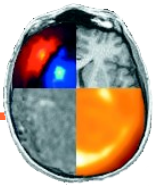
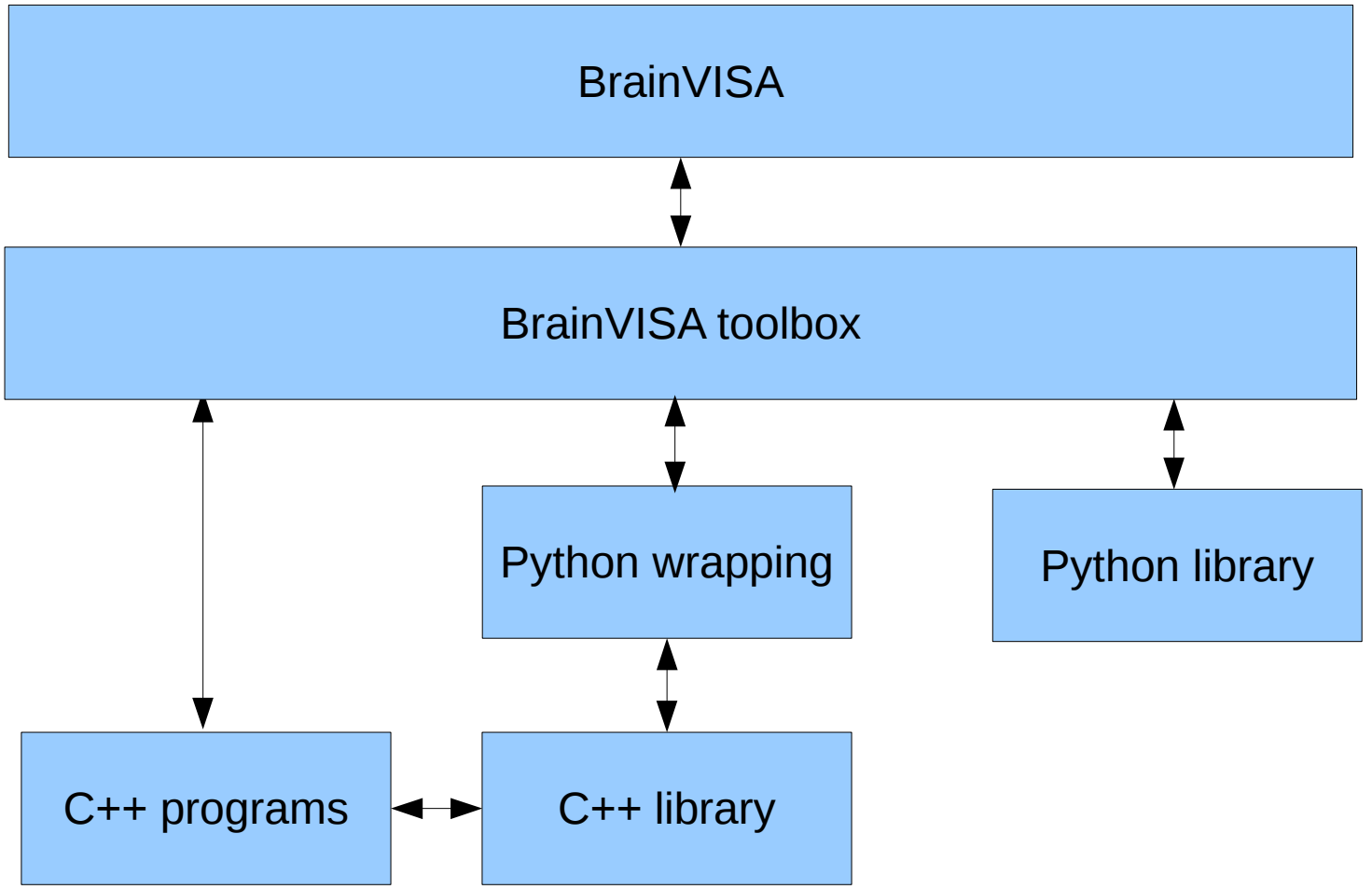
Programming within BrainVISA project

What is the BrainVISA project ?

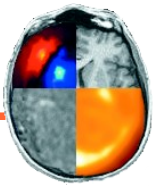


- Complete development environement
- RedMine based forge
- Integration of various programming languages
- Multiplatform programming
- Documentation management
- Packaging and distribution

Typical structure of a BrainVISA project



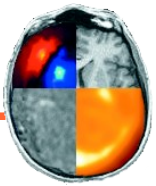
General plan



- Part I – Introduction
- Part II – Programming with BrainVISA
- Part III – Programming with Anatomist
- Part IV – Programming with Aims in Python
- Part V – A complete example

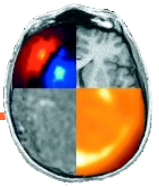
Part I - Introduction

Part I – Introduction



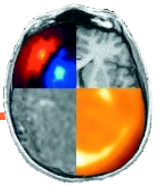
- User environements
- How to start with Python ?
- Useful modules
- Exercises

User environements



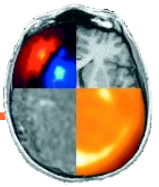
- Shell
  - Mainly used to launch command lines
- Python
  - To be used only if IPython is not available
- **IPython**
  - Should be the main developement environement
- BrainVISA
  - IPython plus BrainVISA specific extensions
- Text editor
  - Linux: kate or gedit

How to start with Python ?



- Tutorial
  - http://docs.python.org/tutorial/
- Introduction to Python for Science
  - http://gael-varoquaux.info/python4science.pdf
- Some videos
  - http://www.archive.org/search.php?query=SciPy%202009%20tutorial
- In french
  - http://dakarlug.org/pat/scientifique/html/index.html

Useful modules



- Standard Python modules (not included by default)
  - SciPy, NumPy
  - Matplotlib
  - PyQt
- BrainVISA project modules
  - soma, aims
  - anatomist
  - brainvisa

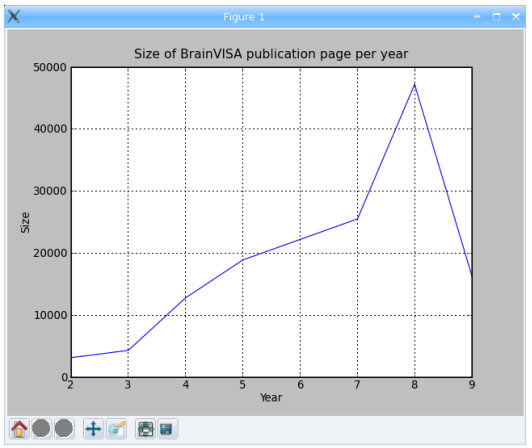
Exercises

- PBV_1: Count the number of occurrences of "BrainVISA" in web page http://brainvisa.info

```
import urllib
occurences = 0
file = urllib.urlopen( 'http://brainvisa.info/' )
for line in file:
  occurences = occurences + line.count( 'brainvisa' )
print occurences
```

- PBV_2: Show a diagram representing the size of BrainVISA bibliography pages between 2002 and 2009

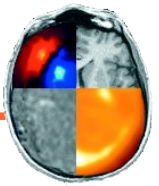
```
import urllib
from pylab import *
url = 'http://brainvisa.info/biblio/en/Year/200%d.html'
years = range( 2, 10 )
values = [ len( urllib.urlopen( url % year ).read() ) for year in years ]
plot( years, values, linewidth=1.0)
ylabel('Size')
xlabel('Year')
title('Size of BrainVISA publication page per year')
grid(True)
show()
```



Part II – Programming with BrainVISA



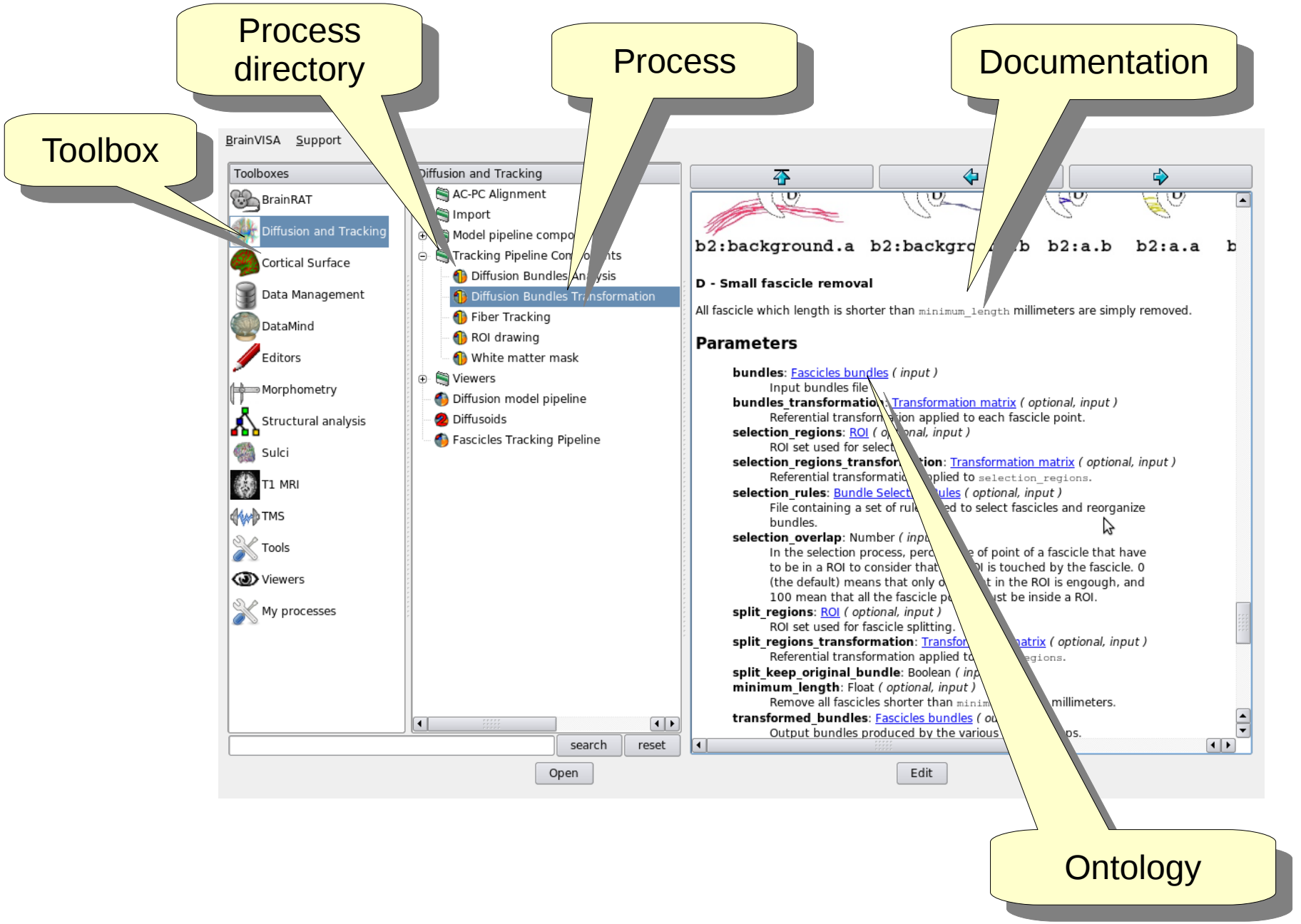
Part II – Programming with BrainVISA



- II.1 – Processes
- II.2 – Databases and Ontologies
- II.3 – Toolboxes

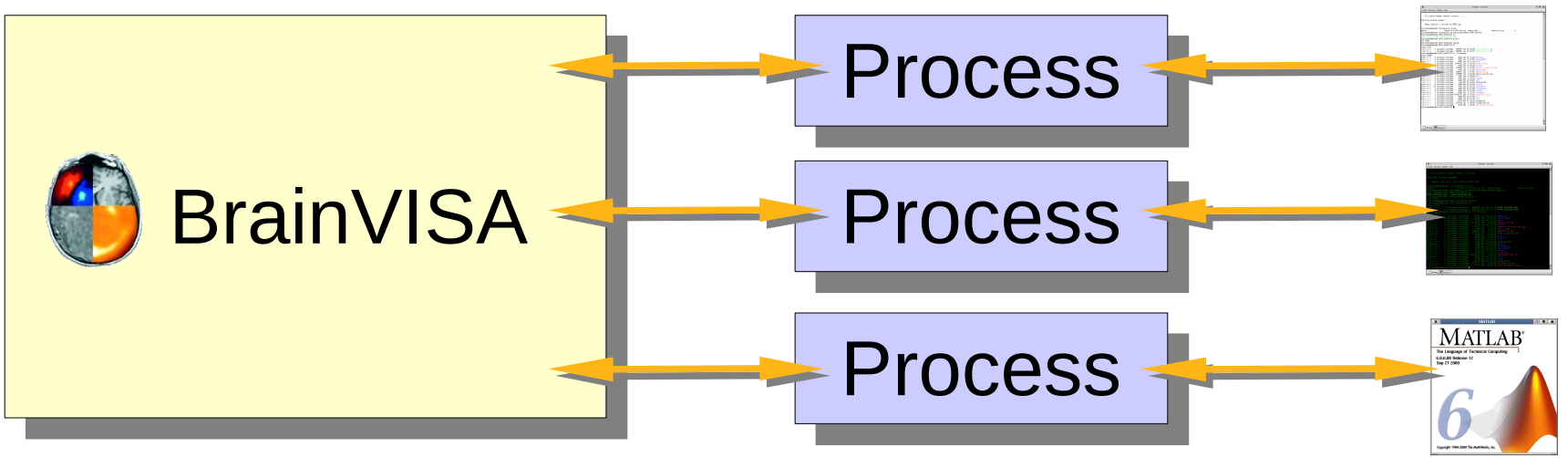
II.1 – Processes

BrainVISA main concepts



What is a process ?

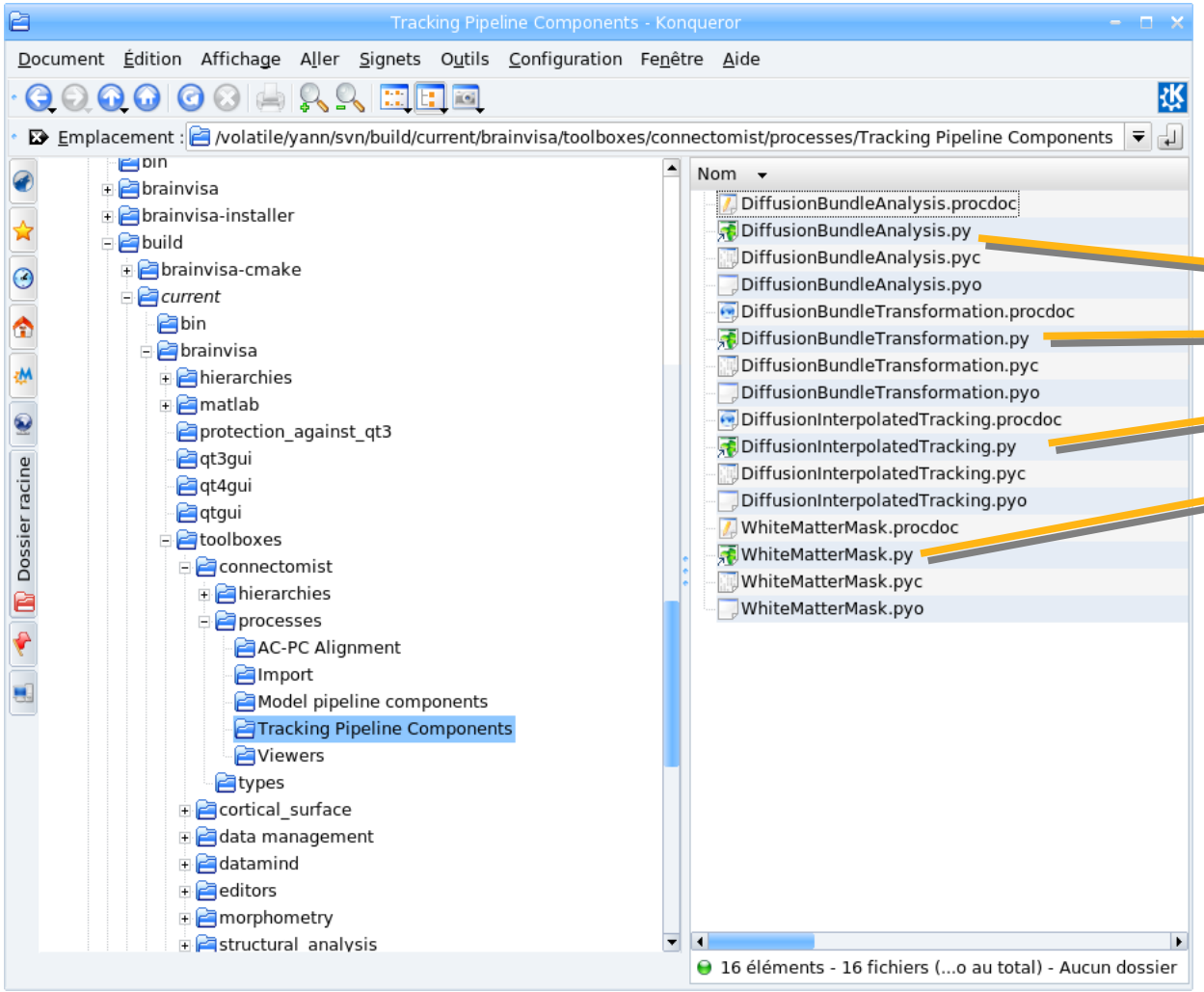
- A process is an interface between BrainVISA and another software

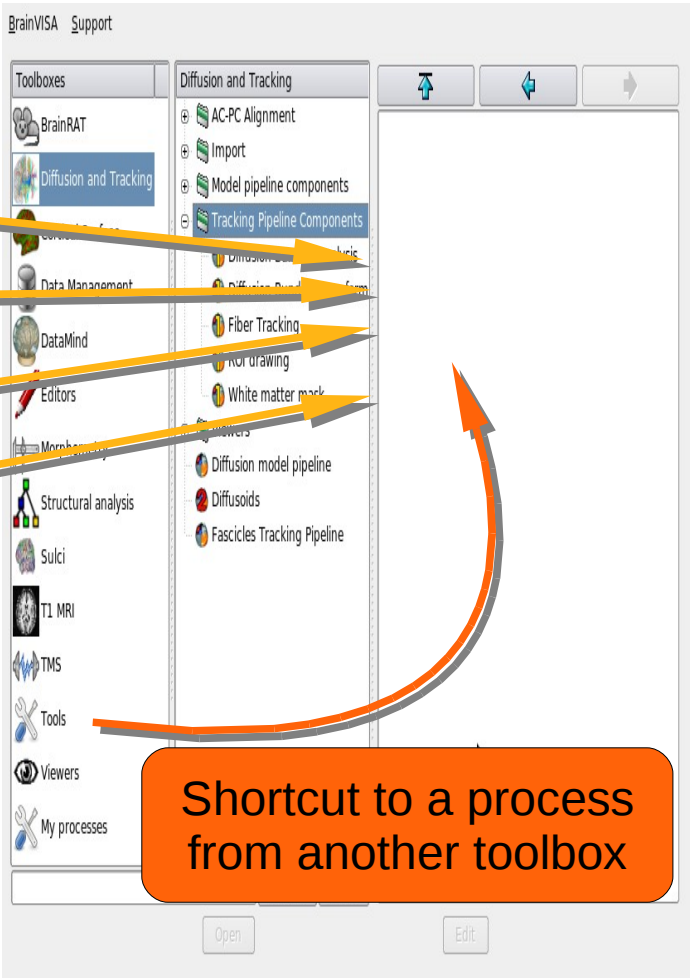


- Reusable component. For instance, a process can call other processes.
- Python file containing one main function

How to create a new process ?

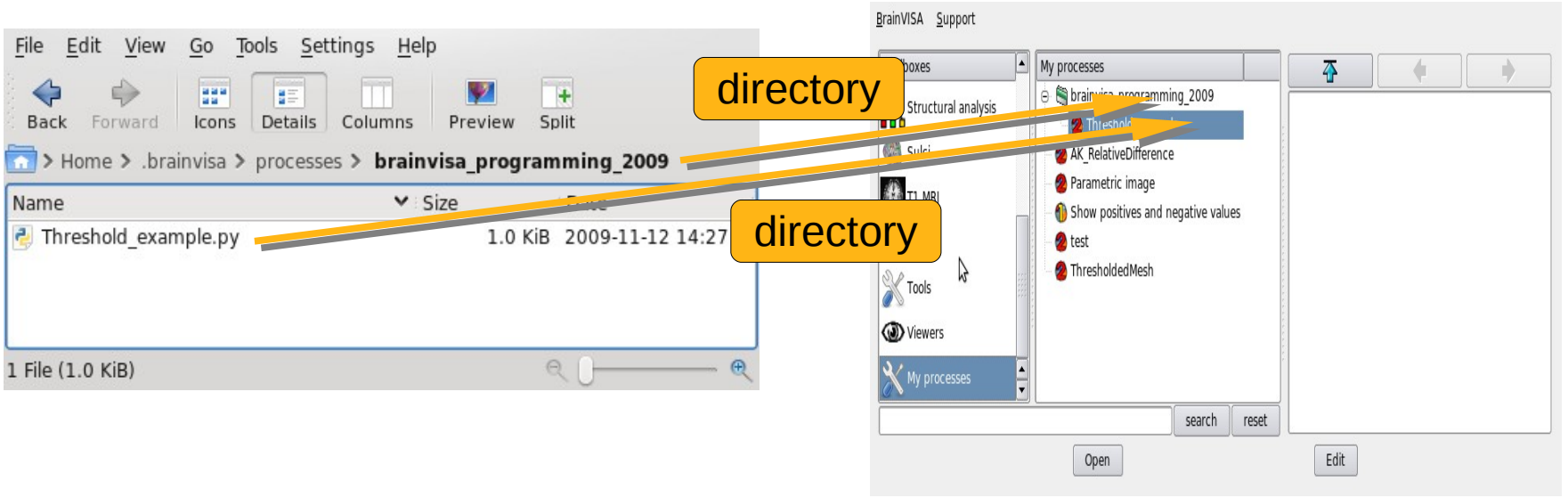
Create a file with **.py** extension in a toolbox **processes** directory





Special toolbox: My processes

- Located in user directory
  - Linux & Mac: $HOME/.brainvisa
  - Windows XP: C:\Documents and Settings\<login>\.brainvisa
  - Windows VISTA: C:\Users\<login>\.brainvisa
- Behave like a regular toolbox



Structure of a BrainVISA process

```
# -*- coding: utf-8 -*-
from neuroProcesses import *
```

header

```
signature = Signature(
  'image_input', ReadDiskItem(
    '4D Volume',
    [ 'NIFTI-1 image', 'SPM image', 'DICOM image', 'GIS image' ] ),
  'image_output', WriteDiskItem(
    '4D Volume',
    'Aims writable volume formats' ),
  'mode', Choice ( ( 'less than', 'lt' ),
    ( 'less or equal', 'le' ),
    ( 'greater than', 'gt' ),
    ( 'greater or equal', 'ge' ),
    ( 'equal', 'eq' ),
    ( 'different', 'di' ),
    ( 'between', 'be' ),
    ( 'outside', 'ou' ) ),
  'threshold1', Float(),
  'threshold2', Float(),
  'binary', Boolean(),
)
```

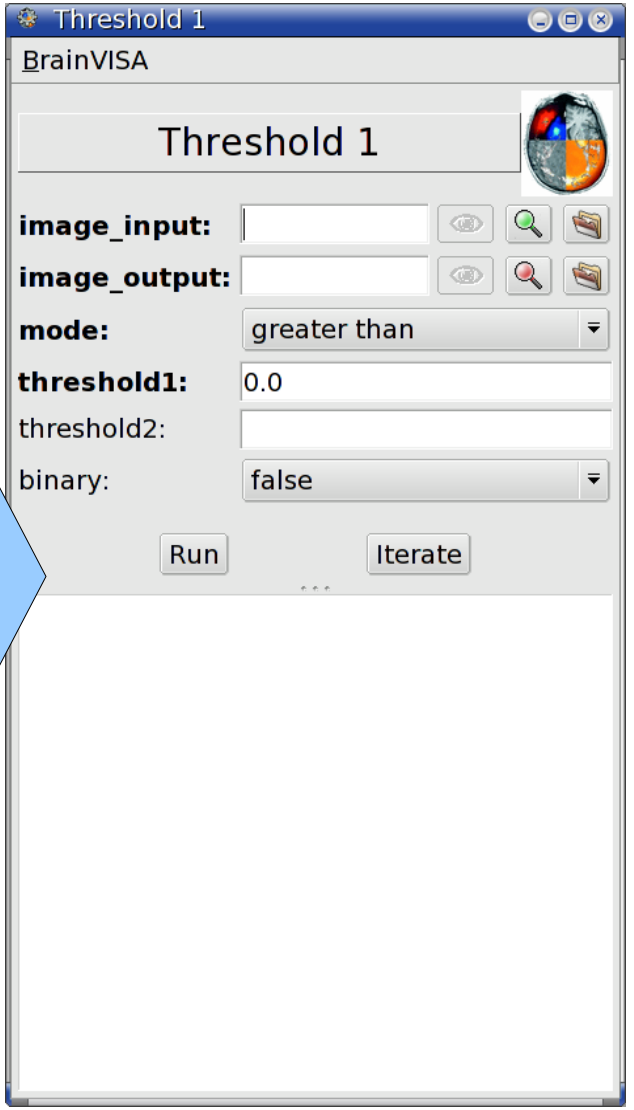
signature

```
def initialization( self ):
  self.setOptional( 'threshold2', 'binary' )
  self.binary = 0
  self.threshold1=0
  self.mode='gt'
```

initialization

```
def execution( self, context ):
  command = [ 'AimsThreshold',
    '-i', self.image_input,
    '-o', self.image_output,
    '-m', self.mode,
    '-t', self.threshold1 ]
  if self.threshold2:
    command += [ '-u', self.threshold2]
  if self.binary:
    command += [ '-b']
  context.system( *command )
```

body



Header of a BrainVISA process

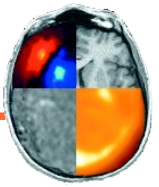
Example of a signature for a thresholding process

```
# -*- coding: utf-8 -*-

from neuroProcesses import *

name = 'Human readable error'

userLevel = 0

roles = ( 'viewer', )

def validation( self ):
   try:
      import my_package.my_module
   except:
      raise ValidationError('my_module is not installed')
```

0 = Visible to everybody
1 = Invisible to basic users
2 = Invisible to basic and advanced users

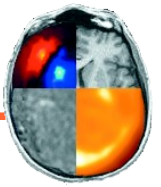
Possible roles:
viewer
editor
converter
importer

Signature of a BrainVISA process (1/2)



- Defines the parameters of a process
- signature = Signature(parameter_list)
  - parameter_list ← name, type, name, type, …
  - Parameter types:
    - String()
    - Number(), Float(), Integer()
    - Boolean()
    - Choice(<value>, (<label>, <value>), …)
    - ReadDiskItem(type, formats)
    - WriteDiskItem(type, formats)

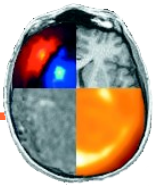
Signature of a BrainVISA process (2/2)



Example of signature for a thresholding process

```
signature = Signature(
  'image_input', ReadDiskItem(
    '4D Volume',
    [ 'NIFTI-1 image', 'SPM image', 'DICOM image', 'GIS image' ] ),
  'image_output', WriteDiskItem(
    '4D Volume',
    'Aims writable volume formats' ),
  'mode', Choice ( ( 'less than', 'lt' ),
    ( 'less or equal', 'le' ),
    ( 'greater than', 'gt' ),
    ( 'greater or equal', 'ge' ),
    ( 'equal', 'eq' ),
    ( 'different', 'di' ),
    ( 'between', 'be' ),
    ( 'outside', 'ou' ) ),
  'threshold1', Float(),
  'threshold2', Float(),
  'binary', Boolean(),
)
```

BrainVISA process initialization



Initialization example

```
def initialization( self ):
  self.binary = 0
  self.threshold1=0
  self.mode='gt'
  self.setOptional( 'threshold2', 'binary' )
  self.linkParameters( 'threshold2', 'threshold1' )



  eNode = SerialExecutionNode( self.name,
    parameterized=self )
  eNode.addChild( 'correction',
                  ProcessExecutionNode( 'DiffusionEPICorrection',
                                        optional=True, selected=True ) )
  eNode.addChild( 'create_mask',
                  ProcessExecutionNode( 'DiffusionT2BrainMask',
                                        optional=True, selected=True ) )
```

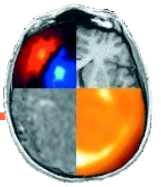
Parameters initialization

Optional parameters

Linked parameters

Pipeline creation

The real work : the process body



Python function

```
def execution( self, context ):
  command = [ 'AimsThreshold',
    '-i', self.image_input,
    '-o', self.image_output,
    '-m', self.mode,
    '-t', self.threshold1 ]
  if self.threshold2:
    command += [ '-u', self.threshold2]
  if self.binary:
    command += [ '-b']
  context.system( *command )
```

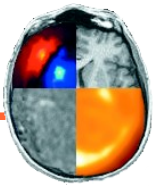
self : get parameters values

Execution context:

Manage interactions with system and user in a controlled and context dependent way

- write, warning, error : prints a message.
- log : writes a message in the BrainVISA log file.
- ask, dialog : asks a question to the user.
- temporary : creates a temporary file.
- system: call a system command.
- runProcess : runs a process.
- matlab : calls a Matlab command.

Exercises



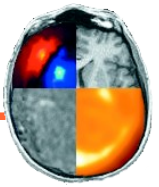
- PBV_3: create a process taking an **image** and a **threshold** and writing a **binary image** containing all voxels **greater than** threshold.

- PBV_4: create a process that takes a **binary image** and create a **3D mesh** from this image.

- PBV_5: create a process **combining PBV_3 and PBV_4**. Input is image and threshold, output is mesh. Thresholded image is only used internally in a temporary file.

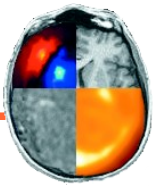
- PBV_6: create a process that **display** an **image** and a **mesh** in the same Anatomist window

Advanced process programming



- Pipeline: combination of existing processes
- Customized parameter links
- Dynamic signature
- Customization of graphical interface

Exercises

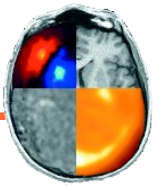


- PBV_7: create a **pipeline** process **chaining PBV_3 and PBV_4**.

II.2 – Databases and Ontologies



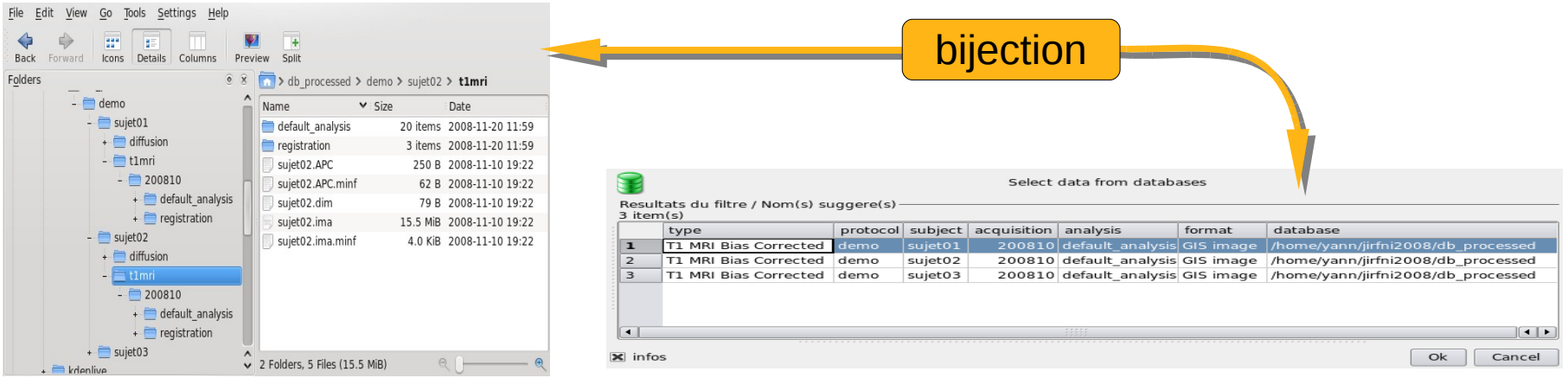
Why using BrainVISA databases ?



- Define a shared data organization
- Reuse data from people who left the lab
- Share data with people from other labs
- Make links between data
  - Find the head mesh corresponding to this MRI scan
- Automation of data processing

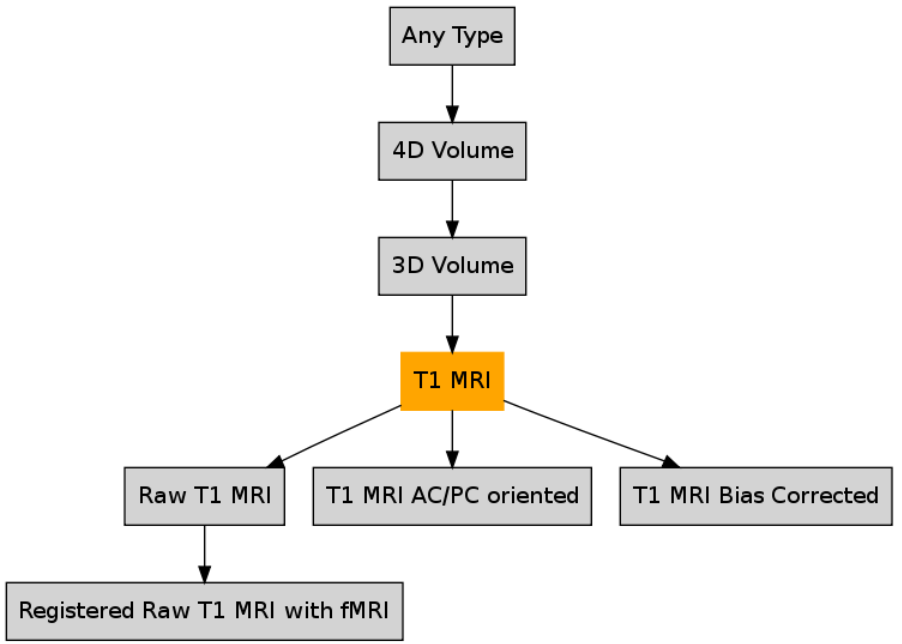
What is a BrainVISA database ?

- What is a BrainVISA database ?
  - A **directory** containing **data files** that are organized in a **hierarchy** that follows an **ontology**
  - A **relational database** built from an **ontology** and allowing to make efficient **selection** requests on **data files**.
- Two ways of seeing the same data connected together by a common ontology



What is a BrainVISA ontology ?

Hierarchy of types



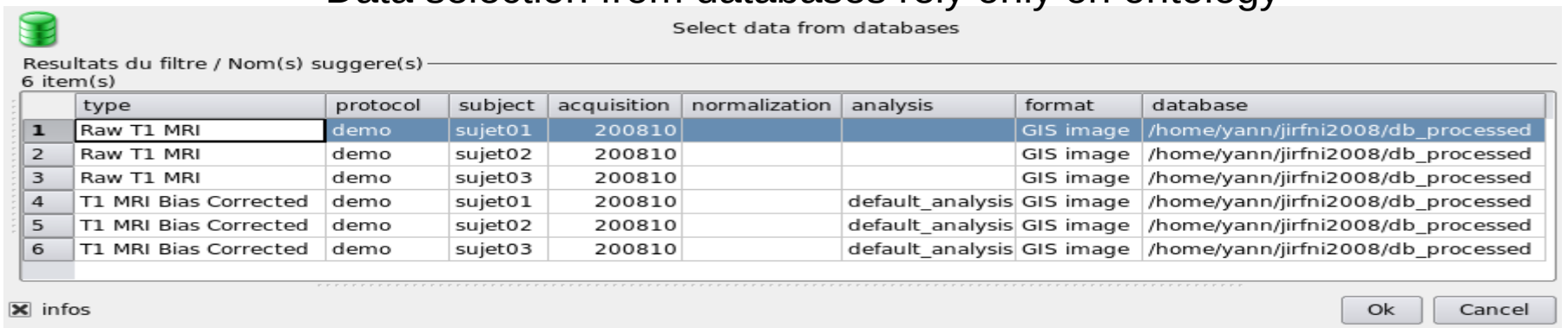
Attributes

Raw T1 MRI
- protocol
- subject
- acquisition
- normalization

T1 MRI Bias Corrected:
- protocol
- subject
- acquisition
- analysis

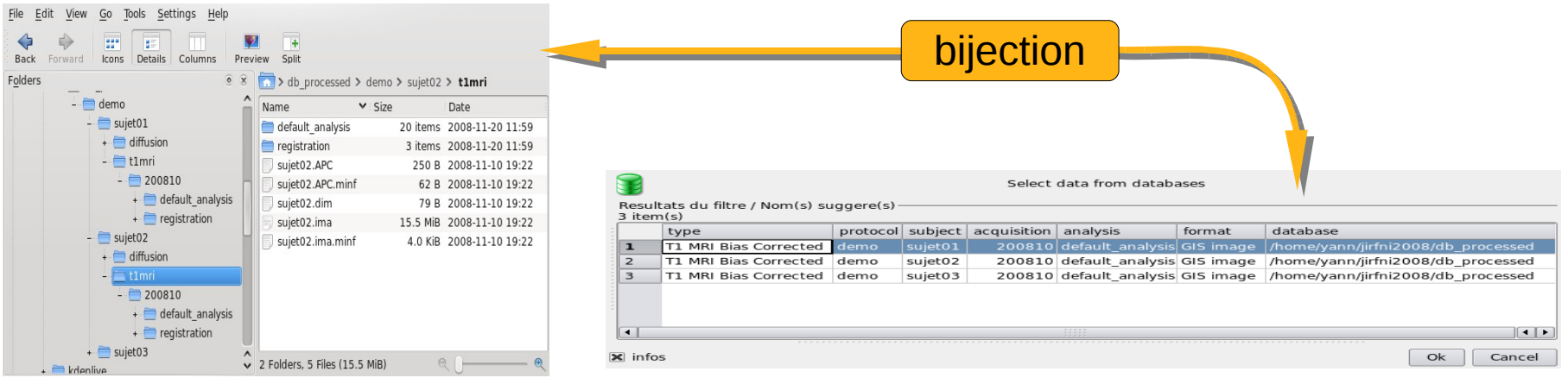
T1 MRI:
- protocol
- subject
- acquisition
- normalization
- analysis

Data selection from databases rely only on ontology



	type	protocol	subject	acquisition	normalization	analysis	format	database
1	Raw T1 MRI	demo	sujet01	200810			GIS image	/home/yann/jirfni2008/db_processed
2	Raw T1 MRI	demo	sujet02	200810			GIS image	/home/yann/jirfni2008/db_processed
3	Raw T1 MRI	demo	sujet03	200810			GIS image	/home/yann/jirfni2008/db_processed
4	T1 MRI Bias Corrected	demo	sujet01	200810		default_analysis	GIS image	/home/yann/jirfni2008/db_processed
5	T1 MRI Bias Corrected	demo	sujet02	200810		default_analysis	GIS image	/home/yann/jirfni2008/db_processed
6	T1 MRI Bias Corrected	demo	sujet03	200810		default_analysis	GIS image	/home/yann/jirfni2008/db_processed

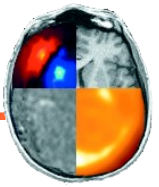
BrainVISA database hierarchy



Raw T1 MRI:
{protocol}/{subject}/t1mri/{acquisition}/<subject>
{protocol}/{subject}/t1mri/{acquisition}/normalized_{normalization}_<subject>

T1 MRI Bias Corrected:
{protocol}/{subject}/t1mri/{acquisition}/{analysis}/nobias_<subject>

Customization of a BrainVISA database



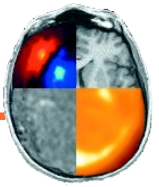
Create types in <toolbox>/types/*.py

```
include( 'builtin' )
include( 'anatomy' )

FileType( 'T1 MRI Bias Corrected', 'T1 MRI' )

Format( 'BrainVISA Gyri Model', "f|*.gyr" )
FileType( 'Gyri Model', 'Any Type', BrainVISA Gyri Model' )
```

Add hierarchy rules in <toolbox>/hierarchies/brainvisa-3.1.0/*.py

```
include( 'base' )

insert( {protocol}/{subject}/t1mri/{acquisition}/{analysis}',
  'nobias_<subject>', SetType( 'T1 MRI Bias Corrected' ),
)
```

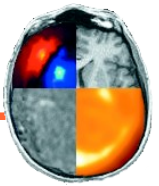
Exercises



- PBV_8: Create a **new type**: "Mesh from Threshold"
- PBV_9: Create a **new hierarchy entry** for "Mesh from Threshold" with the same key attributes as "Fractional Anisotropy"
- PBV_10: Create a **process** that create a mesh of type "**Mesh from Threhold**" from a thresholded "Fractional Anisotropy" image. Include a **link** between input image and output mesh.
- PBV_11: Create a **viewer** for "Mesh from threshold" that display the mesh with the corresponding FA image.

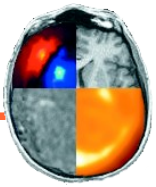
II.3 – Toolboxes

What is a BrainVISA toolbox ?



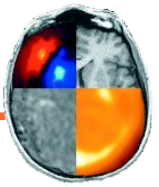
- An set of BrainVISA extensions
- Processes
- Ontology
- Documentation

Why creating a toolbox ?

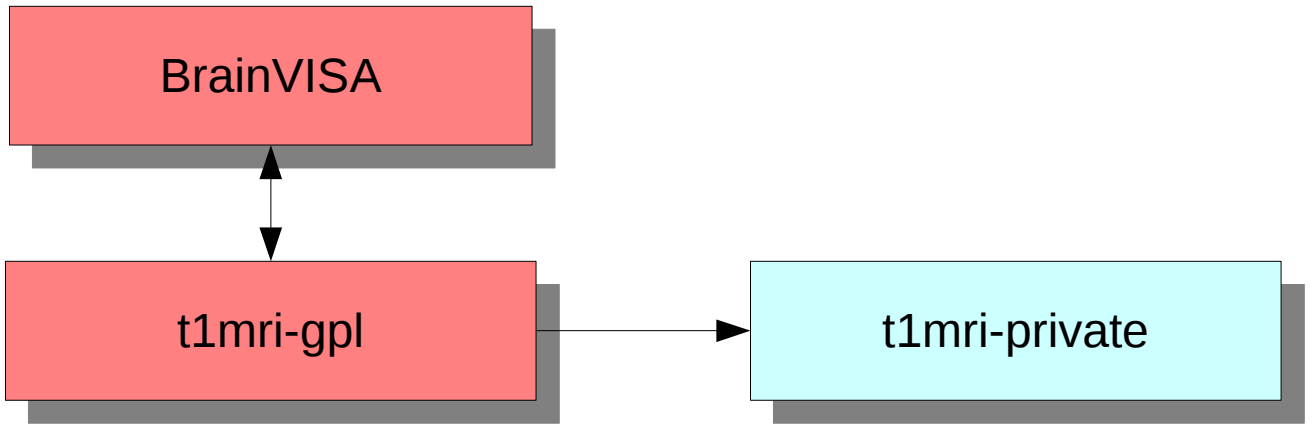


- Organize processes according to a topic
- Distribute BrainVISA extensions

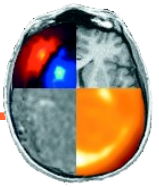
Licensing of a BrainVISA toolbox



- BrainVISA is in CeCill v2 (i.e. GPL)
- Therefore a BrainVISA toolbox must be in GPL
- However, processing libraries can have any licence as long as they do not rely on BrainVISA and can be distributed separately
- Example: T1 MRI Segementation Toolbox

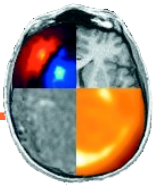


Toolbox creation



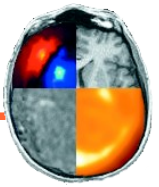
- Create a new **<name>** directory in **toolboxes** directory
- Put processes in **processes** directory
- Put ontology extension in **types** and **hierarchies** directories
- Create a configuration file: **<name>.py**
  - userName = 'User will see this name'
  - icon = file path of the icon that will reprensent the toolbox in graphical interface. Optional, there is a default icon.
  - description : tooltip for the toolbox (default is the name of the toolbox)

Creating links between toolboxes



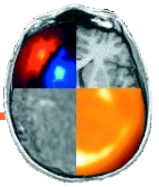
- The same process can appear in several toolboxes
- Links are stored in a minf file: **<name>.minf**
- Example :
  - Diffusion importation processes from Diffusion & tracking toolbox
  - These processes are also in Data management toolbox
- BrainVISA interface can be used to create this file

Packaging and distribution of a toolbox



- Structure of BrainVISA package directory:
  - bin : executables
  - brainvisa/toolboxes : BrainVISA toolboxes
  - include : C/C++ headers
  - lib : dynamic libraries
  - python : Python libraries
  - share : shared data and documentation
- Create an archive that adds files to the existing structure

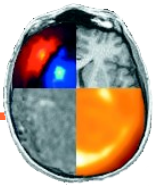
Exercises



- PBV_12: Create a package containing BrainVISA extensions corresponding to exercises PBV_8, PBV_9, PBV_10 and PBV_11

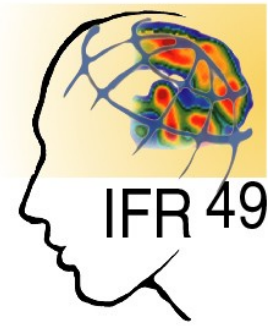
Part IIII – Programming with Anatomist

Anatomist tutorial – programming part



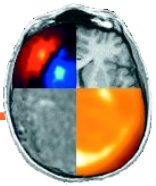
Tutorial about Anatomist python API:

http://brainvisa.info/doc/anatomist/ana_training/en/html/ch08.html



Part IV – Programming with Aims in Python

AIMS tutorial – programming part

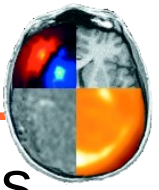


Tutorial about AIMS python API:

http://brainvisa.info/doc/aimsdata/aims_training/en/html/ch10.html

Part V – A complete example

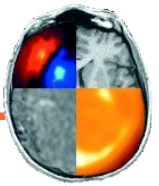
Exercises



- Part 1: Create a BrainVISA process to compute the coordinates of the maximum using aims

```
from neuroProcesses import *
import neuroConfig
from brainvisa import anatomist
import numpy

signature = Signature(
  'image', ReadDiskItem( '4D Volume', 'Aims readable volume formats' ),
)

def execution( self, context ):

  # Read input image
  aims_image = aims.read( self.image.fullPath() )

  # Compute coordinate of maximum voxel with Python
  aims_max_coordinate = ( 0, 0, 0, 0 )
  maximum = image.at( 0, 0, 0, 0 )
  for t in xrange( image.getSizeT() ):
    for z in xrange( image.getSizeZ() ):
      for y in xrange( image.getSizeY() ):
        for x in xrange( image.getSizeX() ):
          v = aims_image.at( x, y, z, t )
          if v > maximum:
            maximum = v
            aims_max_coordinate = ( x, y, z, t )
  context.write( 'Aims says max is', maximum, 'at', aims_max_coordinate )
```

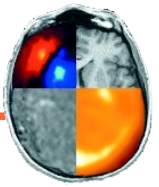
Exercises



- Part 2: Use numpy to process maximum coordinates

```
from neuroProcesses import *
import neuroConfig
from brainvisa import anatomist
import numpy

signature = Signature(
  'image', ReadDiskItem( '4D Volume', 'Aims readable volume formats' ),
)

def execution( self, context ):

  # Read input image
  aims_image = aims.read( self.image.fullPath() )

  numpy_matrix = numpy.array(aims_image, copy = False )
  numpy_max_coordinate = numpy.unravel_index( numpy_matrix.argmax(), numpy_matrix.shape )
  context.write( 'Numpy says max is', aims_image.at( *numpy_max_coordinate ), 'at', numpy_max_coordinate )
```

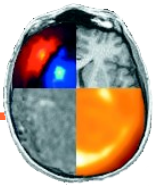
Exercises



- Part 3: Add read of a spherical mesh of size 100 located in standard BrainVISA directory and write it in a temporary file

```
aims_sphere = aims.read( os.path.join( neuroConfig.dataPath[0].directory, 'standardmeshes', 'ico100_7.mesh') )

# Change the size of the sphere and center it on image maximum
voxel_size = aims_image.header()[ 'voxel_size' ]
for vertex in aims_sphere.vertex():
  vertex *= self.sphere_size / 100.0
  vertex[ 0 ] += aims_max_coordinate[ 0 ] * voxel_size[ 0 ]
  vertex[ 1 ] += aims_max_coordinate[ 1 ] * voxel_size[ 1 ]
  vertex[ 2 ] += aims_max_coordinate[ 2 ] * voxel_size[ 2 ]

# Create a temporary file name for the modified sphere
diskitem_sphere = context.temporary( 'Mesh Mesh' )

# Write the modified sphere in temporary file
aims.write( aims_sphere, diskitem_sphere.fullPath() )
```

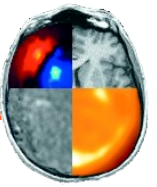
Exercises



- Part 4: Create a texture on the sphere that takes the value from an interpolator for each point of the sphere and save it in a temporary file

```
 # Create an interpolator on the image to be able to get a value for any millimeter coordinate
interpolator = aims.getLinearInterpolator( aims_image )

# Create a texture on the sphere that takes the value from the interpolator for each point of the sphere
vertices_count = aims_sphere.vertex().size()
aims_texture = aims.Texture_FLOAT( vertices_count )
for i in xrange( vertices_count ):
  aims_texture[ i ] = interpolator.value( *aims_sphere.vertex()[ i ] )

# Write the created texture to a temporary file
diskitem_texture = context.temporary( 'Texture' )
aims_time_texture = aims.TimeTexture_FLOAT()
aims_time_texture[ 0 ] = aims_texture
aims.write( aims_time_texture, diskitem_texture.fullPath() )
```

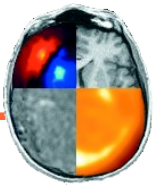
Exercises



- Part 5: Open anatomist and visualize the textured sphere at the maximum of the image

```
# Open Anatomist
a = anatomist.Anatomist()
# Load sphere in anatomist from temporary file
aSphere = a.loadObject( diskitem_sphere.fullPath() )
# Load sphere texture in anatomist from temporary file
aTexture = a.loadObject( diskitem_texture.fullPath() )
# Fusion sphere and texture to create a textured object
aTexturedSphere = a.fusionObjects( (aSphere, aTexture), 'FusionTexSurfMethod' )
# Load image in anatomist
aImage = a.loadObject( self.image.fullPath() )
# Create an Axial window
aWindow = a.createWindow( 'Axial' )
# Display textured sphere and image in window
aWindow.addObjects( (aTexturedSphere, aImage) )
# Move Anatomist cursor to the center of the voxel with maximum value
aWindow.moveLinkedCursor( ( aims_max_coordinate[ 0 ] * voxel_size[ 0 ],
                            aims_max_coordinate[ 1 ] * voxel_size[ 1 ],
                            aims_max_coordinate[ 2 ] * voxel_size[ 2 ] ) )
```

Exercises



- Part 6: finalize the process to not destroy python objects used

```
# Return objects that must not be destroyed immediately
return [ aTexturedSphere, aImage, aWindow ]
```