Nexus Implementation at DESY

Virtual Laboratories – PANDATA WP5 / HDRI

Jan Kotański, Thorsten Kracht, Eugen Wintersberger

Deutsches Elektronen-Synchrotron

May 23, 2013

Why Nexus ?

- The internal structure of the Nexus files can be adopted to specific experimental techniques
- Metadata can be stored for a complete description of the measurement
- Nexus can store many image frames that are created by a measuring sequence in a single file.

This way data can be managed efficiently

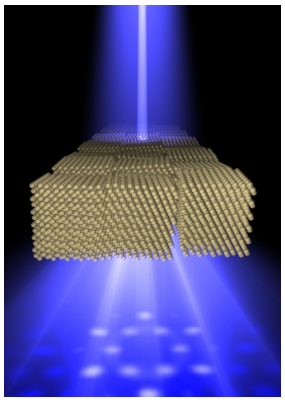
NeXus

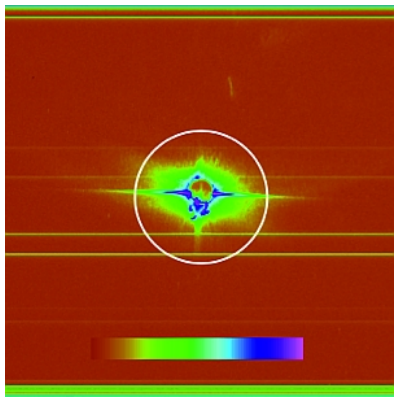
Nexus at PETRA III beamlines

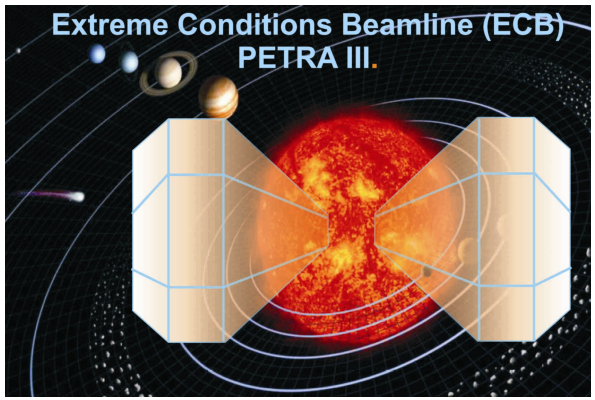
Why Nexus ?

- The internal structure of the Nexus files can be adopted to specific experimental techniques
- Metadata can be stored for a complete description of the measurement
- Nexus can store many image frames that are created by a measuring sequence in a single file.

This way data can be managed efficiently







Nexus at PETRA III beamlines

Why Nexus ?

- The internal structure of the Nexus files can be adopted to specific experimental techniques
- Metadata can be stored for a complete description of the measurement
- Nexus can store many image frames that are created by a measuring sequence in a single file.

This way data can be managed efficiently

NeXus

PNI C++ libraries

Simply creation of Nexus files in C++ (HDRI project)

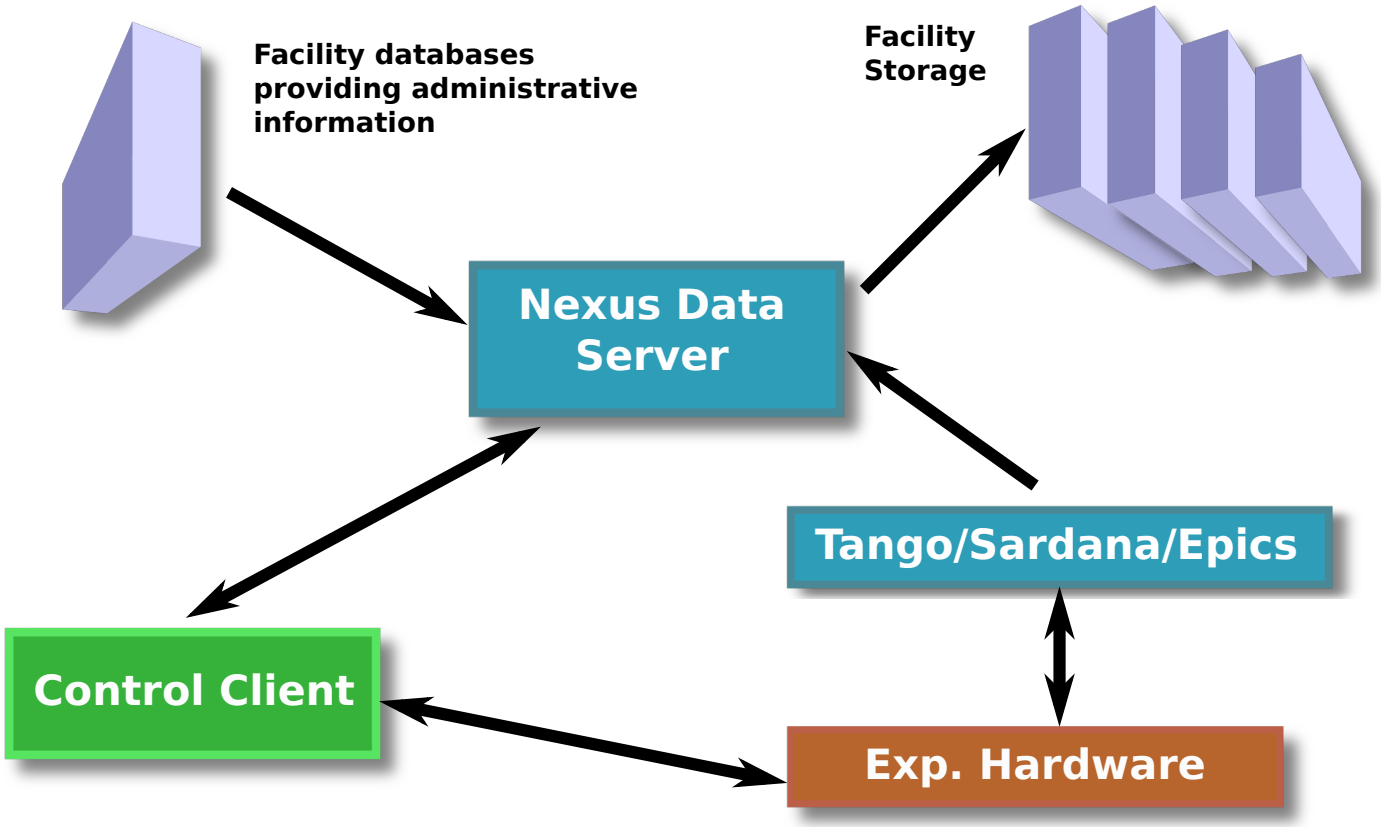
- `libpnicore` provides
  - types of well defined size
  - templates for buffers and arrays
  - reader code to import data from proprietary formats
- `libpniio` classes
  - write Nexus files using HDF5 as its storage back-end
  - make the development independent of the Nexus API
- `python-pniio` Python bindings via the Python package

All libraries are actually in use to develop the software required to establish Nexus as a data format at DESY.

`http://code.google.com/p/pni-libraries/`

`http://www.pni-hdri.de`

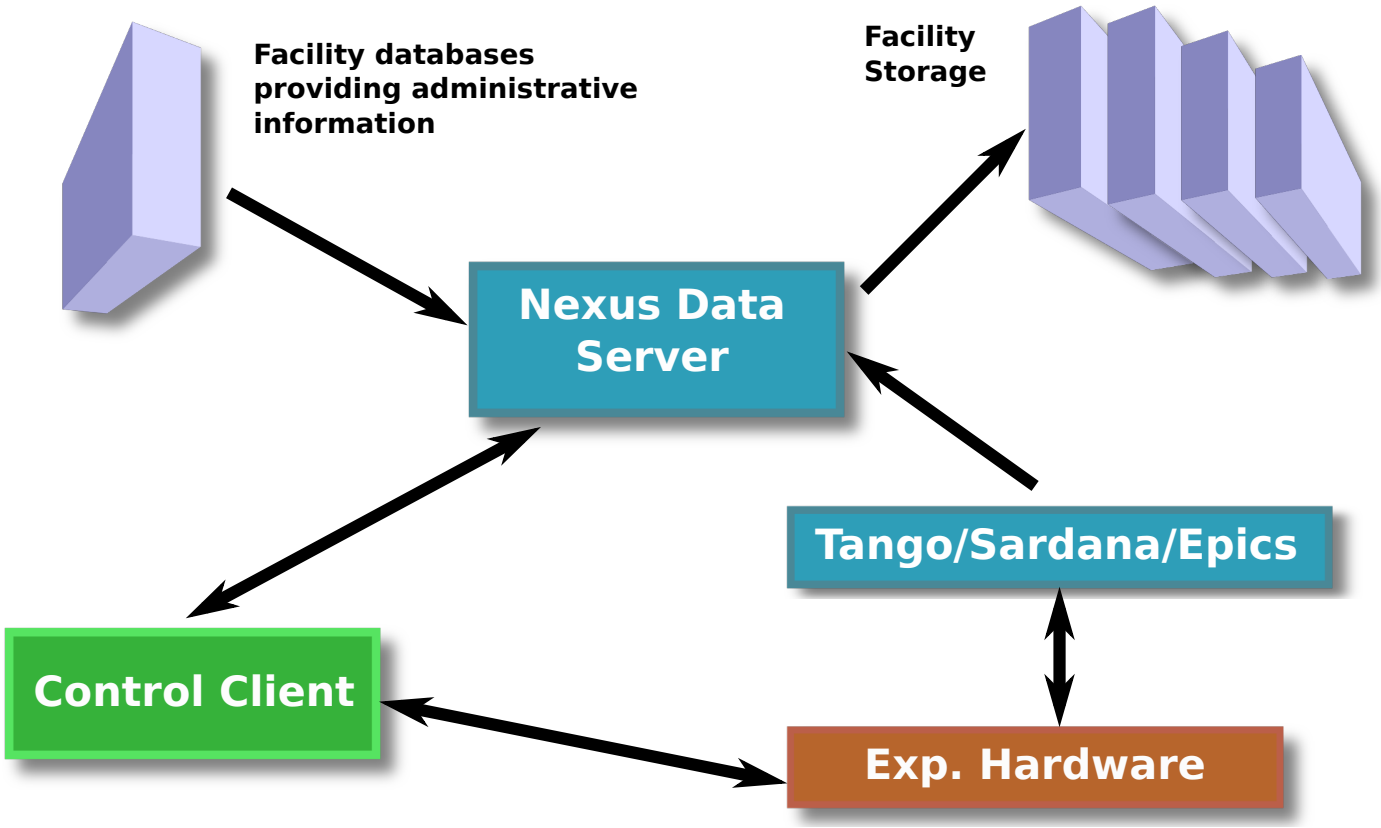
Nexus Data Tango Server - NexDaTaS



We move the responsibility for data IO out of the control client (CC) into a separate Tango server:
the Nexus Data Server (NexDaTaS)

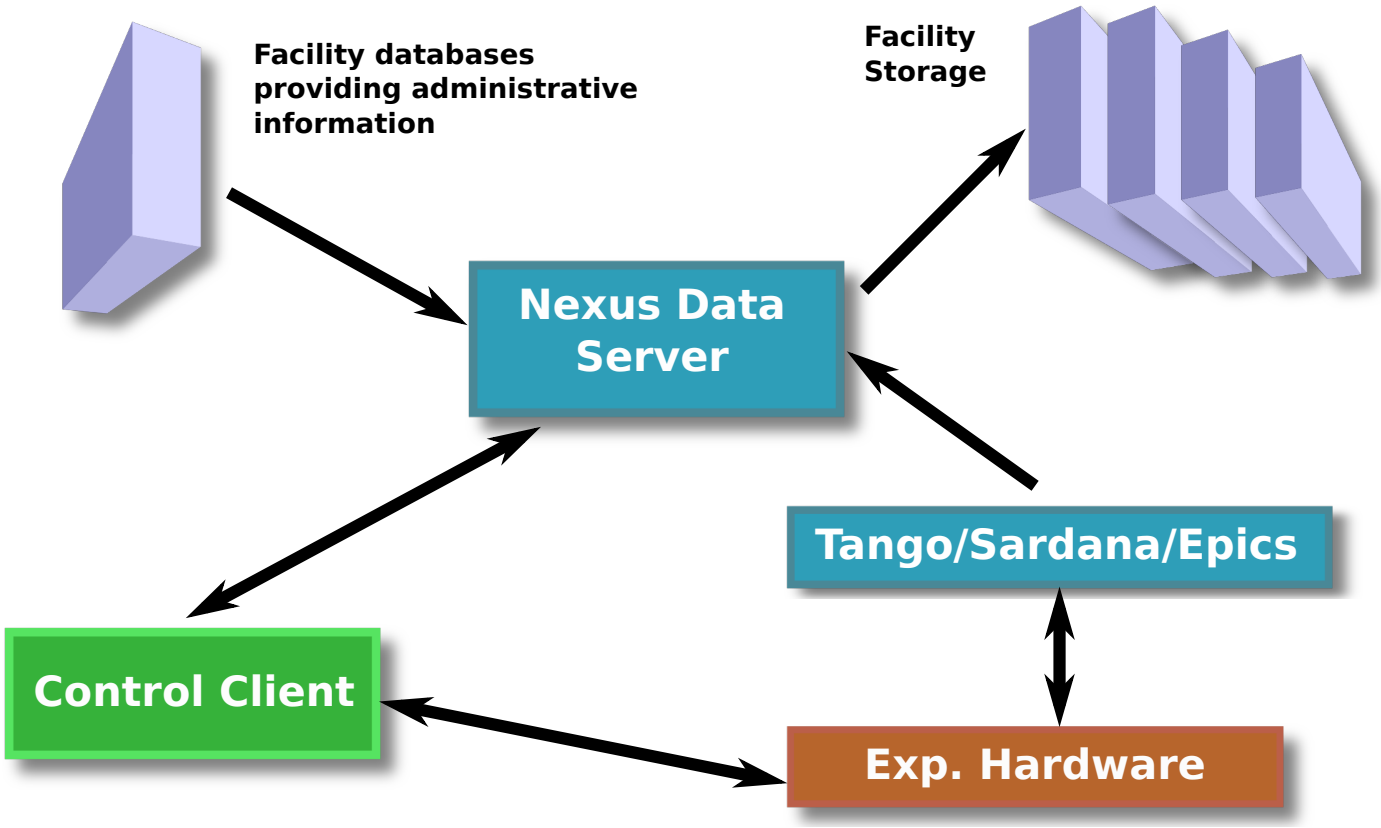
http://code.google.com/p/nexdatas/

Nexus Data Tango Server - NexDaTaS



To Communicate with this server the control system must only be aware of TANGO
for which bindings exist to many languages

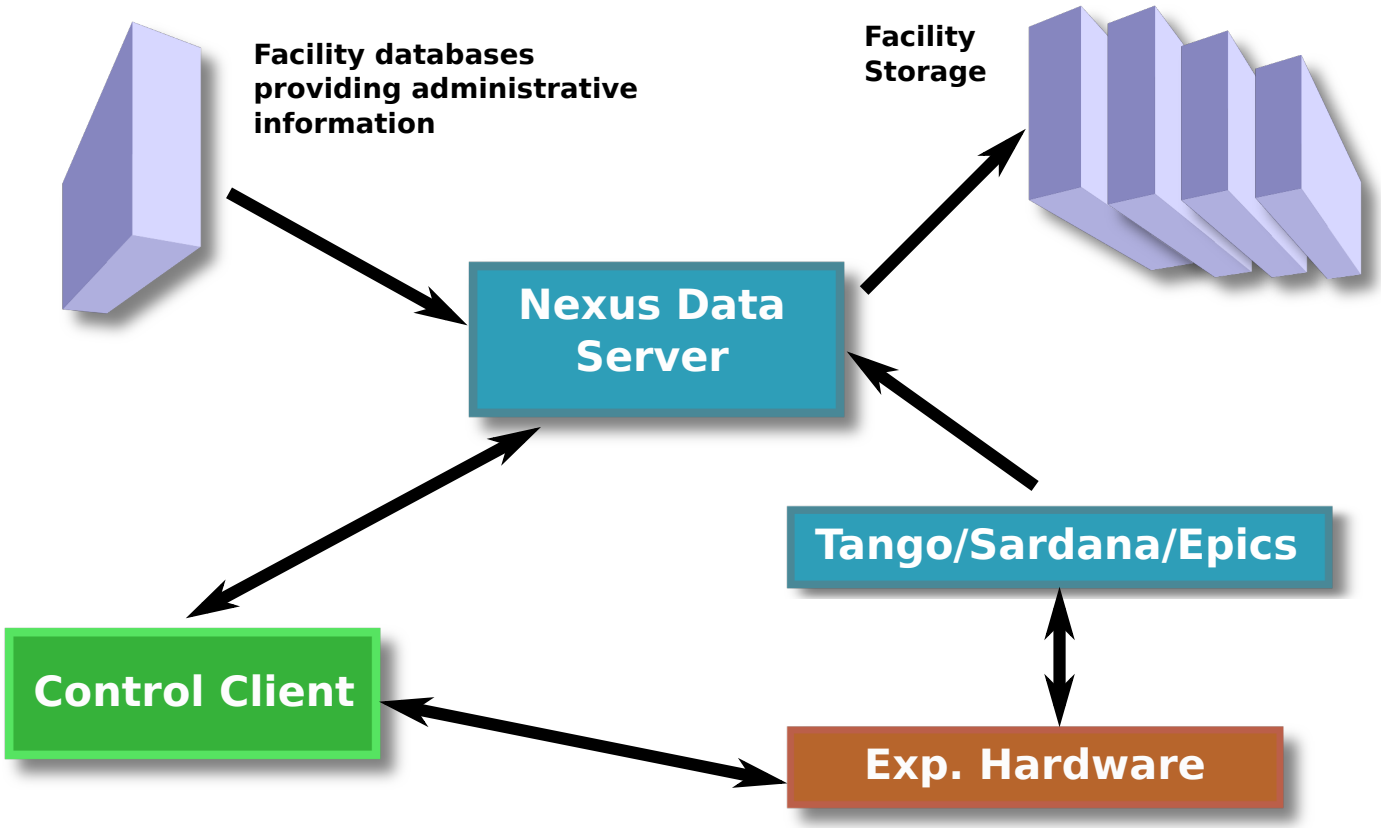
Nexus Data Tango Server - NexDaTaS



The server creates Nexus files and fills its field and attributes from various data sources (DS)

- directly from the CC (also SARDANA) using JSON strings •
- other TANGO servers • databases • external DS •

Nexus Data Tango Server - NexDaTaS



As configuration data the server needs to know

- structure of the Nexus tree for a particular entry
- datasources for each field in the tree
- strategy when and how the data is fetched/written

Extra Tags

Nexus server configuration as NXDL code includes extra tags for storage strategy and data sources

The `<strategy>` attributes:

- `mode` – when the data is fetched:
  - `INIT` – during opening a new entry
  - `STEP` – when the record() command is performed
  - `FINAL` – at the time of closing the entry
  - `POSTRUN` – during post-processing stage
- `trigger` name of the related trigger in STEP mode

The `<datasource>` attributes:

- `type` – type of data source:
  - `CLIENT` – communication with client via JSON strings
  - `TANGO` – data from Tango servers
  - `DB` – data from databases

Extra Tags

Nexus server configuration as NXDL code includes extra tags for storage strategy and data sources

The `<strategy>` attributes:

- `mode` – when the data is fetched:
  - `INIT` – during opening a new entry
  - `STEP` – when the record() command is performed
  - `FINAL` – at the time of closing the entry
  - `POSTRUN` – during post-processing stage
- `trigger` name of the related trigger in STEP mode

The `<datasource>` attributes:

- `type` – type of data source:
  - `CLIENT` – communication with client via JSON strings
  - `TANGO` – data from Tango servers
  - `DB` – data from databases

Extra Tag Example

TANGO datasource in STEP mode

```
<field name="tth" type="NX_FLOAT" unit="degree">
    <strategy mode="STEP" trigger="trigger1"/>
    <datasource type="TANGO">
        <device hostname="haso.desy.de"
                member="attribute"
                name="p09/motor/exp.01"
                port="10000"/>
        <record name="Position"/>
    </datasource>
</field>
```

Extra Tag Example

TANGO datasource in STEP mode

```
<field name="tth" type="NX_FLOAT" unit="degree">
   <strategy mode="STEP" trigger="trigger1"/>
   <datasource type="TANGO">
      <device hostname="haso.desy.de"
              member="attribute"
              name="p09/motor/exp.01"
              port="10000"/>
      <record name="Position"/>
   </datasource>
</field>
```

Extra Tag Examples

CLIENT datasource

```
<datasource type="CLIENT">
    <record name="counter_1"/>
</datasource>
```

DataBase datasource

```
<datasource type="DB">
  <database dbname="tango" dbtype="MYSQL"
            hostname="haso.desy.de"/>
  <query format="SPECTRUM">
    SELECT pid FROM device limit 10
  </query>
</datasource>
```

Extra Tag Examples

CLIENT datasource

```
<datasource type="CLIENT">
    <record name="counter_1"/>
</datasource>
```

DataBase datasource

```
<datasource type="DB">
  <database dbname="tango" dbtype="MYSQL"
            hostname="haso.desy.de"/>
  <query format="SPECTRUM">
    SELECT pid FROM device limit 10
  </query>
</datasource>
```

Example client code

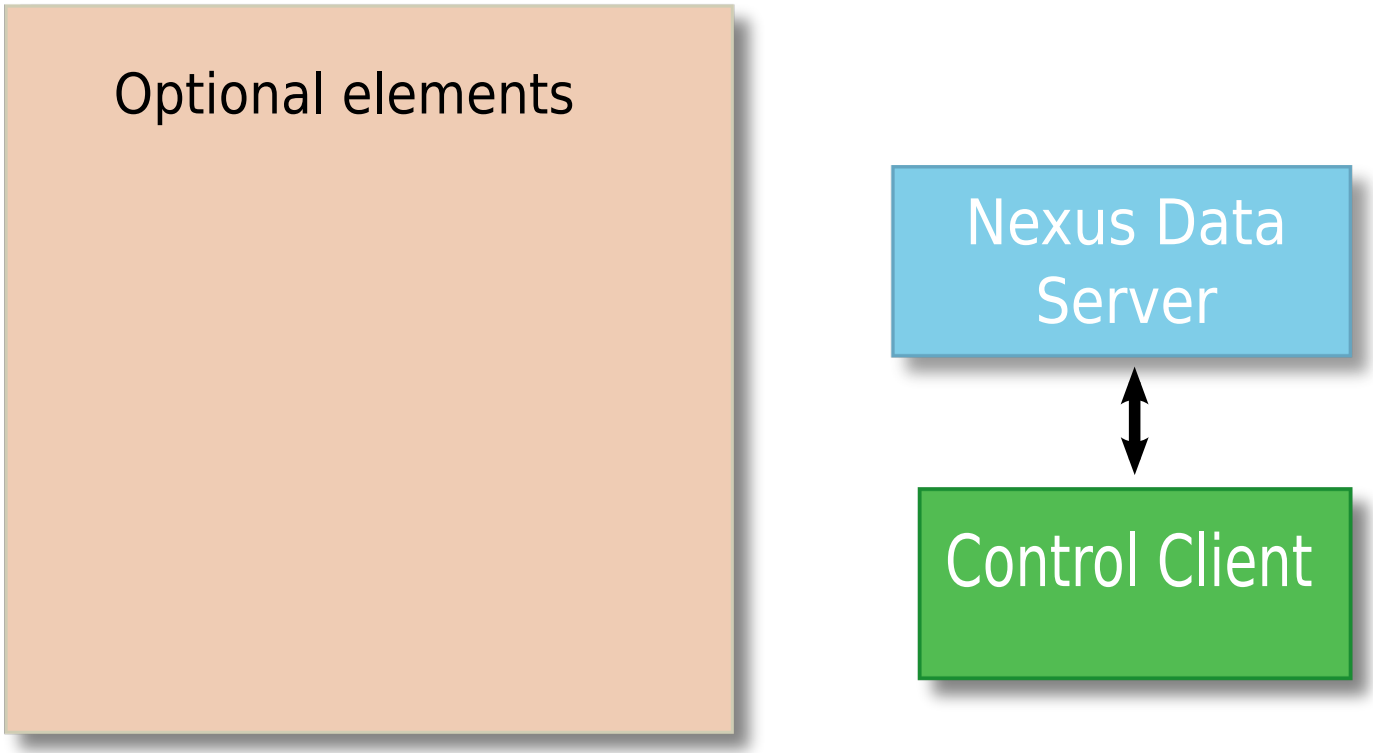
A short example should show how easy it is to use the server from a client application:

```
import PyTango
device = "p09/tdw/r228"
dpx = PyTango.DeviceProxy(device)
dpx.Init()

# open a new file to store data in
dpx.FileName = "test.h5"
dpx.OpenFile()

# send configuration for a new entry
# and write initial data (strategy type INIT)
xml = open("configuration.xml", 'r').read()
dpx.TheXMLSettings = xml
dpx.TheJSONRecord = '{"data": {"parameterA":0.2}}'
dpx.OpenEntry()
```

Example client code

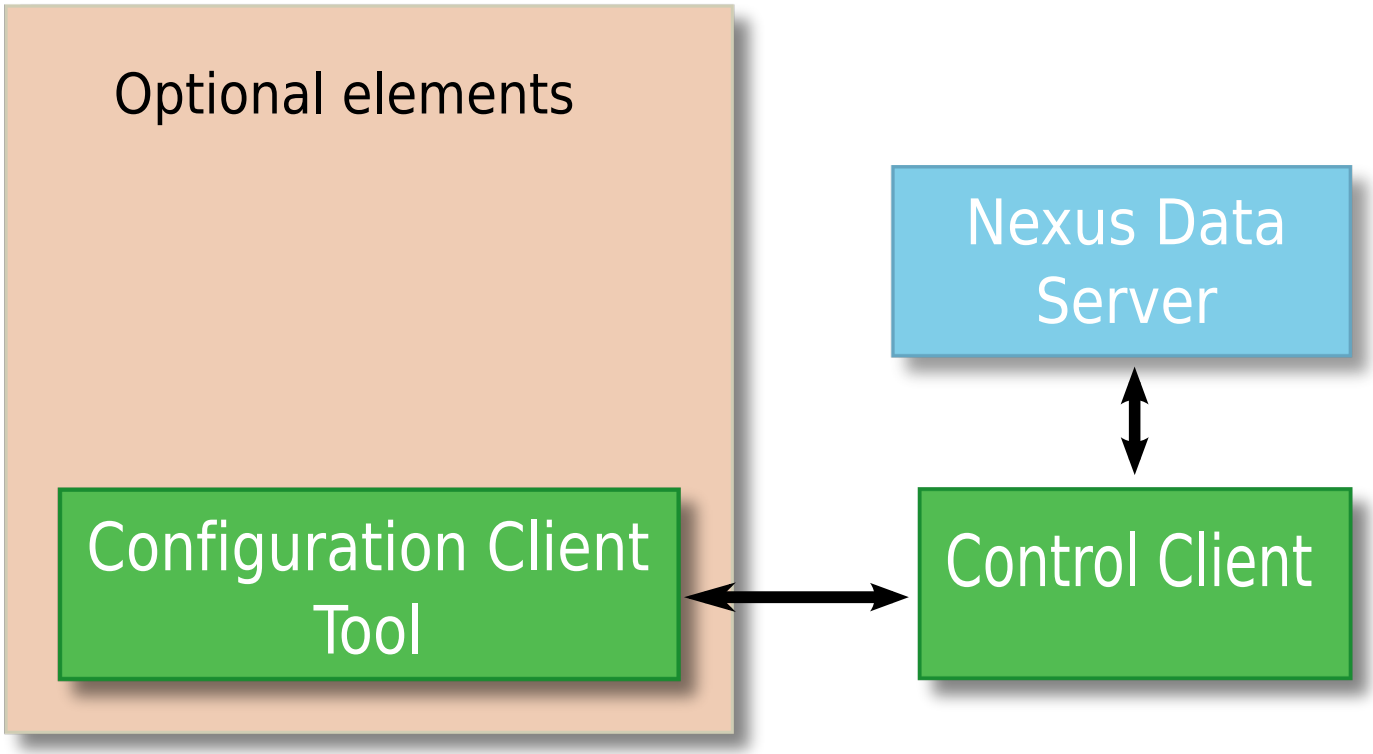
```
# experiment main loop
# (write data with strategy STEP)
for i in range(100):
    dpx.Record('{"data": {"counter.exp01":0.1*i,
                          "counter.exp02":1.1*i}}')

# write final data (strategy FINAL)
# - close the entry - close the file
dpx.TheJSONRecord = '{"data": {"parameterB":0.3}}'
dpx.CloseEntry()
dpx.CloseFile()
```

Configuring the server



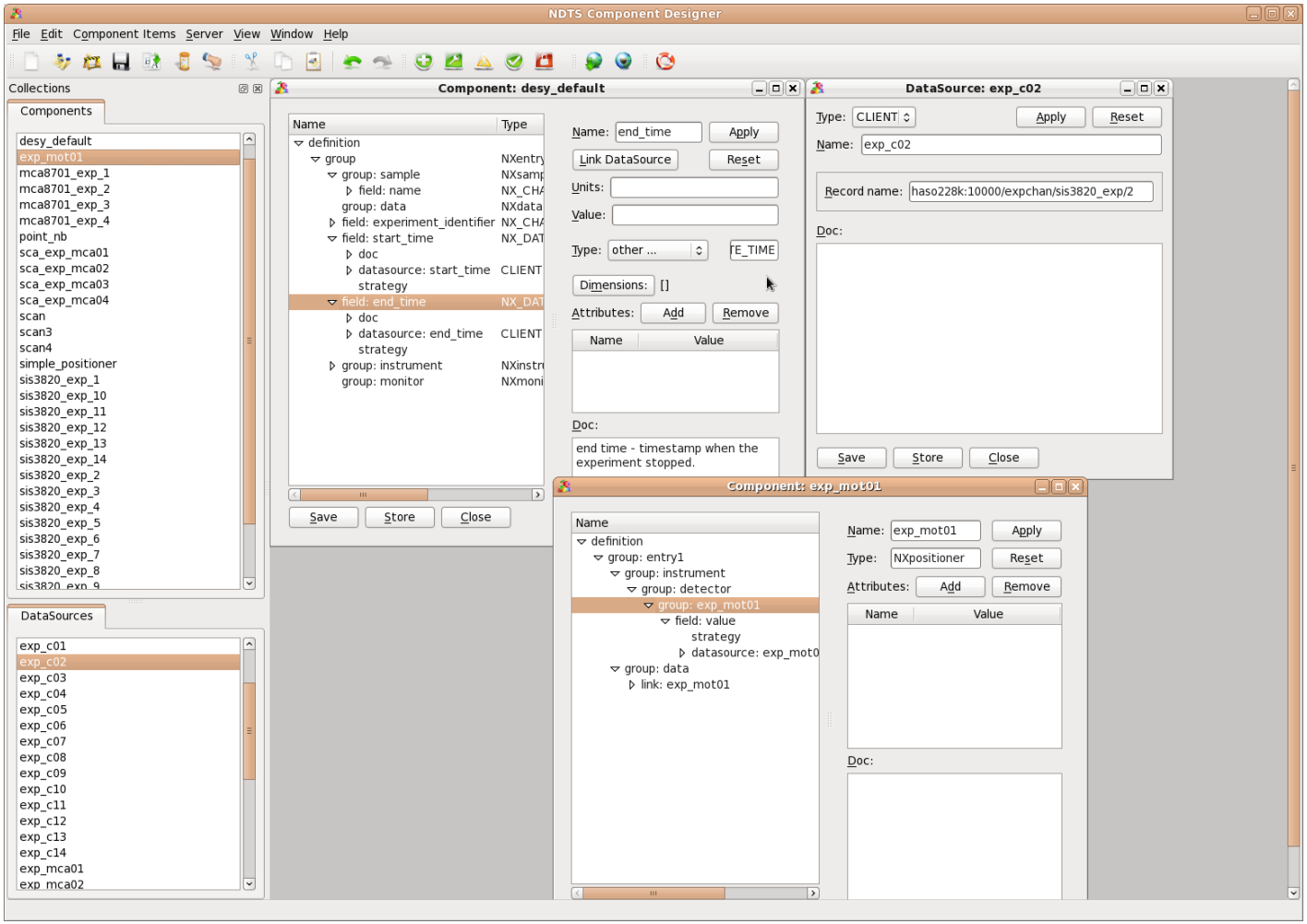
For quite static and not too complex experiments creating the configuration NXDL stream for the Nexus Data Server should be easy and could be done by the control client

Configuring the server



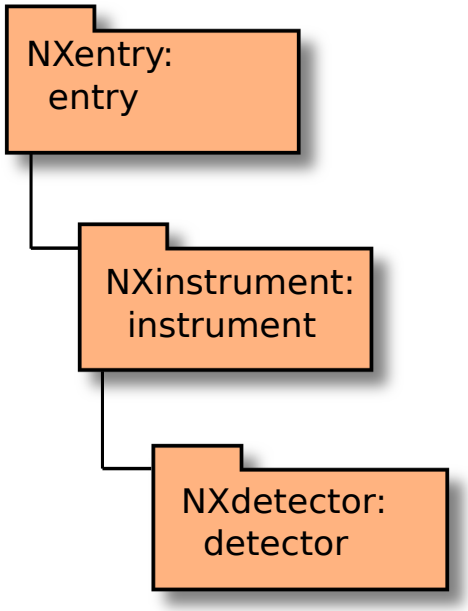
For complex experiments we shift
the complexity of the client code from writing the Nexus file
into creating an advanced NXDL stream

Component Designer



The Configuration Client Tool which allows to create XML configuration files, separate components as well as datasources

Components and their merging

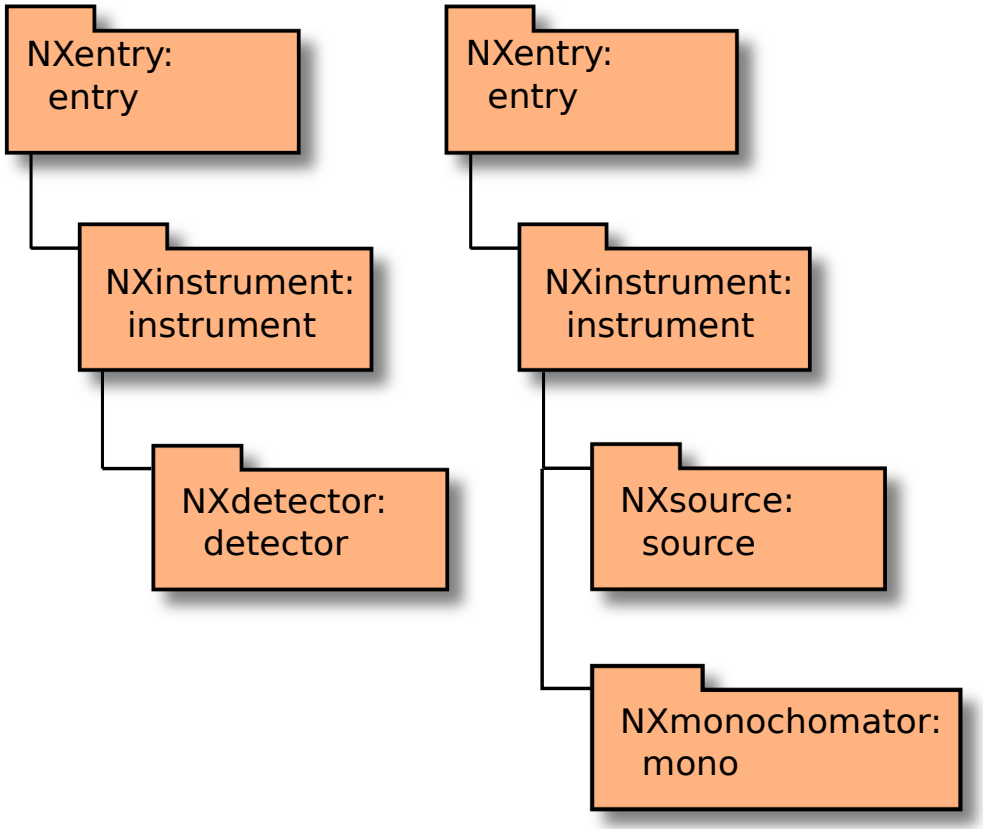


First Component

Second Component

Merged Components

Components and their merging

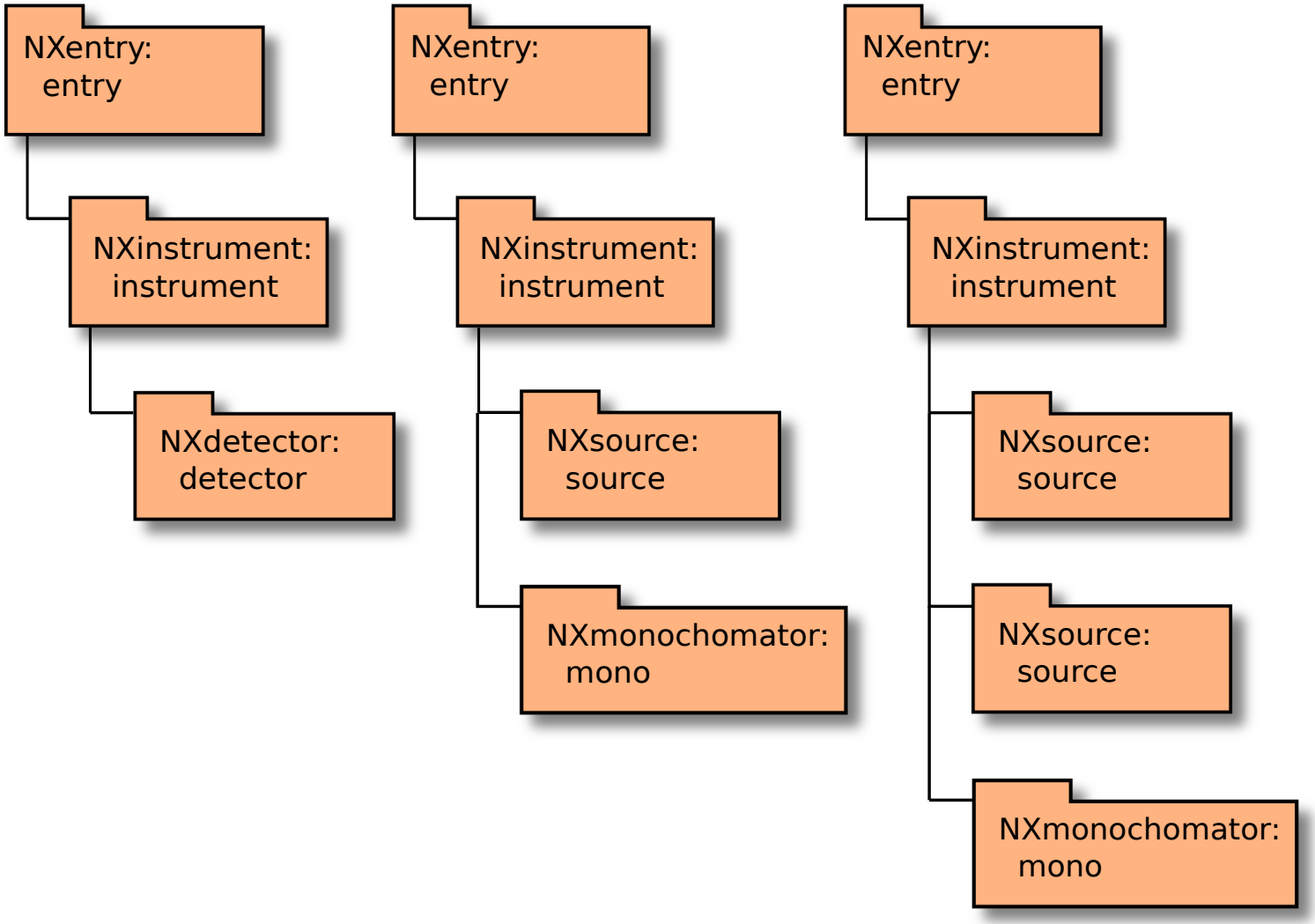


First Component

Second Component

Merged Components

Components and their merging

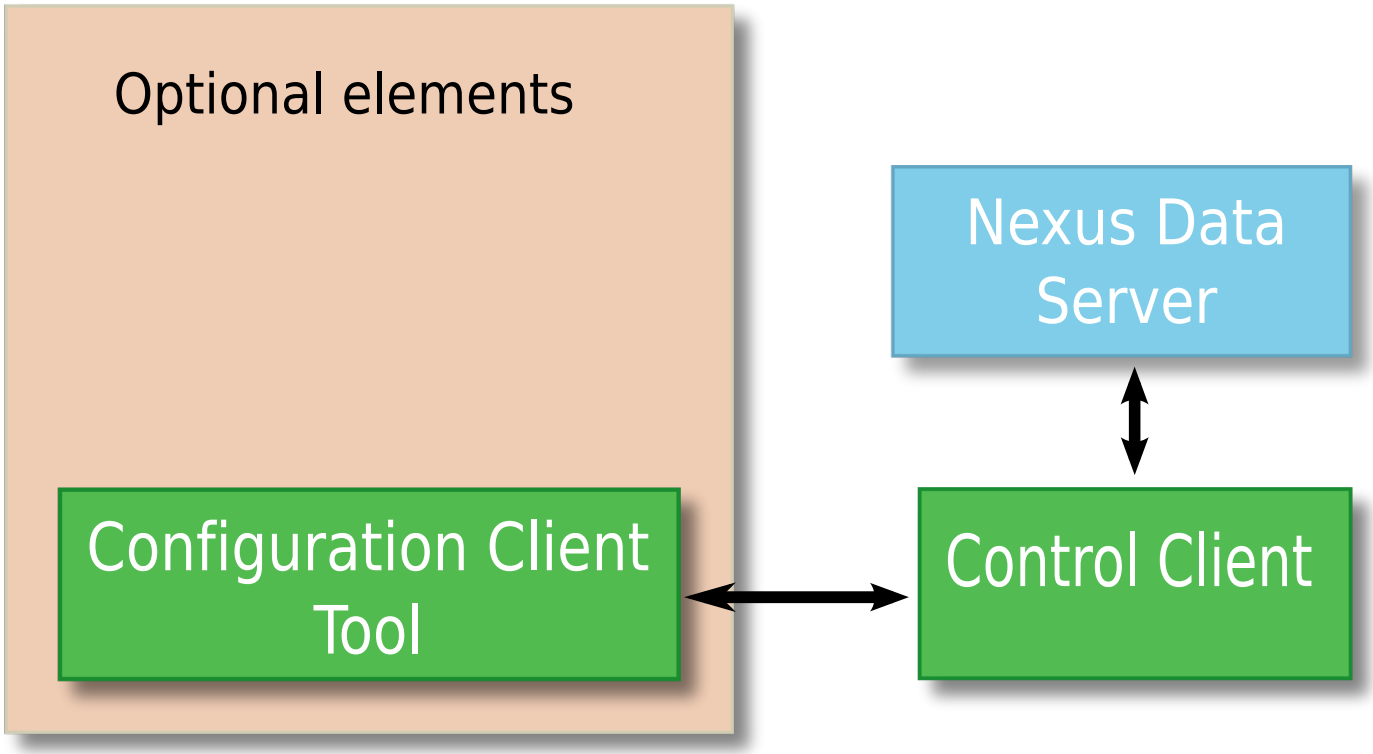


First Component

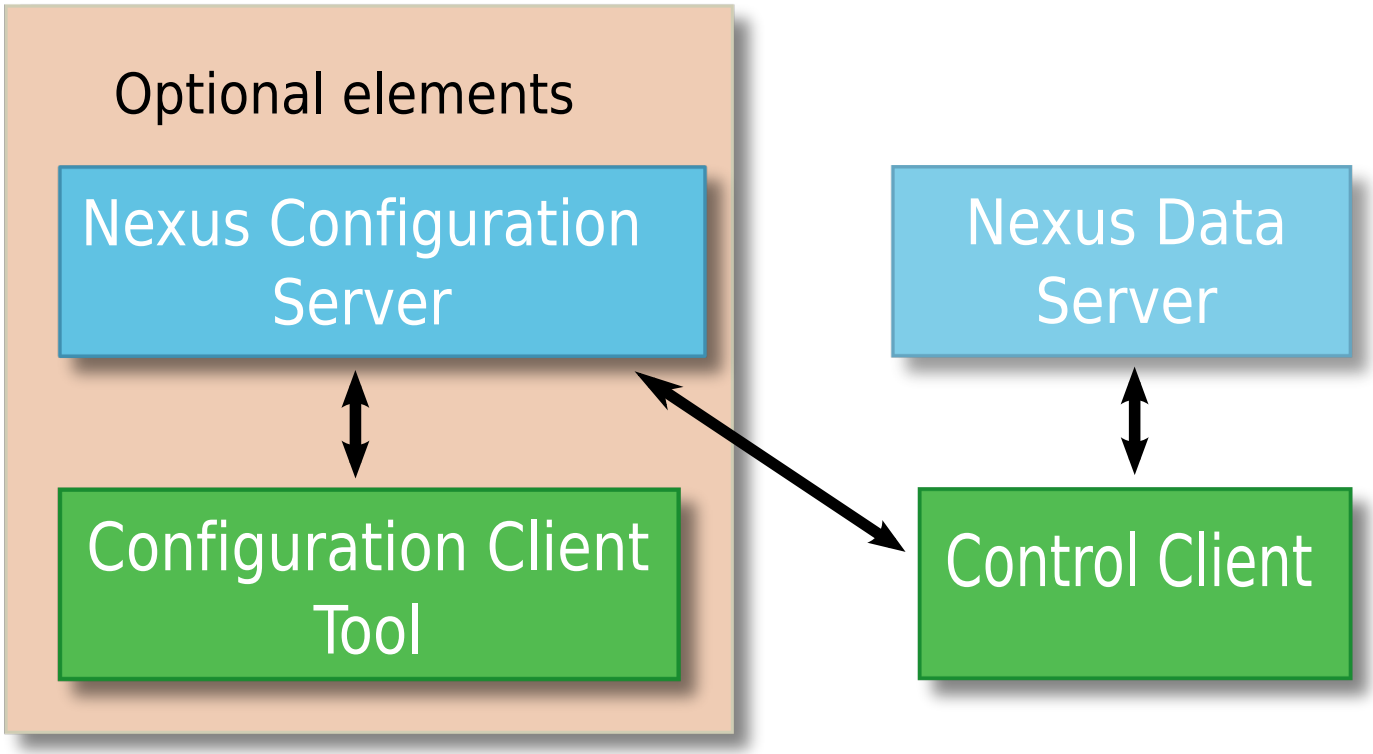
Second Component

Merged Components

Nexus Configuration Server



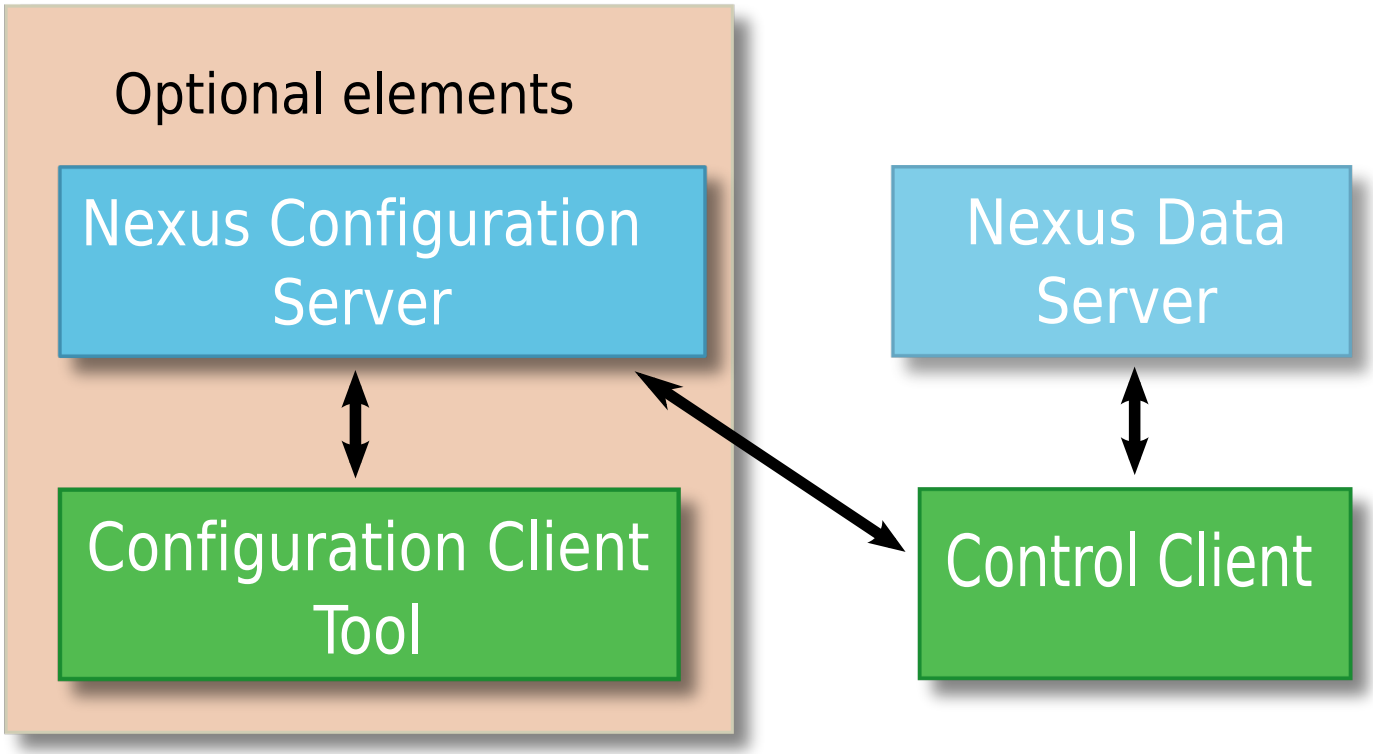
Nexus Configuration Server



Configuration Server

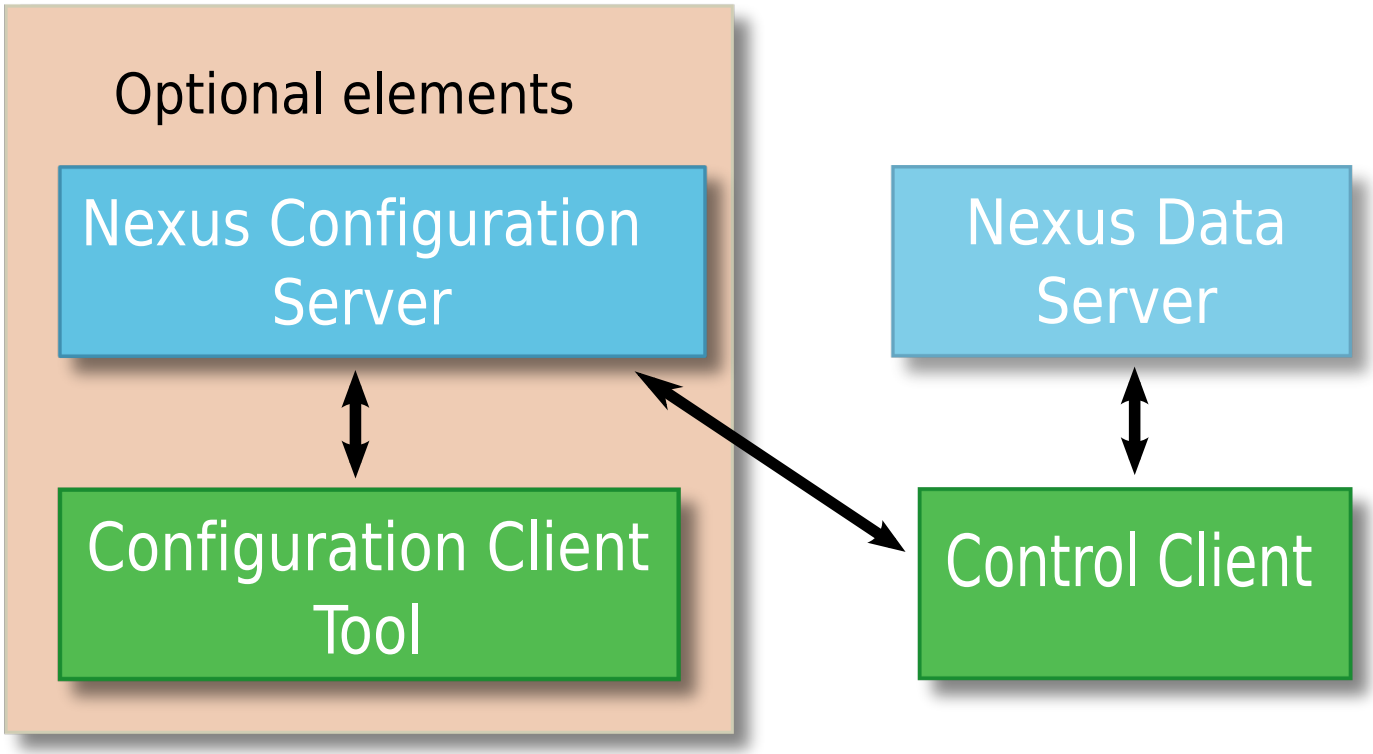
- manages different beamline configurations
- provides the required configuration stream to the Control Client (CC)

Nexus Configuration Server



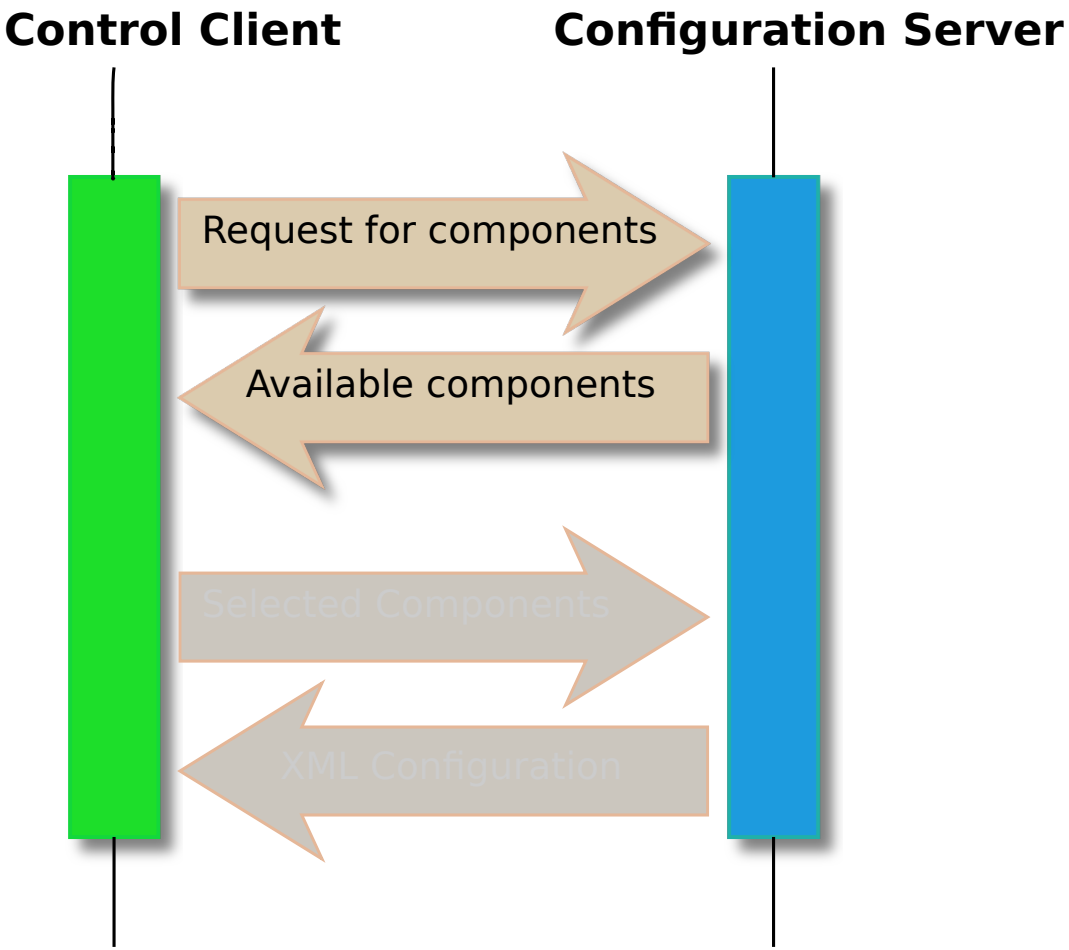
- acts as a central storage facility for experiment configurations
- several Control Clients on a beamline can rely on the same configuration data

Nexus Configuration Server

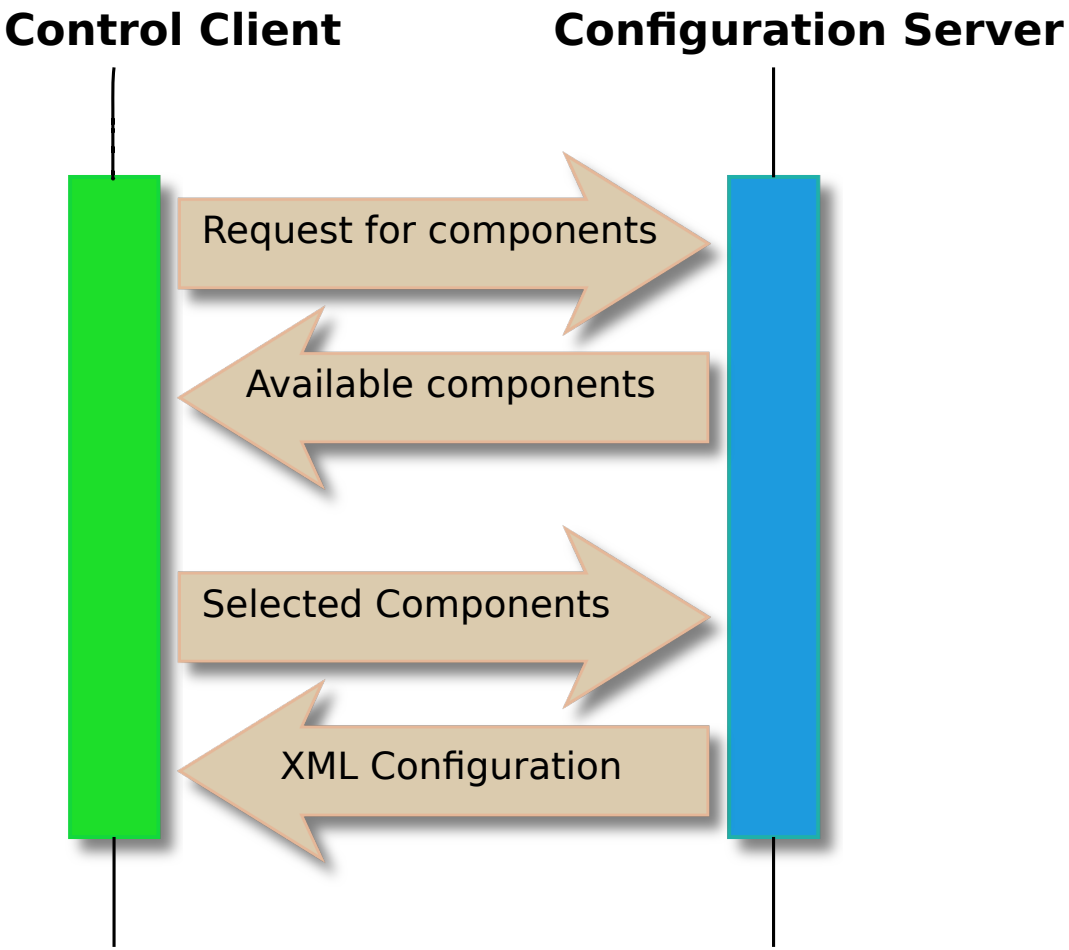


- back-end MySQL DB contains components and datasources
- when final configuration is built components are filled with required datasources and merged

Nexus Configuration Server



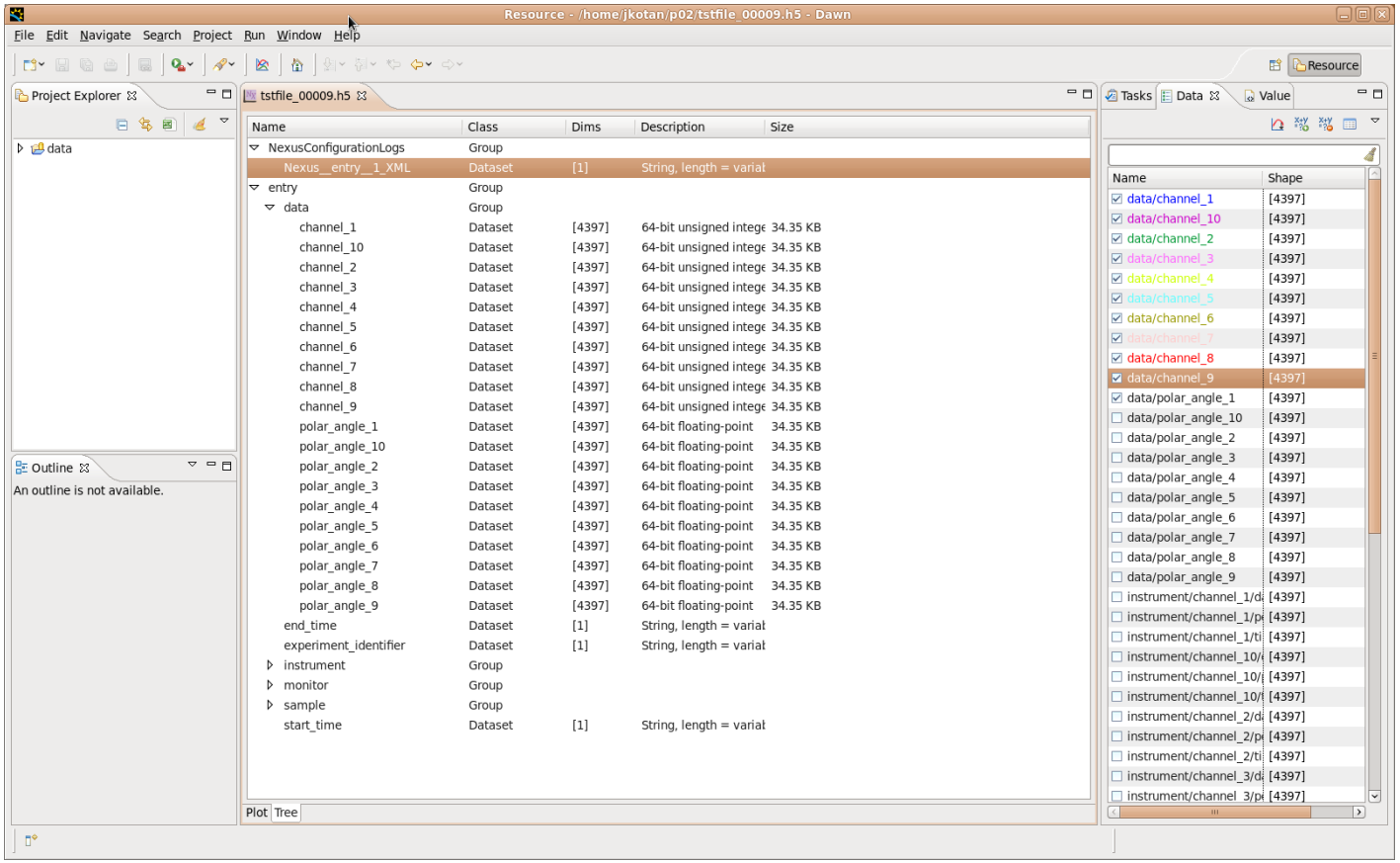
Nexus Configuration Server



Control client does not have to take care about creating the configuration

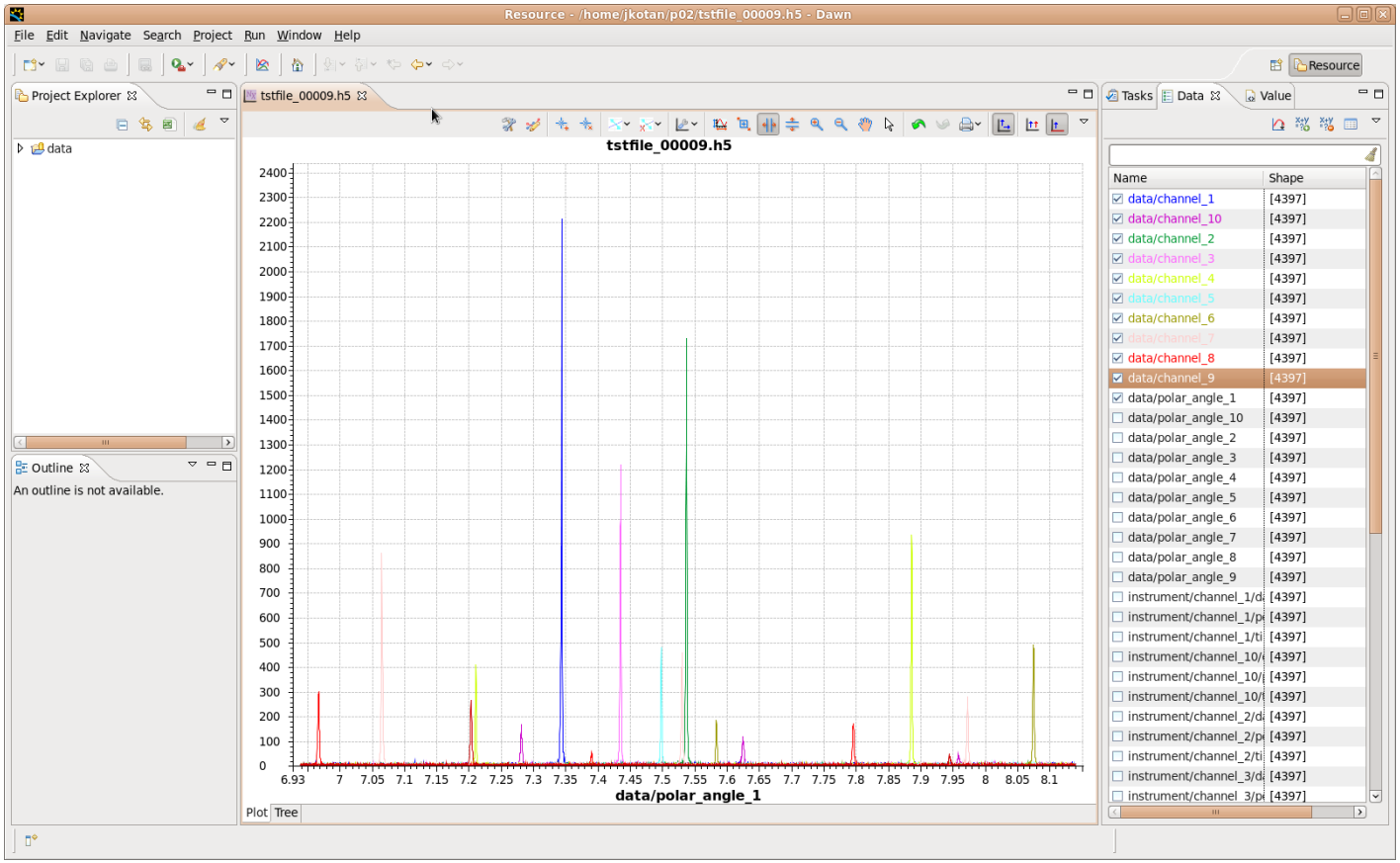
First data on P02

Currently we are deploying the software
at P02/P03 PETRA III beamlines



First data on P02

Currently we are deploying the software
at P02/P03 PETRA III beamlines



Conclusions

Storage Software

- PNI C++ libraries v 0.9.1
- Tango Data Server v 1.1.4
- Component Designer (Configuration Client Tool) v 1.2.1 under tests
- Configuration Server v 1.1.2
- Sardana Control Client v 1.0.2
- Command Line Tools v 1.0.0

Conclusions

Storage Software

- PNI C++ libraries v 0.9.1
- Tango Data Server v 1.1.4
- Component Designer (Configuration Client Tool) v 1.2.1 under tests
- Configuration Server v 1.1.2
- Sardana Control Client v 1.0.2
- Command Line Tools v 1.0.0

Thank You