SeisHub

SeisHub Workshop at the MESS 2013

Lion Krischer

Ludwig-Maximilians-University in Munich
Department of Earth and Environmental Sciences
Geophysics

Sudelfeld, March 14 2013

Rough Schedule

Morning I: Introduction to SeisHub

Morning II: Converting Data to XML

Afternoon: Extending SeisHub

Why a new Tool?

- Most seismic processing tools are mainly limited to classic three-component recordings and cannot easily handle collocated multi-component data (e.g., pressure, temperature, rotational motions, tilt, accelerometer, GPS, GeoTIFF, 4D-Data, . . .)
- Very hard to extend and store other data

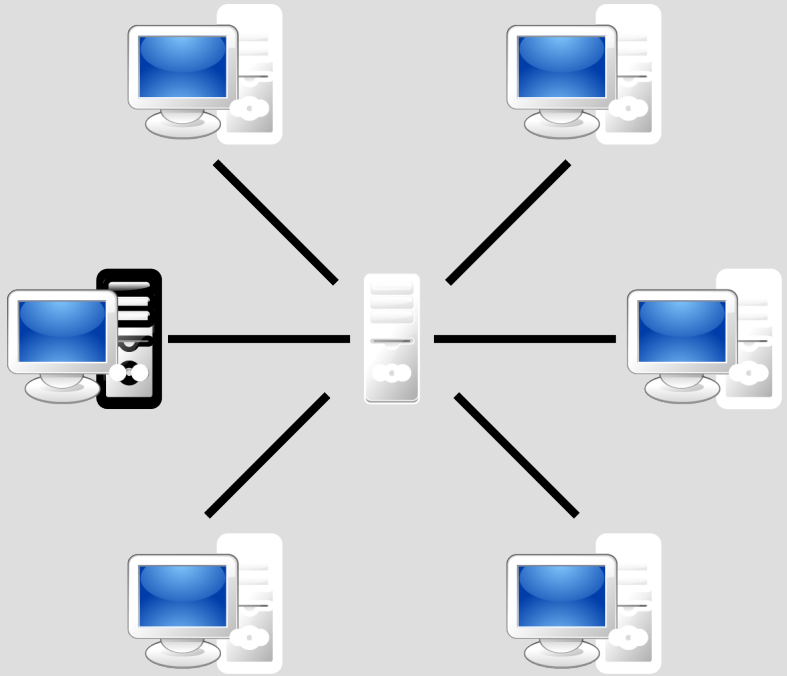
Why SeisHub?

- Can deal with continuous, event, and campaign based data
- Can be accessed from the outside via a web interface
- Multi-User
- Easy to extend
- Flexible - can be adapted on-the-fly without corruption $\rightarrow$ in stark contrast to the classical SQL approach
- Scales to very large data sets
- **Platform independent and open source**
- Most basic client is the browser

SeisHub as a Server

- SeisHub can act a web server
- Multiple users can send requests to it

$\Rightarrow$ Collaborative working independent of location



Crash Course in Technologies Used by SeisHub

XML

XML

- **Extensible Markup Language**
- Well known example: XHTML
- An element is surrounded by an opening and a closing tag
- Elements can be arbitrarily nested
- Tags can be specialized with the help of attributes
- Human and machine readable

```
<?xml version="1.0" encoding="UTF-8" ?>
<seismic_event>
   <magnitude type="Mw">7.0</magnitude>
   <location>
      <longitude>11.669197</longitude>
      <latitude>48.261545</latitude>
   </location>
</seismic_event>
```

XML

A large number of technologies are built on top of XML:

- XPath
- XML Schemata (XSD)
- Transformations (XSLT)
- XML Databases

$\Rightarrow$ **Widely available support and tools**

Examples from seismology:

- QuakeML
- StationXML
- XML-SEED

XPath

- A standardized way to access information inside an XML document
- Complex queries possible but usually not necessary

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<seismic_event>
   <magnitude type="Mw">7.0</magnitude>
   <location>
      <longitude>11.669197</longitude>
      <latitude>48.261545</latitude>
   </location>
</seismic_event>
```

XPath expression to access the longitude:

```
/seismic_event/location/longitude
```

Databases

Relational Databases

- Store data in tables similar to Excel
- The tables have to be defined before data is entered
- Every table row has a fixed data type
- High performance
- One row is one tuple whose items are in relation to each other

Poets

Id	FirstName	Surname	Age
1	Mongane	Afrika	62
2	Stephen	Serote	58
3	Tatumkhulu	Watson	29

Relational Databases

- Data is distributed over several tables with the use of foreign keys
- **Database Normalization**

Poets

Id	FirstName	Surname	Age
1	Mongane	Afrika	62
2	Stephen	Serote	58
3	Tatumkhulu	Watson	29

Poems

Id	Title	Poet
1	Thrones of Darkness	2
2	Wakening Night	1
3	Once	3

SQL

- **Structured Query Language**
- Supported by most relational database systems

Example:

```
SELECT FirstName, Surname, Age
FROM Poets
WHERE Age <= 40
ORDER BY Surname
```

SQLite and PostgreSQL

SQLite	**PostgreSQL**
File Based	Server Based
“No” installation	Complex installation
Slow	Fast
Single User	Multiple Users

XML Databases

- Enable the use of all XML related technologies

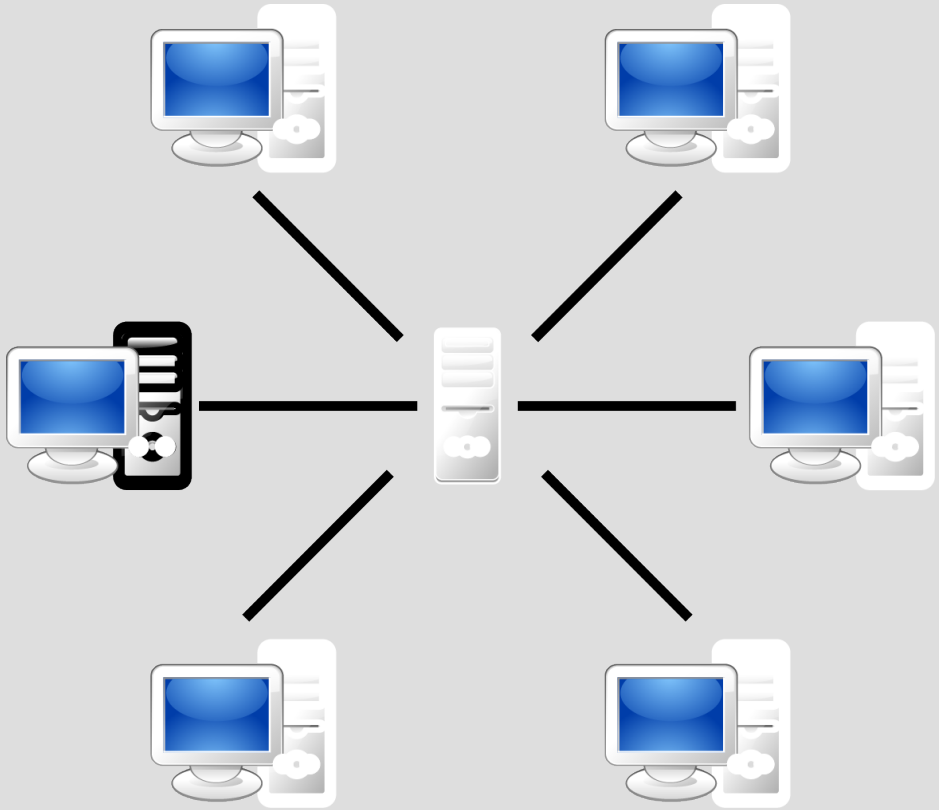
Two different types of XML databases:

- **Native XML databases:** Directly store XML documents - usually very slow
- **XML-enabled databases:** Classical relational database with the benefit of XML input and output

Web Technologies

Client-Server Model

- Clients send requests to the server
- Server answers with a response



HTTP

- **Hypertext Transfer Protocol**
- Used to transfer data in the internet
- Works by request and response, e.g. one party sends a request and the other a response
- Contains headers
- Different request methods:
  - **GET:** Request some data
  - **POST:** Send data
  - **PUT:** Send data, should always directly store data
  - **DELETE:** Delete data
  - ...

  $\Rightarrow$ RESTful service

SeisHub

What is SeisHub?

- Developed by Robert Barsch in the course of his PhD thesis
- Written in Python, backed by Twisted

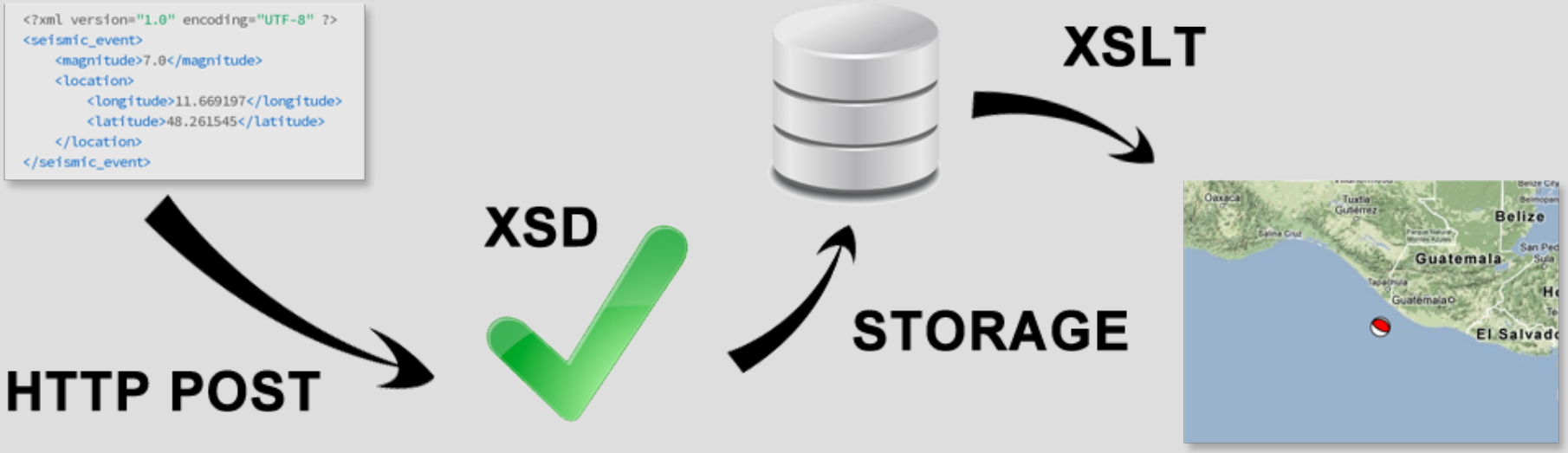
- An XML database
- Not limited to XML data
- An easy way to access arbitrary Python functions from the web

SeisHub's XML database

- Integrated way to handle arbitrary XML data

- **Validation** - assures integrity of your data (XSD)
- **Transformation** - makes it accessible by humans (XSLT)

- Indexes defined values and stores them in a relational database
- Query via SQL, XPath, or HTTP



SeisHub's XML database

- Data is categorized by defining data types
- Upload, download, delete, and modify operations supported
- **Integrated versioning** - Keep track of the evolution of data

- Efficiently search over large datasets
- Extract user-defined values of interest

- Backed by PostgreSQL or SQLite
- Can also store non-XML data but this require more effort

Web Interface

- Complete administration of the XML database via a RESTful interface
- Map URLs to Python functions
- User Management and Access Control

Practical

Goal

- Use the event based data plugin developed yesterday
- Develop a SeisHub plug-in that enhances SeisHub with the possibility of storing reported felt-seismicity records per event

Steps:

1. Convert the data to XML
2. Define the basic plug-in structure
3. Define a mapper producing a map from the stored results

Data

- The data is from the USGS for the L'Aquila earthquake
- Located at */home/mess/Desktop/seishub/data/cdi_zip_laquila.txt*
- One line is one data point and should later correspond to one XML file

```
# Columns: ZIP/Location,CDI,No. of responses,Epicentral ...
"Abetone::Toscana::Italy",3.4,1,294,44.1300,10.6700,0, ...
"Acquafondata::Lazio::Italy",6.2,2,101,41.5500,13.9500, ...
"Acquaviva␣Picena::Marken::Italy",4.6,1,76,42.9300, ...
"Agnone::Molise::Italy",2.0,1,104,41.8000,14.3700,0, ...
...
```

Data Conversion

```
"Abetone::Toscana::Italy",3.4,1,294,44.1300,10.6700,0, ...
```

TO

```
<?xml version='1.0' encoding='UTF-8'?>
<seismic_intensity>
  <event>laquila</event>
  <location>Abetone,Toscana,Italy</location>
  <intensity>3.4</intensity>
  <number_of_responses>1</number_of_responses>
  <epicentral_distance>294</epicentral_distance>
  <latitude>44.1300</latitude>
  <longitude>10.6700</longitude>
</seismic_intensity>
```

Some Hints - Use the lxml element factory

```
from lxml import etree
from lxml.builder import E
doc = (E.root_tag(
    E.element_1("Hello"),
    E.element_2(E.sub_element("World"))))
string_doc = etree.tostring(doc, pretty_print=True,
    xml_declaration=True, encoding="UTF-8")
print string_doc
```

```
<?xml version='1.0' encoding='UTF-8'?>
<root_tag>
  <element_1>Hello</element_1>
  <element_2>
    <sub_element>World</sub_element>
  </element_2>
</root_tag>
```

Some Hints - Relations between event and report

- There needs to be some way to tell which event a felt seismicity is associated with
- For the sake of simplicity this will be the resource name of the event stored in SeisHub
- In a real world example this relation is better expressed as its own resource

```
<?xml version='1.0' encoding='UTF-8'?>
<seismic_intensity>
  <event>laquila</event>
  <location>Abetone,Toscana,Italy</location>
  <intensity>3.4</intensity>
  <number_of_responses>1</number_of_responses>
  <epicentral_distance>294</epicentral_distance>
  <latitude>44.1300</latitude>
  <longitude>10.6700</longitude>
</seismic_intensity>
```

SeisHub Plug-in

SeisHub Plug-ins

- SeisHub alone does not do very much
- Functionality comes with plug-ins

Existing plug-ins:

- **seishub.plugins.seismology:** Continuous data streams at local data center scale, events, and station metadata
- **seishub.plugins.event_based_data:** Event based waveforms and synthetics
- **seishub.plugins.exupery:** Volcano fast response system - GPS, InSAR, seismological, and other data

Plug-in Structure

```
seishub.plugins.mess/
└── seishub/
    ├── __init__.py
    ├── plugins/
    │   ├── __init__.py
    │   └── mess/
    │       ├── __init__.py
    │       ├── mappers.py
    │       └── package.py
    └── setup.py
```

SeisHub Packages

- SeisHub organized data and functionality into so called packages
- A package can be interpreted as a folder

```
from seishub.core.core import Component, implements
from seishub.core.packages.interfaces import IPackage


class MESSPackage(Component):
    implements(IPackage)
    package_id = "mess"
    version = "0.0.0."
```

SeisHub Resource Types

- Each XML resource uploaded to SeisHub has to belong to resource type

```
from seishub.core.core import Component, implements
from seishub.core.packages.interfaces import IResourceType

class SeismicIntensityResourceType(Component):
    implements(IResourceType)
    package_id = "mess"
    version = "0.0.0."
    resourcetype_id = "seismic_intensity"
```

Registering Indices

- Searching for values in the raw XML files would require a full text search in each XML file
- Not feasible once the database reaches a certain size
- Registering an index tells SeisHub which values to monitor
- These values will be stored in the relational database backend
- Fast queries possible

Registering Indices

```
registerIndex(INDEX_NAME, XPATH, INDEX_TYPE)
```

```
from seishub.core.packages.installer import registerIndex

class SeismicIntensityResourceType(Component):
    ...

    registerIndex("event", "/seismic_intensity/event",
        "text")
```

Available index types:

text, **float**, **integer**, **datetime**, **date**, **timestamp**, **boolean**, and **numeric**

Finishing the Plug-In

A preliminary version of the plugin is located at
/home/mess/Desktop/seishub/seishub.plugins.mess

1. Have a look at all files and try to figure out what each part does
2. The *mess* component and the *seismic_intensity* resource type are already defined. Add indices for everything you deem necessary (at least *event*, *latitude*, and *longitude*)
3. When you are done, save everything and launch the SeisHub instance by executing
   /home/mess/Desktop/seishub/SEISHUB_INSTANCE/bin/debug.sh
4. Visit **http://localhost:8080/manage** and activate all components of the plug-in

Uploading the XML Files

- Now it is time to upload the previously generated XML files to the SeisHub database.
- SeisHub implements a RESTful interface for all XML resources
- It can be accessed at the following URL:

ADDRESS:PORT/xml/COMPONENT/RESOURCE_TYPE/RESOURCE_NAME

The following HTTP methods are defined:

- **GET:** Get the specified resource. If no resource name is given, a list of all available resources will be returned.
- **POST:** Upload a new resource. If no resource name is given, a random one will be assigned.
- **PUT:** Upload a new resource/update an existing one.
- **DELETE:** Delete a resource.

Uploading the XML Files

Task: Upload all previously created XML files

Hints:

- Recommended way: The **curl** tool.

```
curl -v --data-binary @FILENAME -X POST "ADDRESS"
```

- **-v:** Verbose output
- **--data-binary @FILENAME:** Specify the file to send as the data part of the request
- **-X POST:** Do a POST request

- Repeat for lots of files:

```
ls *.xml | xargs -I % curl -v --data-binary @% ...
```

Querying the Data I

Task: Request the uploaded data in different ways

Hints:

- Requesting data always uses **GET**
- Directly download a resource:

http://localhost:8080/xml/mess/seismic_intensity/name

Querying the Data II

Task: Request the uploaded data in different ways

Hints:

- Adding parameters to URLs

```
http://URL?arg1=a&arg2=b&arg3=c
```

Querying the Data III

Task: Request the uploaded data in different ways

- Get a list of all resources of a type:

http://localhost:8080/xml/mess/seismic_intensity

- **limit=40** Return more results (Default: 20)
- **offset=40** Return results starting from number 40
- **number_of_responses=1** Return only those with one response
- **min_intensity=8** Return only those with intensity 8 or larger
- **max_intensity=8** Return only those with intensity 8 or smaller
- **format=xhtml** Return an xhtml table (Available formats: xml, xhtml, json)

Summary

- Just a view lines of code are enough to create a flexible, powerful and extensible way of storing and retrieving large amounts of data
- Multiple persons can access and change the data simultaneously
- Web access makes it independent of location
- The indices can be changed at any time $\Rightarrow$ Database adapts to a problem
- May look complicated but is mostly just a matter of copy and paste
- Arbitrary functionality can be executed on top of the data with the help of mappers (see next section)

Things Left Out in This Tutorial

Due to time constraints, the following parts are left out:

- Adding a XSD schema to the resource type
- Used for verification upon uploading

- Adding XSLT style sheets to the seismic intensity resource type
- Used for on-the-fly resource transformation

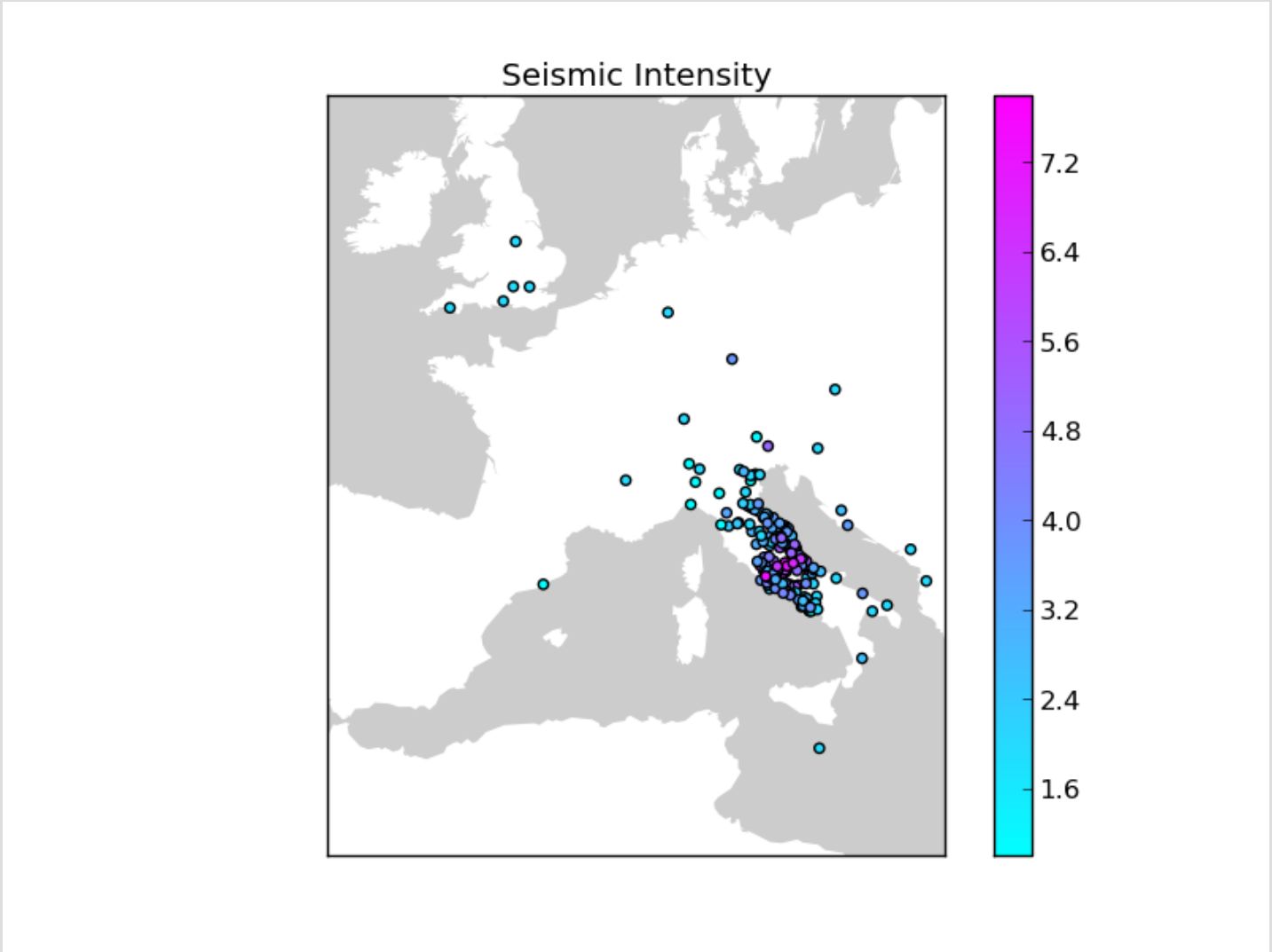
SeisHub Mappers

SeisHub Mappers

- SeisHub mappers are essentially Python functions executed when a certain URL is accessed
- Always executed in a thread $\Rightarrow$ SeisHub servers stays responsive
- Require more careful, defensive programming

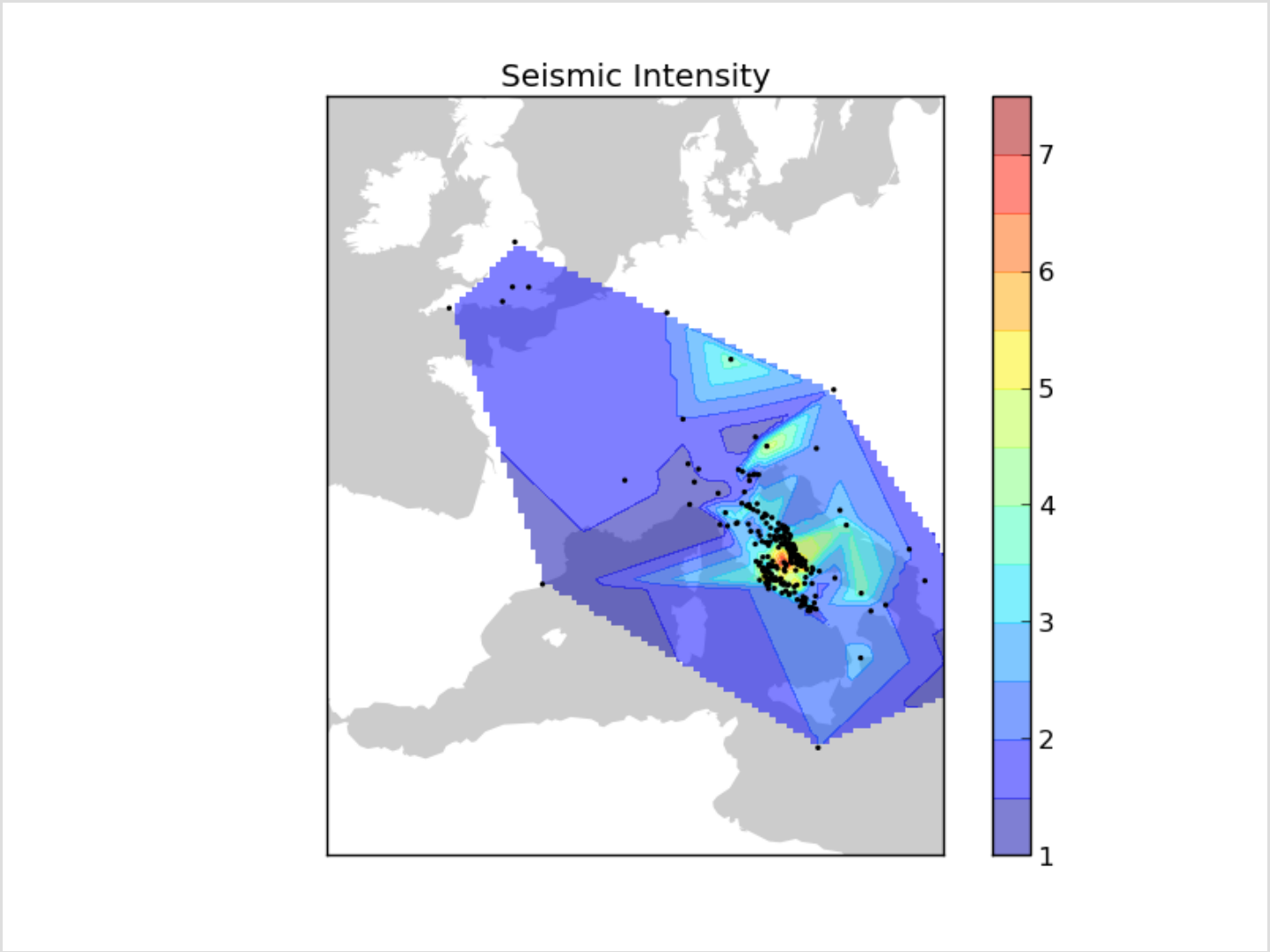
SeisHub Mappers

```
http://localhost:8080/mess/getMap?event=laquila
```



SeisHub Mappers

```
http://localhost:8080/mess/getMap?event=laquila&type=contour
```



SeisHub Mappers

```
from seishub.core.core import Component, implements
from seishub.core.packages.interfaces import IMapper

class MapMapper(Component):
    implements(IMapper)
    package_id = "mess"
    version = "0.0.0."
    mapping_url = "/mess/getMap"

    def process_GET(self, request):
        event = request.args0.get("event", "laquila")
        ...
        request.setHeader("content-type", "image/png")
        return data

    def process_POST(self, request):
        ...
```

Thanks for your Attention!

- http://seishub.org
- https://github.com/krischer/seishub.plugins.how_to_extend_seishub
- https://github.com/krischer/seishub.plugins.event_based_data
- https://github.com/barsch/seishub.plugins.seismology
- https://github.com/barsch/seishub.plugins.exupery