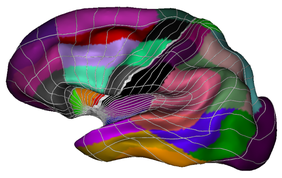
The BrainVISA project: a shared software development infrastructure for biomedical imaging research

http://brainvisa.info

Yann Cointepas[1], Dominique Geffroy[2], Nicolas Souedet[3], Isabelle Denghien[4], Denis Rivière[1]

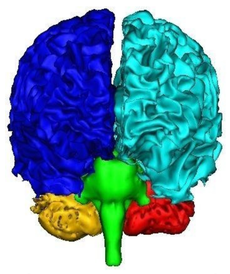
[1]CEA - IFR 49, Gif-sur-Yvette, France, [2]INSERM, GIF/YVETTE, FR, [3]CEA - IFR 49, Fontenay aux Roses, France, [4]INSERM - IFR 49, Gif-sur-Yvette, France



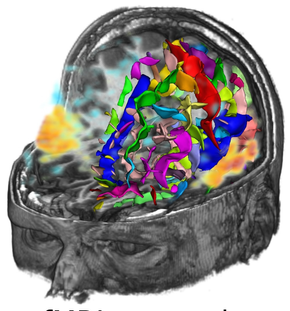
Surface-based analysis

BrainVISA is mainly known as a software containing **application toolboxes** (T1 MRI, sulcal identification and morphometry, cortical surface analysis, diffusion imaging and tractography, fMRI, nuclear imaging, EEG and MEG, TMS, histology and autoradiography, etc.). But BrainVISA is also a **multi-platform modular and customizable software development environment**.

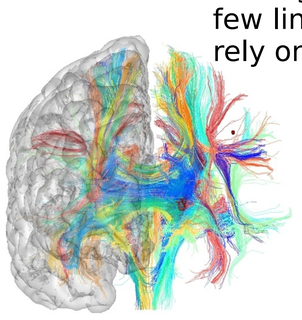
Writing an extension to BrainVISA can be as simple as writing a python script containing a few lines of code but it is also possible to build a complete customized application that can rely on software components already plugged in BrainVISA.



Brain segmentation

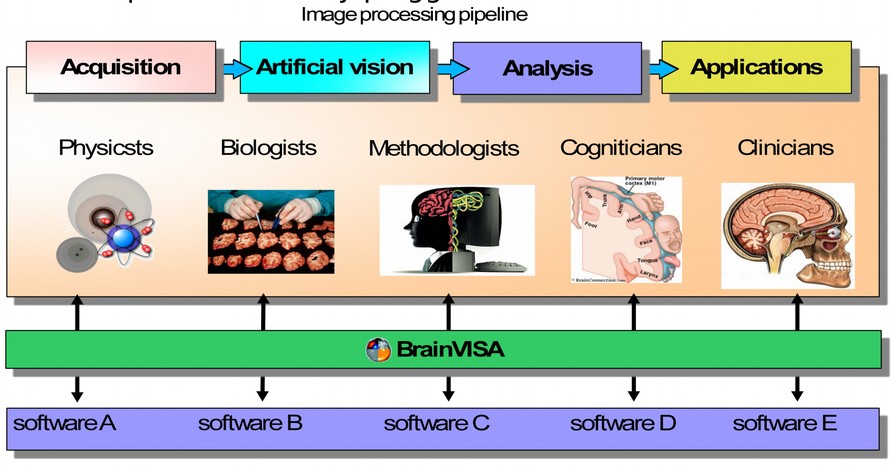


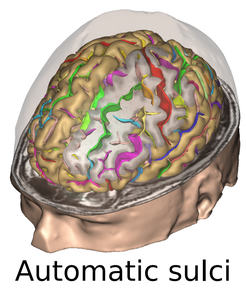
fMRI processing



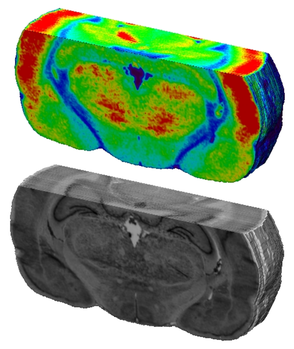
Fiber tracking

Image processing pipeline

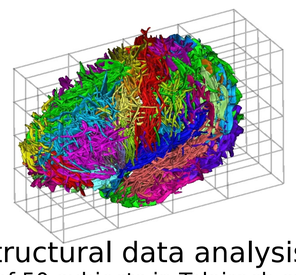




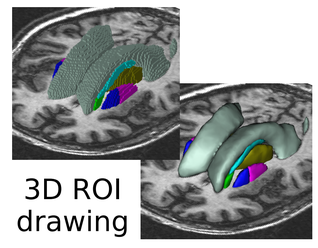
Automatic sulci identification



3D histology



Structural data analysis
Sulci of 50 subjects in Talairach space



3D ROI drawing

BrainVISA is a software platform that has been designed to be highly customizable. Modification can be done at all levels of BrainVISA infrastructure:

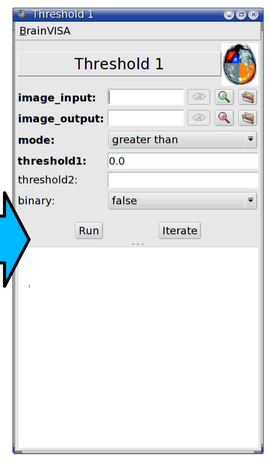
IMAGE PROCESSING: create image processing algorithms either by combining existing methods or by writing new ones.

```
# -*- 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



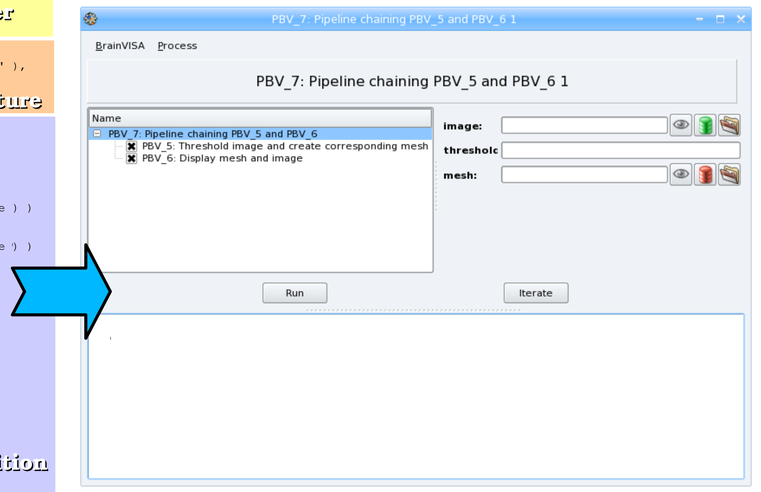
Extensibility: writing a process

SCRIPTING: automatize the execution of processes or pipelines according to specific needs or data organization.

```
# -*- coding: utf-8 -*-
from neuroProcesses import *
name = 'PBV_7: Pipeline chaining PBV_5 and PBV_6'
userLevel = 0
```
header

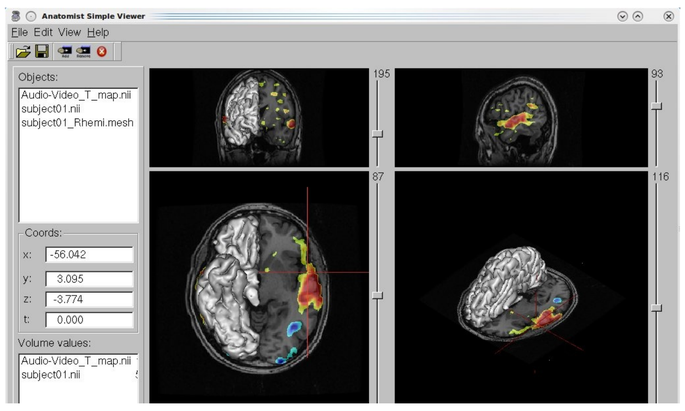
```
signature = Signature(
  'image', ReadDiskItem( '4D Volume', 'Aims readable volume formats' ),
  'threshold', Float(),
  'mesh', WriteDiskItem( 'Mesh', 'Aims mesh formats' ),
)
```
signature

```
def initialization( self ):
  # Create execution tree
  eNode = SerialExecutionNode( self.name,
    parameterized=self )
  eNode.addChild( 'create_mesh',
                  ProcessExecutionNode( 'PBV_5',
                                        optional=True, selected=True ) )
  eNode.addChild( 'display',
                  ProcessExecutionNode( 'PBV_6',
                                        optional=True, selected=True ) )

  # Link PBV_5 parameters
  eNode.addLink( 'image', 'create_mesh.image' )
  eNode.addLink( 'create_mesh.image', 'image' )

  eNode.addLink( 'mesh', 'create_mesh.mesh' )
  eNode.addLink( 'create_mesh.mesh', 'mesh' )

  eNode.addLink( 'threshold', 'threshold' )
  eNode.addLink( 'create_mesh.threshold', 'threshold' )

  # Link PBV_5 parameters
  eNode.addLink( 'image', 'display.image' )
  eNode.addLink( 'display.image', 'image' )

  eNode.addLink( 'mesh', 'display.mesh' )
  eNode.addLink( 'display.mesh', 'mesh' )

  # Attach the execution tree to the process
  self.setExecutionNode( eNode )
```
initialization: pipeline definition



Scripting: pipeline chaining 2 processes

VISUALIZATION: customize existing visualization tools or create new viewers.



AnaSimpleViewer: An example of customized application

A full application in about 700 lines of code. A simpler application fits in less than 300 lines: http://brainvisa.info/doc/pyanatomist-4.0/examples/anaevensimpleviewer.py

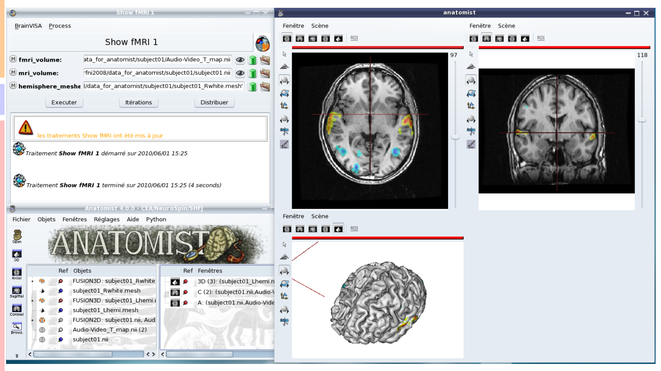
```
# -*- coding: utf-8 -*-

from neuroProcesses import *
from brainvisa import anatomist
import registration

name = 'Show fMRI'
userLevel = 0
roles = ( 'viewer', )
def validation():
    anatomist.validation()
```

```
signature = Signature(
    'fmri_volume', ReadDiskItem( 'fMRI', 'anatomist volume formats' ),
    'mri_volume', ReadDiskItem( 'T1 MRI', 'anatomist volume formats' ),
    'hemisphere_meshes', ListOf( ReadDiskItem( 'Hemisphere mesh',
        'anatomist mesh formats' ) ),
)
```

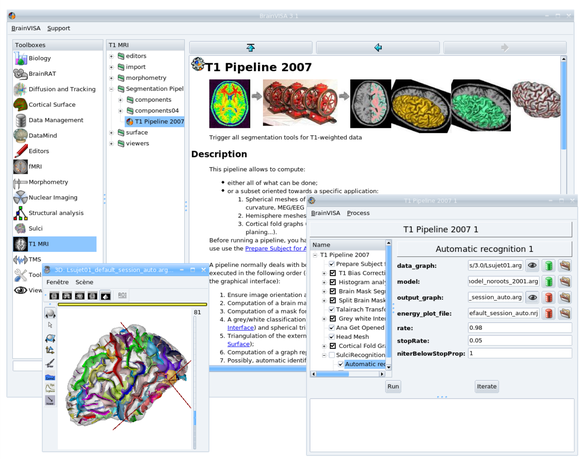
```
def initialization( self ):
    self.linkParameters( 'mri_volume', 'fmri_volume' )
```

```
def execution( self, context ):
    a = anatomist.Anatomist()
    palette = 'tvalues100-200-100'
    mri = a.loadObject( self.mri_volume )
    fmri = a.loadObject( self.fmri_volume, duplicate=True )
    fmri.setPalette( palette, maxVal = 1, minVal = 0 )
    fusion = a.fusionObjects( [mri, fmri], method = 'Fusion2DMethod' )
    a.execute("Fusion2DParams", object=fusion, mode = 'linear_on_defined',
        rate = 0.5, reorder_objects = [ mri, fmri ] )
    objects = [ mri, fmri, fusion ]
    meshes = []
    for meshfile in self.hemisphere_meshes:
      mesh = a.loadObject( meshfile )
      objects.append( mesh )
      fusion3d = a.fusionObjects( [mesh, fmri], method = 'Fusion3DMethod' )
      meshes.append( fusion3d )
      objects.append( fusion3d )
    a.execute( 'ApplyBuiltinReferential', objects=objects )

    block = a.createWindowsBlock()
    window1 = a.createWindow( 'Axial', block=block )
    window2 = a.createWindow( 'Coronal', block=block )
    windows = [ block, window1, window2 ]
    a.addObjects( windows=[ window1, window2 ], objects=[fusion] )
    if len( meshes ) != 0:
        window = a.createWindow( '3D', block=block )
        a.execute( 'Camera', windows=[ window ], zoom=1.2,
            view_quaternion=[ 0.404603, 0.143829, 0.316813, 0.845718 ] )
        windows.append( window )
        window.addObjects( meshes )
    # Keep a reference on objects and windows while the view is needed
    return [ objects, windows ]
```

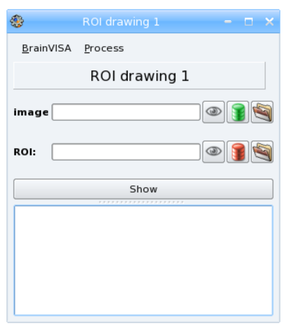
A viewer using Anatomist,
displaying merged volumes and textured meshes



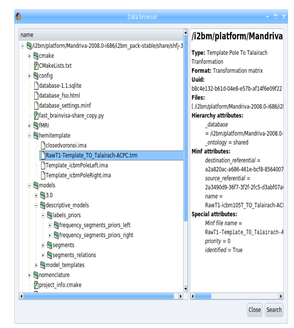
USER INTERFACE: customize a few elements of an existing interface or design your own graphical user interface.



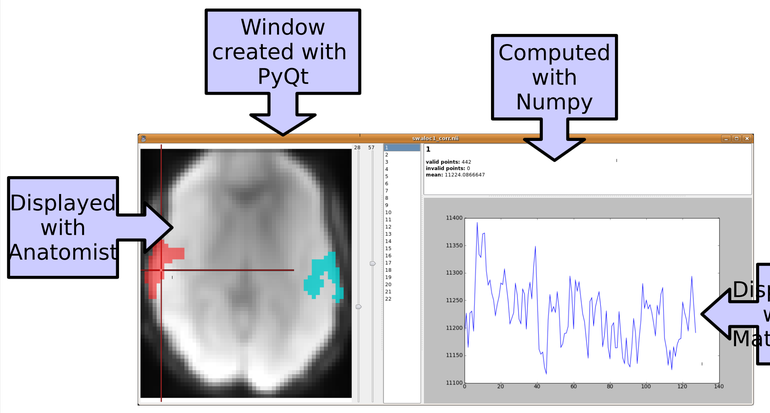
Main graphical user interface



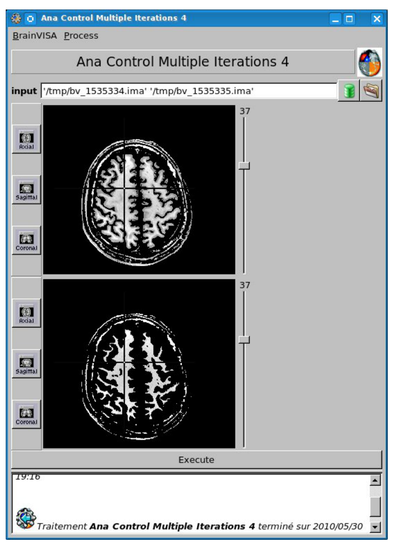
Light customization: only changing buttons



Database browser: Total redefinition of the GUI



Customization of graphical user interface



Customized interface with embedded anatomist views