This presentation is violating the golden rules of good slides Design and aestethic.


PSL : Py-Scene-Language
A refreshing language for Sofa Scenes

A scene generated in Python (pyscn)

```
def createScene(root):
        snake = root.createNode("snake")

        ########### HEAD ###############
        head = snake.createNode("head")
        head.createObject("MechanicalObject")
        head.createObject("OglModel")

        ########### HEAD ###############
        head = root.createNode("body")
        head.createObject("MechanicalObject")
        head.createObject("OglModel")

        ########### PIED ###############
        pied = root.createNode("pied")

        ############ BASE ##############
        base = pied.createNode("base")
        base.createObject("MechanicalObject")
        base.createObject("OglModel")

        ########### TOP ###############
        top = pied.createNode("pied")
        top.createObject("MechanicalObject")
        top.createObject("OglModel")
```

I don’t use PyScn !

```
root):
root.createNode("snak

### HEAD ############
nake.createNode("head
ateObject("Mechanical
ateObject("OglModel")

### HEAD ############
oot.createNode("body"
ateObject("Mechanical
ateObject("OglModel")

### PIED ############
oot.createNode("pied"

############ BASE ############
base = pied.createNode("base"
icalObject")
el")

#####
ed")
calOb
l")
```

Problem statement :

1) Language aesthetic (visual properties)
2) Language functionnality

Ben Schneiderman
(PROGRAM INDENTATION AND COMPREHENSIBILITY)

Problem statement : XML

```
Node name="root">
        <Node name="snake">
                <Node name="head">
                        <MechanicalObject/>
                        <OglModel/>
                </Node>
        </Node>
        <Node name="pied">
                <Node name="base">
                        <MechanicalObject/>
                        <OglModel/>
                </Node>
                <Node name="top">
                        <MechanicalObject/>
                        <OglModel/>
                </Node>
        </Node>
/Node>
```

- Descriptive
  - A DSL
- Visual structure

Problem statement : XML

```
Node name="root">
        <Node name="snake">
                <Node name="head">
                        <MechanicalObject/>
                        <OglModel/>
                </Node>
        </Node>
        <Node name="pied">
                <Node name="base">
                        <MechanicalObject/>
                        <OglModel/>
                </Node>
                <Node name="top">
                        <MechanicalObject/>
                        <OglModel/>
                </Node>
        </Node>
/Node>
```

Problem statement : Python

```
def createScene(root):
        snake = root.createNode("snake")

        ############ HEAD ################
        head = snake.createNode("head")
        head.createObject("MechanicalObject")
        head.createObject("OglModel")

        ############ HEAD ################
        head = root.createNode("body")
        head.createObject("MechanicalObject")
        head.createObject("OglModel")

        ############ PIED ################
        pied = root.createNode("pied")

        ############ BASE ################
        base = pied.createNode("base")
        base.createObject("MechanicalObject")
        base.createObject("OglModel")

        ############ TOP ################
        top = pied.createNode("pied")
        top.createObject("MechanicalObject")
        top.createObject("OglModel")
```

- A general purpose Language
  - Procedural
- Not descriptive… you tell the system how to fabricate it

Problem statement : Python

```
def createScene(root):
    snake = root.createNode("snake")

    ############ HEAD ################
    head = snake.createNode("head")
    head.createObject("MechanicalObject")
    head.createObject("OglModel")

    ############ HEAD ################
    head = root.createNode("body")
    head.createObject("MechanicalObject")
    head.createObject("OglModel")

    ############ PIED ################
    pied = root.createNode("pied")

    ############ BASE ################
    base = pied.createNode("base")
    base.createObject("MechanicalObject")
    base.createObject("OglModel")

    ############ TOP ################
    top = pied.createNode("pied")
    top.createObject("MechanicalObject")
    top.createObject("OglModel")
```

- A general purpose Language
- Procedural
- Poor match to our domain

Visual structure designed for procedural
procedural element :
loop, condition, function

Problem statement : Python

```
def createScene(root):
    snake = root.createNode("snake")

    ########## HEAD ###############
    head = snake.createNode("head")
    head.createObject("MechanicalObject")
    head.createObject("OglModel")

    ########## HEAD ###############
    head = root.createNode("body")
    head.createObject("MechanicalObject")
    head.createObject("OglModel")

    ########## PIED ###############
    pied = root.createNode("pied")

    ########## BASE ###############
    base = pied.createNode("base")
    base.createObject("MechanicalObject")
    base.createObject("OglModel")

    ########## TOP ###############
    top = pied.createNode("pied")
    top.createObject("MechanicalObject")
    top.createObject("OglModel")
```

- A general purpose Language
- Procedural
- Poor match to our domain
- Overhead !! (683)

Problem statement : TheGood

```
ode("root"):
        Node("Snake"):
                Node("Head"):
                        MechanicalObject
                        OglModel
                Node("Body"):
                        MechanicalObject
                        OglModel
        Node("Pied"):
                Node("Base"):
                        MechanicalObject
                        OglModel
                Node("Support"):
                        MechanicalObject
                        OglModel
```

- A DSL
- Descriptive
- Visual Structure
- Small (240 c)

Problem statement : Execution !

```
ef createScene(root):
        for i in range(0, 10):
                root.addChild("Child_"+str(i))
```

Loading → code execution
Code execution → consume input to
To produce a scene as output.

Problem statement : Execution !

```
def createScene(root):
        for i in range(0, 10):
                root.addChild("Child_"+str(i))
```

The « loop » has been lost !

```
def createScene(root):
        root.addChild("Child_0")
        root.addChild("Child_1")
        root.addChild("Child_2")
        root.addChild("Child_3")
        root.addChild("Child_4")
        root.addChild("Child_5")
        root.addChild("Child_6")
        root.addChild("Child_7")
        root.addChild("Child_8")
        root.addChild("Child_9")
        ...
        root.addChild("Child_12354")
        root.addChild("Child_12355")
        root.addChild("Child_12356")
```

Problem statement :

1) Language aesthetic
    - Python is poora (bad choice for scenes)
    - Xml is better (but perfectible)
2) Language functionnality
    - Xml can be saved, Python cannot
    - Python is dynamic, XML need extensions


Problem stated.
What do we do now ?

Problem statement :

XML

PyScn

Problem stated.

What do we do now ?

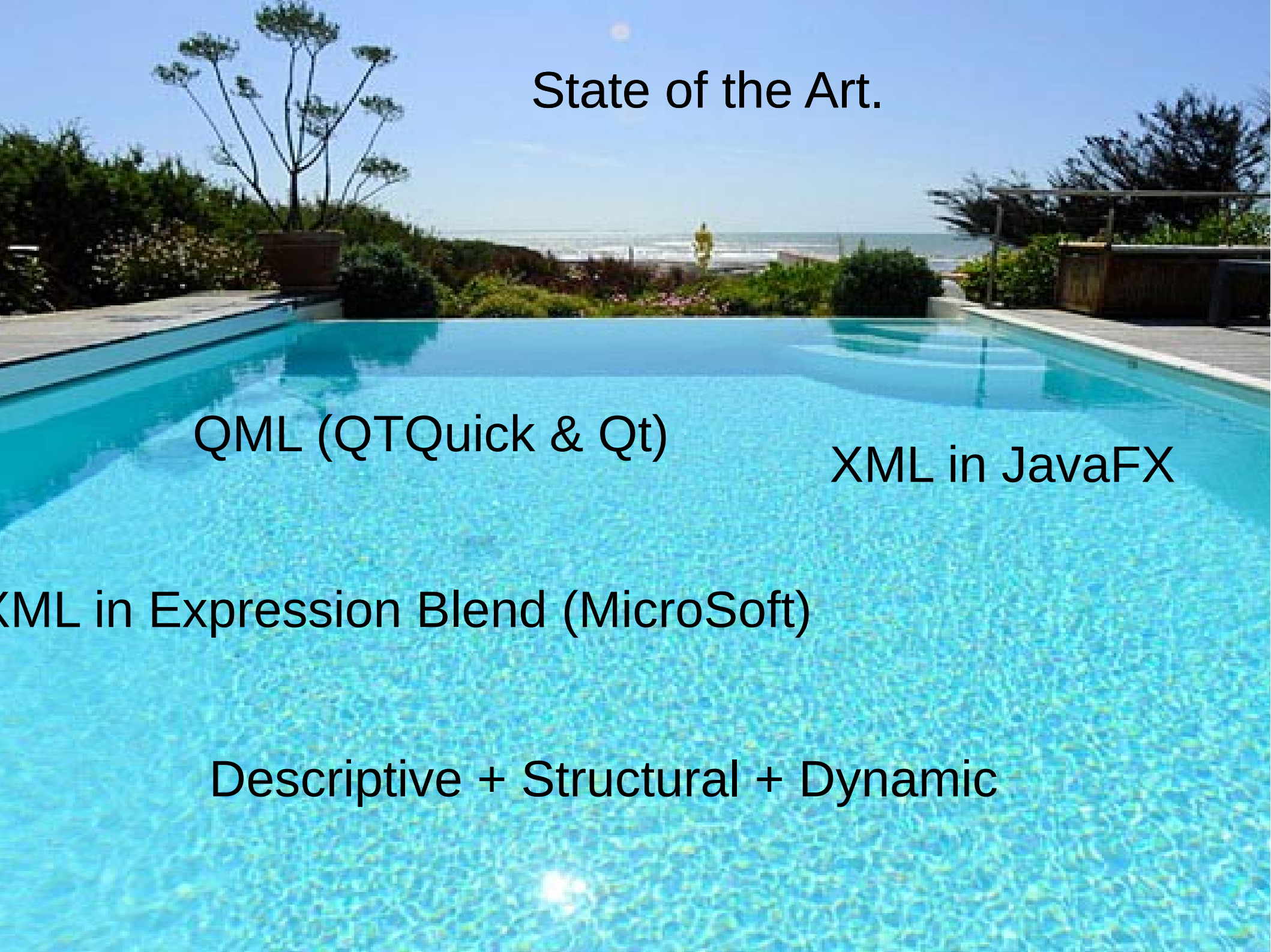

Language are tools.
Having the right tool, designed
for a problem, is fundamental to solve i
C:\


Make a new language


Problem stated.
Dynamic XML with Cheetha


Problem stated.
DSL In Python


PSL Proof of Concept


State of the Art.
QML (QTQuick & Qt)
XML in JavaFX
XML in Expression Blend (MicroSoft)
Descriptive + Structural + Dynamic

Features

Sofa object are first class not loop/procedure/if
Descriptive & Structural
Procedural & dynamic
And it can be Saved
Scenes templates
Scene libraries
Aliases
….

What's next

Dynamic (in GUI) template re-instanciation.
template improvement.
….


Join the Rebel Aliance in the github pool :
https://github.com/SofaDefrost/sofa/blob/add_prefab/applications/plugins/SofaSceneAssist/README.md
Have a nice summer.