InSAR Scientific Computing Environment

Advanced Information Systems Technology Task AIST-08-0023

Paul A. Rosen, JPL (PI)

Howard Zebker, SU (Co-I)

Eric Gurrola, JPL (Co-I)

Giangi Sacco, JPL (Co-I)

Mark Simons, Caltech (Co-I)

Scott Hensley, JPL (Collaborator)

David Sandwell, SIO (Collaborator)

December 14, 2009

Fall AGU, San Francisco, CA 2009

Earth Science Technology Office

InSAR Scientific Computing Environment

PI: Rosen, Paul, Jet Propulsion Laboratory

Objective
- Develop an open-source, modular, extensible InSAR computing environment for the research community
- Incorporate state-of-the-art, highly accurate algorithms to automate InSAR processing for non-experts and experts alike
- Document algorithms, formats and interfaces to facilitate community involvement in continuing development beyond the AIST horizon

A computation suite that facilitates interaction with InSAR data and models

Approach
- Develop community-based requirements for InSAR processing methods and generalized data models
- Develop a modular, extensible, object-oriented processing framework
- Develop modules for the ISCE architecture
- Test and document ISCE framework

Co-Is/Partners

Howard Zebker, Stanford University

Mark Simons, Caltech

Eric Gurrola, Giangi Sacco, Scott Hensley, JPL

David Sandwell, Scripps Institution of Oceanography

Key Milestones
- First draft of requirements document (8/09->10/09)
- First draft of architecture description (1/10)
- Algorithmic accuracy improvements (4/10)
- Framework recasting of processing engines (12/10)
- Principal code elements complete (4/11)
- Completion of testing (2/12)
- Deliver final documentation and software (3/12)

TRL_{in} = 3

InSAR SCE and ROI_PAC background

- InSAR SCE was proposed and accepted as an open source package
- InSAR SCE was founded on a science community workshop funded by NASA
- The science community accepted the proposal based on this notion; the community looks forward to an open source solution.
  - Without open source, this development becomes much less relevant for scientists.

- ROI_PAC was released through EAR and contains similar algorithms. The algorithms are published and in the public domain. Its license has expired.
- Upgrades to ROI_PAC to be included in InSAR SCE are published and just need to be implemented.

The Standard ROI_PAC Flow

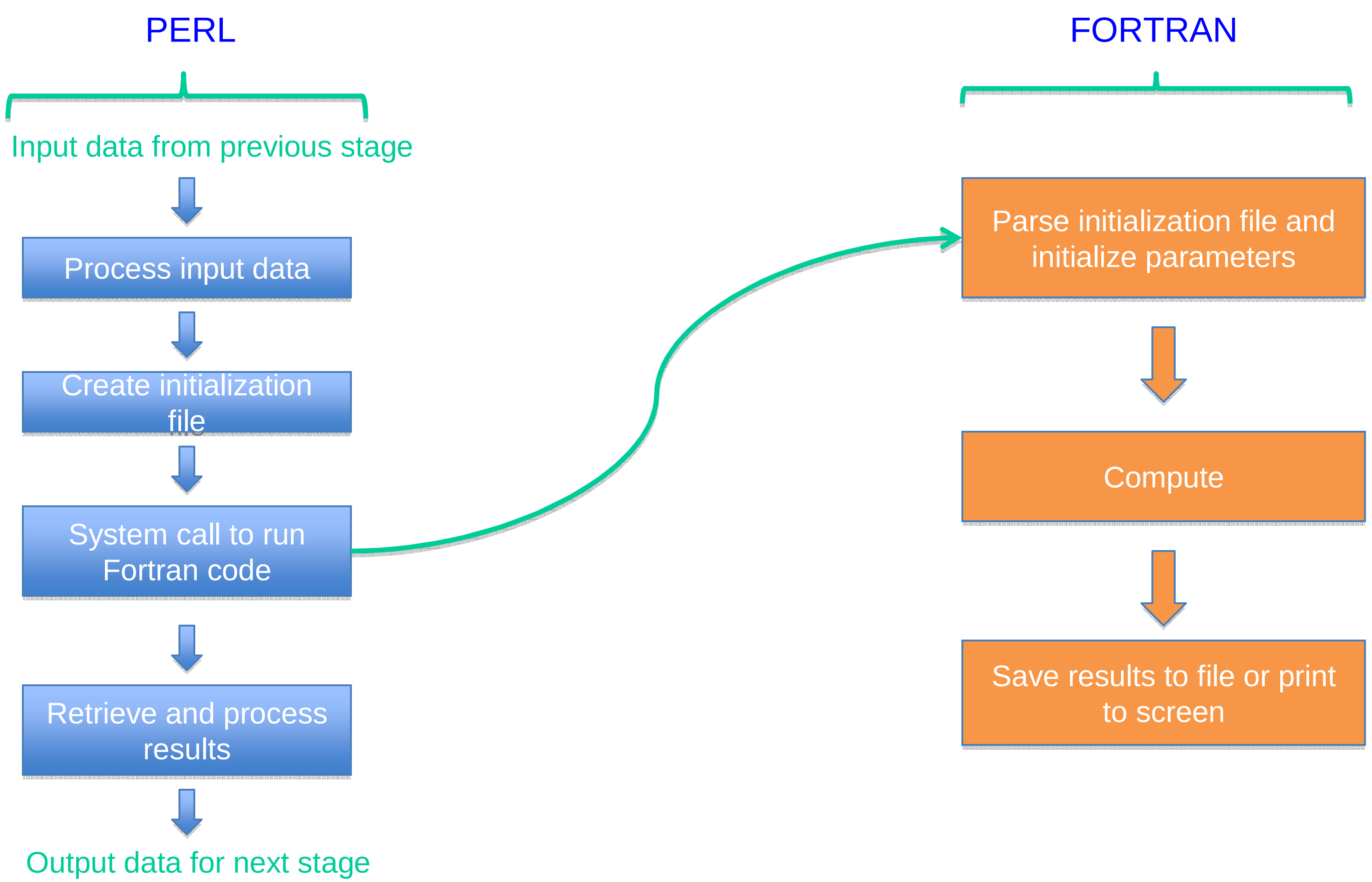
- Standard ROI_PAC recipe creates geocoded unwrapped interferograms from two raw data files, an orbit file, and other ancillary meta-data
- Allows the use of a DEM for terrain correction and topography removal from interferogram
- Allows the use of an a priori deformation model for the purpose of enhancing baseline re-estimation or unwrapping
- Prescription for processing is somewhat inflexible

InSAR SCE Framework Requirements

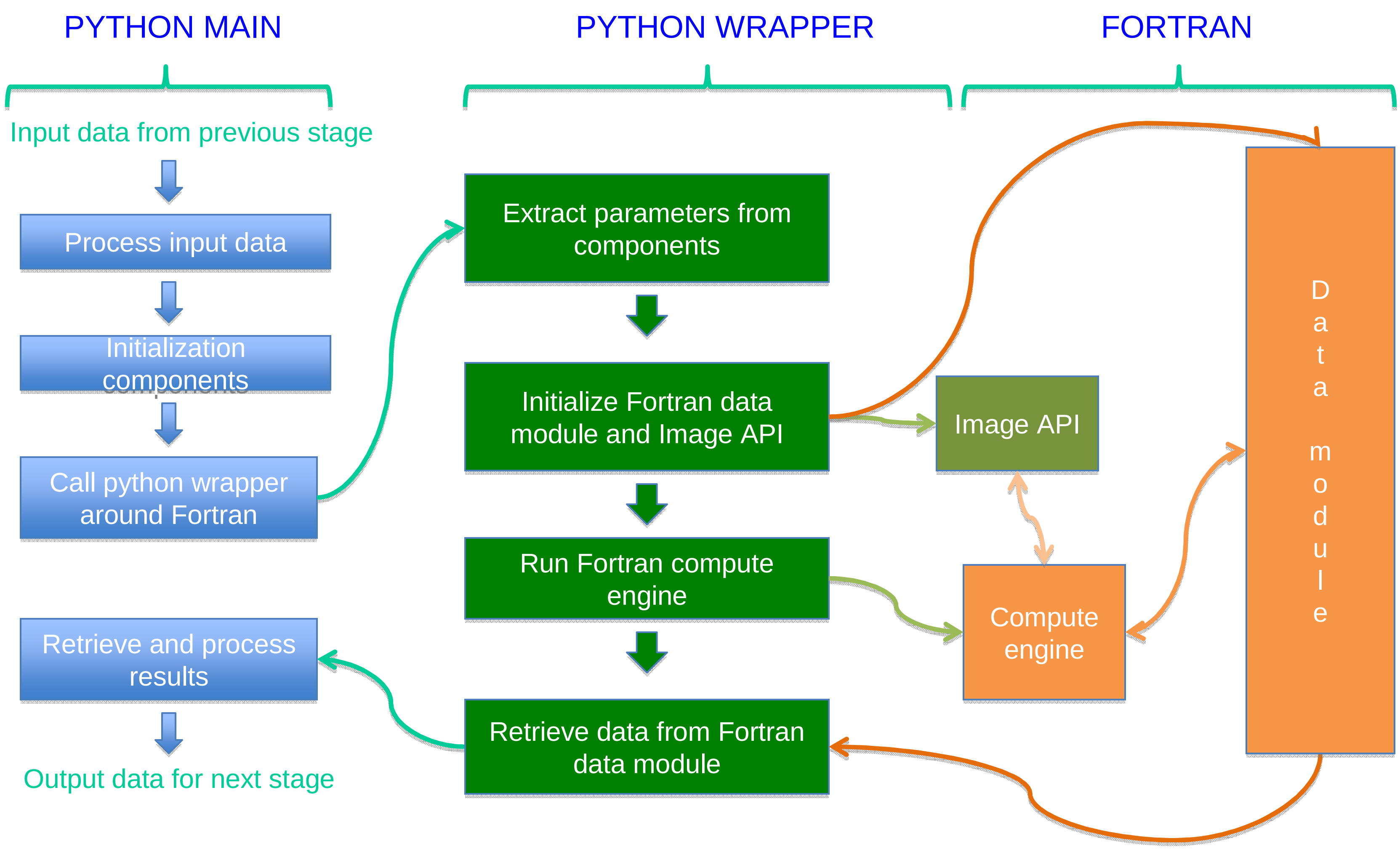
Note: zoom to 400% to read

R2.7	*The ISCE shall consist of a modular, extensible Framework that provides common services to be used by the Core radar processing components and provides a user/developer interface to the Core components that makes it easy for the user/developer to control the data flow and data operations necessary to acheive their desired objective*	The ISCE shall have a framework designed to facilitate the user and developer interface to Core processing code and provide convenient services to the user in prescribing their scientific objective. The framework will also apply modern programming standards that enable and encourage modularity, flexibility, and extensibility at a high level so that it does not impact the development of the optimal Core processing engines that are typically developed with other goals such as fidelity and throughput in mind and with more linear task-oriented coding standards commonly employed by scientist/engineer programmers	Inspection	Complete Package	Inspection by the ISCE team and if possible by feedback from interested parties in the user community	2/1/11	
R2.7.1	*The ISCE Framework shall wrap all Core radar processing engines with controlling components that provides the interface between the Framework services and user interface and the Core components.*	Separates control and integration functionality from the main processing engines to make the overall design more modular and more easilty modifiable	Inspection	Complete Package	Inspection by the ISCE team	6/1/10	
R2.7.1.1	*The ISCE Framework wrappers shall be written in Python with C as intermediary between the Python and Fortran using the standard C Python API and the design shall not preclude the optional use by future customers of third party software such as F2Py or Swig for generating wrappers for their Fortran or C code.*	Python is an open source language with a large scientist/engineer user/developer community that is constantly influencing the course of development of the language and adding new application packages that can be used with the core language. The MROI_PAC development effort that preceded this project used Python for the framework language and the ISCE can make direct use of the wrappers developed under the MROI_PAC effort	Inspection	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	6/1/10	
R2.7.2.2	*The ISCE Framework wrappers shall include formal management of the life cycle of components from intialization to finalization.*						
R2.7.2	*The ISCE Framework shall be extensible*	Provides for greater flexibility and longevity of the ISCE as new algorithmic methods replace old methods, or new capablilities are required of the ISCE, or new data formats and types are used	Test	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	1/1/11	
R2.7.2.1	*At appropriate levels develop functional interfaces and program to the interface not to any particular implementation*	Allows for alternative implementations of major functions	Inspection	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	1/1/11	
R2.7.2.2	*The ISCE shall be written in a way that admits a plug-in style of extension and shall include examples of plug-in code*	Makes use of a proven method for implementing extensibility	Inspection	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	1/1/11	
R2.7.2.3	*The ISCE shall be writen such that acquisition of capabilities for a given software object is done by the object oriented pattern of composition rather than inheritance wherever there is an equal choice between the two methods*	Allows the software to evolve incrementally from the top rather than from the roots up, which requires far less rewriting of already established components	Inspection	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	1/1/11	
R2.7.2.4	*Provide a programmer friendly method for incorporating compiled user code into the framework*	Allows the user and/or developer to use his own software within the context of the framework which allows the user to benefit from the services provided by the framework and may lead to its incorporation into the distributable framework for others to benefit from	Test	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	1/1/11	
R2.7.3	*The ISCE Framework shall be modular*	Increases possibilities for prescribing data flow in unforseen ways and promotes coding standards that solidify the integrity of individual modules.	Inspection	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	6/1/10	
R2.7.3.1	*Modules shall be object oriented*	Encapsulation principal leads to greater modularity	Inspection	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	6/1/10	
R2.7.3.2	*Module components shall be designed with maximal reusability in mind*	Encourages encapsulation of common functions or operations in software modules that may be reused. Simplifies code maintenance	Inspection	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	6/1/10	
R2.7.3.3	*Module components shall be written with minimal functionality necessary to do their principal job with auxiliary functionalities being encapsulated in separate modules*	Reduce number of tasks that a given component does to an appropriate granularity increases its ability to be reused and to be plugged in to a multiply configurable pipes and boxes architecture. Promotes greater flexibility	Inspection	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	6/1/10	
R2.7.3.4	*The ISCE Framework shall organize control parameters and data parameters within logical objects or data structures that will be globally available to be used throughout the Framework and Core components*	Promotes single source maintainers of parameters thereby ensuring consistency throughout the data flow	Inspection	Complete Package	Inspection by the ISCE team and by seeking feedback if possible from other framework and radar processing domain experts	6/1/10	
R2.7.4	*The ISCE Framework shall provide Input/Output services to be used by the radar processing core software*	Separate Input-Output operations from processing code to allow I/O models to be added to the framework without affecting core procesing engines and to allow the user to configure the specific I/O formats at run time thereby maximizing extensibility, flexibility, and ease of use.	Inspection	Input/Output API to be used by the ISCE Core processing components and Framework components	Inspection by the ISCE team and by seeking feedback if possible from other experts in framework development	3/1/10	
R2.7.4.1	*The ISCE Framework I/O services shall handle raw binary radar and orbit data files, provide defaults for a diverse set of modern satellite and airborne or data types (R1.2.1.1), and also be user configurable and extensible for unforseen special or new cases*	The Framework I/O services must at least support, but not be limited to, the raw binary radar and orbit data files from the platforms that are required to be supported in order that that requirement may be met	Test with data sets from supported and goal platforms	I/O API	ROI_PAC End-To-End Test Set and I/O API Unit Test Set	3/1/10	
R2.7.4.2	*The ISCE Framework I/O services shall provide data converters that will allow a diverse set of modern satellite and airborne radar and orbit data (R2.1.1.1) to be ingested into the framework*	To allow for the possibility of conditioning diverse radar and orbit data types to be converted before processing into the standard supported formats in the ISCE code	Test with data sets from supported and goal platforms	I/O API	ROI_PAC End-To-End Test Set and I/O API Unit Test Set	3/1/10	
R2.7.4.3	*The ISCE Framework I/O services shall be capable of ingesting the meta-data for raw binary from a diverse set of modern satellite and airborne radar and orbit data formats (R2.1.1.1) and shall encapsulate the meta data in appropriate data structures*	The meta-data from the supported and goal platforms come in a few different formats and the ISCE Framework must at least support those formats. The design should also allow for support of future meta-data formats. The meta-data associated to the raw data from these platforms are a critical source of the processing parameters and proper encapsulation of this information in data structures at the point of ingestion of this information helps to ensure that all processing components are given properly validated and consistent parameters.	Test with data sets from supported and goal platforms	I/O API	ROI_PAC End-To-End Test Set and I/O API Unit Test Set	3/1/10	
R2.7.4.4	*The ISCE Framework I/O services shall provide configurable components for all other I/O needs of the Framework or Core components such as standard output, error output, and run time logging*	Every I/O function should be configurable by the user at run time, although default configurations will be available and adequate for most circumstances. This allows for maximal flexibility.	Test	I/O API	I/O API Unit Test Set	3/1/10	
R2.7.5	*The ISCE Framework shall provide a User Interface that is flexible and easy to use*	A well thought out User Interface should accommodate both non-radar expert scientist users as well as the radar processing expert. It is critical that both types of users will be able to effectively use the User Interface so that there is not two "tracks" for using the software and so that the potential problems and misunderstandings encountered in using the software are also experienced by the expert.	Test	Framework User Interface	End-to-End Tests by the ISCE team as well as by interested non-experts from the potential user community	2/1/11	
R2.7.5.1	*The ISCE Framework User Interface shall allow an easy to use system to configure and execute given standard workflows and processing configuration parameters*	There should be a list of pre-wired standard workflows that the user can select from and that he can easily configure	Test	Framework User Interface	End-to-End Tests by the ISCE team as well as by interested non-experts from the potential user community	2/1/11	
R2.7.5.1.1	*The ISCE Framework User Interface shall allow the user to create a configuration parameter file and run a standard process workflow from a Unix command line with a single command*	Executing commands from the Unix command line is a familiar User Interface that many of the radar processing experts and potential scientist users are most comfortable with	Test	Framework User Interface	End-to-End Tests by the ISCE team as well as by interested non-experts from the potential user community	2/1/11	
G2.7.5.1.2	*The ISCE Framework User Interface shall provide a GUI and/or web browser interface to select from a list of standard workflows, to set configuration parameters for a selected standard workflow, and to execute that workflow*	The specialized GUI or Internet Services through the use of forms type of interface for executing the standard workflows is also a familiar user interface that could easily be supported as time permits	Test	Framework User Interface	End-to-End Tests by the ISCE team as well as by interested non-experts from the potential user community	2/1/11	
R2.7.5.2	*The ISCE Framework User Interface shall provide user friendly systems to prescribe their own workflow and associated processing configuration parameters*	Both the radar-processing expert and the non-radar scientist at some point will very likely need to use a subset of one of the standard workflows, or will need to add on additional steps onto a standard workflow, or may need to use custom versions to replace certain steps in the standard workflow, or may need to create a whole new workflow from scratch.	Test	Framework User Interface	Tests created by the ISCE team as well as by interested non-experts from the potential user community if possible	2/1/11	
R2.7.5.2.1	*The ISCE Framework User Interface shall allow the user to prescribe a workflow with a Python script provided by the user*	Scripts, whether a Unix shell script or a Perl or Python script, are currently very common and often preferred methods of prescribing workflows by the target user community. Python would be the preferred scripting language, although strictly speaking not the only option, since it would be compatible with the ISCE Framework language and could easily evolve into a standard component of the ISCE Framework. With several simple examples the user can learn to create workflows and to use the Framework API with Python scripts.	Test	Framework User Interface	Test scripts created by the ISCE team as well as by interested non-experts from the potential user community if possible	2/1/11	
R2.7.5.2.2	*The ISCE Framework User Interface shall provide a user friendly system to prescribe user selections for processing parameters and to provide defaults for those parameters*	The User interface should be an aid to the non-expert in terms of requiring a minimum of configuration decisions from him by being built-in with defaults where applicable and the capability to determine calculate configuration parameters with minimal help from the user. The same User Interface should also allow the expert to configure any parameter he needs to without having to alter source code or behind the scenes configuration files. The User Interface should also help the non-expert to understand the meaning of the configuration paramaters he is being asked for without having to become a radar processing expert or having to consult external documents.	Test	Framework User Interface	Test runs created by the ISCE team as well as by interested non-experts from the potential user community if possible	2/1/11	
R2.7.5.2.3	*The ISCE Framework User Interface shall allow for selection options for different processing methods (such as phase unwrapping or FFT methods for instance) and quality metrics*	At a granularity decided by the ISCE team selection of alternative processing methods by the user should be allowed through configuration parameters without requiring rewriting of the frameowork itself. Polymorphism or the ability to choose specific implementations of certain tasks is a common modern design principal allowed for in modern object oriented designs and will be provided for in the ISCE at an appropriate Framework Component level (with the phase unwrapping component example in mind) as well as at an appropriate level internal to Core processing components (with the FFT example in mind) will be planned for by the ISCE Framework development team.	Test	Framework User Interface	Tests created by the ISCE team as well as by interested non-experts from the potential user community if possible	2/1/11	
G2.7.5.2.4	*The ISCE Framework User Interface shall allow the user to prescribe a workflow with an XML document*	XML is a standard method to represent data and process flows which is well supported by the internet community. There are common procedures and software for converting XML process descriptions into software.	Test	Framework User Interface	Tests created by the ISCE team as well as by interested non-experts from the potential user community if possible	2/1/11	
G2.7.5.2.5	*The ISCE Framework User Interface shall be compatible with a third party pipe and boxes visual scripting interface such as that provided by Vistrails*	Third party visual scripting tools such as Vistrails (www.vistrails.org) provide a ready made user interface and are easily applied "on top" of user software that complies with their interface model and with a minimal amount of support code. The MROI_PAC project has already tested the compatibility of Vistrails with our Python-wrapping of the legacy Fortran code. The support code necessary to support user code can easily be encapsulated in the ISCE Framework.	Test	Framework User Interface	Tests with Vistrails	2/1/11	
G2.7.5.2.6	*The ISCE Framework User Interface shall be compatible with a third party user interface such as that provided by Pyre*	Pyre (www.cacr.caltech.edu/projects/pyre) is a third party Framework that provides a command line-driven user interface that includes many features (such as validation and consistency of user input and unit conversion on input) and convenience services. MROI_PAC has developed many of its modules with Pyre in mind.	Test	Framework User Interface	Test with Pyre	2/1/11	
R2.7.5.3	*The ISCE Framework User Interface shall provide a system that preserves, logs, and versions the workflow and data products resulting from the workflow*	Fully documents the pedigree of a particular product resulting from the ISCE framework, allows for reliable reproduction of those results, and facilitates the reprocessing requirement R2.7.5.4 and error handling requirement R2.7.6	Test	Framework User Interface	End to end tests	2/1/11	
R2.7.5.4	*The ISCE Framework User Interface shall allow reprocessing only elements of a process flow that are necessary to be reprocessed due to changes in parameter or flow connections. In other words, do not reprocess data in a flow that does not change as a result of flow changes*	To provide maximum flexibility in terms of altering a flow in mid stream or iterating a particular stage of a flow in order to optimize the results in that stage without any wasted resources and time that would be involved if the entire flow had to be restarted every time a change in the flow were made upstream.	Test	Framework User Interface as well as the ISCE Framework and Core components	End to end tests	2/1/11	
R2.7.6	*The ISCE Framework and Core components shall provide robust error handling*	Robust error handling allows for full documentation of the cause of an error that will make using the ISCE easier to use by non-experts as well as experts.	Test	Complete Package	End to end tests	6/1/10	
R2.7.7	*Provide a memory management model that enables efficient and controllable use of memory within the architecture*	Efficient memory management optimizes the use of resources to enable processing of large amounts of data	Inspection and Test	Complete Package	End to end tests	6/1/10	

The ROI_PAC framework



The mroipac framework



The mroipac motto

do one thing, but do it right

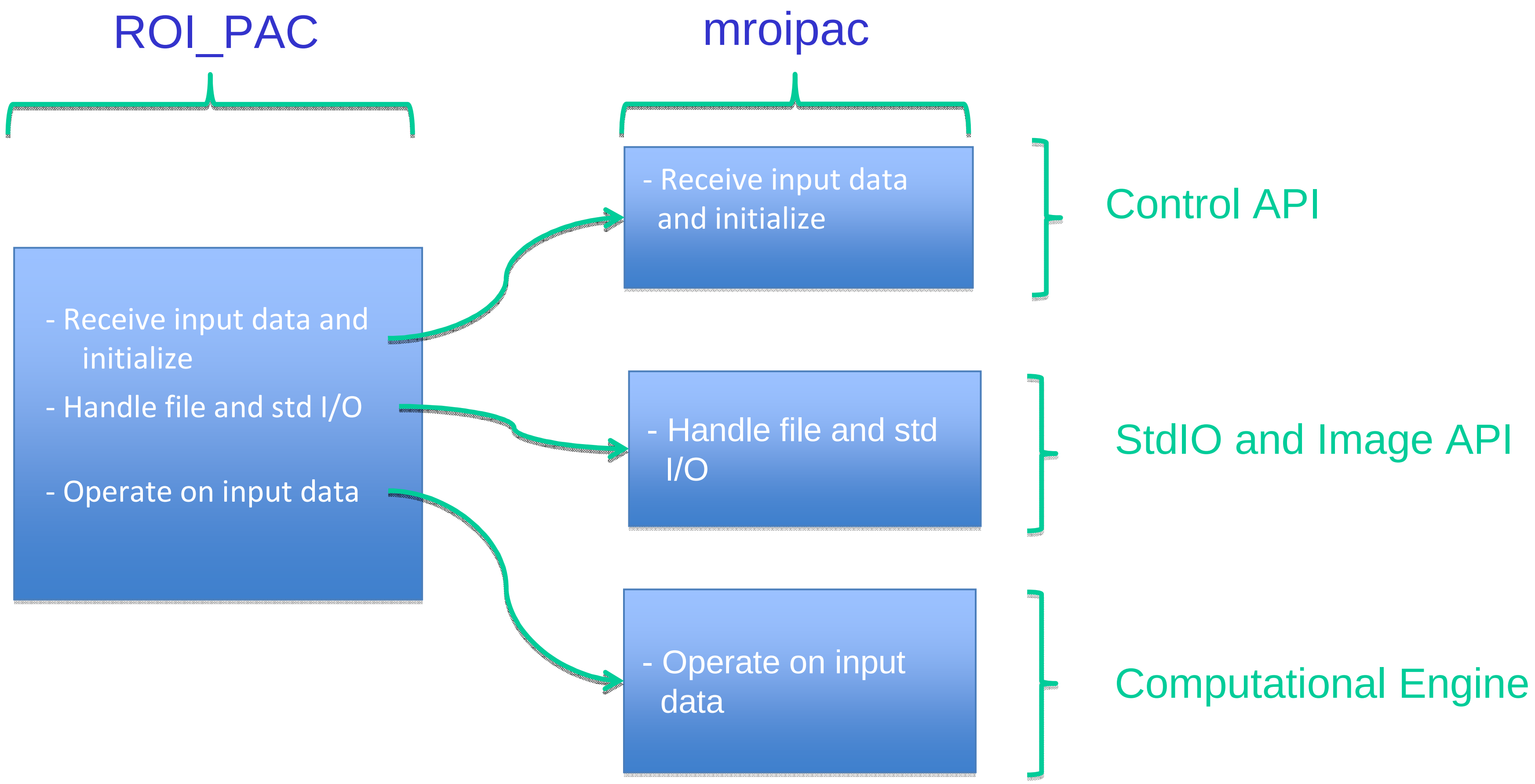
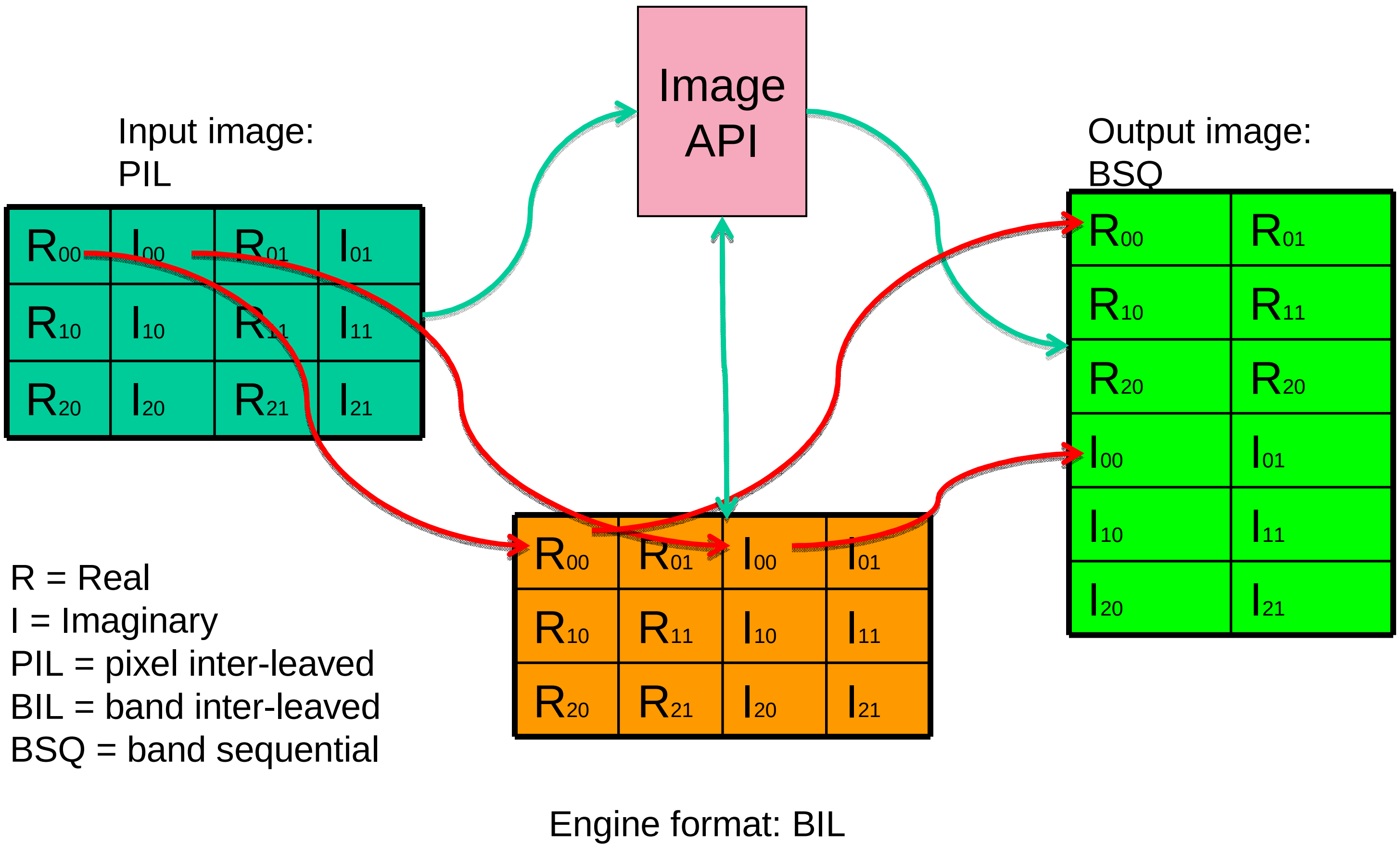


Image API Example

Example: interleaving scheme translation.



Example: FormSLC – the core radar processor

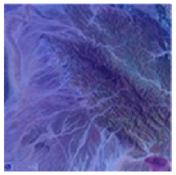
The Driver

```
import FormSLCPy
import obj
...

#initialize the objects from a file
objects = obj.createObjects(file)

#run the module by passing the
#objects
FormSLCPy.run(objects)

#get results after computation
results = FormSLCPy.getResults()
```



```
image.filename = filename
image.width = width
......
```

```
radar.frequency = frequency
radar.chirp = chirp
......
```

```
platform.name = name
platform.height= height
......
```

```
planet.radius = radius
planert.angularSpeed= angularSpeed
......
```


ESTO Earth Science Technology Office

Restructuring of I/O in Fortran

Create image: old way

```
program  OldWay
integer*4 FD
character*80 filename
integer*4 lineWidth
......
! open file from inside the FORTRAN code
open(FD,file=filename,access=‘direct’,recl=lineWidth,form=‘unformatted’)
```

Restructuring of I/O in Fortran

Create image: new way

```
program  NewWay
integer*8 imagePointer
character*80 filename
integer*4 lineWidth, tileHeight
......
! get the image pointer
call getImagePointer(imagePointer)
! initialize image object associated with pointer
call initImage(imagePointer,filename,'read','l','complex',lineWidth,tileHeight)
```

Restructuring of I/O in Fortran

Set or get data: old way

```
program  OldWay
complex*8,dimension(:),allocatable:: data
integer*4 lineNumber
......
! read or write  data directly from or to file
read(FD, rec = lineNumber) (data(k), k = 1, lineWidth)
(or)
write(FD, rec = lineNumber) (data(k), k = 1, lineWidth)
```

Restructuring of I/O in Fortran

Set or get data: new way

```
program  NewWay
complex*8,dimension(:),allocatable:: data
integer*4 lineNumber
......
! get one line of data by passing the associated image pointer
call getLine(imagePointer,data,lineNumber)
(or)
! set one line of data
call setLine(imagePointer,data,lineNumber)
```

Earth Science Technology Office

Image API is available outside the framework

DRIVER	PROGRAM
Fortran	Fortran
C or C++	Fortran
Python	Fortran (mroipac)
C or C++	C or C++
Fortran	C or C++ (not tested)
Python	C or C++ (mroipac)
Python	Python (not implemented, but easy)

DRIVER = who initializes the Image API
PROGRAM = who uses the Image API methods

Image API is Documented

Main Page | Namespaces | Classes | Files

Namespace List | Namespace Members

install::LineAccessorPy

install::LineAccessorPy Namespace Reference

Classes

class **ClassLineAccessor**
This Class provides a set of convinient method

Functions

def **initLineAccessor**
Initializes the accessor object.

def **createLineAccessorObject**
Creates a **LineAccessor** object.

def **createFile**
For a file object opened in write or writeread r

def **getMachineEndianness**
Returns the endianness of the machine runnin

def **finalizeLineAccessor**
Always call this function if a **LineAccessor** obj

def **changeBandScheme**
Changes the file format from BandSchemeIn to

def **convertFileEndianness**
Changes the file endiannes.

def **getTypeSize**
Returns the size of the data type "type".

def **getFileLength**
Provides the number of lines of the file associa

def **getFileWidth**
Provides the number of columns of the file ass

def **getLineAccessorPointer**
Provides pointer associated with the "C" access

def **printObjectInfo**
Prints a series of information related to the file

def **printAvailableDataTypesAndSizes**
Prints the available data types and their sizes.

def **__init__**
Constructor.

Variables

LineAccessorObj
Pointer to the C++ **LineAccessor::LineAccessor** object.

---

def install::LineAccessorPy::getLineAccessorPointer (self)

Provides pointer associated with the "C" accessor object.

Returns:
`int` pointer to the "C" LineAccessor object.

def install::LineAccessorPy::getMachineEndianness (self)

Returns the endianness of the machine running the code.

Does not require that initLineAccessor() be called before execution.

Returns:
`character` 'b' for big endian and 'l' for little endian.

def install::LineAccessorPy::getTypeSize (self, type)

Returns the size of the data type "type".

Parameters:
type data type.

Returns:
`int` size of type.

def install::LineAccessorPy::initLineAccessor (self, filename, filemode, endian, type, row, col)

Initializes the accessor object.

Parameters:

filename	`string` name of the file to be accessed.
filemode	`string` access mode of the file.
endian	`character` endiannes of the data stored in the file. Values are 'b' or 'B' for big endian and 'l' or 'L' for little endian.
type	file data type.
row	`int` number of rows of the buffer tile. Set it to one if no tiling is desired.
col	`int` number of columns of the buffer tile. It must be equal to the number of columns of the associated file.

See also:
printAvailableDataTypesAndSizes().

Summary and Outlook

- InSAR Scientific Computing Environment is being designed according to (our interpretation of) community request for a more modular modern InSAR framework.
  - Requirements are responsive to community workshop recommendations
  - Ability to perform simple operations through complex scripts

- Community desire to have open software is a goal not without obstacles.
  - Security concerns and misconceptions exist with regard to spaceborne SAR data and processing capability
  - InSAR SCE is breaking ground in pushing for open distribution from an FFRDC
  - Some form of licensing scheme is inevitable (GPL, Apache, etc.)

- We are hoping to release beta code for core framework elements by Summer 2010 for the community to test and contribute to.