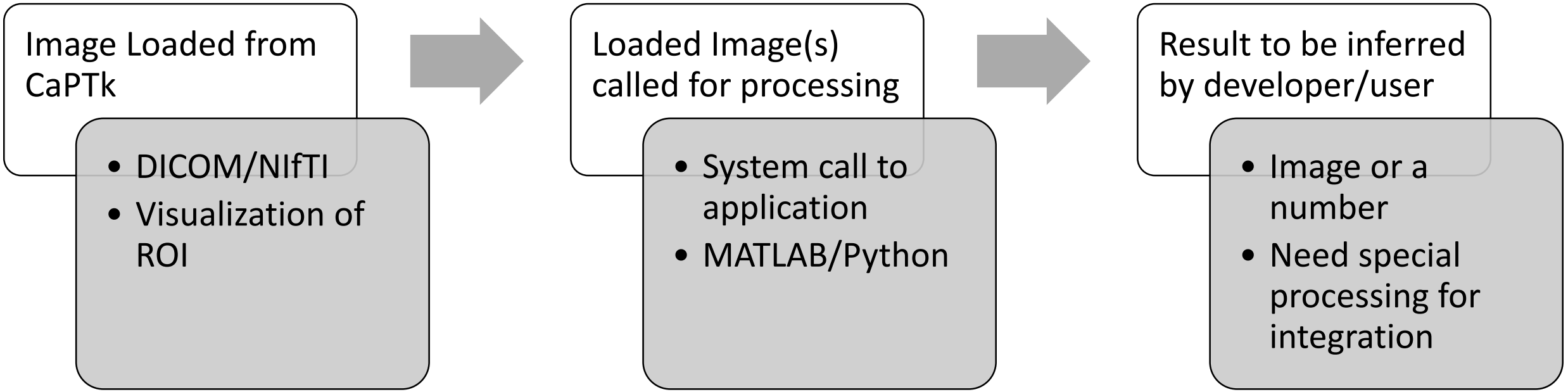
Populate the function calls

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationLIBRASingle()
{
…
  QStringList args;
  args << dicomfilename.c_str() << tempFolderLocation.c_str();

 if (startExternalProcess(scriptToCall.c_str(), args) != 0)
  {
    ShowErrorMessage("LIBRA failed to execute. Please check
installation requirements and retry.");
    return;
  }
 LoadDrawing( tempFolderLocation +
"/Result_Images/totalmask/totalmask.dcm" );
}
```

Flowchart of Standalone App running from CaPTk



Integrating Native Applications

- Written in C++ and follows a Object Oriented Structure

- Well-defined API which uses ITK Image(s) as input and gives either another image or a value as output
  - `SetInput( )` or `SetInputs( )`
  - `Update( )`
  - `GetOutput( )`
  - No file parsing or I/O should be done within the class (sanity checks for data are all done within CaPTk)

Integrating Native Applications – Continued

To set up the package, place the application files (let's use the example of EGFRvIII PHI Estimator) in the following location of ${CaPTk_source}

```
${CaPTk_Source}
/src
    /applications

EGFRvIIISurrogateIndex.h
EGFRvIIISurrogateIndex.cpp
EGFRvIIISurrogateIndex.cxx
```

Application header and implementation files that contain all the processing information

Integrating Native Applications – Continued

To set up the package, place the application files (let's use the example of EGFRvIII PHI Estimator) in the following location of ${CaPTk_source}

```
${CaPTk_Source}
/src
    /applications

EGFRvIIISurrogateIndex.h
EGFRvIIISurrogateIndex.cpp
EGFRvIIISurrogateIndex.cxx
```

Command Line Executable generator (optional)

Integrating Native Applications – Continued

Edit the `applications/CMakeLists.txt` file which sets up the installation of individual applications

```
${CaPTk_Source}
/src
 /applications
  CMakeLists.txt
```

```
SET( APPLICATIONS
  …
  EGFRvIIISurrogateIndex
  …
)

FOREACH(application ${APPLICATIONS})

  IF (${application} STREQUAL EGFRvIIISurrogateIndex)
    ADD_APPLICATION( ${application} )
  ENDIF()

ENDFOREACH()
```

Integrating Native Applications – Continued

Edit the relevant Application List (the applications are delineated by a single space).

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void setupUi(QMainWindow *fMainWindow)
{
…
  auto brainAppList = " WhiteStripe PopulationAtlases confetti
EGFRvIIISurrogateIndex RecurrenceEstimator SurvivalPredictor
MolecularSubtypePredictor";
…
}
```

Integrating Native Applications – Continued

Edit the relevant Application List (the applications are delineated by a single space).

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void setupUi(QMainWindow *fMainWindow)
{
…
  auto brainAppList = " WhiteStripe PopulationAtlases confetti
EGFRvIIISurrogateIndex RecurrenceEstimator SurvivalPredictor
MolecularSubtypePredictor";
…
}
```

Integrating Native Applications – Continued

Add the “slot” that calls the function that does the actual calculation

```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
public slots:

  void ApplicationEGFR();
```

Integrating Native Applications – Continued

Customize the menu text

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::fMainWindow()
{
…
 for (size_t i = 0; i < vectorOfGBMApps.size(); i++) {
    if (vectorOfGBMApps[i].name.find("EGFR") !=
std::string::npos) {
 vectorOfGBMApps[i].action->setText(" Glioblastoma EGFRvIII
Surrogate Index (PHI Estimator)");
      connect(vectorOfGBMApps[i].action, SIGNAL(triggered()),
this, SLOT(ApplicationEGFR()));
    }
  …
  }
…
}
```

Integrating Native Applications – Continued

Connect the menu item with the function slot

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::fMainWindow()
{
…
 for (size_t i = 0; i < vectorOfGBMApps.size(); i++) {
    if (vectorOfGBMApps[i].name.find("EGFR") !=
std::string::npos) {
 vectorOfGBMApps[i].action->setText(" Glioblastoma EGFRvIII
Surrogate Index (PHI Estimator)");
      connect(vectorOfGBMApps[i].action, SIGNAL(triggered()),
this, SLOT(ApplicationEGFR()));
    }
  …
  }
…
}
```

Integrating Native Applications – Continued

Check if a mask is defined or not

```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{
  if (!isMaskDefined())
  {
    ShowErrorMessage("EGFRvIII Estimation requires Near and Far
regions to be initialized");
    help_contextual("Glioblastoma_PHI.html");
    return;
  }
  updateProgress(5);
…
}
```

Integrating Native Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{
  if (!isMaskDefined())
  {
    ShowErrorMessage("EGFRvIII Estimation requires Near and Far
regions to be initialized");
    help_contextual("Glioblastoma_PHI.html");
    return;
  }
  updateProgress(5);
  std::vector< ImageTypeFloat3D::IndexType > nearIndices,
farIndices;
…
}
```

Integrating Native Applications – Continued

If mask is defined, then get the near and far indices that were initialized as ROIs

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{ …
  auto mask = convertVtkToItk< float, 3>(mSlicerManagers[0]-
>mMask);
  ImageTypeFloat3DIterator maskIt(mask, mask-
>GetLargestPossibleRegion());
  for( maskIt.GoToBegin(); !maskIt.IsAtEnd(); ++maskIt; ) {
    if (maskIt.Get() == 1)
      nearIndices.push_back(maskIt.GetIndex());
    else if (maskIt.Get() == 2)
      farIndices.push_back(maskIt.GetIndex());
  }
…
}
```

Integrating Native Applications – Continued

If mask is defined, then get the near and far indices that were initialized as ROIs

```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{ …
  auto mask = convertVtkToItk< float, 3>(mSlicerManagers[0]-
>mMask);
  ImageTypeFloat3DIterator maskIt(mask, mask-
>GetLargestPossibleRegion());
  for( maskIt.GoToBegin(); !maskIt.IsAtEnd(); ++maskIt; ) {
    if (maskIt.Get() == 1)
      nearIndices.push_back(maskIt.GetIndex());
    else if (maskIt.Get() == 2)
      farIndices.push_back(maskIt.GetIndex());
  }
…
}
```

Integrating Native Applications – Continued

If mask is defined, then get the near and far indices that were initialized as ROIs

```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{ …
  auto mask = convertVtkToItk< float, 3>(mSlicerManagers[0]-
>mMask);
  ImageTypeFloat3DIterator maskIt(mask, mask-
>GetLargestPossibleRegion());
  for( maskIt.GoToBegin(); !maskIt.IsAtEnd(); ++maskIt; ) {
    if (maskIt.Get() == 1)
      nearIndices.push_back(maskIt.GetIndex());
    else if (maskIt.Get() == 2)
      farIndices.push_back(maskIt.GetIndex());
  }
…
}
```

Integrating Native Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{ …
 for (unsigned int index = 0; index < mSlicerManagers.size();
index++) {
    if (mSlicerManagers[index]->mImageSubType == IMAGE_TYPE_T1CE)
      T1CEImagePointer = mSlicerManagers[index]->mITKImage;
    else if (mSlicerManagers[index]->mImageSubType ==
IMAGE_TYPE_T2FLAIR)
      T2FlairImagePointer = mSlicerManagers[index]->mITKImage;
    else if (mSlicerManagers[index]->mImageSubType ==
IMAGE_TYPE_PERFUSION)
      perfusionImage = mSlicerManagers[index]-
>mPerfusionImagePointer;
…
}
```

Integrating Native Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{ …
 for (unsigned int index = 0; index < mSlicerManagers.size();
index++) {
    if (mSlicerManagers[index]->mImageSubType == IMAGE_TYPE_T1CE)
      T1CEImagePointer = mSlicerManagers[index]->mITKImage;
    else if (mSlicerManagers[index]->mImageSubType ==
IMAGE_TYPE_T2FLAIR)
      T2FlairImagePointer = mSlicerManagers[index]->mITKImage;
    else if (mSlicerManagers[index]->mImageSubType ==
IMAGE_TYPE_PERFUSION)
      perfusionImage = mSlicerManagers[index]-
>mPerfusionImagePointer;
…
}
```

Integrating Native Applications – Continued

Populate the function calls

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{ …
  EGFRStatusPredictor EGFRPredictor;
  auto EGFRStatusParams = EGFRPredictor.PredictEGFRStatus<
ImageTypeFloat3D, PerfusionImageType >(perfusionImage,
Perfusion_Registered, nearIndices, farIndices, NIfTI);
…
}
```

Integrating Native Applications – Continued

Display the results

```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{ …
  auto msg = "PHI = " + QString::number(EGFRStatusParams[0]) +
"\n\n----------\n\n(Near:Far) Peak Height ratio = " +
    QString::number(EGFRStatusParams[1] / EGFRStatusParams[2]) +
"\n\nNear ROI voxels used = " +
    QString::number(EGFRStatusParams[3]) + "/" +
QString::number(nearIndices.size()) + "\nFar ROI voxels used =
" +
    QString::number(EGFRStatusParams[4]) + "/" +
QString::number(farIndices.size()) +
    "\n\n\n----------\n\nPHI Threshold = 0.1377\n[based on 142
UPenn brain tumor scans]";
…
}
```

Integrating Native Applications – Continued

Display the results

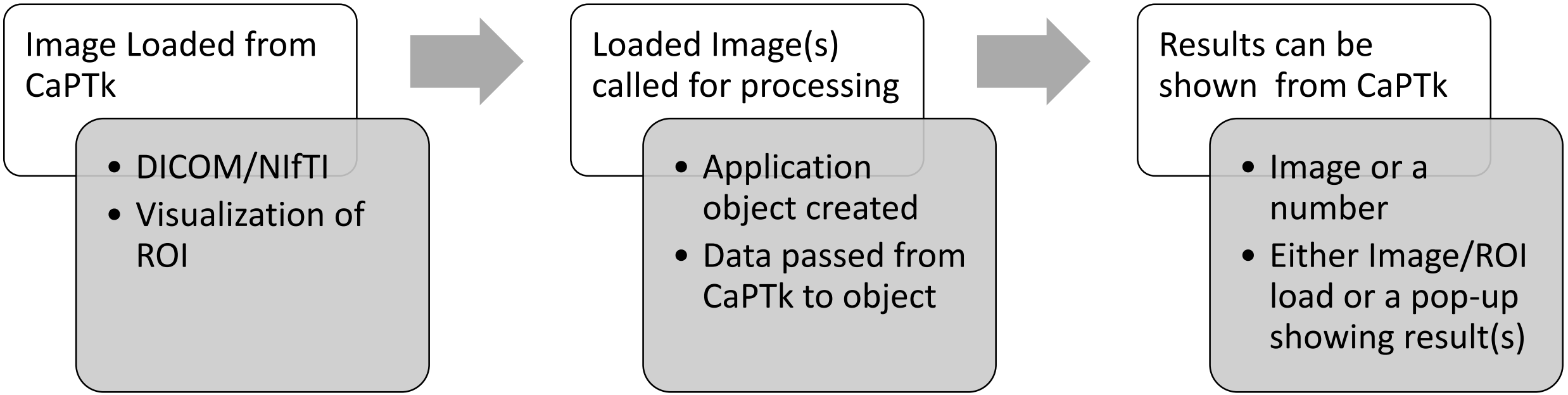
```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
void fMainWindow::ApplicationEGFR()
{ …
  ShowMessage(msg.toStdString());

  updateProgress(100);

}
```

Flowchart of Native App running from CaPTk



Application requiring User Parameterization

Considering WhiteStripe as an example, we follow the same procedures as EGFRvIII

```
${CaPTk_Source}
/src
 /applications

WhiteStripe.h
WhiteStripe.cpp
WhiteStripe.cxx
```

Application header, implementation files and command line executable generator

Application requiring User Parameterization

Since WS needs user-defined parameters, we need to create a dialog box which emits the final "Run" signal with the provided parameters (radius, kernel size, etc.)

```
${CaPTk_Source}
/src
 /gui

ui_fWhiteStripeDialog.h
fWhiteStripeDialog.h
fWhiteStripeDialog.cpp
```

Creation of the dialog box interface and its relevant controls

Integrating Native Applications – Continued

Edit the `applications/CMakeLists.txt` file which sets up the installation of individual applications

```
${CaPTk_Source}
/src
 /applications
  CMakeLists.txt
```

```
SET( APPLICATIONS
  …
  WhiteStripe
  …
)

FOREACH(application ${APPLICATIONS})

  IF (${application} STREQUAL WhiteStripe)
    ADD_APPLICATION( ${application} )
  ENDIF()

ENDFOREACH()
```

Integrating Native Applications – Continued

Edit the relevant Application List (the applications are delineated by a single space).

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void setupUi(QMainWindow *fMainWindow)
{
…
  auto brainAppList = " WhiteStripe PopulationAtlases confetti
EGFRvIIISurrogateIndex RecurrenceEstimator SurvivalPredictor
MolecularSubtypePredictor";
…
}
```

Integrating Native Applications – Continued

Add the “slot” that calls the function that does the actual calculation

```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
public slots:

  void ApplicationWhiteStripe();
```

Integrating Native Applications – Continued

Customize the menu text

```
${CaPTk_Source}
/src
 /gui
  ui_fMainWindow.h
  fMainWindow.h
  fMainWindow.cpp
```

```
void fMainWindow::fMainWindow()
{
…
 for (size_t i = 0; i < vectorOfGBMApps.size(); i++) {
    if (vectorOfGBMApps[i].name.find("WhiteStripe") !=
std::string::npos) {
 vectorOfGBMApps[i].action->setText("  WhiteStripe
Normalization");
      connect(vectorOfGBMApps[i].action, SIGNAL(triggered()),
this, SLOT(ApplicationWhiteStripe()));
    }
  …
  }
…
}
```

Integrating Native Applications – Continued

Connect the menu item with the function slot

```
${CaPTk_Source}
/src
 /gui
   ui_fMainWindow.h
   fMainWindow.h
   fMainWindow.cpp
```

```
void fMainWindow::fMainWindow()
{
…
 for (size_t i = 0; i < vectorOfGBMApps.size(); i++) {
    if (vectorOfGBMApps[i].name.find("WhiteStripe") !=
std::string::npos) {
 vectorOfGBMApps[i].action->setText("  WhiteStripe
Normalization");
      connect(vectorOfGBMApps[i].action, SIGNAL(triggered()),
this, SLOT(ApplicationEGFR()));
    }
  …
  }
…
}
```

Application requiring User Parameterization

Connect the “Run” signal from the dialog with the MainWindow Interface.

```
${CaPTk_Source}
/src
 /gui


ui_fWhiteStripeDialog.h
fWhiteStripeDialog.h
fWhiteStripeDialog.cpp
```

```
void fMainWindow::fMainWindow()
{ …
  connect(&whiteStripeNormalizer,
SIGNAL(RunWhiteStripe(double, int, int, int, double,
double, int, bool, const std::string)), this,
SLOT(CallWhiteStripe(double, int, int, int, double,
double, int, bool, const std::string));

}
```

Application requiring User Parameterization

Connect the “Run” signal from the dialog with the MainWindow Interface.

```
${CaPTk_Source}
/src
 /gui


ui_fWhiteStripeDialog.h
fWhiteStripeDialog.h
fWhiteStripeDialog.cpp
```

```
void fMainWindow::fMainWindow()
{ …
  connect(&whiteStripeNormalizer,
SIGNAL(RunWhiteStripe(double, int, int, int, double,
double, int, bool, const std::string)), this,
SLOT(CallWhiteStripe(double, int, int, int, double,
double, int, bool, const std::string)));

}
```

Application requiring User Parameterization

Connect the “Run” signal from the dialog with the MainWindow Interface.

```
${CaPTk_Source}
/src
 /gui


ui_fWhiteStripeDialog.h
fWhiteStripeDialog.h
fWhiteStripeDialog.cpp
```

```
void fMainWindow::fMainWindow()
{ …
  connect(&whiteStripeNormalizer,
SIGNAL(RunWhiteStripe(double, int, int, int, double,
double, int, bool, const std::string)), this,
SLOT(CallWhiteStripe(double, int, int, int, double,
double, int, bool, const std::string)));

}
```

Application requiring User Parameterization

This checks for the inputs within CaPTk and then loads up the WhiteStripe dialog box

```
${CaPTk_Source}
/src
 /gui


ui_fWhiteStripeDialog.h
fWhiteStripeDialog.h
fWhiteStripeDialog.cpp
```

```
void fMainWindow::ApplicationWhiteStripe()
{ …

 if ((mSlicerManagers[index]->mImageSubType ==
IMAGE_TYPE_T1) || (mSlicerManagers[index]->mImageSubType
== IMAGE_TYPE_T2)) {
    auto tmp = mInputPathName.toStdString();

whiteStripeNormalizer.SetImageModality(mSlicerManagers[i
ndex]->mImageSubType);
    whiteStripeNormalizer.exec();
  }
}
```

Application requiring User Parameterization

This checks for the inputs within CaPTk and then loads up the WhiteStripe dialog box

```
${CaPTk_Source}
/src
 /gui


ui_fWhiteStripeDialog.h
fWhiteStripeDialog.h
fWhiteStripeDialog.cpp
```

```
void fMainWindow::CallWhiteStripe(…)
{ …

 WhiteStripe normalizer;
  normalizer.setParams(twsWidth, sliceStartZ,
sliceStopZ, tissuesMax, smoothMax, smoothDelta,
histSize, T1Image);

  ImageTypeFloat3D::Pointer mask;
  auto normImage =
normalizer.process(mSlicerManagers[index]->mITKImage,
mask);
}
```

Application requiring User Parameterization

This checks for the inputs within CaPTk and then loads up the WhiteStripe dialog box

```
${CaPTk_Source}
/src
 /gui


ui_fWhiteStripeDialog.h
fWhiteStripeDialog.h
fWhiteStripeDialog.cpp
```

```
void fMainWindow::CallWhiteStripe(…)
{ …

 WhiteStripe normalizer;
  normalizer.setParams(twsWidth, sliceStartZ,
sliceStopZ, tissuesMax, smoothMax, smoothDelta,
histSize, T1Image);

  ImageTypeFloat3D::Pointer mask;
  auto normImage =
normalizer.process(mSlicerManagers[index]->mITKImage,
mask);
}
```