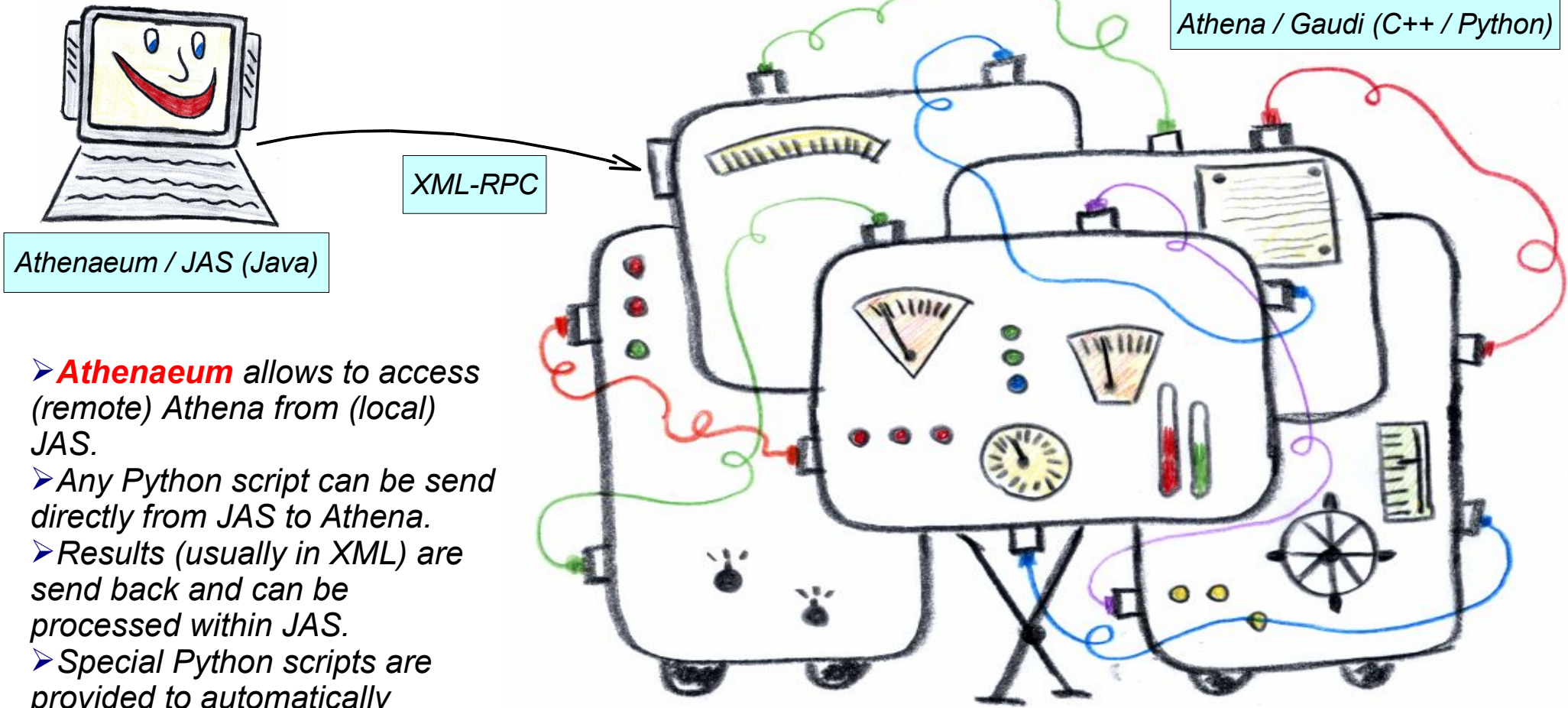




Athenaeum

Using **J**ava **A**nalysis **S**tudio
as an interface to the
Atlas Offline Framework

FREE **H**EP

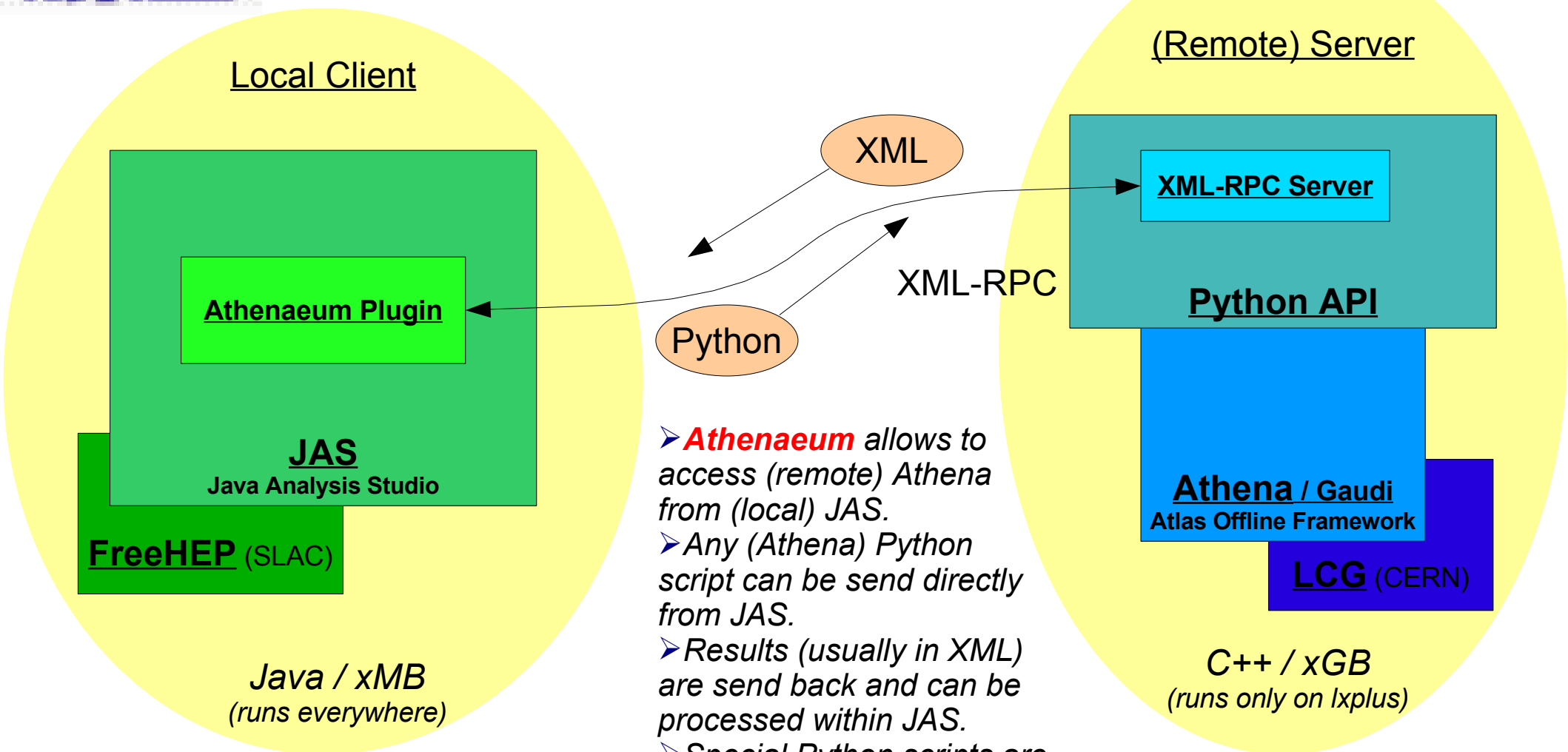


- **Athenaeum** allows to access (remote) Athena from (local) JAS.
- Any Python script can be send directly from JAS to Athena.
- Results (usually in XML) are send back and can be processed within JAS.
- Special Python scripts are provided to automatically present standard Athena data within JAS.



Athenaeum

Using **J**ava **A**nalysis **S**tudio
as an interface to the
Atlas Offline Framework



- **Athenaeum** allows to access (remote) Athena from (local) JAS.
- Any (Athena) Python script can be send directly from JAS.
- Results (usually in XML) are send back and can be processed within JAS.
- Special Python scripts are provided to present Athena data within JAS.



Java Analysis Studio

FREE



GUI

Tree of Objects

Integrated Help with Executable Examples

Java Class

Python/PNuts Script

Python/PNuts Command Line

Graphical/Textual Object Representation

Athena - Athena JAS3 Plugin

- How to Use JAS
- How To Start Athena Python Scripts
- How To Connect to Athena Python
- How to Work with Local Scripts
- How to Work with Remote Scripts
- How to Loop over Events
- How to Work with Proxies of Athena
- How to Write Proxies of Athena
- Existing Proxies:
 - Info (example proxy)
 - Analysis

```

Tuple.py
from java.util import Random
from java.lang import *

af = IAnalysisFactory.create();
tree = af.createTreeFactory().create();
hf = af.createHistogramFactory(tree);
tf = af.createTupleFactory(tree);

t = Random()

columnNames = ["fFlat = 0", "fGauss = 1"]
columnClasses = [Integer.TYPE, Float.TYPE]

tuple = tf.create("tuple", "tupleLabel", columnNames, columnClasses)

for i in range(10000):
    tuple.fill(0, Integer(r.nextInt(20)));
    tuple.fill(1, Float(r.nextGaussian()));
    tuple.fill(2, Float(r.nextFloat()));
    tuple.addRow();

colG = tuple.findColumn("fGauss");
colF = tuple.findColumn("fFlat");
coll = tuple.findColumn("tupleLabel");

h1dI = hf.createHistogram1D(coll);
h1dF = hf.createHistogram1D(coll);
h1dG = hf.createHistogram1D(coll);

```

```

Fit.java
import hep.aida.*;
import java.util.Random;

public class Fit
{
    public static void main(String[] args)
    {
        // Create factories
        IAnalysisFactory analysisFactory = IAnalysisFactory.create();
        ITreeFactory treeFactory = analysisFactory.createTreeFactory();
        ITree tree = treeFactory.create();
        IPlotter plotter = analysisFactory.createPlotterFactory().create("Fit.java Plot");
        IHistogramFactory histogramFactory = analysisFactory.createHistogramFactory(tree);
        IFunctionFactory functionFactory = analysisFactory.createFunctionFactory(tree);
        IFitFactory fitFactory = analysisFactory.createFitFactory();

        IHistogram1D h1 = histogramFactory.createHistogram1D("Histogram 1D", 50, -3, 3);

        Random r = new Random();

        for (int i=0; i<100000; i++) {
            h1.fill(r.nextGaussian());
            h1.fill(r.nextDouble()*10-5);
        }

        IFunction gauss = functionFactory.createFunctionFromScript("gauss", 1, "background+a*exp(-b*x*x)");
        gauss.setParameter("a", h1.maxBinHeight());
        gauss.setParameter("mean", h1.mean());
    }
}

```

	fGauss	fFlat
	-0.030490829	0.6818458
	1.5152388	0.015196621
	-1.3673693	0.53082275
	-0.2744129	0.5291154
	-0.41130975	0.19538856
	1.0221152	0.5871475
	1.2230748	0.27729005
	-1.2411752	0.38542134
	-0.33711454	0.9737252
	1.0957098	0.7615878
	-0.8738934	0.6435949
	0.64157856	0.044230163
	-1.3997235	0.148925698
	-0.096552275	0.8335187
	0.317224	0.8393044
	0.14300768	0.8137784
	0.71282625	0.51521276
	-0.001987687	0.2121237
	-1.1803523	0.51602493
	-1.0319808	0.026451164
	1.1575664	0.6023681

Mean: -8.9443E-4
Rms: 1.016C
OutOfRange: 1

h2d

Entries: 999E
Mean: 1.0055E-3
Rms: 1.015E
OutOfRange: 0.4994C
Mean: -0.2902E
Rms: 0.6

JAS is a GUI based on FreeHEP library. FreeHEP is Java equivalent of CERNLIB, Root, OpenScientist,...

Most Functionality implemented by Plugins. They can be loaded dynamically (over network).

>>> print "Hello"
Hello
>>>

9.59/11.8MB

see <http://jas.freehep.org/jas3> for details



Open Connection to Athena



*On Client
(Any platform with JAS + Athenaem Plugin)*

```
$ athena.py -i -s jobOptions.py
```

```
.....  
XML-RPC server 'atldbdev01.cern.ch:48966' created  
method 'process()' registered  
Waiting for requests...
```

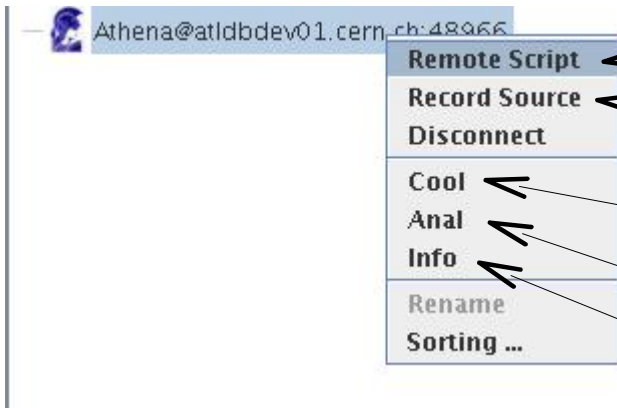
*On Server
(Linux with Athena)*

```
.....  
execfile ("InteractiveServer.py")
```

*Server script written
by Atlantis team
(<http://cern.ch/atlantis>)*

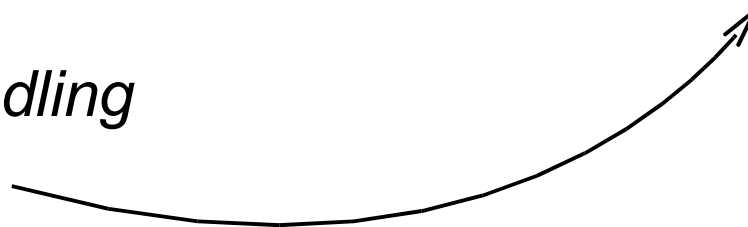


Interact with Athena



- Execute a Python script on Athena Server, get results back
- Steer Athena Event Loop from JAS
- Access Cool (Conditions DB) data
- Access Particle data (prototype)
- Get Information about Athena Server environment (loaded dictionaries, paths,...)

*Registered **Proxies** are implementing concrete handling of connection to specific (Athena) functionality.*





Execute Python on Athena

The screenshot shows the JAS3 web interface. The main window displays a file list with 'Result_1' selected. A callout box labeled 'Script Result' points to this file. Below the file list, a console window shows the execution of a Python script: 'print self'. The output is displayed in a separate window titled 'Result_1@lxplus003.cern.ch:48966', showing the output: '<__main__.InteractiveServer instance at 0xb720830c>', 'SUCCESS'. A callout box labeled 'Output Console' points to the console window. A callout box labeled 'Script to be executed on remote Athena' points to the script content in the console window.

```
File Edit View Tuple Loop Window Athenaem Help
```

Athena@lxplus003.cern.ch:48966

Result_1

```
Athena@lxplus003.cern.ch:48966
```

```
1 print self
```

```
Result_1@lxplus003.cern.ch:48966
```

```
1 <__main__.InteractiveServer instance at 0xb720830c>
```

```
2
```

```
3 SUCCESS
```

```
Connecting to http://lxplus003.cern.ch:48966
```

```
On http://lxplus003.cern.ch:48966 executing:
```

```
-----
```

```
print self
```

```
Result:
```

```
-----
```

```
<__main__.InteractiveServer instance at 0xb720830c>
```

classpath:/org/freehep/jas/web/relnotes.html 4.65/6.13MB

User can mix Python running within JAS and Python running in a (remote) Athena. Athena Python scripts could be moved to JAS.



Steer Athena Event Loop

Next Event,...

Athena interpreted as a set of Records (Events)

```
1 import org.freehep.record.loop.event.RecordAdapter;
2 import org.freehep.record.loop.event.RecordSuppliedEvent;
3
4 import net.hep.atlas.Core.Athenaeum.JAS3Plugin.AthenaClient;
5
6 public class EventLoop extends RecordAdapter {
7
8     public void recordSupplied(RecordSuppliedEvent event) {
9         AthenaClient athena = (AthenaClient)event.getRecord();
10        try{
11            System.out.println(athena.execute("print self"));
12        }
13        catch (Exception e) {
14            System.err.println(e);
15            e.printStackTrace();
16        }
17    }
18 }
19 }
20 }
```

Python script executed on each Event Results analyzed locally

Output Console

```
Connecting to http://lxplus003.cern.ch:48966
On http://lxplus003.cern.ch:48966 executing:
-----
theApp.initialize()
```

classpath:/net/hep/atlas/Core/Athenaeum/JAS3Plugin/doc-files/EventLoop.java 6.41/8.77MB



Remote Proxy

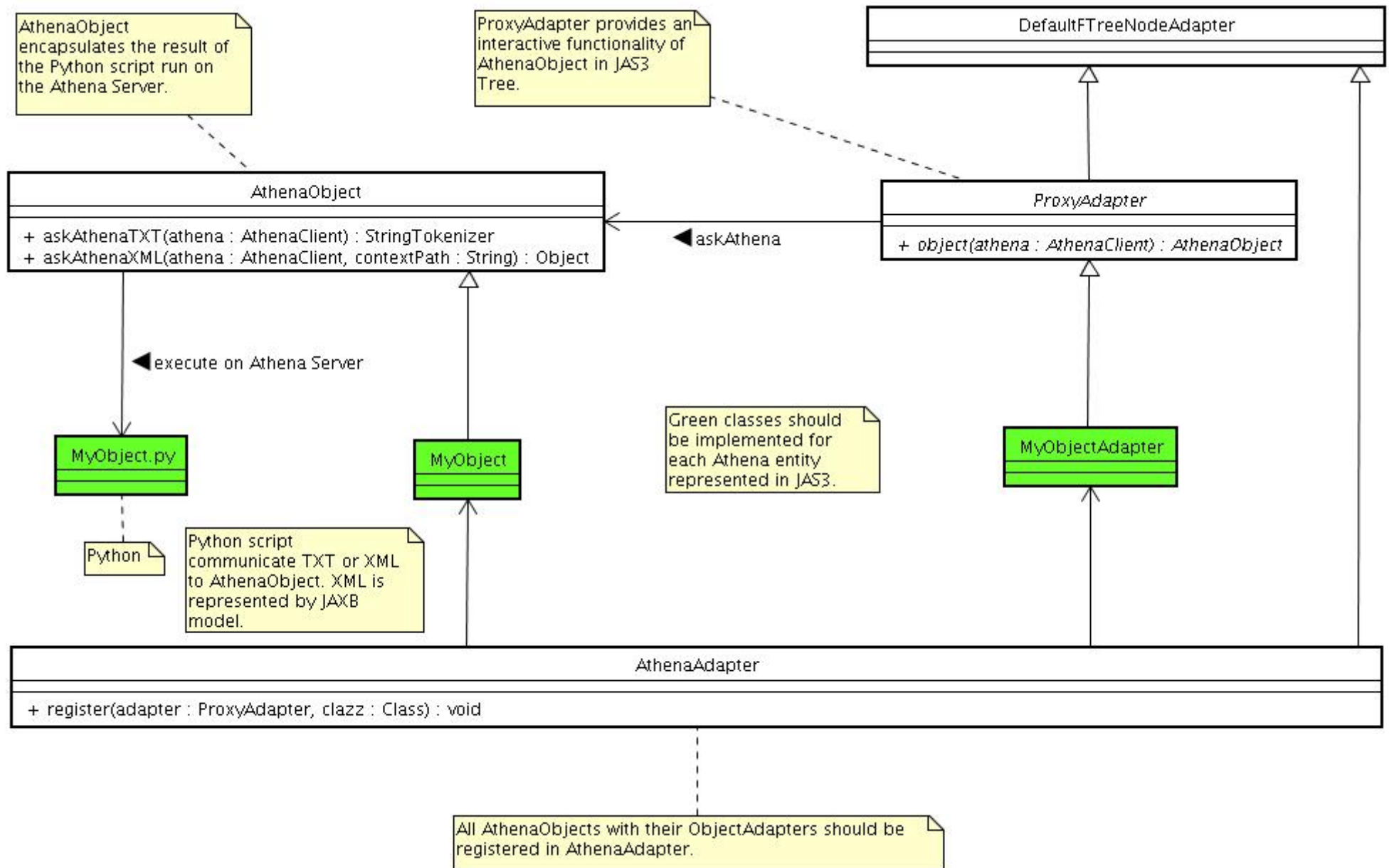
*Registered **Proxies** are implementing concrete handling of connection to specific (Athena) functionality. They are implemented by:*

- Athena Python script to extract data from Athena*
- JAS wrapper to present/handle data inside JAS*
- XML schema to describe data*

When implementing pre-defined interfaces from Athenaeum, those Proxies will make themselves automatically available inside JAS system in an organic way.

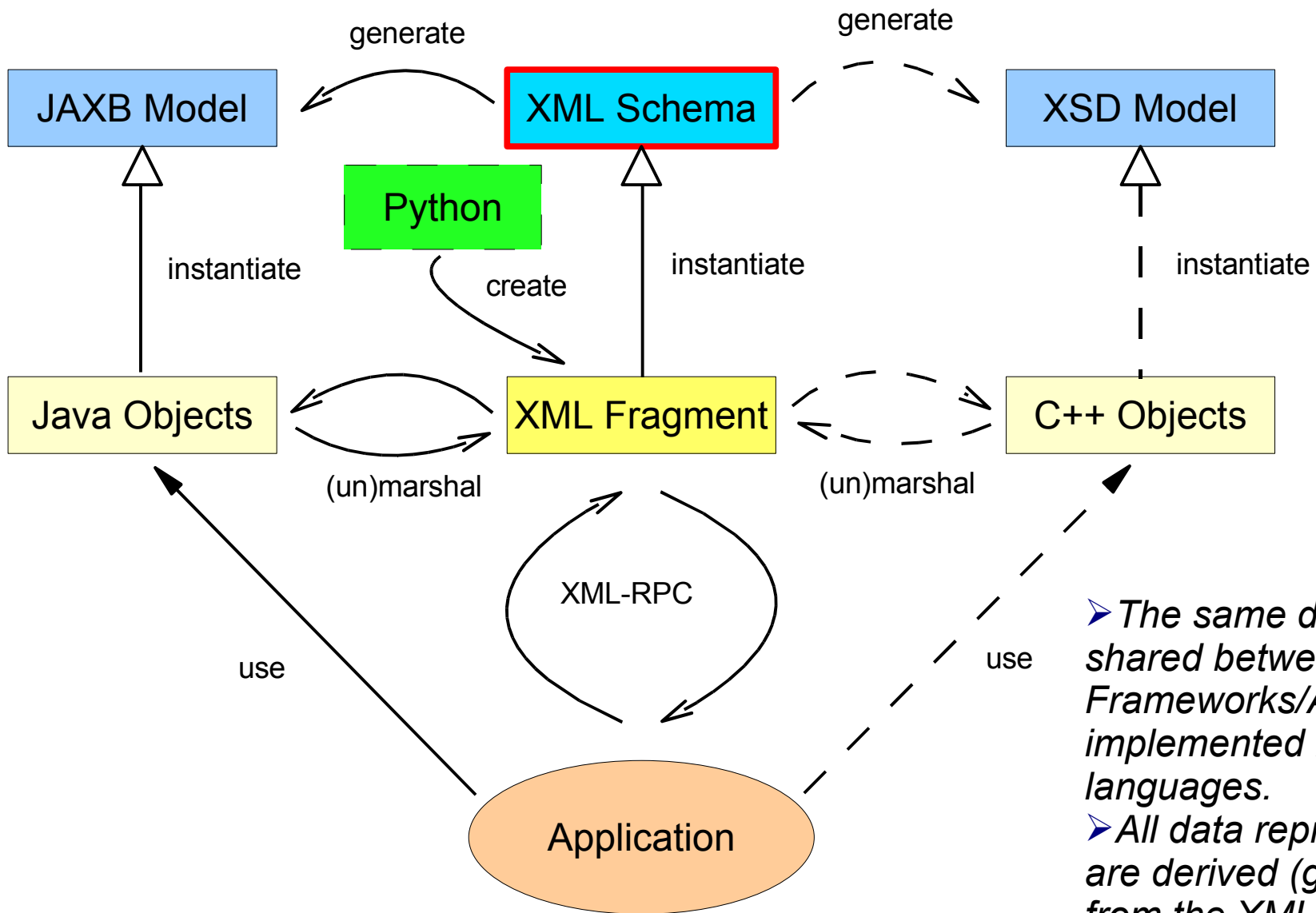


Construction of Proxy





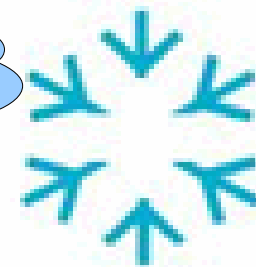
XML Schema Representations



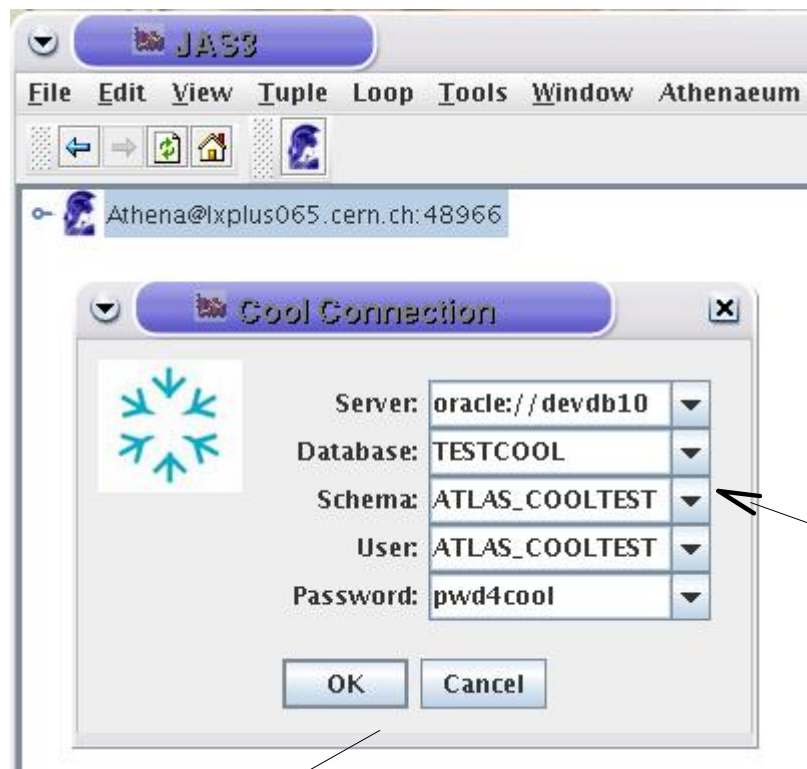
- The same data are shared between different Frameworks/Applications implemented in different languages.
- All data representations are derived (generated) from the XML Schema.



LCG Conditions Database
(C++ / Python / SQL)



Interact with Cool



JAS + Athena
Client

Athena/PyCool
Server



Cool DB
Server

- Open connection to Cool DB
- Interpret data (as AIDA NTuples)
- Show data as HTML
- Show data as XML
- Analyse data
- Show Python script used to get data

Athena@lxplus058.cern.ch: 14420

Cool@oracle:devdb10[TESTCOOL,ATLAS_COOLTEST]

- Convert into NTuple
- Show as HTML
- Show XML
- Analyse
- Show Script
- Rename



Cool XML

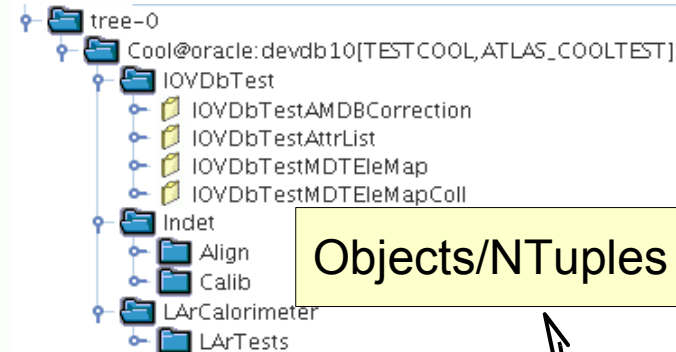


/IOVDbTest/IOVDbTestAttrList[1326]

Inserted: 24/10/2005 at 4:58:13
FOLDER_IOVTABLENAME: TESTCOOL_F1326_IOVS
FOLDER_TAGTABLENAME: TESTCOOL_F1326_TAGS
FOLDER_IOV2TAGTABLENAME: TESTCOOL_F1326_IOV2TAG
Channels: 0,
TypeName: AthenaAttributeList
TimeStamp: run-event
Symlinks:
ServiceType: 71
Clid: 40774348

HTML View

period	object	channel	insert	xPosition [float]	id [int]	name [string]
run:0 event:0 run:3 event:3	8	0	24/10/2005 at 5:0:17	25.0	7	TestAttrList
run:3 event:3 run:4 event:3	14	0	24/10/2005 at 5:0:41	25.0	7	TestAttrList
run:4 event:3 run:2147483647 event:4294967295	13	0	24/10/2005 at 5:0:41	25.0	7	TestAttrList



Objects/NTuples

Java

/IOVDbTest/IOVDbTestAttrListColl[1327]

Inserted: 24/10/2005 at 4:58:46
FOLDER_IOVTABLENAME: TESTCOOL_F1327_IOVS
FOLDER_TAGTABLENAME: TESTCOOL_F1327_TAGS
FOLDER_IOV2TAGTABLENAME: TESTCOOL_F1327_IOV2TAG
Channels: 16, 26, 36, 46, 56,
TypeName: CondAttrListCollection
TimeStamp: run-event
Symlinks:
ServiceType: 71
Clid: 1238547719

```

<folder name='/IOVDbTest/IOVDbTestMDTEleMapColl'
  id='1331' day='24' month='10' year='2005' hour='5' minute='2' second='36' >
  <attributes>
    <attribute name='FOLDER_IOVTABLENAME' value='TESTCOOL_F1331_IOVS'/>
    <attribute name='FOLDER_TAGTABLENAME' value='TESTCOOL_F1331_TAGS'/>
    <attribute name='FOLDER_IOV2TAGTABLENAME' value='TESTCOOL_F1331_IOV2TAG'/>
  </attributes>
  <channels>
    <channel>0</channel>
  </channels>
  <description>
    <timeStamp>run-event</timeStamp>
    <addrHeader><address_header service_type="71" clid="155887251" /></addrHeader>
    <typeName>IOVDbTestMDTEleMapColl</typeName>
  </description>
  <signature>
    <item name='PoolRef' type=' string'/>
  </signature>
  <payload since='run:0 event:0' until='run:2147483647 event:4294967295'
    object='1' channel='0' day='24' month='10' year='2005' hour='5' minute='3' second='6' >
    <entry name='PoolRef' value='[...]'/>
  </payload>
</folder>
  
```

XML View

XSLT

XML Schema

Python



Work with Cool (1)

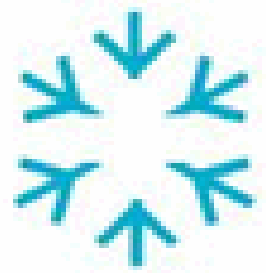


- Data can be represented as
 - XML
 - Objects
 - Tree
 - (AIDA) NTuples
 - HTML
- and accessed
 - via GUI
 - using scripting interface (Java, Python, Pnuts)
 - using API (Java, Python)

```
athenaem
├── server="lxplus058.cern.ch:14420"
├── cool
│   ├── dbname="TESTCOOL"
│   ├── password="pwd4cool"
│   ├── schema="ATLAS_COOLTEST"
│   ├── server="oracle://devdb10"
│   └── user="ATLAS_COOLTEST"
│   ├── #comment
│   ├── #comment
│   └── folder
│       ├── day="12"
│       ├── hour="15"
│       ├── id="2707"
│       ├── minute="58"
│       ├── month="0"
│       ├── name="/IOVDbTest/IOVDbTestAMDBCo"
│       ├── second="50"
│       └── year="2006"
│       ├── attributes
│       ├── channels
│       ├── description
│       ├── signature
│       └── payload
│           ├── channel="0"
│           ├── day="12"
│           ├── hour="16"
│           ├── minute="1"
│           ├── month="0"
│           ├── object="8"
│           ├── second="40"
│           ├── since="1970/01/01 Thu 00:00:00"
│           ├── until="1970/01/01 Thu 00:00:35"
│           └── year="2006"
│       ├── entry
│       └── #comment
│       └── payload
│       └── folder
│       └── folder
```




Work with Cool (2)



Athena@ixplus065.cern.ch:48966

Cool

tree-0

Cool@oracle:devdb10[TESTCOOL, ATLAS_COOLTEST]

IOVDbTest

- IOVDbTestAMDBCorrection
- IOVDbTestAttrList
 - since
 - until
 - object
 - channel
 - day
 - month
 - year
 - hour
 - minute
 - second
 - xPosition
 - id
 - name
- IOVDbTestMDTEleMap
- IOVDbTestMDTEleMapColl

Indet

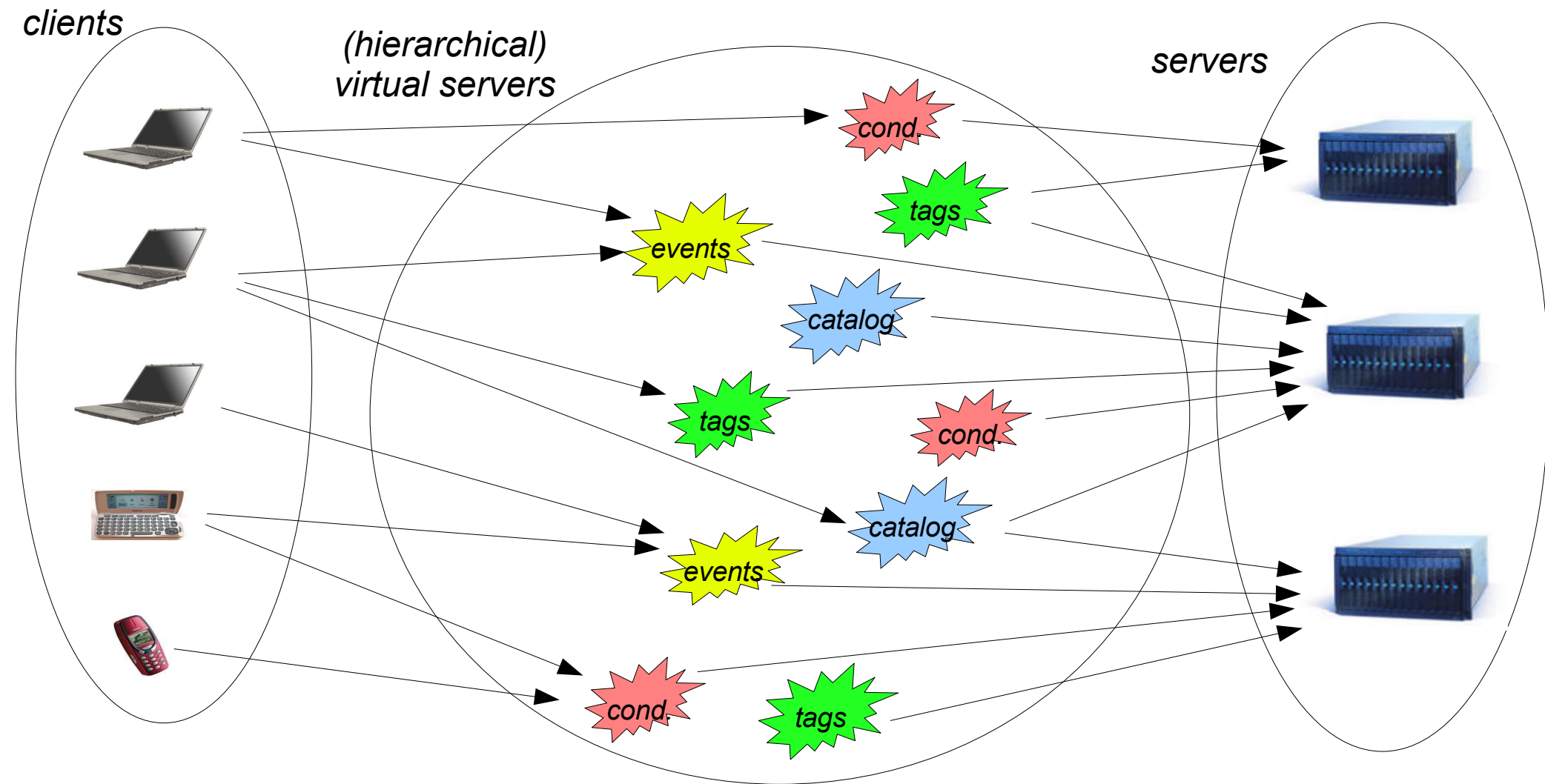
- Align
 - TRT_DF_B0
 - TRT_DF_B1
 - TRT_DF_B2
- Calib

since	until	obj...	cha...	day	month	year	hour	minute	second	xPosition	id	name
run:0 e...	run:3 e...	8	0	31	9	20...	18	46	28	25.0	7	TestAttrList
run:3 e...	run:4 e...	14	0	31	9	20...	18	46	51	25.0	7	TestAttrList
run:4 e...	run:21...	13	0	31	9	20...	18	46	51	25.0	7	TestAttrList

- Data can be represented as
 - XML
 - Objects
 - Tree
 - (AIDA) NTuples
 - HTML
- and accessed
 - via GUI
 - using scripting interface (Java, Python, Pnuts)
 - using API (Java, Python)



Distributed Interactive Environment Architecture Project



- Only user code + access layer in clients
- Data access and standard processing in servers
- Orchestration and optimization in virtual servers
- Passed data described by common (XML) Schema

- **Athenaeum**
- SQLTuple/ColMan (see poster 331)
- Sequoia (see poster 331)



Architecture Advantages

- Light local client
 - Running on any platform, any release
 - Fully interactive GUI, scripting and API in several languages
 - Easily extensible by modular plugins
- Server on a powerful machine, close to data, replicated and hierarchised when useful
- Standard communication protocols
 - XML-RPC for the Control Flow and small data
 - Eventually performant protocols (JDBC, xrootd,...) for big data



Problems

- PyAthena (Python API to Athena)
 - **Incomplete** (only a subset of C++ API is available via Python)
 - **Undocumented** (C++ Doxygen is not enough for documentation of its Python API; it is not easy to guess the meaning of weakly-typed methods; code fragments on Web/Wiky are often out-of-date)
 - **Unstable** (too many things change too often)
 - The **C++** Framework is **still there**, it just hidden (its problems will pop up from time to time)
- Data
 - **No abstract data definition** is available, the actual data model is hidden very deep in the C++ header files forest
 - Athenaeum **XSD Schema** has been written for data passed around; XML, Java, Python and C++ incarnations can be created from them



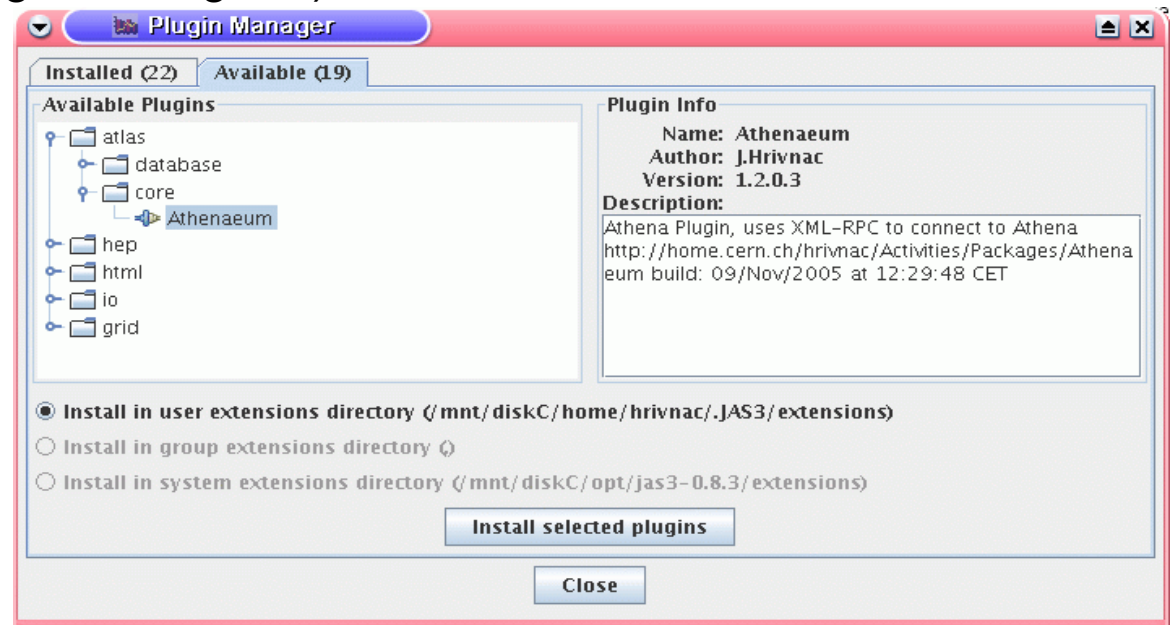
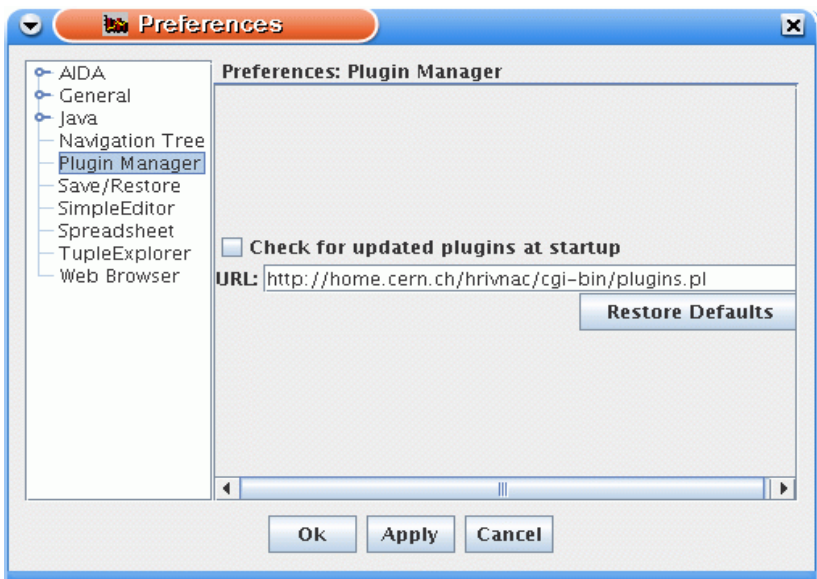
To Do Next

- Generalization for other Monolithic Frameworks
 - there is nothing special about Athena/Gaudi, any Framework with functional XML-RPC server would work fine
- Lazy & Compressed data transport (to speed up)
 - XML-aware compression, MPEG-7 compression, binary XML,... can give size down to about 2x compressed Root files size of the same data
- User-customizable XSLT
- More Proxies (Analysis objects, Generic StoreGate access, ...)
- Athena (remotely) startable from Athenaeum (so that user does not have to start the server herself)
- Deployment of a network of hierarchical *Athena Servers*



How To Start

- Within CERN AFS:
 - `./afs/cern.ch/sw/java/share/bin/setjdk sun 1.5.0_02`
 - `/afs/cern.ch/atlas/offline/external/JAS/jas3/jas3`
- Elsewhere (any platform):
 - Get Java 1.5
 - Get JAS from <http://jas.freehep.org/jas3> (Linux, MS, MacOSX,...)
 - Set Plugin Server (View - Preferences...)
 - Get Plugin (View – Plugin Manager...)





Help

- <http://home.cern.ch/hrivnac/Activities/Packages/Athenaeum>
- <https://uimon.cern.ch/twiki/bin/view/Atlas/HowToUseJAS>
- JAS integrated Help (with executable examples)

Athenaeum - Athena JAS3 Plugin

- [How to Use JAS](#)
- [How To Start Athena Python Server](#)
- [How To Connect to Athena Python Server](#)
- [How to Work with Local Scripts](#)
- [How to Work with Remote Scripts](#)
- [How to Loop over Events](#)
- [How to Work with Proxies of Athena Objects](#)
- [How to Write Proxies of Athena Objects](#) (for experts)
- Existing Proxies:
 - [Info](#) (example proxy)
 - [Analysis](#)
 - [Cool](#)
- [How to Use in a Standalone Environment](#)
- [Where to Find More](#)

How to Use JAS

The JAS3 documentation is [available](#). JAS3 also contains its integrated help system with executable examples.

How To Start Athena Python Server

Athena XML-RPC Server can be started using simple [Python Script](#) (this script has been developed by the [Atlas](#) team):

- Place the script into you run directory.
- Put following lines at the end of your 'joboptions.py' script:

```
# initialise Athena and go the first Event
# (following two commands can be omitted here
# and called remotely once the Server is started)
theApp.initialize()
theApp.nextEvent()
# load the Server
execfile ("InteractiveServer.py")
# instantiate the Server
server = InteractiveServer()
# start the Server
server.start()
```
- Call Athena using following command:

```
athena.py -i joboptions.py -s
```
- The message with the Server URL will be printed out. It should be used for the connection from JAS.

How To Connect to Athena Python Server