OGC Testbed-14: Exploitation Platform Thread

Final Demonstration meeting
ESA/ESRIN, January 2019
Agenda

- Introductions
- Problem statement
- Solution Architecture
- Demonstration video
- Components
  - Catalogue
  - Client
  - Application package
  - Execution Management Service
  - Application Deployment & Execution Service
- Technologies
- Applications
- Challenges
- Standardization-related issues
- Conclusion/Achievements
INTRODUCTIONS
Introductions

- CRIM
- CubeWerx Inc.
- Geomatys
- Solenix
- Spacebel
PROBLEM STATEMENT
Problem: complex ecosystem

Platform: $X$ platforms, $Y$ implementers

Users:
- Confused scientists

I see a lot of:
- Power
- Capacity

I want to:
- Develop once
- Benefit from $X \times Y \times Z$ explosion
- Not care about details

Infrastructure:
- $Z$ suppliers
  - aws
  - CREODIAS
  - SOLENIX
  - ONDA
  - SOBLOO
  - W3kEO
Let’s help the confused scientist by:

- Defining a standardised Application Package (AP):
  - To convey different kinds of information that all applications have/need (executable, interface, required data sources, etc.)
  - So that heterogeneous platforms/implementations and their components can exchange information about an application in a standard and interoperable way
  - So that heterogeneity and infrastructure details are hidden but the same application can be executed on different infrastructure

Let’s help platform developers by:

- Proposing two standard/essential EP building blocks: the EMS & the ADES
- Documenting Best Practices and Interfaces for engineering them:
  - EMS: control logic for deploying/executing applications in different MEPs & TEPs, the chaining thereof, overall coherence of the execution chain
  - ADES: single application deployment/execution on specific platform
Pre-requisite: Dockerized application

Prior (TB13):
- AP based on OWS Context
- Simpler architecture: just AMC and ADES, i.e. no Execution Management Service (EMS).
- Interface based on WPS/XML, two alternative implementations:
  - Standard WPS 2.0 and two dedicated - but regular - WPS processes for deploying and undeploying processes/applications in the ADES;
  - WPS-T, not standardized but described in a OGC discussion paper.

After (TB14):
- WPS-T, both for client<->EMS and EMS<->ADES interface.
- AP based on WPS-T DeployProcess
- REST/JSON rather than XML: WPS REST/JSON under public review by WPS SWG -> WPS 2.0 SWG and TB-14 EOC coordination
- CWL for application chaining (workflows)
SOLUTION ARCHITECTURE
High-Level Architecture
Detailed Architecture
Detailed sequence of steps

1. Alice deploys a newly developed application.
2. Alice’s sister discovers the new application and builds a workflow including that application.
3. Alice’s sister deploys the workflow.
4. Bob discovers the new workflow and executes it.
5. The EMS performs the EO Image Discovery.
6. The EMS triggers the deployment on the ADES.
7. The ADES executes the application.
COMPONENTS
List of Components

- Authentication (WSO2 bridge) [ATOS]
- Catalogue (FedEO gateway) [Spacebel]
- Client [Solenix]
- Application Package (format, not software)
- Servers
  - Execution Management Service (EMS) [CRIM, Geomatys, Spacebel]
  - Application Deployment & Execution Service (ADES) [CubeWerx, Geomatys, Spacebel]
- Applications
  - Feature extraction [CRIM]
  - Multi-sensor NDVI [Geomatys]
  - Stacker applications [CRIM, Geomatys]
AUTHENTICATION (WSO2 bridge)
WSO2 Identity Server

- Is an open source Identity and Access Management system.

- It bridges multiple SSO protocols (e.g. SAML, OIDC, WS-Federation) to provide a unified experience by doing protocol translations and transforming identity tokens between heterogeneous identity protocols.

- It has a mapping mechanism between the user attributes received from external authenticators and the attributes sent to service providers.

- In OGC Testbed 14, WSO2 Identity Server acts as a proxy/bridge to allow service providers to choose identity providers that use a different authentication protocol, e.g. a user could login to an OIDC application with a SAML identity provider.
WSO2 Identity Server as a Bridge in the OGC TB14 Context

External Authenticators

- EO IAM SAML IdP
- Cubewerx OIDC Provider

Configurable Authentication Methods

- CRIM OGC Test OIDC Applications
- Geomatys OIDC Application
- Solenix OIDC Application
- Magpie-ogc-ems OIDC Application
CATALOGUE
1. Execution request contains collection, area and time of interest.
2. Products are discovered through the FedEO gateway.
3. MEP hosting the data as execution target.

Objectives achieved by the Gateway:
- Discover products data
- Execute process close to data
Fedeo OpenSearch Gateway

• ESA FedEO OpenSearch Gateway
  – Common access point (API) for EO data discovery (EO Collection and Product metadata including product download links).
  – Standards-based access:
    • CEOS OpenSearch Best Practices – incl. "two-step search"
    • OGC 13-026r8 - OpenSearch Extension for Earth Observation
    • OGC 13-032r8 - OpenSearch Geo and Time Extensions
    • OGC 17-003 - GeoJSON(-LD) EO Product Metadata
    • OGC 17-047 - GeoJSON(-LD) OpenSearch Response.
    • Many flavours of responses and EO metadata including Atom, ISO, EOP O&M, GeoJSON(-LD), RDF/XML, JSON-LD, DIF-10, (Geo)DCAT(-AP)*
  – Deployments:
    • ESA FedEO (CEOS Context)
    • ESA Catalog (OADS) - transfer into operation Q1/19.
    • German Space Agency (DLR) CODE-DE
    • Copernicus CDS 3.0

OGC®
* Experimental

Copyright © 2018 Open Geospatial Consortium
CLIENT

OGC®
Making location count.
TEP Client

- https://tools.solenix.ch/ogctb14/
APPLICATION PACKAGE
Application Package in the architecture

- **Facet 1 (for users)** is CWL file -> user-provided or built using client (CWL editor)
- **Facet 2 (for M2M)** is WPS-T DeployProcess document -> The execution unit can be a Docker container for simple applications or a CWL file for workflows

<table>
<thead>
<tr>
<th>WPS-T Deploy executionUnit points to Docker</th>
<th>WPS-T Deploy executionUnit points to CWL workflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>WPS-T Deploy deploymentProfileName indicates it’s a Docker</td>
<td>WPS-T Deploy deploymentProfileName indicates it’s CWL</td>
</tr>
</tbody>
</table>

1 AP format (WPS-T DeployProcess in JSON) = 2 uses

For CWL-supporting ADES, CWL solves open issue from TB-13 concerning how to map inputs/outputs for use by Docker
EXECUTION MANAGEMENT SERVICE (EMS)
Execution Management Service

- 3 Implementations provided by:
  - CRIM
  - Geomatys
  - Spacebel
Execution Management Service (EMS)

- WPS-T using PyWPS / OWSLib with REST API
- OAuth2 support with WSO2 Gateway
- User permissions for visibility / execution of each application
- OpenSearch queries to obtain EO Image Data from collections
- Dynamic ADES deployment & execution based on collection data location
- CWL Workflow execution using **CWLTools** engine
- Error handling & logging, e.g. EMS, remote ADES monitoring, …
- WPS 2.0 JSON API proxy using spring boot / swagger codegen pojo.
- Validates user credentials and ACL for deployment/execution
- Performs product search on the FEDEO OpenSearch catalogue
- Transfers WPS request to the relevant ADES based on collection parameter.
- Merges result / quotation / status when multiple ADES are involved
- Executes CWL Workflow (manual parsing), dispatching each step to different ADES and merging the results

Github : [https://github.com/Geomatys/Testbed14/tree/master/EMS](https://github.com/Geomatys/Testbed14/tree/master/EMS)
• Java
• Based on 52° North Implementation
• Updated our WPS-T Extension from Testbed 13
• Docker & CWL workflow modules
• Workflow: rewrite the CWL to be executed on any CWL engine
• WPS-T REST proxy in front of XML implementation
• Proxy stub generated with Swagger Generator
APPLICATION DEPLOYMENT & MANAGEMENT SERVICE (ADES)
ADES

- Implemented in C
- Implements OGC WPS 1.0 & 2.0 Interfaces
- Implements REST-JSON API defined in this thread
  - Supports WPS transactional extension
  - Supports quotation and billing resources
  - Supports access control via /visibility resources
  - Messages encoding in JSON, XML or HTML
  - Implements notification extension (in addition to polling)
  - RFC 2616 content negotiation & ‘f’ parameter for minting encoding-specific URLs
- Can execute containers on local host OR using AWS Fargate compute engine
Secured using OAuth 2
Access controlled
- Who can execute what WPS operation
- Who can see, execute or undeploy what process
  - A process is owned by the user who deploys it
  - Can grant describe, execute and undeploy privileges to other users and roles
- Security requirements advertised in server’s OpenAPI doc
- Implements CORS support
- Secured server available here.
- Unsecured server available here.
ADES

- Based on the Examind-server WPS 2.0. It's a java based software, using Geotoolkit library.
- Handles WPS 1.0/2.0 in XML and JSON.
- Implements "Transactional" and "Quotation" profile
- Validates user credentials and ACL for deployment/execution
- Deploys/Executes CWL process using CWLTools Engine
- Located on VITO data center to access PROBAV product files
- Computes quotations and bills (based on input size)

Github: [https://github.com/Geomatys/Testbed14/tree/master/ADES](https://github.com/Geomatys/Testbed14/tree/master/ADES)

DEMONSTRATION VIDEO
Demo

- Alice deploys individual application(s)
  - Deploy MultiSensorNDVI (from Geomatys)
  - Deploy Stacker (from Geomatys)
- Alice’s sister deploys a workflow MultiSensor NDVI Stack Generator (chaining the two individual applications - MultiSensorNDVI and Stacker)
- Bob executes workflow
• Other videos:
  • Spacebel: https://portal.opengeospatial.org/files/?artifact_id=82366
    (acknowledgements: Neon Science – Microsoft – NASA – Spacebel)
  • CubeWerx: https://portal.opengeospatial.org/files/?artifact_id=82448
Not shown in video but in CFP

- **Billing/quotiation**
  - Proposed API covers some of it and some servers implement it

- **Application visibility/execute-ability**
  - Proposed API covers visibility and some servers implement it

- **Error handling**
  - Standard REST/JSON architecture approach over HTTP: JSON messages and standard HTTP error codes
Containers -VS- Virtual Machine

- Containers are an “app level” thing
  - They encapsulate everything needed to run an application
- Virtual machine are an “infrastructure level” thing
  - They turn one machine in many servers
Together containers and VMs offer a significant amount of flexibility to optimally deploy, manage and scale applications...
Common Workflow Language (CWL)

- A specification for describing analysis workflows and tools.
- Used in TB14 for:
  - Specifying the execution of a Docker application
  - Describing application chaining in a machine-readable way
- Advantages
  - Open standard supported by the scientific community
  - Multiple engine implementations (workstations, cluster, cloud, HPC)
  - Built for Docker (which is consistent with the AP deployment)
  - JSON is both human and machine-readable
  - Editors and viewers are available at no cost
  - Can be embedded in other applications
TB14 Workflow in CWL: Application Chaining

```json
{
    "cwlVersion": "v1.0",
    "class": "Workflow",
    "requirements": [...],
    "inputs": {...},
    "outputs": {...
        "classout": {
            "type": "File",
            "outputSource": "Stacker/output"
        }
    },
    "steps": {
        "NDVIMultiSensorCollection1": {
            "run": "GeomatysNDVIMultiSensor.cwl",
            "in": {
                "files": "image-collection1"
            },
            "out": ["output"]
        },
        "NDVIMultiSensorCollection2": {...},
        "Stacker": {
            "run": "GeomatysNDVIStacker.cwl",
            "in": {
                "files": {
                    "linkMerge": "merge_flattened",
                    "source": ["NDVIMultiSensorCollection1/output",
                                "NDVIMultiSensorCollection2/output"]
                }
            },
            "out": ["output"]
        }
    }
}
```

- Describes workflow
- Defines inputs / outputs
- Declares output from steps
- Workflow steps defined as CWL!
- Implicit chaining by inputs
TB14 Application in CWL: Execution

Describes application

Docker image reference

CWL engine automatically executes docker image

File type triggers volume mounting in docker container...

...and output retrieval

CWL resolves an open issue from Testbed-13

```json
{
  "cwlVersion": "v1.0",
  "class": "CommandLineTool",
  "requirements": {
    "DockerRequirement": {
      "dockerPull": "docker-registry.crim.ca/ogc-public/snap6-stack-creation:v2.12"
    }
  },
  "inputs": {
    "files": {
      "inputBinding": {
        "position": 1,
        "prefix": "-Pfiles=",
        "separate": false,
        "itemSeparator": ","
      },
      "type": {
        "type": "array",
        "items": "File"
      }
    }
  },
  "outputs": {
    "output": {
      "outputBinding": {
        "glob": "output.dim.zip"
      },
      "type": "File"
    }
  }
}
```
APPLICATIONS
Applications

- Stacker
- Feature extraction
- Flood detection
- MultisensorNDVI
- Workflow(s) chaining some of these
Applications

- Stacker (Overlapping region detection & alignment)
- Structural Feature Set
- Flood Detection

Packaged in Docker images
- Provides everything an application needs to run anywhere
- Wraps SNAP workflows in a docker

Delivered with CWL File
- Executed using standard interface

Built a complex workflow using chaining of the 3 applications!
Stacker Application

Overview
- Automatically selects overlapping regions of all input files
- Selects infrared bands
- Outputs BEAM_DIMAP stack ready for processing

Can be run on all MEP to produce a stack
- Sentinel (1, 2 or Radarsat, using collection Sentinel_2)
- ProbaV (Level 2 or 3, using collection PROBAV_S1-TOA_1KM_V001)
- Deimos (Format tiff, using collection DE2_MS4_L1B)

Can also be chained with itself
- Creates a single stack from results of 3 MEP stacks
## SFS & Flood Detection Application

### Structural Feature Set (SFS)
- Computes features for each pixel and produces a features stack
- Currently extracts standard deviation

### Flood Detection
- Uses SFS features
- Produces a stack of binary images of water-covered areas

---

<table>
<thead>
<tr>
<th>SFS Applications</th>
<th>Flood Detection Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sentinel-2 A</strong></td>
<td><strong>Uses SFS features</strong></td>
</tr>
<tr>
<td><strong>Deimos-1</strong></td>
<td><strong>Produces a stack of binary</strong></td>
</tr>
<tr>
<td><strong>Proba-V S1</strong></td>
<td><strong>images of water-covered</strong></td>
</tr>
<tr>
<td></td>
<td><strong>areas</strong></td>
</tr>
</tbody>
</table>

---

![Images of SFS and Flood Detection applications](images)
Example of a Complex Workflow

Flood Detection Workflow

- Sentinel-2 images
- Proba-V images
- Deimos images

Stacker → SFS → Flood Detection

Sentinel-2 MEP
Proba-V MEP
Deimos MEP
PFC Boreal Cloud
MultiSensorNDVI

- **Overview**
  - Takes as input 1..n Sentinel-2 (Zipped SAFE file), Proba-V (HDF5 file) and/or Deimos (TIFF file)
  - Provides as output one NDVI image in TIFF format per input image
  - Python script using GDAL/Numpy provided as docker image

- **Launch**

  ```
  # Work directory is current directory
  export WORKDIR=`pwd`
  # The file to process is located under ${WORKDIR}
  export S2FILE=S2A_MSIL1C_20180610T154901_N0206_R054_T18TXR_20180610T193029.SAFE.zip
  # Compute NDVI processing
  docker run -v ${WORKDIR}/${S2FILE}:${S2FILE} -v ${WORKDIR}:/outputs --workdir=/outputs images.geomatys.com/ndvims:latest /${S2FILE}
  ```

  [GitHub Link](https://github.com/Geomatys/Testbed14/tree/master/application-packages/NDVIMultiSensor)
Stacker

- **Overview**

  - Takes as input 2 .. \( n \) GeoTIFF images
  - Provides as output a single GeoTIFF in the same CRS as input images (or in EPSG:4326 if input CRSes differ) at the lowest resolution of input resolutions
  - Python script using GDAL provided as docker image

- **Launch**

  ```bash
  # Work directory is current directory
  export WORKDIR=`pwd`
  # The file to process is located under ${WORKDIR}
  export PV1FILE=PROBAV_L1C_20160505_232748_3_V101.tif
  export PV2FILE=PROBAV_L1C_20160505_232949_3_V101.tif
  # Compute NDVI processing
  docker run -v ${WORKDIR}:${PV1FILE} -v ${PV1FILE} -v ${WORKDIR}:${PV2FILE} -v ${WORKDIR}:/outputs --workdir=/outputs images.geomatys.com/ndvis:latest ${PV1FILE} ${PV2FILE}
  ```

  [https://github.com/Geomatys/Testbed14/tree/master/application-packages/NDVIStacker](https://github.com/Geomatys/Testbed14/tree/master/application-packages/NDVIStacker)
CHALLENGES & FUTURE WORK
Issues of interest

- Mapping of catalogue results
- Handling of multiple outputs
- Inputs/Outputs format
- EO image access on MEP
- CORS and other connectivity issues
- Short validity time of authentication token
- Many others (read D009 – “EMS and Best Practices ” ER)
Issue 1 – Mapping of Catalogue Results

1. EMS Catalog Query: Collection, TOI, AOI
2. Multiple Catalog results.
3. Results mapped to N' inputs
Real Cases: Multiple Combinations

How do we map Results to Inputs?

Query

Collection
TOI
AOI
Unique Collection

Collection
TOI
AOI
Multiple Collections

Too many results?

Input 1
Result 1
Result 2
Result 3
Single File Input

Input 2
Result 1
Result 2
Result 3
Result 4
Multiple File Input

Mapping of Collections to Inputs

Multiple TOI / AOI?
• Include Catalog Query in the Processing Chain
  → allows to define appropriate rules
Issue 2 – Handling of Multiple Outputs

- WPS 2.0 does not support Output arrays!
- Evaluated approaches:

<table>
<thead>
<tr>
<th>Method</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archive (e.g. zip)</td>
<td>Resource consuming for large files</td>
</tr>
<tr>
<td>Metalink</td>
<td>Parser required, file reference only</td>
</tr>
<tr>
<td>Multipart/mixed</td>
<td>Not supported by all browsers.</td>
</tr>
<tr>
<td></td>
<td>No statement about individual files.</td>
</tr>
</tbody>
</table>

→ Recommendations:
- Metalink if external output reference is available
- Otherwise: WPS 3.0 support?
File Mime Type: TIFF, JPG, ZIP

→ Not satisfactory : not precise enough

Processes consumes specific formats:

• Optical Sensors: Sentinel-2, Sentinel-3, Spot, NOAA, …
• Radar Sensors: Sentinel-1, TerraSAR, Kompsat-S, …
• Vector Data: Shapefile, GeoJSON, …
• Generic Data: JP2000, GeoTiff, ENVI, …
As opposed to Testbed 13, the external (HTTP) URL of data is provided by the Catalogue.

ADES needs to map the URL to the local file location


/nfs/eodata/Sentinel-2/MSI/L1C/2018/01/30/S2B_MSIL1C_20180130T034959_N0206_R104_T47PPT_20180130T064159.SAFE
Issue 5 - Network-level issues

To be addressed in operational/realistic context:

- CORS
  - Either configure servers to accept all origins:
    - Access-Control-Allow-Origin: *
  - Bypass using browser plugin (Chrome-specific):
    - https://chrome.google.com/webstore/detail/allow-control-allow-origin/nlfbmbojpeacfgkhkpbjhdhjklkiljb?hl=en
- Mixed HTTP/HTTPS content (particularly due to WSO2 endpoint):
  - Browser-specific solution:
    - Firefox and Chrome: shield icon on address bar to bypass
    - IE: “Show all content” at bottom of screen
Issue 6 - Token Validity

- Issue #1
  - Web Client should update the access_token on a regular basis to ensure that requests sent to the EMS/ADES provide fresh access_token.
  - Concern if access_token expires during workflow execution.

- Issue #2
  - To deploy ADES on MEPs, EMS must provide a user’s access_token that give rights to deploy process on ADES. Since the execute request is sent by Bob, the EMS has access to the access_token from Bob and not from Alice.
  - Two solutions
    - We give Bob the right to deploy process on ADES.
    - We introduce an "ems user" associated to EMS. This user can deploy process on ADES. Each request from EMS to ADES uses the credentials of this user instead of the real user (i.e. Alice or Bob).
Future Work

- CRs to WPS SWG and/or Best Practices for use of WPS-T DeployProcess as EP AP
- Consolidate the WPS REST API
- Initiate standardization path for WPS-T
- Broader issues:
  - How to transform message-based protocols such as WPS (GetCapabilities, DescribeProcess, Execute) into resource-based protocols such as WPS REST (*processes, jobs* resources, HTTP GET/POST/PUT) -> straightforward conversion may result in non-intuitive and semantically unclear APIs
  - How to transform/encode XML into JSON content
- Relevance of CWL to OGC -> potential Community Standard
- Many other suggestions (read D009 – “EMS and Best Practices ” ER)
STANDARDIZATION-RELATED ISSUES
• Participated in WPS-T 2.0 draft (OGC 13-071r1)
• Some WPS-T implementations (e.g. TB-13, SSEGrid, …)
→ Cooperated with NextGen Thread (WPS-T ER)
New WPS-T Schemas

Includes all required elements for deploying an application package

- **DESCRIPTION OF INPUTS/OUTPUTS**
- **WORKFLOW CWL**
- **DOCKER IMAGE**
- **WORKFLOW OR DOCKER**
Remaining Issue

- WPS 2.0: Processing Offering = ProcessDescription + Attributes

- Testbed-14: Confusing structure

- Recommendation:
REST-JSOn API

- Based on work already being done by the WPS SWG to define a REST API
- Based on the NextGen service pattern defined by WFS 3.0
- The API is described using OpenAPI so it is developer and tool friendly
## REST-JSON API - Discovery

<table>
<thead>
<tr>
<th>Resource/Path</th>
<th>HTTP Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>GET</td>
<td>The landing page provides links to the API definition, the Conformance statements and the metadata about the processes offered by this API</td>
</tr>
<tr>
<td>/api</td>
<td>GET</td>
<td>Get’s the server’s OpenAPI document</td>
</tr>
<tr>
<td>/conformance</td>
<td>GET</td>
<td>Lists all requirements classes specified in the standard (WPS REST/JSON Binding Core) to which the server conforms</td>
</tr>
</tbody>
</table>
## REST-JSOn API - Processes

<table>
<thead>
<tr>
<th>Resource/Path</th>
<th>HTTP Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>/processes</td>
<td>GET</td>
<td>Retrieve available processes</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Deploy a process</td>
</tr>
<tr>
<td>/processes/{id}</td>
<td>GET</td>
<td>Retrieve a process description</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>Undeploy a process</td>
</tr>
<tr>
<td>/processes/{id}/quotations</td>
<td>GET</td>
<td>Retrieve the list of quotation ids for a given process</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Request a quotation for a given process</td>
</tr>
<tr>
<td>/processes/{id}/quotations/{quotationID}</td>
<td>GET</td>
<td>Retrieve quotation information</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Execute a quoted process</td>
</tr>
</tbody>
</table>
## REST-JSON API - Jobs

<table>
<thead>
<tr>
<th>Resource/Path</th>
<th>HTTP Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>/processes/{id}/jobs</td>
<td>GET</td>
<td>Retrieve the list of jobs for a process</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Execute a process</td>
</tr>
<tr>
<td>/processes/{id}/jobs/{jobID}</td>
<td>GET</td>
<td>Retrieve the status of a job</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>Dismiss a job</td>
</tr>
<tr>
<td>/processes/{id}/jobs/{jobID}/result</td>
<td>GET</td>
<td>Retrieve the result(s) of a job</td>
</tr>
</tbody>
</table>
# REST-JSON API – Quotation

<table>
<thead>
<tr>
<th>Resource/Path</th>
<th>HTTP Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>/quotations</td>
<td>GET</td>
<td>Retrieve the list of all quotation ids</td>
</tr>
<tr>
<td>/quotations/{quotationID}</td>
<td>GET</td>
<td>Retrieve quotation information</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Execute a quoted process</td>
</tr>
<tr>
<td>/bills</td>
<td>GET</td>
<td>Retrieve the list of all bill identifiers</td>
</tr>
<tr>
<td>/bills/{billID}</td>
<td>GET</td>
<td>Retrieve bill information</td>
</tr>
</tbody>
</table>
## REST-JSON API – Access Control

<table>
<thead>
<tr>
<th>Resource/Path</th>
<th>HTTP Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>/processes/{id}/visibility</td>
<td>GET</td>
<td>Retrieve the visibility status for a process</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td>Change the visibility status for a process</td>
</tr>
</tbody>
</table>
Quotation & Billing

• EMS
  – The server keeps in memory a mapping between the EMS generated JobId and one or more subsequent ADES jobIDs. The same mechanism is applied for quotations and bills.
  – When receiving a request, if multiple ADES are involved, all ADES quotations are merged in one single quotation

• ADES
  – Minimalistic approach: the estimated time and price are calculated based on a ratio of input files' size.
  – On successful quotation request, the quotation response is stored with the Execute request within the ADES database.
  – The quotation has a validity period. An execution request based on the quotation outside of the validity period is refused.
  – Bill is provided as a link sent within the results response.
CONCLUSION/ACHIEVEMENTS
Conclusion/Achievements

• 5 different companies working together in the OGC manner, i.e. consensus-based, to:
  • Propose and write-down a standard Application Package format (D008 – “Application Package” ER)
  • Propose a WPS-T JSON/REST API - partly developed in coordination with the WPS 2.0 SWG - and write-down best practices and experiences about the EMS & ADES, two essential components of a EP (D009 – “EMS & ADES Best Practices and Results” ER)

• 1 client, 3 different EMS and 3 different ADES implementations able to interact in a standardized and interoperable way:
  • Using the proposed AP format
  • Using the proposed WPS-T REST/JSON API
  • Integrating several different TEPs & MEPs, authN/authZ, catalogue in flow

• With the support of the OGC, brought ESA closer to the goal of standardizing parts of the very complex EP ecosystem
Acknowledgements

- ESA: Cristiano Lopes, with support from Antonio Romeo, Marco Leonardi
- NRCan: Brian Low
- OGC: Ingo Simonis, Scott Serich, Greg Buehler
- Deimos-Imaging
- VITO
- FedEO Team at Spacebel
- ATOS CVC Romania

Participants:
- CRIM: Tom Landry, David Byrns, Kevin Heffner
- CubeWerx: Peter Vretanos
- Geomatys: Jérôme Gasperi, Guilhem Legal, Vincent Heurteaux
- Solenix: Paulo Sacramento, Dan Robinson
- Spacebel: Patrick Jacques, Christophe Noël, Yves Coene