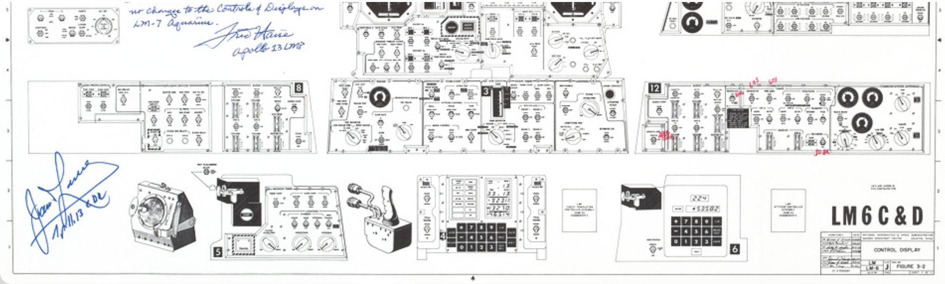
Karlsruhe Institute of Technology

Experiment Control for High-Speed Tomography

M. Vogelgesang, T. Farago, T. Rolo, A. Kopmann, W.Mexner and T. Baumbach

Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation



Motivation

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Motivation

We have *strong* X-ray light sources

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Motivation

We have *strong* X-ray light sources , *fast* detectors,

Motivation

We have *strong* X-ray light sources , *fast* detectors, *distributed* device access via TANGO

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Motivation

We have *strong* X-ray light sources , *fast* detectors, *distributed* device access via TANGO and *huge* processing capacity.

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Motivation

We have *strong* X-ray light sources , *fast* detectors, *distributed* device access via TANGO and *huge* processing capacity.

- Let's do awesome stuff with that!
- Process data and monitor changes on-line
- Build feedback-based control algorithms

UFO Project

Collaborative effort to

- Build hardware and software for high-speed tomography experiments
- Develop fast 2D detector and library for direct access
- Implement GPU-based data processing framework
  - Running on heterogenous compute systems
  - $\approx$ 10 to 100 times *faster* reconstruction
- Do on-line reconstruction and data analysis
- Provide image-based feedback control

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

UFO Project

Collaborative effort to

- Build hardware and software for high-speed tomography experiments
- Develop fast 2D detector and library for direct access
- Implement GPU-based data processing framework
  - Running on heterogenous compute systems
  - $\approx$ 10 to 100 times *faster* reconstruction
- Do on-line reconstruction and data analysis
- Provide image-based feedback control

We need a system that *glues* all components together and is accessible through a simple user interface.

UFO Project

Collaborative effort to

- Build hardware and software for high-speed tomography experiments
- Develop fast 2D detector and library for direct access
- Implement GPU-based data processing framework
  - Running on heterogenous compute systems
  - $\approx$ 10 to 100 times *faster* reconstruction
- Do on-line reconstruction and data analysis
- Provide image-based feedback control

We need a system that *glues* all components together and is accessible through a simple user interface.

That's what *Concert* will be for.

Scope of Concert

What it is about

- A Python framework for conducting high-speed experiments
- Local instead of a distributed system
- Standard procedures for common tomography tasks
- *Prototype* for high-speed tomography experiments at ANKA

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Scope of Concert

Karlsruhe Institute of Technology

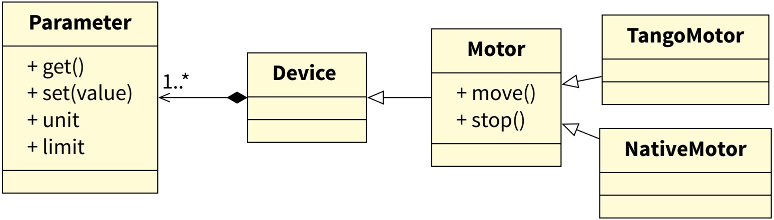
What it is about

- A Python framework for conducting high-speed experiments
- Local instead of a distributed system
- Standard procedures for common tomography tasks
- *Prototype* for high-speed tomography experiments at ANKA

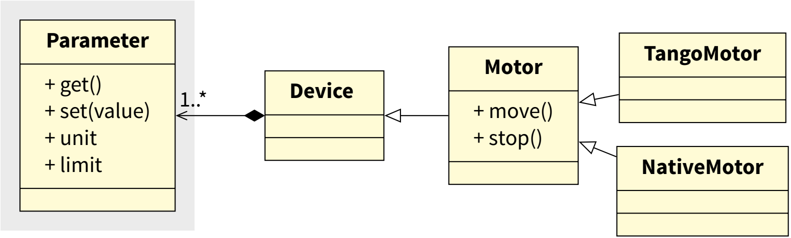
What it is not

- A general solution for all beamline problems
- Data archival system (e.g. meta data)
- Providing a GUI (Taurus?)

Partial Class Hierarchy Overview



Partial Class Hierarchy Overview

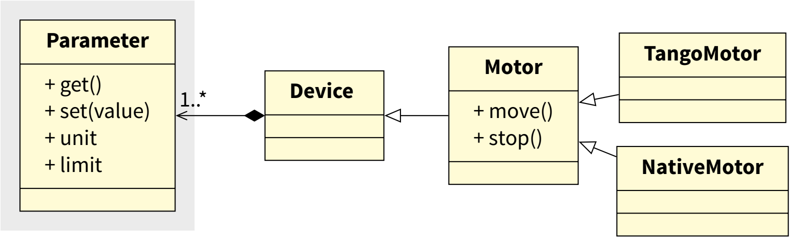


A parameter

- Controls *one* aspect
- Has device specific getters & setters,
- optional SI units (via quantities),
- limits and descriptive doc string

Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

Partial Class Hierarchy Overview

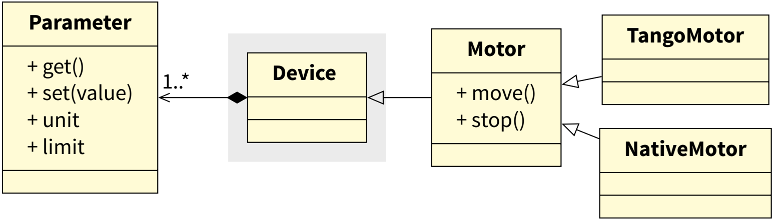


Benefits:

- Validation of user input units
- Automatic access logging

Partial Class Hierarchy Overview

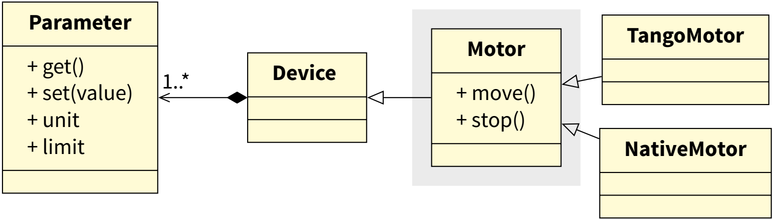
Karlsruhe Institute of Technology



A device consists of

- One or more parameters and
- Auxiliary methods

Partial Class Hierarchy Overview



Base device class provides

- Type-safe device distinction
- Common interface and methods

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Usage

- Enumerate parameters

```
motor = TangoMotor()

for param in motor:
    print(param)    # prints parameters value and unit
```


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Usage

- Enumerate parameters

```
motor = TangoMotor()

for param in motor:
    print(param)    # prints parameters value and unit
```

- *Dict access* for Parameter objects

```
print(motor['position'].unit)
```


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron
Radiation

Usage

- Enumerate parameters

```
motor = TangoMotor()

for param in motor:
    print(param)    # prints parameters value and unit
```

- *Dict access* for Parameter objects

```
print(motor['position'].unit)
```

- *Attribute access* for setting/getting values

```
print(motor.position)
x = motor.position
```


Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

Usage

- Enumerate parameters

```
motor = TangoMotor()

for param in motor:
    print(param)    # prints parameters value and unit
```

- *Dict access* for Parameter objects

```
print(motor['position'].unit)
```

- *Attribute access* for setting/getting values

```
print(motor.position)
x = motor.position
```

- Invalid assignment fails gracefully with an exception

```
>>> motor.position = 1 * q.keV
Sorry, 'position' can only receive values of unit 1 m (meter) but got 1.0 keV
```


Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

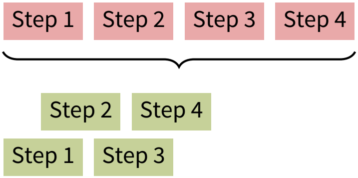
Asynchronous Device Access

- Careless synchronization can lead to excessive latencies

Step 1	Step 2	Step 3	Step 4

Asynchronous Device Access

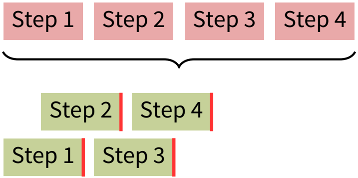
- Careless synchronization can lead to excessive latencies
- Latencies are reduced by executing tasks in parallel



Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

Asynchronous Device Access

- Careless synchronization can lead to excessive latencies
- Latencies are reduced by executing tasks in parallel
- We *must* be notified when a task is finished



Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Parallel Execution

Karlsruhe Institute of Technology

Asynchronous execution

- Futures instead of raw threads
    - A future promises to return the result of a task at some point in the future
- Callbacks are called, no matter *when* they are attached
- Synchronization via device locks

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Parallel Execution

Karlsruhe Institute of Technology

Asynchronous execution

- Futures instead of raw threads
  - A future promises to return the result of a task at some point in the future
- Callbacks are called, no matter *when* they are attached
- Synchronization via device locks

Monitoring and notification

- Messaging bus for process-wide notification
- Subscribers sign up for messages and are notified upon message arrival
- Light-weight monitoring mechanism

Futures in Action

Karlsruhe Institute of Technology

- “Regular” attribute-like accesses are synchronous

```
m = MotorImpl()
m.position = 1.5 * q.mm          # Blocks until finished
```


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Futures in Action

- “Regular” attribute-like accesses are synchronous

```
m = MotorImpl()
m.position = 1.5 * q.mm         # Blocks until finished
```

- Accessors are asynchronous parameter methods and return a future

```
f1 = m.set_position(1.5 * q.mm) # Does not block
f2 = m.get_position()
f3 = m['position'].get()
```


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Futures in Action

Karlsruhe Institute of Technology

- “Regular” attribute-like accesses are synchronous

```
m = MotorImpl()
m.position = 1.5 * q.mm         # Blocks until finished
```

- Accessors are asynchronous parameter methods and return a future

```
f1 = m.set_position(1.5 * q.mm) # Does not block
f2 = m.get_position()
f3 = m['position'].get()
```

- Query futures and add callbacks

```
print("Done yet? {0}".format(f1.done()))
f1.add_done_callback(do_something)
```


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Futures in Action

Karlsruhe Institute of Technology

- "Regular" attribute-like accesses are synchronous

```
m = MotorImpl()
m.position = 1.5 * q.mm          # Blocks until finished
```

- Accessors are asynchronous parameter methods and return a future

```
f1 = m.set_position(1.5 * q.mm) # Does not block
f2 = m.get_position()
f3 = m['position'].get()
```

- Query futures and add callbacks

```
print("Done yet? {0}".format(f1.done()))
f1.add_done_callback(do_something)
```

- Wait for the result synchronously and do something with it

```
future = m.get_position()
result = future.result()
print(result)
```

Asynchronous Methods

- @async decorator turns any method into an asynchronous one

```
class Motor(Device):
    @async
    def move(self, delta):
        self.position += delta
```


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Asynchronous Methods

Karlsruhe Institute of Technology

- @async decorator turns any method into an asynchronous one

```
class Motor(Device):
    @async
    def move(self, delta):
        self.position += delta
```

- Usage is the same as for the parameter access:

```
m = MotorImpl()
f = m.move(-5 * q.cm)
print("Still running? {0}".format(f.running()))
```


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron
Radiation

Messaging

- Single message *dispatcher* is used for subscription

Messaging

- Single message *dispatcher* is used for subscription
- Caller provides a callback handler …

```
def alert(sender):
    msg = "We ran into a limit, current position is {0}"
    print(msg.format(sender.position))
```

Messaging

Karlsruhe Institute of Technology

- Single message *dispatcher* is used for subscription
- Caller provides a callback handler …

```
def alert(sender):
    msg = "We ran into a limit, current position is {0}"
    print(msg.format(sender.position))
```

- …and subscribes on the bus

```
m = MotorImpl()
dispatcher.subscribe(m, m.LIMIT, alert)
```

Process Abstractions

Motivation

- Common procedures are recurring over and over again
- Separation of high level algorithm from low-level device access encourages code re-use

Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

Process Abstractions

Karlsruhe Institute of Technology

Motivation

- Common procedures are recurring over and over again
- Separation of high level algorithm from low-level device access encourages code re-use

Solution

- Provide *abstract* skeletons for recurring tasks
- Let the scientist compose complete processes

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Scanning

- Correlate scan parameter and feedback values
- Feedback can be of any complexity
- For example, a detector calibration procedure calculates the sensitivity over a range of exposure times

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Simplified EMV1288 Sensitivity Procedure

```
detector = UcaCamera('pco')
shutter = Shutter()

def compute_parameters():
    shutter.close().wait()
    mean_dark = np.mean(detector.grab())
    shutter.open().wait()
    mean_bright = np.mean(detector.grab())
    return (mean_bright - mean_dark)

scanner = Scanner(detector['exposure-time'], compute_parameters)
scanner.minimum = 10 * q.microsecond
scanner.maximum = 1 * q.second

# Wait for the scan to complete and resolve the future
exp_times, sensitivity = scanner.run().result()
plt.plot(exp_times, sensitivity)
```


Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

Simplified EMV1288 Sensitivity Procedure

Karlsruhe Institute of Technology

```
detector = UcaCamera('pco')
shutter = Shutter()

def compute_parameters():
    shutter.close().wait()
    mean_dark = np.mean(detector.grab())
    shutter.open().wait()
    mean_bright = np.mean(detector.grab())
    return (mean_bright - mean_dark)

scanner = Scanner(detector['exposure-time'], compute_parameters)
scanner.minimum = 10 * q.microsecond
scanner.maximum = 1 * q.second

# Wait for the scan to complete and resolve the future
exp_times, sensitivity = scanner.run().result()
plt.plot(exp_times, sensitivity)
```

Specify the feedback

Simplified EMV1288 Sensitivity Procedure

Karlsruhe Institute of Technology

```
detector = UcaCamera('pco')
shutter = Shutter()

def compute_parameters():
    shutter.close().wait()
    mean_dark = np.mean(detector.grab())
    shutter.open().wait()
    mean_bright = np.mean(detector.grab())
    return (mean_bright - mean_dark)

scanner = Scanner(detector['exposure-time'], compute_parameters)
scanner.minimum = 10 * q.microsecond
scanner.maximum = 1 * q.second

# Wait for the scan to complete and resolve the future
exp_times, sensitivity = scanner.run().result()
plt.plot(exp_times, sensitivity)
```

Scan from 10 μs to 1 s

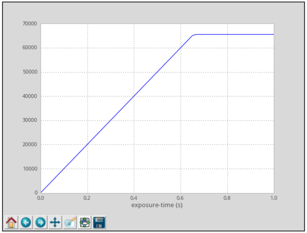
Simplified EMV1288 Sensitivity Procedure

```
detector = UcaCamera('pco')
shutter = Shutter()

def compute_parameters():
    shutter.close().wait()
    mean_dark = np.mean(detector.grab())
    shutter.open().wait()
    mean_bright = np.mean(detector.grab())
    return (mean_bright - mean_dark)

scanner = Scanner(detector['exposure-time'], compute_parameters)
scanner.minimum = 10 * q.microsecond
scanner.maximum = 1 * q.second
```

```
# Wait for the scan to complete and resolve the future
exp_times, sensitivity = scanner.run().result()
plt.plot(exp_times, sensitivity)
```



Processing Data

- Same abstraction is used for processing data with our GPU-based framework

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Processing Data

- Same abstraction is used for processing data with our GPU-based framework
- Create a task graph of dependent operations

```
graph.connect_nodes(detector, backproject)
graph.connect_nodes(backproject, writer)
```


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Processing Data

Karlsruhe Institute of Technology

- Same abstraction is used for processing data with our GPU-based framework
- Create a task graph of dependent operations

```
graph.connect_nodes(detector, backproject)
graph.connect_nodes(backproject, writer)
```

- …and a “process” object using it

```
process = UfoProcess(graph, backproject, 'axis-pos')
```

Processing Data

- Same abstraction is used for processing data with our GPU-based framework
- Create a task graph of dependent operations

```
graph.connect_nodes(detector, backproject)
graph.connect_nodes(backproject, writer)
```

- …and a “process” object using it

```
process = UfoProcess(graph, backproject, 'axis-pos')
```

- As before, we can use this object directly

```
future = process.run()      # process in the background
```


Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

Processing Data

Karlsruhe Institute of Technology

- Same abstraction is used for processing data with our GPU-based framework
- Create a task graph of dependent operations

```
graph.connect_nodes(detector, backproject)
graph.connect_nodes(backproject, writer)
```

- …and a “process” object using it

```
process = UfoProcess(graph, backproject, 'axis-pos')
```

- As before, we can use this object directly

```
future = process.run()      # process in the background
```

- …or scan the exposed parameter

```
scanner = Scanner(process['axis-pos'], do_something)
```


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Sessions

- Encapsulate experiment types into pre-defined sessions
- Combine sessions via import

```
import tomography

rot_motor.set_velocity(10 * q.deg / q.second)
shutter.open().wait()
pco_dimax.start_record()
...
```

- Starting a session launches an IPython shell (for now)

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

NeXus

Karlsruhe Institute of Technology

- Started to think about saving NeXus data sets
- Protoype stores tomographic scan data sets using Nexpy

```
def do_nothing():
   pass

tomo_scanner = StepTomoScanner(detector, rotary_stage)
dataset = get_tomo_scan_result(tomo_scanner).result()
dataset.nxsave('scan.hdf5')
```

- We are currently investigating DESY's pni-libraries as a backend

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Quality Assurance

Karlsruhe Institute of Technology

- Continuous integration with Jenkins
- 75 unit tests
- flake8 (pep8 + pyflakes) & pylint checks
- Sphinx documentation at `concert.readthedocs.org`
- Usable with pip and virtualenv

Conclusion

Karlsruhe Institute of Technology

Summary

- We built an open prototype to integrate control and data processing
  - `github.com/ufo-kit/concert`
  - `pypi.python.org/pypi/concert`
- Interoperability with TANGO, UFO framework, NeXus, …
- Parallel execution with defined synchronization points and messaging

Next steps

- Provide stable control loops based on python-control
- Use `IPython.traits` for unit-less parameters

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Thanks for your attention! Questions?

Implementation Details

- Runs on Python 2.6+
- Data processing with the UFO framework
- General device access via Tango
- Detectors accessed with libuca
- quantities, logbook, PyTango and IPython

Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

Synchronization

- Two or more tasks that access the same devices asynchronously must be synchronized (“Start acquisition only when shutter is open …”)

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Synchronization

- Two or more tasks that access the same devices asynchronously must be synchronized (“Start acquisition only when shutter is open …”)
- Devices implement the context manager protocol and keep a lock when used

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Synchronization

Karlsruhe Institute of Technology

- Two or more tasks that access the same devices asynchronously must be synchronized (“Start acquisition only when shutter is open …”)
- Devices implement the context manager protocol and keep a lock when used
- Multiple devices are locked with multicontext object or Python 2.7's enhanced `with` statement

```
# In-process safe device access
with motor, detector:
    motor.set_position(0.5 * q.mm)
    frame = detector.grab()
```

Requirements

Such an approach requires a system that

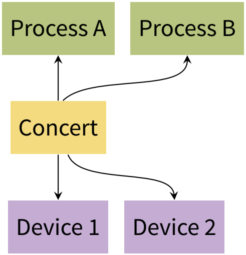
Concert

Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Requirements

Such an approach requires a system that

- Controls devices and processes under study

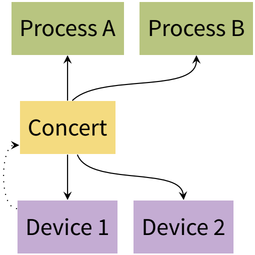


Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

Requirements

Such an approach requires a system that

- Controls devices and processes under study
- Acquires data

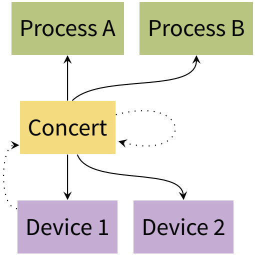


Institute for Data Processing and Electronics & Institute for Photon Science and Synchrotron Radiation

Requirements

Such an approach requires a system that

- Controls devices and processes under study
- Acquires data
- Reacts on data analysis results

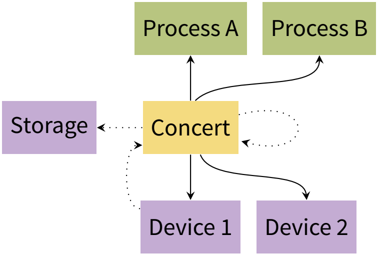


Institute for Data Processing and Electronics &
Institute for Photon Science and Synchrotron Radiation

Requirements

Such an approach requires a system that

- Controls devices and processes under study
- Acquires data
- Reacts on data analysis results
- Stores data



The Zen of Concert

1. Focus on usage and favor
   - User before instrument
   - Scientist before developer
2. Local over distributed processing
3. Small, high quality core
4. Code re-use wherever, whenever possible