Scene = .mrml file

(workspace state of .mrml)

Scene / fiducial

PHASE 1 : CORE MENTAL MODEL

1. Module vs extension

- module : tool (python package) → UI, logic, menus inside running Slicer instance
- extension : bundle of modules

2. MRML Nodes

- MRML node → data object living in the Slicer scene → Volume (CT, MRI, image), Model (mesh), Segmentation, Markups (fiducials, curves), My parameters
- nodes → stored in the scene, can be saved/loaded, support undo/redo, can be observed (events)

SLICER → UI + logic that manipulates MRML nodes

3. Parameter Node

- Param Node → special MRML node that stores your module's settings (selected input module, threshold value...)

used to persist module state, everything the module needs to remember to behave the same later

4. Module / Widget / Logic

config mold

A. Module class

- purpose : registration and metadata
- "tell Slicer I exist"

party on UI

B. Widget class

- purpose : UI
- "translate user actions into parameters"
- responsibilities → Buttons, Sliders, Node selectors, Connecting UI → param Node = R/W paramNode

{ Widget → Param Node → Logic → MRML Nodes }

opera

C. Logic Class

- purpose : brains of module
- "Pure computation + MRML manipulation"
- Responsibilities → pure algo, create / update nodes, observe changes, be UI-independent

PHASE 2 : MRML Deep Dive

0. Mental Model

- 3D Slicer = reactive scene graph of MRML Nodes. Everything you see → volumes, models, markups, transforms, segmentations, ... is JUST DATA in the MRML scene and UI reacts to changes in that data

1. the MRML Scene

- the MRML scene is → Container for all Nodes; Global, singleton like; responsible for → save/load, undo/redo, node lifetime, event dispatch
- When scene resets nodes are destroyed, widgets must reconnect

2. MRML Nodes

- A node is a { typed data obj; stored in the scene; identified by an ID }
- Each node has → Data (image, mesh, parts); → Display node (how it looks); → Events (what changed)

qt signal → msg emitted by GUI obj when sth happens to it (i.e: btn clicked)

5. observers : how slicer reacts

SLICER IS EVENT DRIVEN

- A node emits events when { Data changes; Display changes; References change } you observe these events
- observers are: call me when this node changes

  node.AddObserver(vtkCommand.ModifiedEvent, callback)
- Scene events (nodeAdded, nodeRemoved, etc)

- GUI widgets are views — WIDGET —— observes → SCENE
- MRML nodes are state — WIDGET —— observes → ParamNode
- UI observe MRML nodes via signals

- GUI → ParamNode : updateParameterNodeFromGui() (connect qt signals to it) { user acts; selects node; edits value }
- ParamNode → GUI : updateGuiFromParameterNode() (observer) { scene loads; logic modifies params }

FIDUCIAL → MRML Node stores list of 3D points in a scene (landmarks)

- vtk = visualization toolkit (library)

(.nii, .nii.gz, .hdr/.img) • .nii = file format stores brain imaging data

ROADMAP MAIN LEARNINGS

proj 1

proj 2

- we need an observer for selected node (if paramNode references a new inputNode)
- widget components don't know about the scene just because the parent does, we need top-level to signal each component's setMRMLScene slot

* BioFace3D Module 1: MATLAB passes files_to_process.txt to python scripts instead of .nii path

proj 3

- fiducial node: data object to store point-based annotations
- create fiducial → update paramNode to hold reference to it → updateGUIFromParamNode attaches observers to it to react to
  - point modified
  - point added
  - point removed

proj 4

- param Node emmagatzema tant
  - → fiducial node actual
  - → posicions inicials landmarks
- pel fiducial node, observers
  - → point added / point removed / point modified
  - → vtk modified event (fires when anything about the node changes)
    - toggle locked status in markups
    - rename fiducial node in Data

  handles node interaction outside of our module

integration plan

- Slicer comes with its own Python environment
- python version tied to Slicer version
- Approach: LEVERAGE Slicer existing environment

1 - Create a Custom Module

a. Modules → Developer tools → create Extension (SL - tutorials - Category tutorials)

b. Add Module to Extension (t- Apply threshold)

c. Fix bug wrong category

d. Add module to favorites

2. Python Interact, PyCharm. Debug & 3D Slicer pipeline

a. Functionality of template module (apply module to volume → new volume named volume)

b. Python interactor

c. Debug using PyCharm

d. Pipeline study
- → Module class
- → widget class
- → Logic class (process data

e. optimize template code

f. summary

3. Developer / Slicer Area, Neat Blanks, GUI Hello World

a. Developer Area vs Slicer Area

b. Neat blanks clean up

c. GUI edit

d. connect signal / slot

4. Code style, Method Groups & vs sequence viewer

a. Install IGT extensions

b. Code style

c. Rules to make module robust

d. Method groups

5. Lamina landmark labeling

- relació extensió - modul
- què és un node / param Node
- estructura codi mòdul (pipeline)
- neat blanks
- scene file
- robust : Node Reference & Explicit UI Update
  + parameter Node Update

SlicerMorph → designed for morphological analysis of biological specimens

ALPACA

- traslladar landmarks a altres models, veure comparació
- enllaç amb mòdul markups per a → visualitzar / → editar } landmarks