TEP Use Cases, Architecture & Standards

Salvatore Pinto

20/05/2016
Outline

• TEP use cases
• TEP architecture & interfaces (work-in-progress)
• OGC Testbed 13 & Open Source Framework
Scenario: A scientific user has developed a software application (working on files, e.g. Level 1, maybe composed by different modules) and he or another user wants to run it into the platform (close to the data)

Use case:

1. User S integrates his software application into the platform
   a. Defines the software application package and execution information
   b. Uploads the software application package into the platform processing service catalogue

2. User P selects the service from the processing service catalogue

3. User P selects the input products from the data catalogue

4. User P executes the processing service
   a. Processing service discovers additional auxiliary data
   b. Processing service resolves the location of the input and auxiliary data
   c. Processing service executes software applications to create some output data
   d. Output data is registered into the platform and provided as output of the processing service

5. User P visualizes the processing results and downloads them
TEP use case #2: Scientific user exploitation (workflow)

Scenario: A scientific user wants to run a set of processing services linked together and make this available to other users.

Use case:

1. User S creates a new processing service as a set of processing services already in the platform or in other platforms
   a. Defines the relations between the processing services in a workflow package
   b. Uploads the workflow package and registers it into the platform processing service catalogue

2. User P selects the newly created processing service from the processing service catalogue

3. User P selects the input products from the data catalogue

4. User P executes the processing service
   a. Processing service discovers additional auxiliary data
   b. Processing service resolves the location of the input and auxiliary data
   c. Processing service executes software applications to create some output data
   d. Output data is registered into the platform and provided as output of the processing service

5. User P visualizes the processing results and downloads them
TEP use case #3: Commercial data

Scenario: A user wants to process commercial data from commercial cloud providers, this includes the access to the data at the cloud provider premises

1. User S integrates his processing algorithm (software applications) into TEP
2. User P selects the input products from the data catalogue of commercial MEP
3. User P checks an estimate of the cost of the intended processing
4. User P executes the processing service on TEP
   a. TEP A deploys the processing service on commercial MEP
   b. TEP A executes the processing service on commercial MEP, the processing is accounted by commercial MEP to User P
   c. Processing service resolves the input data location on MEP
   d. Processing service on MEP, if needed, discovers and gathers additional auxiliary data from TEP
   e. Output data generated on MEP is published into the TEP
5. User P visualizes the processing results on the TEP and downloads them from TEP
6. User P visualizes the accounted processing resources from the commercial MEP on TEP
7. User P gets billed by the TEP which will share revenue with the MEP according to the accounted resource consumption
Scenario: A commercial user wants to offer his commercial service based on a commercial software application via the platform to customers, in a pay-per-use model

Use case:

1. User C integrates his processing algorithm (software applications) into the platform. Commercial software license is not included in the package.
2. User C issues a commercial license for the platform users with given T&C
3. User P selects the service from the processing service catalogue
4. User P selects the input products from the data catalogue
5. User P checks the cost of the intended processing (e.g. simulated data and processing costs according to AOI definition and number of inputs)
6. User P submits the processing, output data is registered into the platform
7. User P usage is accounted for the resources consumed including the use of the software with the relative AOI and parameters, user C bills the user P
8. User P visualizes the processing results and downloads them
Scenario: A commercial or scientific user wants to generate a collection of value added products and update it in near-real time or in bulk.

Use case:

1. User C integrates his processing algorithm (software applications) into the platform.
2. User C starts a bulk processing campaign to process the an AOI over the entire mission data collection, the output is automatically published into the catalogue.
3. User C monitors the processing status of the bulk processing.
4. User C setups a near-real time processing campaign to process new data as soon as it is acquired, the output is published into the catalogue.
5. User C monitors the processing status of the near-real time processing.
6. User Q search for the products into the catalogue, optionally he has a cost for visualizing the data or for downloading the data.
7. Platform exports product collection to a DataCube.
8. User P accesses the data cube for execution of interactive post-analysis tasks.
TEP use case #5: Collaboration

Scenario: An user shares development, processing results or processing services execution with the community

Use case:

1. User S integrates his processing algorithm (software applications) in a collaborative development (e.g. using collaboration functions of GitHub).
2. User S selects the processor, input data and executes the processing, the output is published into the platform for User S access only.
3. User S shares the results of the processing with another user, group or the public.
4. User Q can comment or add annotation on User S results.
5. User S shares the processing execution
   a. A virtual experiment package is created from the processing execution, linking to all the information to reproduce the execution.
   b. The virtual experiment is shared with the user, group or the public.
6. User Q reproduces the experiment (eventually changing some parameters).
7. User S references in his scientific paper the virtual experiment (or the input data or the results).
8. User Q re-runs the experiment in the future (e.g. after 2 years).
TEP use case #6: Interactive Application

Scenario: A user wants to use an interactive application (e.g. MATLAB, Sentinel Toolbox, ...) but he has no access to the input data, no license or no skills on how to configure and run the application

Use case:
1. User S integrates an interactive application for post-analysis
2. User Q selects an interactive service from the service catalogue
3. User Q selects the collection from the service data catalogue
4. User Q requests to start the service
5. Platform provides user Q with a web interface to perform interactive processing
6. Platform automatically pauses the interactive application if it is in idle. If paused, User Q can request to restart it
7. User Q publishes results generated in the interactive application into the platform
8. User Q stop and interactive processing service
TEP use case #7: Development

Scenario: A commercial user wants to use the platform to develop an algorithm or a software application using the platform tools (e.g. iPython, AppScale, ...) and data

Use case:

1. User S accesses interactive development service
2. User S discovers the input data using the API of the development service
3. User S develops the software application using the API of the development service
4. User S exports the software application for execution into the platform
   a. Platform defines the software application package and execution information
   b. Platform uploads the software application package into the platform processing service catalogue
5. User P executes the processing service
Scenario: in all the use cases, all the user shall be authenticated with single credentials from their own organization for web and M2M interfaces and they shall be able to delegate the platform to act on their behalf.

User Application
- Native App
- Browser App (javascript/ajax)
- Script
- ...

Application first calls the Login Services (provides a token) to get back a token. The token is propagated. It needs to be accepted by IDP#2 (federation?).

Web Services (catalogue, WMS browse & maps, WPS processing, Product download, ...)
Login Services (provides a token)
Outline

• TEP use cases

• **TEP architecture & interfaces (work-in-progress)**

• OGC Testbed 13 & Open Source Framework
The **Exploitation Platform Open Architecture** definition is an on-going project, building on experience from the TEP and related projects. Currently, the **DRAFT3** has been released with Creative Commons Attribution-Share Alike 4.0 International License.
Full components list

User Access Portal

Resource Management

Execution Environment

Service Integration

Available on the TEP website (http://tep eo esa.int) or directly here
## EP Open Architecture last release

<table>
<thead>
<tr>
<th>Documents last baseline</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP Open Architecture</td>
<td>DRAFT3</td>
</tr>
<tr>
<td>EP Processing Service Execution Interface</td>
<td>DRAFT1</td>
</tr>
<tr>
<td>EP Processing Service Packaging</td>
<td>DRAFT1</td>
</tr>
<tr>
<td>EP Resource Access Interface</td>
<td>DRAFT1</td>
</tr>
<tr>
<td>EP Resource Catalogue Interface</td>
<td>DRAFT1</td>
</tr>
<tr>
<td>EP Resource Ingestion Interface</td>
<td>DRAFT1</td>
</tr>
<tr>
<td>EP Virtual Experiment Packaging</td>
<td>DRAFT1</td>
</tr>
<tr>
<td>EP Common Core Components</td>
<td>DRAFT1</td>
</tr>
</tbody>
</table>

Available in the TEP website, [EP Open Architecture Draft 3 Folder](#)
## EP Interfaces & Standards

<table>
<thead>
<tr>
<th>Documents released (DRAFT1)</th>
<th>Baseline standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP Processing Service Execution Interface</td>
<td><strong>WPS 2.0.0</strong></td>
</tr>
<tr>
<td>EP Processing Service Packaging</td>
<td>-</td>
</tr>
<tr>
<td>EP Resource Access Interface</td>
<td><strong>OGC Download BP</strong></td>
</tr>
<tr>
<td>EP Resource Catalogue Interface</td>
<td><strong>OGC OpenSearch, WMS, WCS, WFS</strong></td>
</tr>
<tr>
<td>EP Resource Ingestion Interface</td>
<td>-</td>
</tr>
<tr>
<td>EP Virtual Experiment Packaging</td>
<td><strong>OGC OWS Context</strong></td>
</tr>
</tbody>
</table>

For details you can refer to the documents referenced in the [EP Open Architecture Draft 3](#).
Key points for the standards selection in the EP:

- **Web/developer oriented API**
- M2M communication
- Simple client
- Efficient encoding

Which translates to preference to the following technologies:

- **JSON over XML**
- **REST over SOAP**
- Pre-defined namespace vs custom namespaces

Moreover, standards shall contain Best Practices for Implementation, with:

- Clear definition of **namespaces, nomenclature and conventions**
- Definition of End-to-end Scenario involving the interface/product
- **Samples for all messages exchange** in the end-to-end scenario
OGC Web Processing Service
Evolution of Processing service execution

Proposed evolution for the Processing service standard

1. JSON rendering of processing status response (including processing logs, cost, etc...)
2. Dynamic UI parameters (e.g. scripting for default and/or constrained parameters)
3. Interactive processing support (job pause and interactive feedback)
4. Quota support (daily, monthly, yearly, total, concurrent and absolute quota, quota status retrieval, set quota for users)
5. Cost support, a-priori (eg. via SimulateEx call) and a-posteriori
6. Add pre-defined output/input data parameters (e.g. input parameters are OpenSearch references, outputs are OpenSearch references to files and processing logs)
7. Add possibility to request massive processing (time-driven or data driven)
Proposed evolution for the Processing service standard

1. JSON rendering of processing status response (including processing logs, cost, etc...)

Rationale: simplicity of the implementation

```
"wps:ProcessDescriptions": {=}
  "-service": "WPS",
  "-version": "1.0.0",
  "-xml:lang": "en-GB",
  "-xsi:schemaLocation": "http://www.opengis.net/wps/1.0.0",
  "-xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
  "-xmlns:xlink": "http://www.w3.org/1999/xlink",
  "-xmlns:wps": "http://www.opengis.net/wps/1.0.0",
  "-xmlns:ows": "http://www.opengis.net/ows/1.1",
  "ProcessDescription": {=}
    "-wps:processVersion": "1.0.0",
    "-storeSupported": "true",
    "-statusSupported": "true",
    "ows:Identifier": "MIRAVI",
    "ows:Title": "MIRAVI Geo",
    "ows:Abstract": "Service access point for the MIRAVI service. The request must specify the operation and input parameters to be processed. The output parameters can be specified to retrieve the results of the processing."
  }[=]
  "Input": {=}
    "-minOccurs": "0",
    "-maxOccurs": "1",
    "ows:Identifier": "ce",
    "ows:Title": "Computing Element",
    "LiteralData": {=}
      "ows:DataType": {=}
        "-ows:reference": "http://www.w3.org/TR/xmlschema-2/#string",
        "#text": "string"
      },
      "ows:AllowedValues": {=}
        "ows:Value": [=]
```
OGC Web Processing Service
Evolution of Processing service execution

Proposed evolution for the Processing service standard

1. JSON rendering of processing status response (including processing logs, cost, etc...)

2. **Dynamic UI parameters (e.g. scripting for default and/or constrained parameters)**
   
   **Rationale:** for use case #1 (Scientific Exploitation)

```xml
<Input minOccurs="0" maxOccurs="1">
  <ows:Identifier>wi_frame</ows:Identifier>
  <ows:Title>wi_frame:</ows:Title>
  <LiteralData>
    <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
    <ows:DefaultValue>javascript:30*proj.value</ows:DefaultValue>
  </LiteralData>
</Input>

<Input minOccurs="0" maxOccurs="1">
  <ows:Identifier>proj</ows:Identifier>
  <ows:Title>Projection</ows:Title>
  <ows:Abstract>Plate-carre;Plate-carre;UTM;UTM;Stereographic;Stereographic</ows:Abstract>
  <LiteralData>
    <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
    <ows:AllowedValues>
      <ows:Value>Plate-carre</ows:Value>
      <ows:Value>UTM</ows:Value>
      <ows:Value>Stereographic</ows:Value>
    </ows:AllowedValues>
    <ows:DefaultValue>javascript:wi_frame.enabled=not proj.enabled;</ows:DefaultValue>
  </LiteralData>
</Input>
```
Proposed evolution for the Processing service standard

1. **JSON rendering of processing status response** (including processing logs, cost, etc...)
2. **Dynamic UI parameters** (e.g. scripting for default and/or constrained parameters)
3. **Interactive processing support (job pause and interactive feedback)**

**Rationale:** for use case #6 (Post-Analysis)
OGC Web Processing Service
Evolution of Processing service execution

Proposed evolution for the Processing service standard

1. JSON rendering of processing status response (including processing logs, cost, etc...)
2. Dynamic UI parameters (e.g. scripting for default and/or constrained parameters)
3. Interactive processing support (job pause and interactive feedback)
4. Quota support (daily, monthly, yearly, total, concurrent and absolute quota, quota status retrieval, set quota for users)

**Rationale:** for use case #1 (Scientific Exploitation)

**e.g.:** GET /wps/?service=WPS&Request=GetQuota&identifier=MIRAVI
OGC Web Processing Service
Evolution of Processing service execution

Proposed evolution for the Processing service standard

1. JSON rendering of processing status response (including processing logs, cost, etc…)
2. Dynamic UI parameters (e.g. scripting for default and/or constrained parameters)
3. Interactive processing support (job pause and interactive feedback)
4. Quota support (daily, monthly, yearly, total, concurrent and absolute quota, quota status retrieval, set quota for users)
5. Cost support, a-priori (e.g. via SimulateEx call) and a-posteriori

**Rationale:** for use case #2 (Commercial Exploitation)
OGC Web Processing Service
Evolution of Processing service execution

Proposed evolution for the Processing service standard

1. JSON rendering of processing status response (including processing logs, cost, etc...)
2. Dynamic UI parameters (e.g. scripting for default and/or constrained parameters)
3. Interactive processing support (job pause and interactive feedback)
4. Quota support (daily, monthly, yearly, total, concurrent and absolute quota, quota status retrieval, set quota for users)
5. Cost support, a-priori (e.g. via SimulateEx call) and a-posteriori
6. Add pre-defined output/input data parameters (e.g. input parameters are OpenSearch references, outputs are OpenSearch references to files and processing logs)

Rationale: for simplicity of the implementation for all use cases and in particular for use case #3 (Massive processing) and #5 (Platform interoperability)
Create a new standard or Best Practices for packaging processing services

Proposed new standard key elements:

1. Support both not-interactive processing (e.g. Workflow) and interactive processing services (e.g. App)
2. “WPS describe process”—like for basic job input parameters definition
3. Processing graph definition
4. Support possibility to chain workflow and import/link other workflow packages
5. Includes software packaging standard definition (eg. RPM, Docker) and wrappers

**Rationale:** from all the integration use cases and in particular use case #5 (Platform Interoperability)
OGC Web Processing Service
Evolution of Processing service execution

Proposed evolution for the Processing service standard

1. JSON rendering of processing status response (including processing logs, cost, etc…)
2. Dynamic UI parameters (e.g. scripting for default and/or constrained parameters)
3. Interactive processing support (job pause and interactive feedback)
4. Quota support (daily, monthly, yearly, total, concurrent and absolute quota, quota status retrieval, set quota for users)
5. Cost support, a-priori (eg. via SimulateEx call) and a-posteriori
6. Add pre-defined output/input data parameters (e.g. input parameters are OpenSearch references, outputs are OpenSearch references to files and processing logs)
7. Add possibility to request massive processing (time-driven or data driven)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Occurrences</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFRESH_PERIOD</td>
<td>Min. 0 Max. 1</td>
<td>LiteralData: date</td>
<td>Describes time period of the checks for new data to be processed</td>
</tr>
<tr>
<td>REFRESH_QUERY</td>
<td>Min. 0 Max. N</td>
<td>LiteralData: anyURI</td>
<td>OpenSearch descriptor for the data query requests for new data to be processed</td>
</tr>
</tbody>
</table>

**Rationale:** from use case #3 (Massive Processing)
Create a new standard or Best Practices for packaging processing services

Proposed new standard key elements:

1. **Support both not-interactive processing (e.g. Workflow) and interactive processing services (e.g. App)**
Create a new standard or Best Practices for packaging processing services

Proposed new standard key elements:

1. Support both not-interactive processing (e.g. Workflow) and interactive processing services (e.g. App)

2. “WPS describe process”-like for basic job input parameters definition

```xml
  <creator>Salvatore Pinto</creator>
  <created>2016-04-22T15:45:752</created>
  <title>Sample package for ROI-PAC application</title>
  <abstract>This is an example of an interactive and non-interactive packaging of the ROI-PAC application. One non-interactive application to calculate interferograms via ROI-PAC is provided with two different workflows (using orbit files correction and not) and one interactive application is provided to allow GUI access to ROI-PAC with freedom for the user to control all the different application features.</abstract>
  <packageVersion>1.0</packageVersion>
  <processingService id="roipac_insar" type="non-interactive">
    <version>1.0</version>
    <inputParameters>
      <wps:Input minOccurs="1" maxOccurs="1">
        <ows:Identifier INPUTDATA</ows:Identifier>
        <wps:LiteralData>
          <ows:Format mimeType="text/plain" default="true"/>
          <ows:LiteralDataDomain default="true"/>
          <ows:DataType
            csw:reference="http://www.w3.org/2001/XMLSchema#anyURI">anyURI</ows:DataType>
        </wps:LiteralData>
      </wps:Input>
    </inputParameters>
  </processingService>
</package>
```
Create a new standard or Best Practices for packaging processing services

Proposed new standard key elements:

1. Support both not-interactive processing (e.g. Workflow) and interactive processing services (e.g. App)

2. “WPS describe process”-like for basic job input parameters definition

3. Processing graph definition

```xml
<inputs>
  <outputs>
    <outputFolder>/application/dem/output/</outputFolder>
  </outputs>
</jobConfiguration>

<node id="roipac">
  <executable type="rpm">
    <executablePackage>https://myrepository.server/rpm/broadgdac/roi-pac-x64-1.0.1.rpm</executablePackage>
  </executable>
  <inputs>
    <inputsFolder>/application/roipac/input/</inputsFolder>
    <input name="INPUTDATA" source="inputParameters">INPUTDATA</input>
    <input name="slave" source="inputParameters">slave</input>
    <input name="dem" source="node">dem</input>
    <input name="useaux" source="inputParameters">useorbitdata</input>
    <input name="aux" source="node">aux</input>
  </inputs>
  <outputs>
    <outputFolder>public=true</outputFolder>
  </outputs>
</node>
</jobConfiguration>
```
Create a new standard or Best Practices for packaging processing services

Proposed new standard key elements:

1. Support both not-interactive processing (e.g. Workflow) and interactive processing services (e.g. App)
2. “WPS describe process”-like for basic job input parameters definition
3. Processing graph definition
4. **Support possibility to chain workflow and import/link other workflow packages**
Create a new standard or Best Practices for packaging processing services

Proposed new standard key elements:

1. Support both not-interactive processing (e.g. Workflow) and interactive processing services (e.g. App)
2. “WPS describe process”-like for basic job input parameters definition
3. Processing graph definition
4. Support possibility to chain workflow and import/link other workflow packages
5. Includes software packaging standard definition (eg. RPM, Docker) and wrappers

```xml
<node id="roipac">
  <executable type="rpm">
    <executablePackage>https://myrepository_server/rpm/roipac.rpm</executablePackage>
    <executablePath>/application/roipac/run.sh</executablePath>
  </executable>
  <inputsFolder>/application/roipac/input/</inputsFolder>
  <input name="INPUTDATA" source="inputParameters">INPUTDATA</input>
  <input name="slave" source="inputParameters">slave</input>
  <input name="dem" source="node">dem</input>
  <input name="useaux" source="inputParameters">useorbitdata</input>
  <input name="aux" source="node">aux</input>
</node>
```
Proposed evolution for the Data Access best practice

1. Add JSON and HTML renderings
2. Quota and cost management
3. User ACL management
Proposed evolution for the Data Access best practice

1. Add JSON and HTML renderings

**Rationale:** for use case #1 (Scientific Exploitation)
OGC Download Service for EO Products BP
Evolution of Data Access standard

Proposed evolution for the Data Access best practice

1. Add JSON and HTML renderings
2. Quota and cost management
3. User ACL management

<table>
<thead>
<tr>
<th>Metadata Class</th>
<th>Metadata Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl</td>
<td>owner</td>
<td>Owner of the data</td>
</tr>
<tr>
<td></td>
<td>group</td>
<td>Group owner of the data</td>
</tr>
<tr>
<td></td>
<td>permission</td>
<td>Read/write/access permissions in POSIX format (e.g. 770)</td>
</tr>
<tr>
<td>quota</td>
<td>period</td>
<td>Period of refresh of the quota associated to the resource collection</td>
</tr>
<tr>
<td></td>
<td>total</td>
<td>Total number of downloads allowed per quota period associated to the resource collection</td>
</tr>
<tr>
<td></td>
<td>left</td>
<td>Total number of allowed downloads left per quota period associated to the resource collection</td>
</tr>
<tr>
<td>cost</td>
<td>amount</td>
<td>Amount to be paid for access the resource expressed in credits</td>
</tr>
<tr>
<td></td>
<td>paid_period</td>
<td>Period of time associated to the paid amount for a single purchase. This is expressed in seconds.</td>
</tr>
<tr>
<td></td>
<td>paid_period_left</td>
<td>Period of time left associated to the last purchase. This is expressed in seconds.</td>
</tr>
</tbody>
</table>

```json
{
  acl: { owner: "admin", group:"admin,normalusers", permission:"640" },
  quota: { period: "year", total:10, left:1},
  cost: { amount: 1000, paid_period: 10, paid_period_left: 0 }
}
```

**Rationale:** for all cases involving download, in particular for #2 (Commercial Exploitation)
Proposed evolution for the OpenSearch standard:

1. Add JSON rendering with support to JSON-LD
2. Define metadata for catalogue of Processing Services, Interactive Applications, Processing Results
3. Standardize data preview format for geo-coded data (e.g. via OGC Context or KML)
4. Add catalogue statistics (spatial, temporal coverage and custom statistics)
5. Add temporal WKT support
6. Clarify relations with CSW 3.0 OpenSearch endpoint
Proposed evolution for the OpenSearch standard:

1. Add JSON rendering (e.g. with support to JSON-LD or GeoJSON)

Rationale: for simplicity of the implementation and better intercompatibility
Proposed evolution for the OpenSearch standard:

1. Add JSON rendering with support to JSON-LD

2. Define metadata for catalogue of Processing Services, Interactive Applications, Processing Results

**Rationale:** from all use cases

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Occurrences</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>application:id</td>
<td>Unique identifier of the application. Mandatory.</td>
<td>minOccurs=1</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maxOccurs=1</td>
<td></td>
</tr>
<tr>
<td>application:name</td>
<td>Name of the application. Mandatory.</td>
<td>minOccurs=1</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maxOccurs=1</td>
<td></td>
</tr>
<tr>
<td>application:description</td>
<td>A short description of the application. Optional.</td>
<td>maxOccurs=1</td>
<td>String</td>
</tr>
<tr>
<td>application:abstract</td>
<td>A description of the application in terms input output, minimum requirements to run the application. Optional.</td>
<td>maxOccurs=1</td>
<td>String</td>
</tr>
<tr>
<td>application:owner</td>
<td>Link to the OSSD of the person, institute, company (see contactInfo) who own the application. Mandatory</td>
<td>minOccurs=1</td>
<td>anyUri</td>
</tr>
<tr>
<td>application:publisher</td>
<td>Link to the OSSD of the person, institute, company (see contactInfo) who has published the application on the platform. Mandatory</td>
<td>minOccurs=1</td>
<td>anyUri</td>
</tr>
<tr>
<td>application:addedon</td>
<td>Date when the application has been uploaded to the platform. Mandatory.</td>
<td>minOccurs=1</td>
<td>dateTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maxOccurs=1</td>
<td></td>
</tr>
<tr>
<td>application:lastupdate</td>
<td>Date of the last update of the application. Mandatory.</td>
<td>minOccurs=1</td>
<td>dateTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maxOccurs=1</td>
<td></td>
</tr>
<tr>
<td>application:publications</td>
<td>Link to the OSSD of the publications related to the application</td>
<td>minOccurs=1</td>
<td>anyUri</td>
</tr>
<tr>
<td>application:dependency</td>
<td>Link to the OSSD of the list of all the applications which are required to run the current application (see application)</td>
<td>minOccurs=0</td>
<td>anyUri</td>
</tr>
</tbody>
</table>
Proposed evolution for the OpenSearch standard:

1. Add JSON rendering with support to JSON-LD
2. Define metadata for catalogue of Processing Services, Interactive Applications, Processing Results
3. Add management interface (data ingestion, data edit, data delete)

**Rationale:** from all use cases

**NOTE:** HTTP RESTful, e.g.: PUT /{Product URI}/atom
OpenSearch
Evolution of Resource Search standard

Proposed evolution for the OpenSearch standard:

1. Add JSON rendering with support to JSON-LD
2. Define metadata for catalogue of Processing Services, Interactive Applications, Processing Results
3. Standardize data preview format for geo-coded data (e.g. via OGC Context or KML)

Rationale: from all use cases which involves visualization
OpenSearch
Evolution of Resource Search standard

---

Proposed evolution for the OpenSearch standard:

1. Add JSON rendering with support to JSON-LD
2. Define metadata for catalogue of Processing Services, Interactive Applications, Processing Results
3. Standardize data preview format for geo-coded data (e.g. via OGC Context or KML)
4. Add catalogue statistics (spatial, temporal coverage and custom statistics)

---

Get **Spatial Coverage Stats**: Coverage statistics as WMS layer

Get **Temporal Coverage Stats**: Coverage statistics

---

**Rationale**: from all use cases which involves data selection, in particular use case #1 (Scientific Exploitation)
Proposed evolution for the OpenSearch standard:

1. Add JSON rendering with support to JSON-LD
2. Define metadata for catalogue of Processing Services, Interactive Applications, Processing Results
3. Standardize data preview format for geo-coded data (e.g. via OGC Context or KML)
4. Add catalogue statistics (spatial, temporal)
5. Add temporal WKT support

**Rationale:** from all use cases which involves data selection, in particular use case #1 (Scientific Exploitation)
Proposed evolution for the OpenSearch standard:

1. Add JSON rendering with support to JSON-LD
2. Define metadata for catalogue of Processing Services, Interactive Applications, Processing Results
3. Standardize data preview format for geo-coded data (e.g. via OGC Context or KML)
4. Add catalogue statistics (spatial, temporal coverage and custom statistics)
5. Add temporal WKT support
6. **Clarify relations with CSW 3.0 OpenSearch endpoint**
Create a new Best Practices for ingesting input data, processing services, processing results or other data into the platform, including binary representation upload and metadata definition.

This shall

1. Use HTTPs for upload
2. Allow specification of metadata or extraction
3. Includes the following sub-VE definitions:
   a. Product Set: only input/output data and commentary annotations
   b. Service Test: only input, processing service and validation info

**Rationale:** from all use cases
Create a new Best Practices for representing Virtual Experiments as the collection of resources related to a scientific processing performed with an Exploitation Platform.

This shall

1. Represent and link all the resources related to a scientific processing:
   a. Processing services (including processing parameters)
   b. Input and output data sets
   c. Commentary annotations
   d. Qualification information (thirty party verified quality level)
   e. Validation information

2. Support DOI (via DataCite/DOI.org standard)

**Rationale:** from use case #4 (Collaboration)
Outline

- TEP use cases
- TEP architecture & interfaces (work-in-progress)
- OGC Testbed 13 & Open Source Framework
ESA is evaluating the possibility to sponsor a new OGC Testbed to push evolution to standards and interoperability.

TEP and other use cases and details on evolution of the standards will be given to the OGC for evaluation of the feasibility of a testbed.

Question: How do HMA members want to contribute to this?
Within the TEP, ESA is drafting a common OSS Core Components list which can be used as base to build an Exploitation Platform.

ESA intends to promote and coordinate **industrial OSS framework activities** to develop and evolve OSS implementations for the exploitation platforms core components.

---

**2.3 Execution Environment**

**2.3.1 Execution Cluster**

<table>
<thead>
<tr>
<th>OSS</th>
<th>Project reference</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>HT Condor</td>
<td><a href="https://research.cs.wisc.edu/htcondor/">https://research.cs.wisc.edu/htcondor/</a></td>
<td>Developed in C++/Perl, it is an high-throughput, software framework for management intensive tasks on a dedicated computer clusters</td>
</tr>
<tr>
<td>Hadoop</td>
<td><a href="http://hadoop.apache.org/">http://hadoop.apache.org/</a></td>
<td>Developed in Java, Hadoop is a distributed storage and processing tool for large data sets on computer clusters</td>
</tr>
<tr>
<td>Torque</td>
<td><a href="http://www.adaptivecomputing.com/products/torque/">http://www.adaptivecomputing.com/products/torque/</a></td>
<td>Torque is an open-source, distributed resource manager providing management of jobs and distributed compute nodes clusters</td>
</tr>
<tr>
<td>Jenkins</td>
<td><a href="https://github.com/jenkinsci/jenkins/">https://github.com/jenkinsci/jenkins/</a></td>
<td>Jenkins is an open-source automation server which is able to execute jobs on physical or cloud resources.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OSS</th>
<th>Project reference</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jenkins</td>
<td><a href="https://github.com/jenkinsci/jenkins/">https://github.com/jenkinsci/jenkins/</a></td>
<td>Developed in Java, Jenkins is an open-source automation server which is able to execute jobs on physical or cloud resources. It allows to</td>
</tr>
</tbody>
</table>

**Question:** How do HMA members want to contribute to this?