Building complex UWS services

Ivan Zolotukhin
IRAP / UPS

Our use case

- Project STOP: **S**pectral **To**ols **P**latform (collaboration with S. Bottinelli, J.-M. Glorian, E. Caux, D. Quénard)
- Legacy / heavy codes in FORTRAN / C: LIME and LVG
- Complex to install, inconvenient to access, troublesome to maintain :)
- Computationally intensive (impossible to use on a workstation)

Our use case

- Wrapping code into UWS cures most of the usage / maintenance problems
- Many client applications can now access it
- Dispatch jobs on a large cluster (SLURM)

UWS solutions market

As of June 2015:

- CDS library (Java)
- VO-PDC library by M. Servillat in progress (python / bottle)
- No working client in python (`uws-client` broken)

UWS solutions market

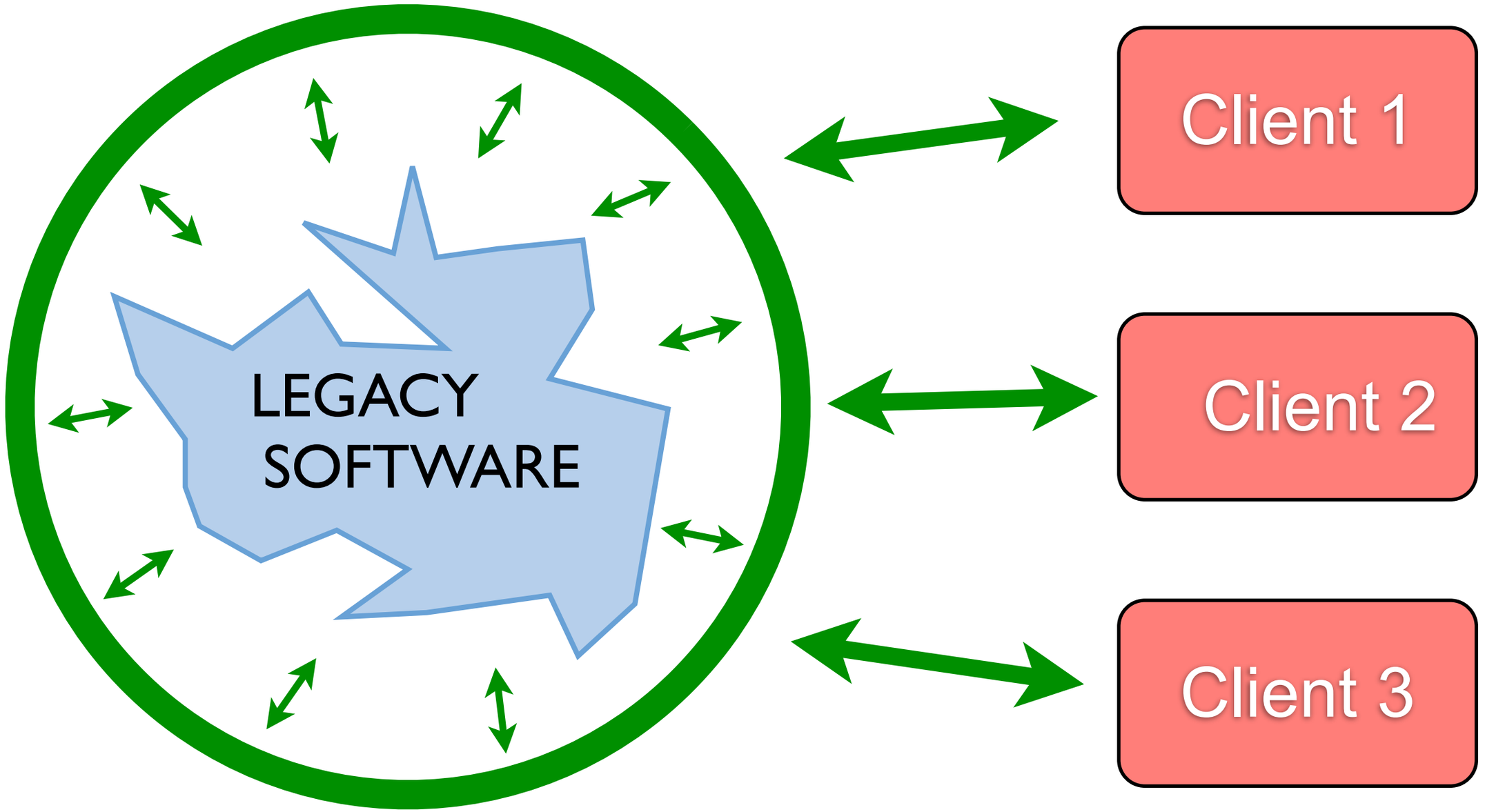
Progress in July / August:

- DaCHS (dev version)
- `uws-client` fixed

UWS with DaCHS

- Very powerful solution (arbitrary binary wrap, arbitrary job input environment, arbitrary arguments through UWS, etc)
- Power of python embedded into `q.rd` – the only service configuration file

UWS server



DaCHS UWS setup

- http://docs.g-vo.org/DaCHS/ref.html#custom-uwses
  - `gavo imp uws enable_useruws`
  - **Prepare** `q.rd`
  - `gavo imp -m q.rd`

Minimal q.rd

```
<resource schema="uws_test">
   <meta name="title">UWS test webservice</meta>
   <table id="results">
      <meta name="description">Test</meta>
      <column name="column_1" ucd="" description="Test column"/>
   </table>
   <data id="parse_output">
      <reGrammar topIgnoredLines="1">
         <names>column_1</names>
      </reGrammar>
   <make table="results">
      <rowmaker idmaps="*"/>
   </make>
</data>

<service id="r" allowed="uws.xml,form">
   <pythonCore>
      <inputTable>
         <inputKey name="molecule" type="text"
               multiplicity="single"
               description="Name of the molecule"/>
      </inputTable>

      <outputTable original="results"/>
```

```
        <coreProc>
          <setup>
            <par name="grid_template">
              <![CDATA["""%(molecule)s"""]]>
            </par>
            <code>
import subprocess
print 'Hello world!'
            </code>
          </setup>

          <code>
params = {
  "molecule": inputTable.getParam("molecule"),
}
with open("config.txt", "w") as f:
  f.write(grid_template%pars)

subprocess.call([rd.getAbsPath("binary_code")])
return rsc.makeData(rd.getById("parse_output"),
  forceSource=os.path.abspath("%s-sel_lines.dat" % (pars['molecule'])))
          </code>
        </coreProc>
      </pythonCore>
    </service>
</resource>
```

DaCHS UWS debug

- Quite cumbersome :(
- Hence, best debugging tool is Markus :)
- This means you may want your abstraction python layer outside of `q.rd` with e.g. `argparse` command line interface – then you debug it with usual methods and simply call it with `subprocess.call()` from there

DaCHS UWS debug

- In case of XML tag mismatch: copy-paste whole `q.rd` into e.g. http://www.w3schools.com/xml/xml_validator.asp to validate, it says mismatched tag names
- BadCode errors: do `gavo --traceback uwsrun $jobid`, then copy-paste generated code to an editor with python syntax highlight

UWS client requests

- `uws-client` fixed (very helpful!)
- Array arguments and file upload examples

```
uws -H http://stop-dev1.irap.omp.eu:8080/stop/lvg/r/uws.xml job
new -r molecule=co temperature="1 2 3" linewidth="2 5 3"
isotopic_ratio="0.02 0.04" structure_type="0 1"
```

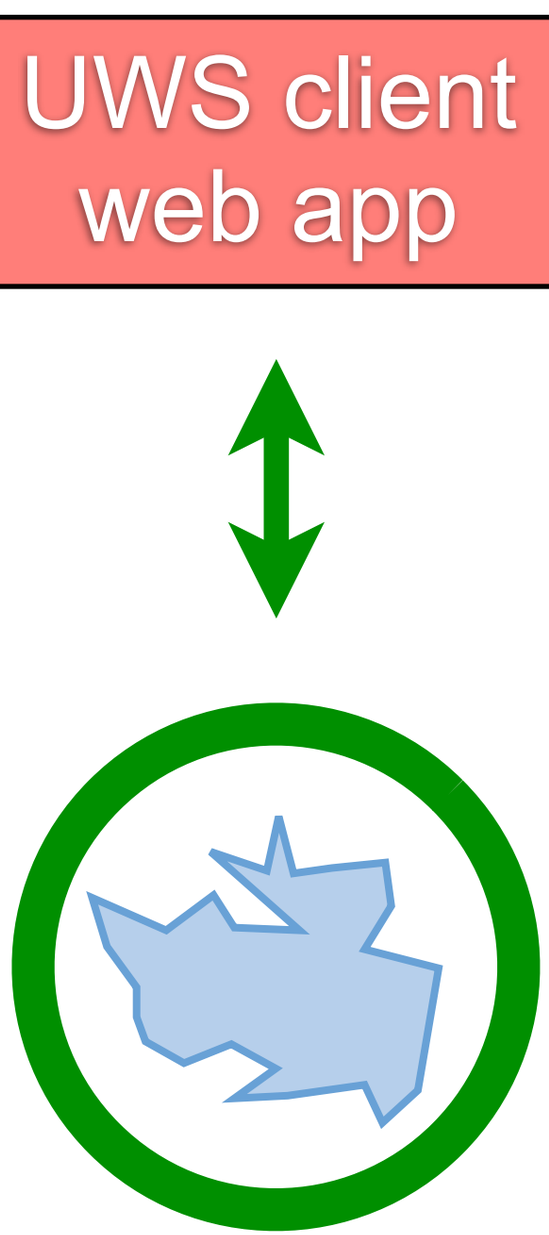
```
uws -H http://stop-dev1.irap.omp.eu:8080/stop/lime/r/uws.xml job
new # get jobId

> import requests

> requests.post("http://stop-dev1.irap.omp.eu:8080/stop/lime/r/
uws.xml/MXGoNT/parameters", {'UPLOAD':
'models_tarball,param:upl'}, files = {'upl': open('/home/
izolotukhin/models.tar')})

uws -H http://stop-dev1.irap.omp.eu:8080/stop/lime/r/uws.xml job
run MXGoNT
```

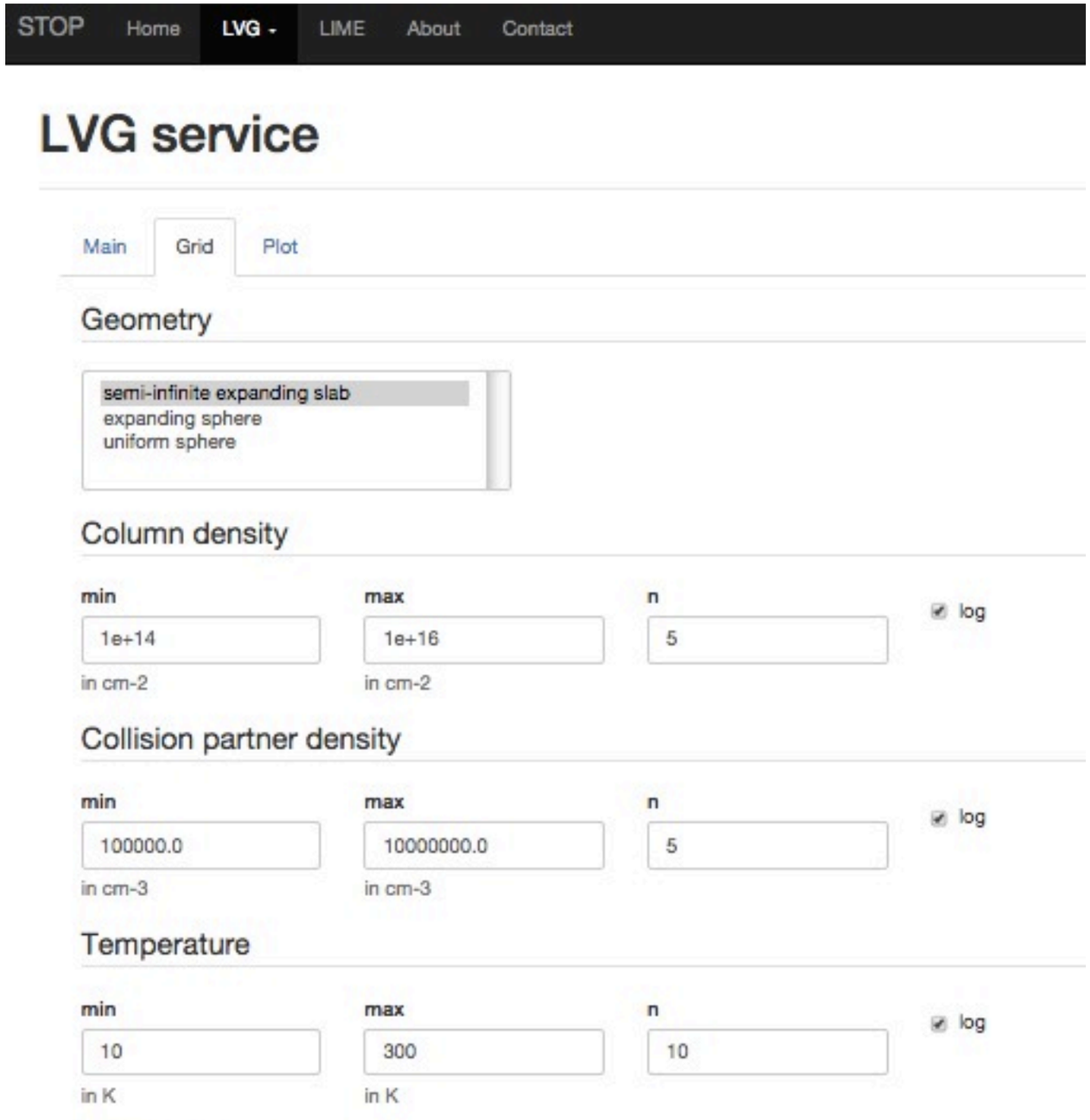
Need for special UWS client

- Complex input parameters
  - File uploads
  - Array / dictionary arguments
  - Complex relationships between input arguments
- **Cannot be handled by default / automatic client**

Special UWS client

UWS client
web app





Lessons learned

- UWS + DaCHS: stable and mature
- Some inconveniences do exist (lack of dictionary arguments, cumbersome file uploads, ...)
- Those are mitigated by the need to build custom clients
- Generic / automated clients or service descriptions are not really possible (e.g. PDL is of no help)

Thanks