PyTine – v 0.1

Andres Pazos

EMBL

Introduction

- Python
- Motivation
- What is PyTine?
- Evaluated Methods
- PyTine Capabilities
- Current Status
- PyTine API
- Examples

Couple of words about Python!

- Very used as a scripting language (also in the scientific community)…
- … but also a powerful programming language
- Dynamic Object Oriented
- Multi-platform
- Open Source
- Possible to compile and create executables
- Very well supported
- Graphical Support (PyQT and others)
- Multiple open libraries available

Our Motivation

1. High level scripting language for our TINE servers

2. Generic Interface for connecting TINE with a existing Python Framework (MxCube)

Our Scripting Requirements

- Easy to learn (for the developers and for the users)
- Easy to maintain
- Flexible (possible to refactor)
- Dynamic (doesn’t need variable declarations)
- With a defined syntax
- Well documented
- Possible to control the accessible functionality
- Separated of the device specific layer
- Command-line support
- Sequencer support (graphical?)
- Reliable
- User proof
- Multi-platform
- Open-source

What is PyTine?

- Interface to access TINE from the Python programming language

- Two modules
  - PyTine.Client (already implemented)
  - PyTine.Server (not required for the moment)

Evaluated Methods: General

1. Native Python Library (discarded)
   - Low level integration of TINE inside Python
   - Reimplementation of TINE (we don't want to reinvent the wheel)
   - Long-term project with complex network implementations
   - Big effort to maintain in the future

2. Bindings to the TINE libraries (suitable)
   - Interface an existing TINE native library to Python
   - Already used in TINE (for Labview for example)
   - Two native TINE libraries: C and Java (??)

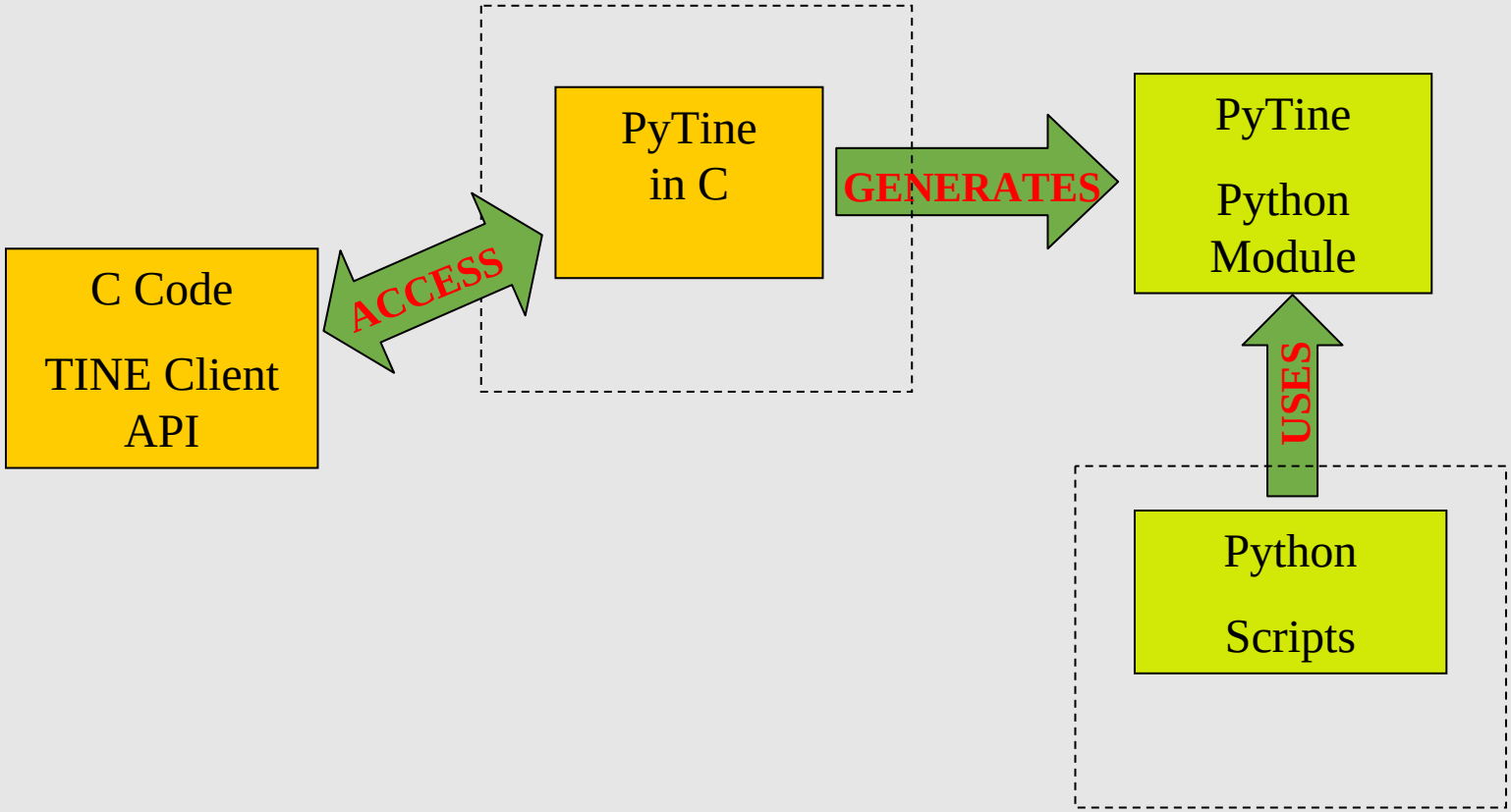
Evaluated Methods: Python Bindings

- Java TINE API (discarded)
  - Possible to call Python from Java (JPython)
  - But not that well supported to call Java from Python
- C TINE API (suitable)
  - Original TINE implementation
  - Existing interface between Python and C
    - From Python to C
    - From C to Python
  - Provides as a result Python modules independent of the C programming language
  - Better experience with the C-TINE API

Evaluated Methods: Python in C

- Use of a translator library (discarded)
  - Boost.Python vs Swig (by Peter Konarev)
    - Boost supports more functionality
    - Boost is more extended than Swig
    - Boost belongs to a “standard” set up C/C++ libraries
  - Is not fully automatic
- Use of the native Python library inside C (decided)
  - Binding included inside C (Python.h)
  - We do not depend on a third party library
  - Used by Boost and Swig
  - First approach came by Daniel Franke (TINE Workshop 2007)

Python – TINE C Binding



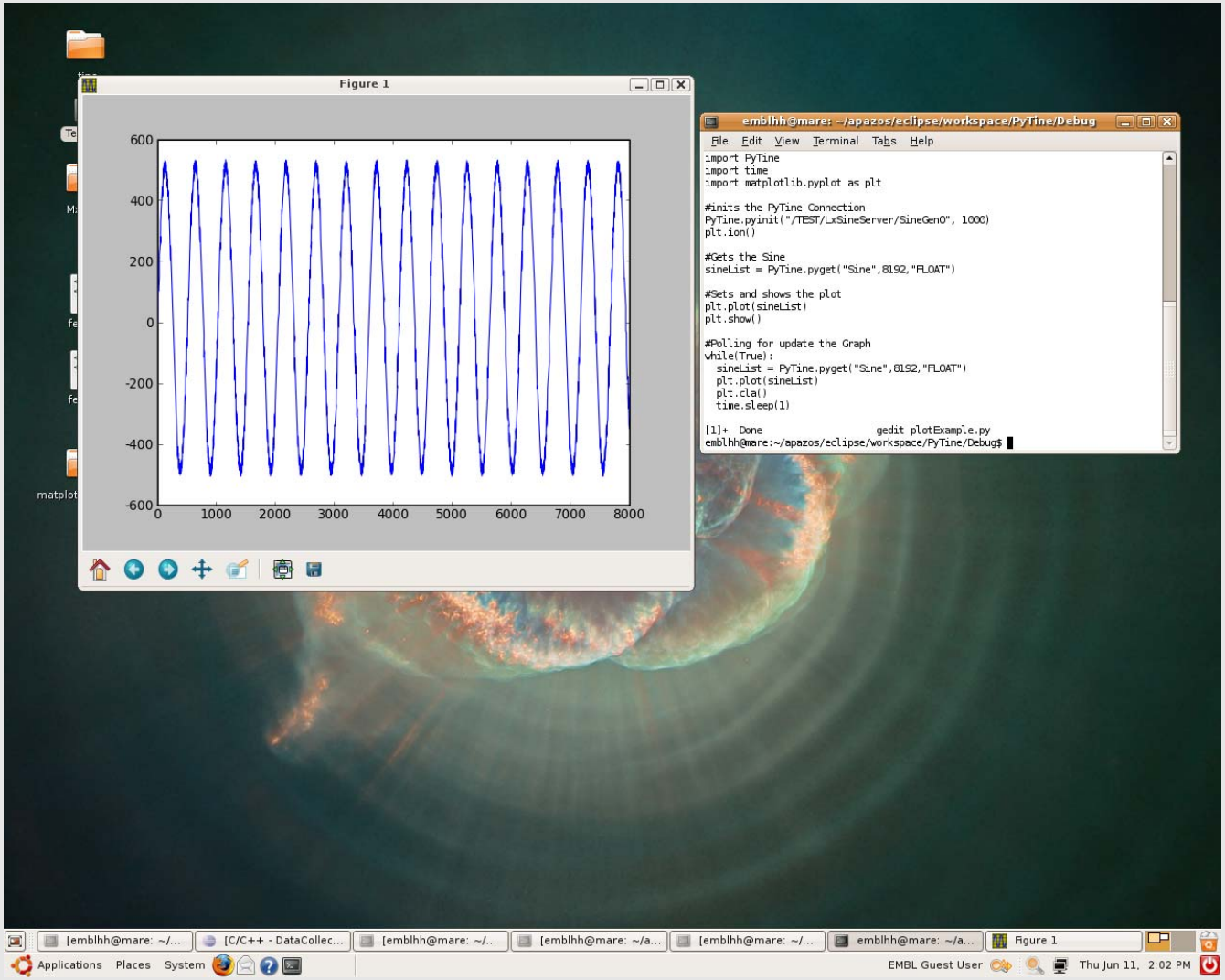
PyTine

- Client TINE-API implemented (90 %)
- Callback support
- TINE Structure Support
- Multi-platform
- Tested with the MxCube TINE integration (with by Peter Konarev)
- First prototype available
- Can be integrated in Java applications (Jython)
- Can be integrated in Labiew applications (already tested with LabPython)
- Plot functionality can be integrated (PyPlot)

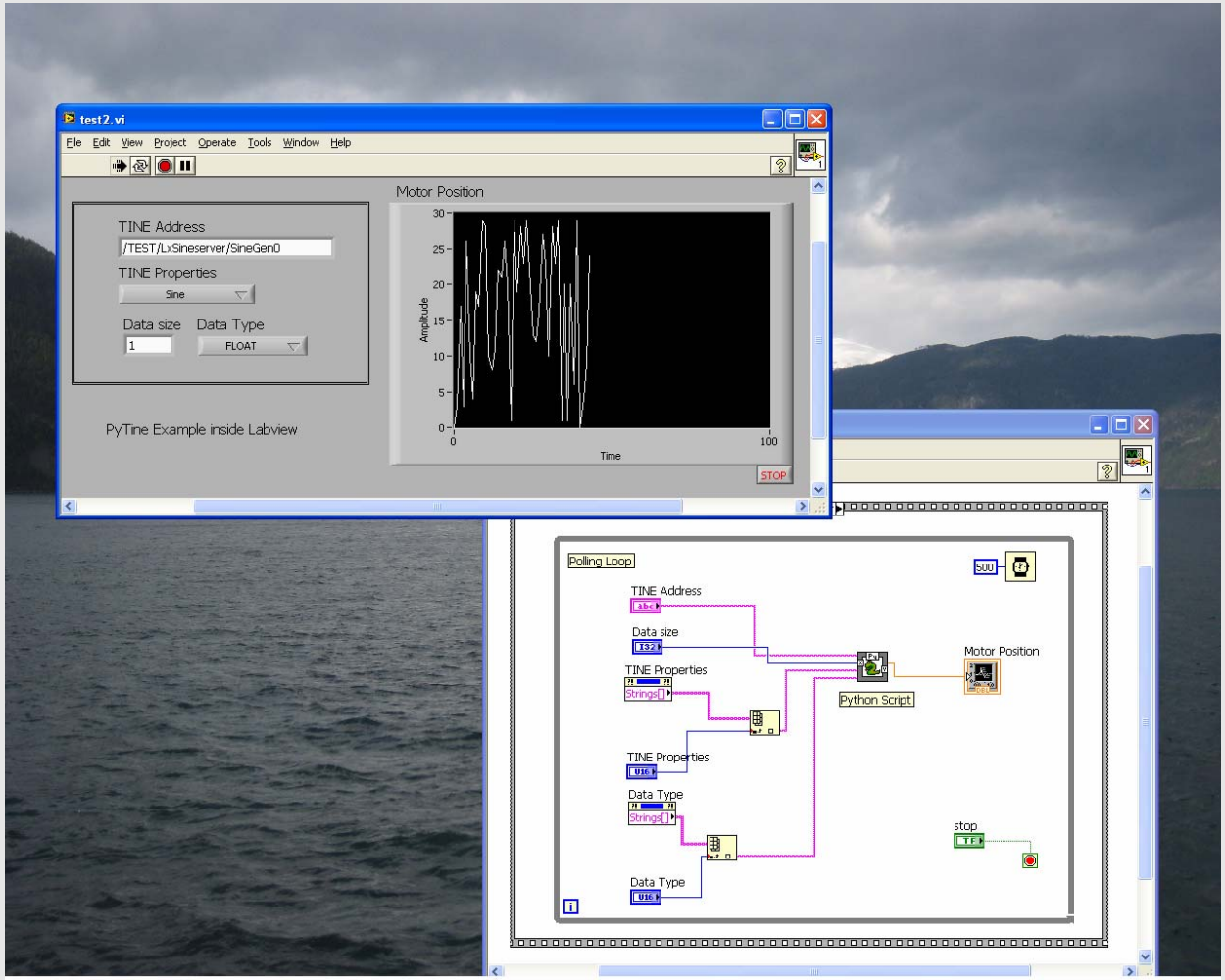
To be done

- Finish the TINE Client API
- Finish the Documentation
- More tests
- Prepare first release
- Implementation of Python scripts.
- Anybody is interested?

SinGenerator Client



Other Client inside Labview



PyTine API

- Initialization

```
PyTine.pyInit("TEST/DataCollection/Axis",1000)
```

- Get Data

```
PyTine.pyget (,“Position“,”FLOAT”,1)
PyTine.pygetEx("TEST/DataCollection/Axis",“Position“,”FLOAT”,1)
```

- Put Data:

```
PyTine.pyput("Name","NAME32",["Resolution"])
PyTine.pyputEx("TEST/DataCollection/Axis","Name","NAME32",["Resolution“])
```

- Put/Get Data:

```
print PyTine.pyputget(“Position","NAME32",["Distance"])
```

- TINE Structs:

```
PyTine.pygetStructFormat("SineInfo")
PyTine.pygetStruct("SineInfo")
PyTine.pyputStruct("SineInfo", ['250',"3","29","1","2511","Test"]
```

PyTine API

- Callback Example

```
def callbackFunction (idc, cc, listData):
      print (listData)

PyTine.pyinit("TEST/DataCollection/Axis",1000)
PyTine.pysetHandler(callbackFunction)
PyTine.pygetAsync("axisPosition",1,"FLOAT", "CM_DATACHANGE")
PyTine.pyrun()
```

- Other functions

```
PyTine.pygetProperties()
PyTine.pygetSrvAddress()
PyTine.pygetCurrentLinkStatus(1)
etc..
```

Example 1

```
import PyTine

class dataCollection():

   def __init__(self,TINEaddr, rate):
        PyTine.pyinit(TINEaddr,rate)

   def moveAxis(destPosition, axis):
        PyTine.pyput("axisMove","NAME32",   [axis
        ,str(destPosition), 1])

   def exposeFrame(axis, exposure, startphi, phirange, run, dir
   ,prefix):
        PyTine.pyput(“daqExposeFrame”,”STRUCT”,[exposure,
        startPhi, phirange, run, dir, prefix])
```

Example: collect data

```
import dataCollection

//set the exposure parameters
prefix = tst1
dir = /home/marccd/images
run = 1
distance = 320
startphi = 0
phirange = 1.0
exposure = 1.0
frames = 10

# move distance to start synchronous
dataCollection.moveAxis(325, “Distance”)

# start data collection
i = 1
while i <= frames:
  status = dataCollection.exposeFrame(PHI, exposure, startphi, phirange, run, dir
    ,prefix)
 print “Exposing frame ”, i , “ result: “, status
 i++

print “Data Collection Finished”
```