WEB PROCESSING: STANDARDIZATION AND CLOUD CONCEPTS

Matthes Rieke, Benjamin Proß

Geospatial Sensing | Virtual 2020
OVERVIEW AND AGENDA

1. Web-Processing
2. Standardization / Interoperability
   a. WPS 2.0
   b. OGC API Processes
3. Software Solutions
   a. javaPS
   b. wps-js-ng
   c. Angular Map Client
4. “Hands-on” / Use Case
   a. Docker-based Process Execution
**Why Web-based Geoprocessing and WPS?**

Motivation and Background
Main focus of 52°NORTH

Sensor Web Enablement

Geoprocessing

Desktop Apps

Web Apps

SDIs, SOA, Big Data

User icons made by Freepik from www.flaticon.com are licensed as CC BY 3.0
Geoprocessing

Raw data

Processing

Value-added data products

Processing

Information products
GEOPROCESSING – EARLIER APPROACH

Desktop GIS

Input data
Output data
**Web Service - Approach**

- **Web Applications**
- **ShakeMap Service**
- **ShakeMap Computation**
- **Desktop**
- **Risk Service**
  - Exposure, Vulnerability Service
- **Data Service**
- **Other Web Services**

- Standardized APIs
- + Data Formats
**OGC Web Processing Service (WPS)**

- **WMS** – Maps as Images (jpg, tiff, …)
- **WCS** – Coverages (geoTiff, netCDF, …)
- **WFS** – Vector Data (GML, shp)
- **SOS** – Observations (O&M, SweCommon, …)
- **WPS** – Geoprocesses, Simulations, …
Geoprocessing in the Web, because...

...the set-up of software is complicated

...processing is tightly coupled to data that is available only remotely

...I lack the computational capacity in my own environment

...I want to reuse processing in different environments

...I want to couple different technologies ("Processing Mash-Ups").
...the analysis should be automated and controllable via the web
STANDARDIZATION FOR GEOPROCESSING

OGC and related Concepts
OGC Web Processing Service - Overview

- Official OGC Standard since 2007, Version 2.0.0 since 2015:
  - [http://www.opengeospatial.org/standards/wps](http://www.opengeospatial.org/standards/wps)
- Standardized description of geoprocessing functionality („processes“)
  - Identifier
  - Textual description
  - Input and output parameters
- Predefined service operations for the description and execution of processes (synchronous, asynchronous)
  - DescribeProcess, Execute, GetResult
- Software:
  - 52°North WPS, PyWPS, Zoo WPS, ArcGIS Server, ERDAS Imagine, …
OGC Web Processing Service – Basic Operations

Clients
- Desktop Apps
- Web Apps
- SDIs, SOA, Cloud Environments

WPS
- Process 1
- Process 2
- Process n

Operations:
- GetCapabilities
- List of processes
- DescribeProcess
- Detailed process description
- Execute
- Processing result
WPS – WHICH PROCESSES ARE AVAILABLE?

Clients
- Desktop Apps
- Web Apps
- SDIs, SOA, Big Data

GetCapabilities

List of processes

WPS
- Process 1
- SimpleBufferAlgorithm
- Process n
WPS – GET CAPABILITIES OPERATION

• Request via URL (HTTP GET with Key-Value-Pair encoding) or XML Request (HTTP POST)
• Returns service description of the WPS
• Basic information:
  • Endpoints
  • Technical request mechanisms
  • Information about the service provider/access constraints
• Short information about the offered processes
WPS – What does the process do? What are the inputs/outputs?

**Clients**

- Desktop Apps
- Web Apps
- SDIs, SOA, Big Data

**WPS**

- GetCapabilities
  - List of processes
  - DescribeProcess

  Detailed process description

- Process 1
  - SimpleBufferAlgorithm

- Process n
WPS: Describe Process Operation

- Request via URL (HTTP GET with Key-Value-Pair encoding) or XML Request (HTTP POST)
- Returns detailed description of a process based on the process id
- Defines the inputs and outputs
  - IDs
  - Default data formats
  - Further supported data formats
WPS – How do I execute a process?

Clients

- Desktop Apps
- Web Apps
- SDIs, SOA, Big Data

GetCapabilities

WPS

- Process 1
- SimpleBufferAlgorithm
- Process n

List of processes
DescribeProcess
Detailed process description
Execute
Processing result
WPS – Execute Operation

• XML Request (HTTP POST)
• Execution of an offered process
• Request:
  • Must contain id and input parameters according to the process description
• Result
  • Can be returned directly or as reference to a web accessible resource
• Can be executed asynchronously for long running processes
  Client doesn’t directly get the result, but can request the status of the execution
**WPS – How to transfer the input data**

**Option 1: Direct transfer**

**Option 2: Reference to web accessible resource**

**Option 3: Query on encapsulated data**
**WPS – Synchronous Execution**

- **Client**
  - Send request
  - Waits for response
  - Receives result and uses it

- **WPS**
  - Receives request
  - Processes result
  - Sends result
WPS – ASYNCHRONOUS EXECUTION (PUSH-MODEL)

Sequence Diagram

- **Client**
  - Sends request
  - Does something different
  - Receives the result and uses it

- **WPS**
  - Receives request
  - Processes result
  - Sends result

Client sends request to WPS, which processes the request differently and sends the result back to the client, who then uses the result.
WPS Spec 2.0 – Asynchronous Execution

Clients

- Desktop Apps
- Web Apps
- SDIs, SOA, Big Data

WPS

- Process 1
  - SimpleBufferAlgorithm
  - ...
  - Process n

Execute(mode=async,...)

- JobID, StatusInfo
- GetStatus(JobID)
- StatusInfo
- GetStatus(JobID)
- StatusInfo
- GetResult(JobID)
- Result
OGC API - Processes

- Currently in the process of standardization
  - Next step: Request for public comments
- Several implementations are available for testing
- Core and extensions

https://github.com/opengeospatial/wps-rest-binding
Motivation for OGC APIs

• WPS 2.0 and other OGC specifications normally define a SOAP/XML Binding as necessary interface
  • Exceptions: OGC Web Map Tiling Server and Sensor Things API

• REST APIs with JSON Encodings leaner for implementations of Web clients and easier to use
  • Focus on resources (not on operations)
  • Use of standard HTTP operations for CRUD of resources

• In OGC Testbed 12 REST APIs for different OGC services were tested and described in the Testbed 12 REST Architecture Engineering Report

http://docs.opengeospatial.org/per/16-035.html
OGC API - Processes: Resource Model

• Based upon the general process model of WPS 2.0

• Hypermedia approach
  • Landing Page contains link to ProcessList
  • ProcessList contains links to single Process –ressources
  • ...

## Endpoints for Resource Access

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>HTTP Operation</th>
<th>Endpoint</th>
<th>Message Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landing Page</td>
<td>Request of the service description</td>
<td>HTTP GET</td>
<td>{baseURL}</td>
<td></td>
</tr>
<tr>
<td>Process list</td>
<td>Request of the list of processes</td>
<td>HTTP GET</td>
<td>{baseURL}/processes</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>Request of a single process description</td>
<td>HTTP GET</td>
<td>{baseURL}/processes/{processID}</td>
<td></td>
</tr>
<tr>
<td>Job list</td>
<td>Request of the list of jobs (executions) of a process</td>
<td>HTTP GET</td>
<td>{baseURL}/processes/{processID}/jobs</td>
<td></td>
</tr>
<tr>
<td>Job info</td>
<td>Execution of a process/creation of a new job</td>
<td>HTTP POST</td>
<td>{baseURL}/processes/{processID}/jobs</td>
<td>Execute Request in JSON</td>
</tr>
<tr>
<td>Job info</td>
<td>Request of the status of a job</td>
<td>HTTP GET</td>
<td>{baseURL}/processes/{processID}/jobs/{jobID}</td>
<td></td>
</tr>
<tr>
<td>Results</td>
<td>Request of the results of a job</td>
<td>HTTP GET</td>
<td>{baseURL}/processes/{processID}/jobs/{jobID}/results</td>
<td></td>
</tr>
</tbody>
</table>
52N JAVAPS-REST

• Implements OGC API - Processes

• GitHub Repo:
  • https://github.com/52North/javaPS

• Example instance for testing:
  • http://geoprocessing.demo.52north.org:8080/javaps/rest/
**Request Landing Page**

**HTTP GET <baseURL>**

- No predefined pattern for endpoint-URL
- Returns Capabilities document containing the list of available processes

General service information:

```
"title": "52° North OGC API - Processes",
"description": "52° North OGC API - Processes, powered by javaPS",
"links": [
    {
        "href": "http://geoprocessing.demo.52north.org:8080/javaps/rest",
        "rel": "self",
        "type": "application/json",
        "title": "this document"
    },
    {
        "href": "http://geoprocessing.demo.52north.org:8080/javaps/rest/api/",
        "rel": "service",
        "type": "application/openapi-json;version=3.0",
        "title": "the API definition"
    },
    {
        "href": "http://geoprocessing.demo.52north.org:8080/javaps/rest/conformance/",
        "rel": "conformance",
        "type": "application/json",
        "title": "Conformance classes implemented by this server"
    },
    {
        "href": "http://geoprocessing.demo.52north.org:8080/javaps/rest/processes/",
        "rel": "processes",
        "type": "application/json",
        "title": "The processes offered by this server"
    }
]
```

Link to process list:

http://geoprocessing.demo.52north.org:8080/javaps/rest
REQUEST PROCESS LIST

HTTP GET <baseURL>/processes

• Returns a list of short process summaries containing links to detailed process descriptions

```
[{
  "id": "org.n52.javaps.test.EchoProcess",
  "title": "org.n52.javaps.test.EchoProcess",
  "keywords": [],
  "metadata": [],
  "version": "1.0.0",
  "jobControlOptions": [],
  "sync-execute",
  "async-execute",
  "outputTransmission": [],
  "value",
  "reference"
},

"links": [
  {
    "rel": "process description",
    "type": "application/json",
    "title": "Process description"
  }
]
```

Link to detailed process description
REQUEST PROCESS DESCRIPTION

HTTP GET <baseURL>/processes/<process-id>

• Returns the detailed description of a process including input and output parameters
REQUEST PROCESS DESCRIPTION

Execute-URL

Input parameter

Output parameter

http://geoprocessing.demo.52north.org:8080/javaps/rest/processes/org.n52.javaps.test.EchoProcess
Execution of a Process (1)

HTTP POST
<baseURL>/processes/<process-id>/jobs

- Parameter:
  - Execute Request in JSON (see to the right)

- A new job resource (process execution) is created

```
1 {  "inputs": [  
2      {  "id": "complexInput",  
3        "input": {  
4          "format": {  
5            "mimeType": "application/xml"  
6          },  
7          "value": {  
8            "inlineValue": "<test/>"  
9          }  
10        }  
11      }  
12    ],  
13    "outputs": [  
14      {  "id": "complexOutput",  
15        "format": {  
16          "mimeType": "application/xml"  
17        },  
18        "transmissionMode": "value"  
19      }  
20    ],  
21    "response": "document",  
22    "mode": "async"
23 }
```

Execution of a process (II)

- Asynchronous execution:
  - HTTP 201 with link to job resource
  - After execution is finished link to results

```json

{
    "status": "successful",
    "jobID": "2ceb5c7d-69cb-4064-a941-1c812506bf08",
    "links": [
        {
            "href": "http://geoprocessing.demo.52north.org:8080/javaps/rest/processes/org.n52.javaps.test.EchoProcess/jobs/2ceb5c7d-69cb-4064-a941-1c812506bf08",
            "rel": "self",
            "type": "application/json",
            "title": "This document"
        },
        {
            "href": "http://geoprocessing.demo.52north.org:8080/javaps/rest/processes/org.n52.javaps.test.EchoProcess/jobs/2ceb5c7d-69cb-4064-a941-1c812506bf08/results",
            "rel": "results",
            "type": "application/json",
            "title": "Job results"
        }
    ]
}
```

Finished job with link to results
EXECUTION OF A PROCESS (III)

• Synchronous execution:
  • JSON result document
    (success|failure)
REQUEST PROCESSING RESULTS

HTTP GET:

<baseURL>/processes/<process-id>/jobs/<job-id>/outputs

• Returns JSON result document

```json
{
  "outputs": [
    {
      "id": "complexOutput",
      "value": {
        "inlineValue": "<test/>"
      }
    }
  ]
}
```
JAVAPS REST TESTCLIENTS

HTML view of execute endpoints, e.g.:

Jobs

Show this page as JSON document

- 9cceb20e-c47d-4e57-a886-97b2347c50a
- 2e8e5c7d-69cb-40b4-a941-1c812508f9d8

Submit new job

Load example request

```json
{
  "Inputs": [
    {
      "id": "complexInput",
      "Input": {
        "format": {
          "value": "application/xml"
        },
        "inlinenvalue": "<text/>"
      }
    },
    {
      "id": "literalInput",
      "Input": {
        "datatype": "double",
        "value": "0.05"
      }
    },
    {
      "id": "boundingboxInput",
      "Input": {
        "bbox": [
          91.9,
          7,
          92,
          7.1
        ],
        "crs": "EPSG:4326"
      }
    }
  ]
}
```

Create job
EXAMPLE APPLICATIONS

Practical Use Cases
EXAMPLE APPLICATION: USGS

https://cida.usgs.gov/gdp/
**Example Application: TAMIS**
**Example Application: WaCoDiS**

**Domain User**

- Order EO Product
- Discover Data

**Data Sources**

- EO Products

**Data Sources**

- Schedule Processing

**Acquire Data**

**Process Data**

**Deliver EO Product**

**WPS**

**Icons created by Alice Design, Andi Nur Abdillah, Creative Mania, Emily van den Heever, fœ freya, flamingo, Graphic Tigers, ProSymbols, rivercon, Scott Dunlap, Tinashe Mugayi, Xicons.co from the Noun Project**
SOFTWARE SOLUTIONS - SERVER & CLIENT

52N Software, other Open Source solutions
Server Implementations

• OGC lists 69 implementations for WPS 1.0/2.0

• Commercial:
  • ESRI, FME, Intergraph, Envitia, ERDAS, …

• Open Source:
  • 52°North, pyWPS, ZOO, Geoserver, …
52°NORTH WPS

• Beta-Release of version 4.0.0 available
• Supports (all) features and operations of the WPS specification Version 1.0.0 and 2.0
• Support of:
  • GRASS 7, Sextante, R, Java/Python
• Java Process/R Script upload via Web UI
• Parser/Generators for common data formats:
  • SHP-Files, GML, GeoJSON, GeoTIFF, NetCDF, …
52°NORTH JAVAPS

• Version 1.5.0
• Complete new implementation using state-of-the-art technology
• Supports WPS 2.0 and OGC API - Processes
• Supports deployment of new processes (transactional extension)
• Parser/Generators for common data formats:
  • SHP-Files, GML, GeoJSON, GeoTIFF, NetCDF, …
52° North WPS Deployment Patterns

- Extending an existing WPS
- Custom Binaries with configuration files
- Rich Data Interfaces
- Deployment of annotated scripts (currently R)
**WPS Extension**

- Direct extension of an existing WPS Server implementation
  - Reuse of I/O handlers
- Example: Implementation of an algorithm in Java as direct extension of the 52N javaPS
**Execution of Custom Binaries**

- Execution of a program in binary code
- Configuration via config files
  - Parameters for controlling the process
  - Inputs/Outputs are stored locally and are referenced using config parameters
**WPS as Rich Data Interface**

- WPS as interface for a large data store
- Dynamic views can be generated using input parameters
  - Inputs don’t contain data for processing
DEPLOYMENT OF R SCRIPTS

- Annotation in R scripts define inputs/outputs
- Upload of annotated R scripts allows deployment as WPS processes
CLIENT IMPLEMENTATIONS

• 52°North ArcGIS WPS Client
  • ArcGIS Extension in collaboration with ESRI Inc.
  • Available as Open Source

• 52°North WPS-JS
  • JavaScript library for the creation of Web clients

• Quantum-GIS WPS – Plugin

• Custom clients for specific applications, e.g. USGS
52°NORTH WPS-NG-CLIENT

- **wps-js:**
  - JavaScript client library
- **Wps-ng-client:**
  - JavaScript client
  - Based on the Angular Framework

https://github.com/52North/wps-ng-client
52°NORTH ArcGIS WPS Client (I)

Adding of a WPS
52°NORTH ArcGIS WPS Client (II)

Toolbox for processes

Input form for the execution
**52° NORTH ArcGIS WPS Client (III)**

Integration in the Model Builder

Result map
Developing a Process with Python / Jupyter Notebook and making it available on the Web
INTERFACE FOR CLOUD-BASED GEOPROCESSING

Quelle:
http://www.opengeospatial.org/node/2526#Cloud
**Hosted Processing**

**Clients**
- Desktop Apps
- Web Apps
- SDIs, SOA, Cloud Environments

**WPS**
- DeployProcess(DockerImage)
- ID of the new process
- DescribeProcess
- Detailed process description
- Execute
- Processing result

- Docker Process 1
- Process 2
- Process n
CONCEPT & APPROACH

- What is the current trend of developing “processing” functionality in the research community and the data science/analysis domain?
  → Jupyter Notebooks or R [Notebooks, Markdown]

- Idea:
  a. develop a (simple) process
  b. make it executable in an environment-independent way
  c. host it in the Cloud [, close to the data]
CONCEPT & APPROACH

• Approach:
  a. develop a (simple) process
     → Jupyter Notebooks
  b. make it executable in an environment-independent way
     → Docker with Python Kernel
  c. host it in the Cloud [, close to the data]
     → OGC API Processes, with Docker Execution Backend
**Jupyter Notebook Process**


- How can we make the (interactive) Notebook executable without manual interaction?

→ **Papermill** allows parameterization
Docker Image Preparation

1. Reasonable Base Image: `continuumio/miniconda3`

2. Setup environment: import required libraries (geopandas)

3. Define command that executes the Notebook (Papermill)
**Deploy the Process on the Web**

**ADES:** Application Deployment and Execution Service

- Extension to OGC API Processes
- Introduces transactional API (registration, management of processes)
- Makes use of “Execution Units” (e.g. Docker images, other programming language environments)

Demo with Web Client

https://52north.github.io/wps-ng-client/

OGC API Processes (draft) instance:
https://testbed.dev.52north.org/ades/rest/

OGC WPS 2.0 instance:
https://testbed.dev.52north.org/ades/service
ADES - Things to Keep in Mind

- No defined interface between “Application Package” und “Execution Unit”:
  - The way how Inputs and Outputs are communicated between these two interfaces depends (at the moment!) on the implementation
  - Example: https://github.com/matthesrieke/gsv2020/blob/master/Dockerfile#L11
- Docker Execution is most suitable for heavyweight computations as the bootstrapping of Docker containers introduces an overhead → less relevant for longer running processes
ADES - ROAD AHEAD

- Feasibility Study in established Cloud environments
  - Kubernetes / Google Cloud Platform
  - AWS
- Generic Approach for Notebooks → improvements in javaPS <-> container interaction
- Extension to additional frameworks (R notebooks, ...)

52°NORTH  HTTPS://52NORTH.ORG
SUMMARY
Topics Covered

- Geoprocessing motivation
- WPS 2.0 as the established standard
- OGC “API Processes” as the emerging next generation API
- Client & Server Software
- Cloud-based Processing
  - Jupyter Notebook
  - Docker
  - ADES - OGC API Processes
THANKS FOR THE ATTENTION!

QUESTIONS?

Matthes Rieke (m.rieke@52north.org)
Benjamin Proß (b.pross@52north.org)
RESOURCES

• OGC Resources:
  • OGC API Processes: https://github.com/opengeospatial/wps-rest-binding
  • Application Package: http://docs.opengeospatial.org/per/17-023.html
• Papermill: https://github.com/nteract/papermill
• 52N javaPS: https://github.com/52North/javaps
• WPS 2.0 Web Client: https://github.com/52North/wps-ng-client