The CEDA Web Processing Service for rapid deployment of earth system data services

Stephen Pascoe
Ag Stephens
Phil Kershaw
Centre of Environmental Data Archival
Overview of CEDA-WPS

• History – first implementation and deployment
• Architecture
• Generalising first implementation
• Test Beds
• Operational Services
History

UKCIP09 User Interface

Requirements

- Map and non-map plots
- Interactive and configurable
- Long-running tasks
  - Large data extractions
  - Secondary simulations (Weather Generator)
- Resilient to variable load

Service Oriented Architecture
OGC Web Services at CEDA

- CEDA has developed web portals based on OGC Web Services since 2006
  - WMS clients and servers
  - WCS/WFS for complex GML features (CSML)
- NERC Data Grid
- NERC Portals Project
- ISIC Visualisation
Architecture: OGC Web Processing Service

- GET DescribeProcess resource to discover process arguments
- POST to create a process execution resource (unique URL)
- GET to poll status of process execution
- Navigate to outputs when available
CEDA OGC Web Services Framework: COWS

http://cows.badc.rl.ac.uk/

Architecture

Apache + mod_python, FastCGI, Python HTTPD

Pylons

Middleware

HTTP Server

WSGI

e.g. Authentication

COWS

WMS

WCS

W*S

WSGI

Pylons

Web Application

Data Model

Library

Standard Interface

Application code
Architecture

CEDA OGC Web Services Framework: COWS
http://cows.badc.rl.ac.uk/

COWS Server Implementations

- **cowsserver**: WMS/WCS server component
- **cowsclient**: WMS/WCS web-application
- **cows-wps**: WPS server

COWS library

- **cows**: OWS operation controllers, KVP encoding and exception handling
- **cows.pylons**: XML templates, Stub server
- **cows.model**: OWS-Common model implementation
- **cows.service**: Stub service implementations, WMS, WCS, WFS, W*S
- **cows.csmlbackend**: Data back-end for Climate Science Markup Language (CSML)
Architecture

Process Modules

- Implement a process as a Python module
- Configuration file defines
  - Inputs / outputs
  - Synchronous / Asynchronous
  - Caching
  - Workflow

```python
class ExampleProcess(ProcessBase):
    def _executeProc(self, context):
        # Call standard_setup
        self._setup(context)
        # Now set status to started
        context.setStatus(STATUS.STARTED, 'Job is now running', 0)
        # Add output file
        outputNC = os.path.join(context.outputDir, "output.nc")
        outputPNG = os.path.join(context.outputDir, "output.png")
        # ...
        # DO STUFF
        # ...
        nc_size = os.path.getsize(outputNC)
        png_size = os.path.getsize(outputPNG)
        # Add the stations file to the outputs
        self._addFileToOutputs(outputNC, 'NetCDF File', size=nc_size)
        self._addFileToOutputs(outputPNG, 'PNG File', size=png_size)
        process_support.finishProcess(context, self.fileName,
                                       self.startTime, keep=True)

def validateInputs(self):
    # Runs specific checking of arguments and their compatibility.
    pass
```

---

Example COWS-WPS Process

---

# ... IMPORTS HIDDEN

---

8
UKCP09 Deployment

User Interface Layer
Service Layer (WPS, WMS)
Job Execution Layer

Master server
Physical servers
Virtual Machines
Temporary servers for high-demand launch period
Generalised Service
CEDA-WPS

WPS 1.0.0 Features

- A web service interface, using POST or GET.
- Asynchronous reporting and control of jobs.
- A defined XML interface for responses, including exceptions.
- A common format for passing arguments to the server.
- Job status interrogation.

Exensions in CEDA-WPS

- Add new processes as Python modules.
- Web-client to auto-generate process submission forms and interrogate current and previous jobs.
- Connection to a parallelised processing back-end.
- Output caching
- Inform users via e-mail when a job has completed (or failed).
- Integration with CEDA Security middleware.
- Estimate the job size and duration.
- Zip up output files
CEDA-WPS Web Client

The UI automatically generates submission forms for each process. This includes bounding box, date/time, float, integer and string types.
Test Beds: Service Chaining WCS

- COWS WPS UI generates a form for the “WCSWrapper” process.
- User can view the WCSEndPoint options.
- User clicks the “Update form” button to load the available “Coverages”
- The Coverages are extracted from a call to “GetCapabilities” at the WCSEndPoint.
- Further options are loaded.
- Once all selections have been made the user can click “Submit” to make a request to the WPS.
- The WPS calls the WCS “GetCoverage” method at the WCSEndPoint
Test Beds: MashMyData

- Proof-of-concept web portal
- Scientists will be able to simultaneously visualize data from many sources, including their own uploaded data
- Scientists will be able to perform simple quantitative comparison calculations
- Data access will require Authentication and Authorisation
Test Beds: NDGSecurity

- Single sign-on with OpenID and PKI credentials
- WSGI filters for Authentication and Authorisation
- Centralised authorisation policy described by XACML (Oasis Standard)
- XACML policy generated from COWS-WPS configuration
Test Beds: MashMyData

1. A user accesses the MashMyData Portal.

2. They select some datasets for an intercomparison operation requiring a process run by the WPS.

3. The portal requests the BADC's WPS execute this calculation on its behalf, but it responds indicating that access credentials are required to access secured datasets.

4. The portal prompts the user to sign in with OpenID.

5. Once the user has signed in, the portal can translate their authenticated status by obtaining a short-lived certificate representing them from the Credential Translation Service. This credential can be used by the Portal to authenticate the user on their behalf with other services.

6. The portal uses the certificate acquired from the Credential Translation Service to authenticate with the WPS.

7. The portal also provisions a Delegation Service with a proxy certificate enabling the WPS to perform credential retrievals.

8. The WPS can retrieve a proxy certificate from the Delegation Service to enable it to authenticate with an OPeNDAP service on behalf of the user.

9. When the WPS has completed execution of the given job, it exposes the results to the Portal for retrieval.
Test Beds: OPeNDAP

- WPS NetCDF Outputs available over OpeNDAP
- Implemented via an embedded PyDap server
- Interrogate NetCDF Metadata and Subset outputs
- Secured with NDGSecurity
Operational Service MIDAS

- New Service at BADC (beta)
- UK Met Office MIDAS database
- Land surface observations data from the Met Office station network
- Daily and hourly weather measurements
- Extract by UK county of bounding box
- CSV output
Future

• CDO Operators available as WPS Processes (ExArch Project)
• Run WPS Processes on the CMIP5 Archive
• Porting process execution backend to other schedulers (TORQUE)
• Improved WPS-1.0.0 compliance
Lessons Learned

- OGC Standards provide a useful blueprint for implementing operational SOA
- Slavish adherence to the standards can lead to limited payback for NWP/atmospheric science community
  - Process is slow
  - Premature standardisation
  - Focus is GIS not atmospheric science
- Our approach is to comply with commonly deployed versions whilst staying within striking distance of the latest standards
Thanks

Stephen Pascoe, Ag Stephens, Phil Kershaw

http://ceda-wps.badc.rl.ac.uk
http://cows.badc.rl.ac.uk/cows_wps.html
Architecture: COWS-WPS

- **Pylons**
  - Service Container
  - WPS Service

- **cows-wps**
  - Library
  - Process Module

- **GridEngine**
  - Execute host
CEDA-WPS Workflow

(A) “cost-only” call.
(B) poll to update the job status.
(C) retrieve job details and outputs
(D) Previous jobs can be displayed
(E) outputs can be downloaded.
CEDA-WPS Workflow

(A) “cost-only” call.
(B) poll to update the job status.
(C) retrieve job details and outputs
(D) Previous jobs can be displayed
(E) outputs can be downloaded.