Master System Design Document

EOEPCA.SDD.001

TVUK System Team

Version Draft for 1.1, In Progress:

Master System Design

EO Exploitation Platform Common Architecture

Master System Design Document

EOEPCA.SDD.001

COMMENTS and ISSUES	**PDF**
If you would like to raise comments or issues on this document, please do so by raising an Issue at the following URL https://github.com/EOEPCA/master-system-design/issues.	This document is available in PDF format here.
EUROPEAN SPACE AGENCY CONTRACT REPORT The work described in this report was done under ESA contract. Responsibility for the contents resides in the author or organisation that prepared it.	**TELESPAZIO VEGA UK Ltd** 350 Capability Green, Luton, Bedfordshire, LU1 3LU, United Kingdom. Tel: +44 (0)1582 399000 www.telespazio-vega.com

AMENDMENT HISTORY

This document shall be amended by releasing a new edition of the document in its entirety. The Amendment Record Sheet below records the history and issue status of this document.

Table 1. Amendment Record Sheet

ISSUE	DATE	REASON
1.1	InProgress	Updates during development of Reference Implementation
1.0	02/08/2019	Issue for domain expert ITT
0.6	18/07/2019	Added XACML description + PDF template modifications
0.5	09/07/2019	Added Client Library + Billing
0.4	21/06/2019	Added Platform API
0.3	13/05/2019	Added content for Processing & Chaining and Resource Management
0.2	25/04/2019	Re-work IAM approach
0.1	24/04/2019	Initial in-progress draft

Chapter 1. Introduction

1.1. Purpose and Scope

This document presents the Master System Design for the Common Architecture.

1.2. Structure of the Document

Section 2 - Context

Provides the context for Exploitation Platforms within the ecosystem of EO analysis.

Section 3 - Design Overview

Provides an overview of the Common Architecture and the domain areas.

Section 4 - User Management

Describes the User Management domain area.

Section 5 - Processing and Chaining

Describes the Processing & Chaining domain area.

Section 6 - Resource Management

Describes the Resource Management domain area.

Section 7 - Platform API

Describes the Platform API, covering all domain areas.

Section 8 - Web Portal

Describes the Web Portal, covering all domain areas.

1.3. Reference Documents

The following is a list of Reference Documents with a direct bearing on the content of this document.

Reference	Document Details	Version
[EOEPCA-UC]	EOEPCA - Use Case Analysis EOEPCA.TN.005 https://eoepca.github.io/use-case-analysis	Issue 1.0, 02/08/2019
[EP-FM]	Exploitation Platform - Functional Model, ESA-EOPSDP-TN-17-050	Issue 1.0, 30/11/2017
[TEP-OA]	Thematic Exploitation Platform Open Architecture, EMSS-EOPS-TN-17-002	Issue 1, 12/12/2017

Reference	Document Details	Version
[WPS-T]	OGC Testbed-14: WPS-T Engineering Report, OGC 18-036r1, http://docs.opengeospatial.org/per/18-036r1.html	18-036r1, 07/02/2019
[WPS-REST-JSON]	OGC WPS 2.0 REST/JSON Binding Extension, Draft, OGC 18-062, https://raw.githubusercontent.com/opengeospatial/wps-rest-binding/develop/docs/18-062.pdf	1.0-draft
[CWL]	Common Workflow Language Specifications, https://www.commonwl.org/v1.0/	v1.0.2
[TB13-AP]	OGC Testbed-13, EP Application Package Engineering Report, OGC 17-023, http://docs.opengeospatial.org/per/17-023.html	17-023, 30/01/2018
[TB13-ADES]	OGC Testbed-13, Application Deployment and Execution Service Engineering Report, OGC 17-024, http://docs.opengeospatial.org/per/17-024.html	17-024, 11/01/2018
[TB14-AP]	OGC Testbed-14, Application Package Engineering Report, OGC 18-049r1, http://docs.opengeospatial.org/per/18-049r1.html	18-049r1, 07/02/2019
[TB14-ADES]	OGC Testbed-14, ADES & EMS Results and Best Practices Engineering Report, OGC 18-050r1, http://docs.opengeospatial.org/per/18-050r1.html	18-050r1, 08/02/2019
[OS-GEO-TIME]	OpenSearch GEO: OpenSearch Geo and Time Extensions, OGC 10-032r8, http://www.opengeospatial.org/standards/opensearchgeo	10-032r8, 14/04/2014
[OS-EO]	OpenSearch EO: OGC OpenSearch Extension for Earth Observation, OGC 13-026r9, http://docs.opengeospatial.org/is/13-026r8/13-026r8.html	13-026r9, 16/12/2016
[GEOJSON-LD]	OGC EO Dataset Metadata GeoJSON(-LD) Encoding Standard, OGC 17-003r1/17-084	17-003r1/17-084

Reference	Document Details	Version
[GEOJSON-LD-RESP]	OGC OpenSearch-EO GeoJSON(-LD) Response Encoding Standard, OGC 17-047	17-047
[PCI-DSS]	The Payment Card Industry Data Security Standard, https://www.pcisecuritystandards.org/document_library?category=pcidss&document=pci_dss	v3.2.1
[CEOS-OS-BP]	CEOS OpenSearch Best Practise, http://ceos.org/ourwork/workinggroups/wgiss/access/opensearch/	v1.2, 13/06/2017
[OIDC]	OpenID Connect Core 1.0, https://openid.net/specs/openid-connect-core-1_0.html	v1.0, 08/11/2014
[OGC-CSW]	OGC Catalogue Services 3.0 Specification - HTTP Protocol Binding (Catalogue Services for the Web), OGC 12-176r7, http://docs.opengeospatial.org/is/12-176r7/12-176r7.html	v3.0, 10/06/2016
[OGC-WMS]	OGC Web Map Server Implementation Specification, OGC 06-042, http://portal.opengeospatial.org/files/?artifact_id=14416	v1.3.0, 05/03/2006
[OGC-WMTS]	OGC Web Map Tile Service Implementation Standard, OGC 07-057r7, http://portal.opengeospatial.org/files/?artifact_id=35326	v1.0.0, 06/04/2010
[OGC-WFS]	OGC Web Feature Service 2.0 Interface Standard – With Corrigendum, OGC 09-025r2, http://docs.opengeospatial.org/is/09-025r2/09-025r2.html	v2.0.2, 10/07/2014
[OGC-WCS]	OGC Web Coverage Service (WCS) 2.1 Interface Standard - Core, OGC 17-089r1, http://docs.opengeospatial.org/is/17-089r1/17-089r1.html	v2.1, 16/08/2018
[OGC-WCPS]	Web Coverage Processing Service (WCPS) Language Interface Standard, OGC 08-068r2, http://portal.opengeospatial.org/files/?artifact_id=32319	v1.0.0, 25/03/2009

Reference	Document Details	Version
[AWS-S3]	Amazon Simple Storage Service REST API, https://docs.aws.amazon.com/AmazonS3/latest/API	API Version 2006-03-01

1.4. Terminology

The following terms are used in the Master System Design.

Term	Meaning
Admin	User with administrative capability on the EP
Algorithm	A self-contained set of operations to be performed, typically to achieve a desired data manipulation. The algorithm must be implemented (codified) for deployment and execution on the platform.
Analysis Result	The *Products* produced as output of an *Interactive Application* analysis session.
Analytics	A set of activities aimed to discover, interpret and communicate meaningful patters within the data. Analytics considered here are performed manually (or in a semi-automatic way) on-line with the aid of *Interactive Applications*.
Application Artefact	The 'software' component that provides the execution unit of the *Application Package*.
Application Deployment and Execution Service (ADES)	WPS-T (REST/JSON) service that incorporates the Docker execution engine, and is responsible for the execution of the processing service (as a WPS request) within the ‘target’ Exploitation Platform.
Application Descriptor	A file that provides the metadata part of the *Application Package*. Provides all the metadata required to accommodate the processor within the WPS service and make it available for execution.
Application Package	A platform independent and self-contained representation of a software item, providing executable, metadata and dependencies such that it can be deployed to and executed within an Exploitation Platform. Comprises the *Application Descriptor* and the *Application Artefact*.
Bulk Processing	Execution of a *Processing Service* on large amounts of data specified by AOI and TOI.
Code	The codification of an algorithm performed with a given programming language - compiled to Software or directly executed (interpretted) within the platform.
Compute Platform	The Platform on which execution occurs (this may differ from the Host or Home platform where federated processing is happening)

Term	Meaning
Consumer	User accessing existing services/products within the EP. Consumers may be scientific/research or commercial, and may or may not be experts of the domain
Data Access Library	An abstraction of the interface to the data layer of the resource tier. The library provides bindings for common languages (including python, Javascript) and presents a common object model to the code.
Development	The act of building new products/services/applications to be exposed within the platform and made available for users to conduct exploitation activities. Development may be performed inside or outside of the platform. If performed outside, an integration activity will be required to accommodate the developed service so that it is exposed within the platform.
Discovery	User finds products/services of interest to them based upon search criteria.
Execution	The act to start a *Processing Service* or an *Interactive Application*.
Execution Management Service (EMS)	The EMS is responsible for the orchestration of workflows, including the possibility of steps running on other (remote) platforms, and the on-demand deployment of processors to local/remote ADES as required.
Expert	User developing and integrating added-value to the EP (Scientific Researcher or Service Developer)
Exploitation Tier	The Exploitation Tier represents the end-users who exploit the services of the platform to perform analysis, or using high-level applications built-in on top of the platform's services
External Application	An application or script that is developed and executed outside of the Exploitation Platform, but is able to use the data/services of the EP via a programmatic interface (API).
Guest	An unregistered User or an unauthenticated Consumer with limited access to the EP's services
Home Platform	The Platform on which a User is based or from which an action was initiated by a User
Host Platform	The Platform through which a Resource has been published
Identity Provider (IdP)	The source for validating user identity in a federated identity system, (user authentication as a service).
Interactive Application	A stand-alone application provided within the exploitation platform for on-line hosted processing. Provides an interactive interface through which the user is able to conduct their analysis of the data, producing *Analysis Results* as output. Interactive Applications include at least the following types: console application, web application (rich browser interface), remote desktop to a hosted VM.

Term	Meaning
Interactive Console Application	A simple *Interactive Application* for analysis in which a console interface to a platform-hosted terminal is provided to the user. The console interface can be provided through the user's browser session or through a remote SSH connection.
Interactive Remote Desktop	An Interactive Application for analysis provided as a remote desktop session to an OS-session (or directly to a 'native' application) on the exploitation platform. The user will have access to a number of applications within the hosted OS. The remote desktop session is provided through the user's web browser.
Interactive Web Application	An Interactive Application for analysis provided as a rich user interface through the user's web browser.
Key-Value Pair	A key-value pair (KVP) is an abstract data type that includes a group of key identifiers and a set of associated values. Key-value pairs are frequently used in lookup tables, hash tables and configuration files.
Kubernetes (K8s)	Container orchestration system for automating application deployment, scaling and management.
Login Service	An encapsulation of Authenticated Login provision within the Exploitation Platform context. The Login Service is an OpenID Connect Provider that is used purely for authentication. It acts as a Relying Party in flows with external IdPs to obtain access to the user's identity.
Network of EO Resources	The coordinated collection of European EO resources (platforms, data sources, etc.).
Object Store	A computer data storage architecture that manages data as objects. Each object typically includes the data itself, a variable amount of metadata, and a globally unique identifier.
On-demand Processing Service	A *Processing Service* whose execution is initiated directly by the user on an ad-hoc basis.
Platform (EP)	An on-line collection of products, services and tools for exploitation of EO data
Platform Tier	The Platform Tier represents the Exploitation Platform and the services it offers to end-users
Processing	A set of pre-defined activities that interact to achieve a result. For the exploitation platform, comprises on-line processing to derive data products from input data, conducted by a hosted processing service execution.
Processing Result	The *Products* produced as output of a *Processing Service* execution.
Processing Service	A non-interactive data processing that has a well-defined set of input data types, input parameterisation, producing *Processing Results* with a well-defined output data type.

Term	Meaning
Products	EO data (commercial and non-commercial) and Value-added products and made available through the EP. *It is assumed that the Hosting Environment for the EP makes available an existing supply of EO Data*
Resource	A entity, such as a Product, Processing Service or Interactive Application, which is of interest to a user, is indexed in a catalogue and can be returned as a single meaningful search result
Resource Tier	The Resource Tier represents the hosting infrastructure and provides the EO data, storage and compute upon which the exploitation platform is deployed
Reusable Research Object	An encapsulation of some research/analysis that describes all aspects required to reproduce the analysis, including data used, processing performed etc.
Scientific Researcher	Expert user with the objective to perform scientific research. Having minimal IT knowledge with no desire to acquire it, they want the effort for the translation of their algorithm into a service/product to be minimised by the platform.
Service Developer	Expert user with the objective to provide a performing, stable and reliable service/product. Having deeper IT knowledge or a willingness to acquire it, they require deeper access to the platform IT functionalities for optimisation of their algorithm.
Software	The compilation of code into a binary program to be executed within the platform on-line computing environment.
Systematic Processing Service	A *Processing Service* whose execution is initiated automatically (on behalf of a user), either according to a schedule (routine) or triggered by an event (e.g. arrival of new data).
Terms & Conditions (T&Cs)	The obligations that the user agrees to abide by in regard of usage of products/services of the platform. T&Cs are set by the provider of each product/service.
Transactional Web Processing Service (WPS-T)	Transactional extension to WPS that allows adhoc deployment / undeployment of user-provided processors.
User	An individual using the EP, of any type (Admin/Consumer/Expert/Guest)
Value-added products	Products generated from processing services of the EP (or external processing) and made available through the EP. This includes products uploaded to the EP by users and published for collaborative consumption
Visualisation	To obtain a visual representation of any data/products held within the platform - presented to the user within their web browser session.
Web Coverage Service (WCS)	OGC standard that provides an open specification for sharing raster datasets on the web.

Term	Meaning
Web Coverage Processing Service (WCPS)	OGC standard that defines a protocol-independent language for the extraction, processing, and analysis of multi-dimentional coverages representing sensor, image, or statistics data.
Web Feature Service (WFS)	OGC standard that makes geographic feature data (vector geospatial datasets) available on the web.
Web Map Service (WMS)	OGC standard that provides a simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases.
Web Map Tile Service (WMTS)	OGC standard that provides a simple HTTP interface for requesting map tiles of spatially referenced data using the images with predefined content, extent, and resolution.
Web Processing Services (WPS)	OGC standard that defines how a client can request the execution of a process, and how the output from the process is handled.
Workspace	A user-scoped 'container' in the EP, in which each user maintains their own links to resources (products and services) that have been collected by a user during their usage of the EP. The workspace acts as the hub for a user’s exploitation activities within the EP

1.5. Glossary

The following acronyms and abbreviations have been used in this report.

Term	Definition
AAI	Authentication & Authorization Infrastructure
ABAC	Attribute Based Access Control
ADES	Application Deployment and Execution Service
ALFA	Abbreviated Language For Authorization
AOI	Area of Interest
API	Application Programming Interface
CMS	Content Management System
CWL	Common Workflow Language
DAL	Data Access Library
EMS	Execution Management Service
EO	Earth Observation
EP	Exploitation Platform
FUSE	Filesystem in Userspace
GeoXACML	Geo-specific extension to the XACML Policy Language
IAM	Identity and Access Management

Term	Definition
IdP	Identity Provider
JSON	JavaScript Object Notation
K8s	Kubernetes
KVP	Key-value Pair
M2M	Machine-to-machine
OGC	Open Geospatial Consortium
PDE	Processor Development Environment
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RBAC	Role Based Access Control
REST	Representational State Transfer
SSH	Secure Shell
TOI	Time of Interest
UMA	User-Managed Access
VNC	Virtual Network Computing
WCS	Web Coverage Service
WCPS	Web Coverage Processing Service
WFS	Web Feature Service
WMS	Web Map Service
WMTS	Web Map Tile Service
WPS	Web Processing Service
WPS-T	Transactional Web Processing Service
XACML	eXtensible Access Control Markup Language

Chapter 2. Context

The Master System Design provides an EO Exploitation Platform architecture that meets the service needs of current and future systems, as defined by the use cases described in [EOEPCA-UC]. These use cases must be explored under 'real world' conditions by engagement with existing deployments, initiatives, user groups, stakeholders and sponsors within the user community and within overlapping communities, in order to gain a fully representative understanding of the functional requirements.

The system design takes into consideration existing precursor architectures (such as the Exploitation Platform Functional Model [EP-FM] and Thematic Exploitation Platform Open Architecture [TEP-OA]), including consideration of state-of-the-art technologies and approaches used by current related projects. The master system design describes functional blocks linked together by agreed standardised interfaces.

The importance of the OGC in these activities is recognised as a reference for the appropriate standards and in providing mechanisms to develop and evolve standards as required in the development of the architecture. In order to meet the design challenges we must apply the applicable existing OGC standards to the full set of federated use cases in order to expose deficiencies and identify needed evolution of the standards. Standards are equally important in all areas of the Exploitation Platform, including topics such as Authentication & Authorization Infrastructure (AAI), containerisation and provisioning of virtual cloud resources to ensure portability of compute between different providers of resource layer.

Data and metadata are fundamental considerations for the creation of an architecture in order to ensure full semantic interoperability between services. In this regard, data modelling and the consideration of data standards are critical activities.

The system design must go beyond the provision of a standalone EO Exploitation Platform, by intrinsically supporting federation of similar EO platforms at appropriate levels of the service stack. The Network of EO Resources seeks, ‘to unite the available - but scattered - European resources in a large federated and open environment’. In such a context, federation provides the potential to greatly enhance the utilization of data and services and provide as stimulus for research and commercial exploitation. From the end-user point of view, the federated system should present itself as a single consolidated environment in which all the federated resources are made available as an integrated system. Thus, the system design must specify federation-level interfaces that support this data and service-level interoperability in such a way that is seamless to the end users.

The goal is to create an Integrated Data Exploitation Environment. Users will apply their workflows close to the hosted data, supplemented by their own data. Processing outputs may be hosted as new products that can themselves contribute to the global catalogue. This paradigm can then be extended to encompass the federated set of Exploitation Platforms within the Network of EO Resources. The result is a Federated, Integrated Data Analysis Environment.

A Reference Implementation of the full architecture will be developed to prove the concepts and also to provide an off-the-shelf solution that can be instantiated by future projects to implement their EO Exploitation Platform, thus facilitating their ability to join the federated Network of EO Resources. **Thus, the Reference Implementation can be regarded as a set of re-usable platform**

services, in the context of a re-usable platform architecture.

Chapter 3. Design Overview

The overall system design has been considered by taking the ‘Exploitation Platform – Functional Model’ [EP-FM] as a starting point and then evolving these ideas in the context of existing interface standards (with some emphasis on the OGC protocol suite) and the need for federated services.

3.1. Domain Areas

The system architecture is designed to meet the use cases as defined in [EOEPCA-UC] and [EP-FM]. [EOEPCA-UC] makes a high-level analysis of the use-cases to identify the main system functionalities organised into domain areas: 'User Management', 'Processing & Chaining' and 'Resource Management'. The high-level functionalities are often met by more than one domain area, and User Management (specifically Identity & Access Management) cuts across all use cases, and forms the basis of all access control restrictions that are applied to platform services and data.

Figure 1 depicts the domain areas as top level component blocks in a Platform ‘A’. The arrows may be read as “uses”, each implying one or more service interfaces.

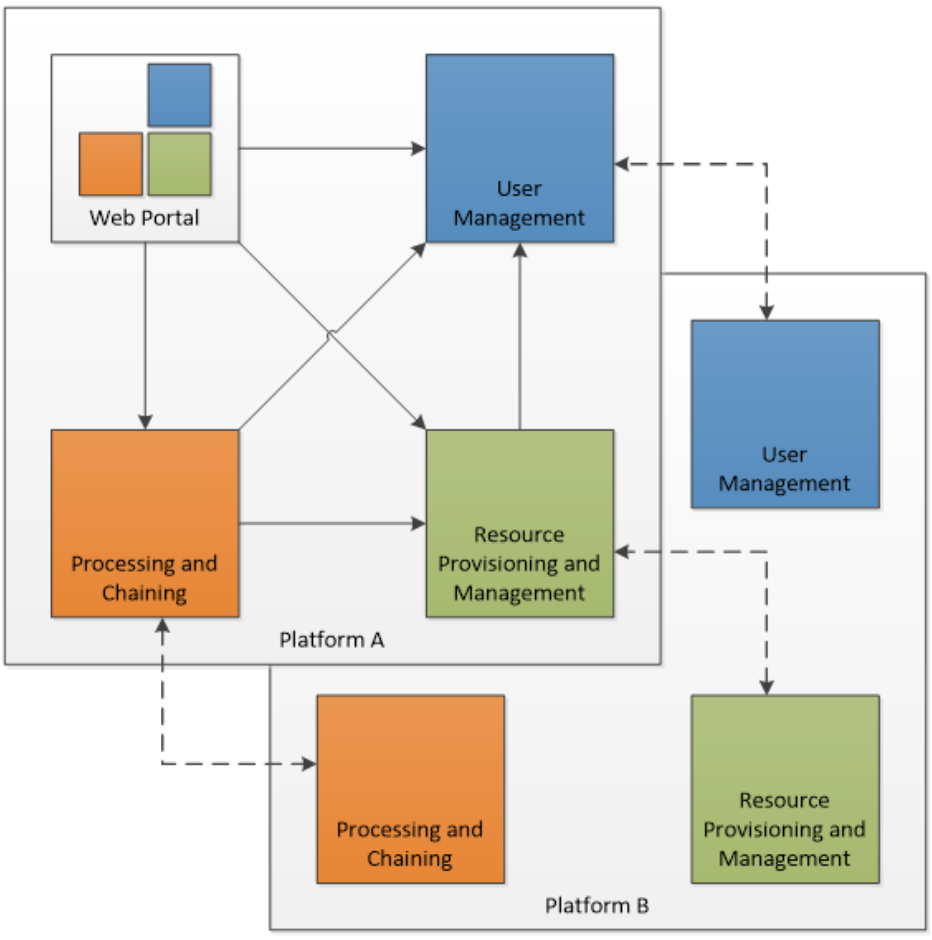

Figure 1. Top-level Architecture

A potential federation concept is represented by interactions between corresponding blocks in a collaborating Platform ‘B’. The architecture aims to minimise dependencies and is conducive to the principle of subcontracting the implementation to experts in the respective domains. The web portal integrates various client components to form a rich user-facing interface. **The Web Portal is depicted as it has interfaces with the other domain areas - but it is not a priority concern for**

the Common Architecture. Each exploitation platform would be expected to develop its own web interfaces according to its needs.

3.1.1. User Management

Responsible for all aspects related to user identity, authentication, authorization, accounting and billing in a federated system-of-systems environment.

It provides authentication, authorization and accounting services that are required to access other elements of the architecture, including the processing-oriented services and resource-oriented services. Individual resources may have an associated access and/or charging policy, but these have to be checked against the identity of the user. Resource consumption may also be controlled e.g. by credits and/or quotas associated with the user account. In the Network of EO Resources, a user should not need to create an account on multiple platforms. Therefore some interactions will be required between the User Management functions, whether directly or in directly via trusted third party.

3.1.2. Processing and Chaining

Provides access to a variety of processing functions, tools and applications, as well as execution environments in which to deploy them.

Provides a deployment and execution environment for processing tasks, analysis tools and interactive applications. Supports the chaining of processing tasks in workflows whose execution can include steps executed external to the origin exploitation platform. Handles and abstracts the low-level complexities of the different underlying compute technologies, and ensures the compute layer is scaled in accordance with current demand. Provides an integrated development environment to facilitate development of new processing algorithms and applications. Facilitating the network of EO resources by providing a federated interface to other processing services within the wider EO network.

The development and analysis environment provides a platform for the expert user to develop their own processing chains, experiments and workflows. It integrates with platform catalogue services (for data, processing services and applications) for discovery of available published datasets and processing elements. Subject to appropriate controls and permissions, the user can publish their own processing services and results. Workflows can be executed within the context of the processing facility, with the possibility to execute steps ‘remotely’ in collaborating platforms, with the results being collected for the continuation of the workflow.

3.1.3. Resource Management

Responsible for maintaining an inventory of platform and federated resources, and providing services for data access and visualisation.

Storage and cataloguing of all persistent resources. First and foremost, this will contain multidimensional geo-spatial datasets. In addition it may include a variety of heterogeneous data and other resources, such as documentation, Docker images, processing workflows, etc. Handles and abstracts the low-level complexities of different underlying storage technologies and strategies. Facilitating the network of EO resources by providing a federated interface to other data services

within the wider EO network.

The catalogue holds corresponding metadata for every published resource item in the local platform storage, as well as entries for resources that are located on remote collaborating platforms. Catalogue search and data access is provided through a range of standard interfaces, which are used by the local Web Portal and Processing & Chaining elements and may be exposed externally as web services.

Access to services and resources is controlled according to an associated authorization policy as defined by the IAM approach. This component may interact with corresponding peer components on other platforms - for example to synchronise catalogue entries.

The user has a personal workspace in which to upload files, organise links to resources of interest (services/application/data), and receive/manage results output from processing executions. Shared workspaces for collaboration can be similarly provisioned. The ingestion of new data is controlled to ensure the quality of any published resource, including associated metadata, and to maintain the integrity of the catalogue.

3.1.4. Platform API

Defines standard interfaces at both service and programmatic levels.

The Service API and its associated Client Library together present a standard platform interface against which analysis and exploitation activities may be developed, and through which platform services can be federated. The Platform API encourages interoperation between platforms and provides a consistent and portable programming paradigm for expert users.

3.1.5. Web Portal

Presents the platform user interface for interacting with the local resources and processing facilities, as well as the wider network of EO resources.

The Web Portal provides the user interface (themed and branded according to the owning organisation) through which the user discovers the data/services available within the platform, and the analysis environment through which they can exploit these resources. It provides a rich, interactive web interface for discovering and working with all kinds of resources, including EO data, processing and documentation. It includes web service clients for smart search and data visualisations. It provides a workspace for developing and deploying processing algorithms, workflows, experiments and applications, and publishing results. It includes support and collaboration tools for the community.

Web Portal integrates together various web service clients that uses services provided by the specialist domains (Processing, Resource, User) on the local platform and collaborating platforms.

3.2. Architecture Layers

Figure 2 provides a simplified architectural view that illustrates the broad architecture layes of the Exploitation Platform, presented in the context of the infrastructure in which it is hosted and the end-users performing exploitation activities.

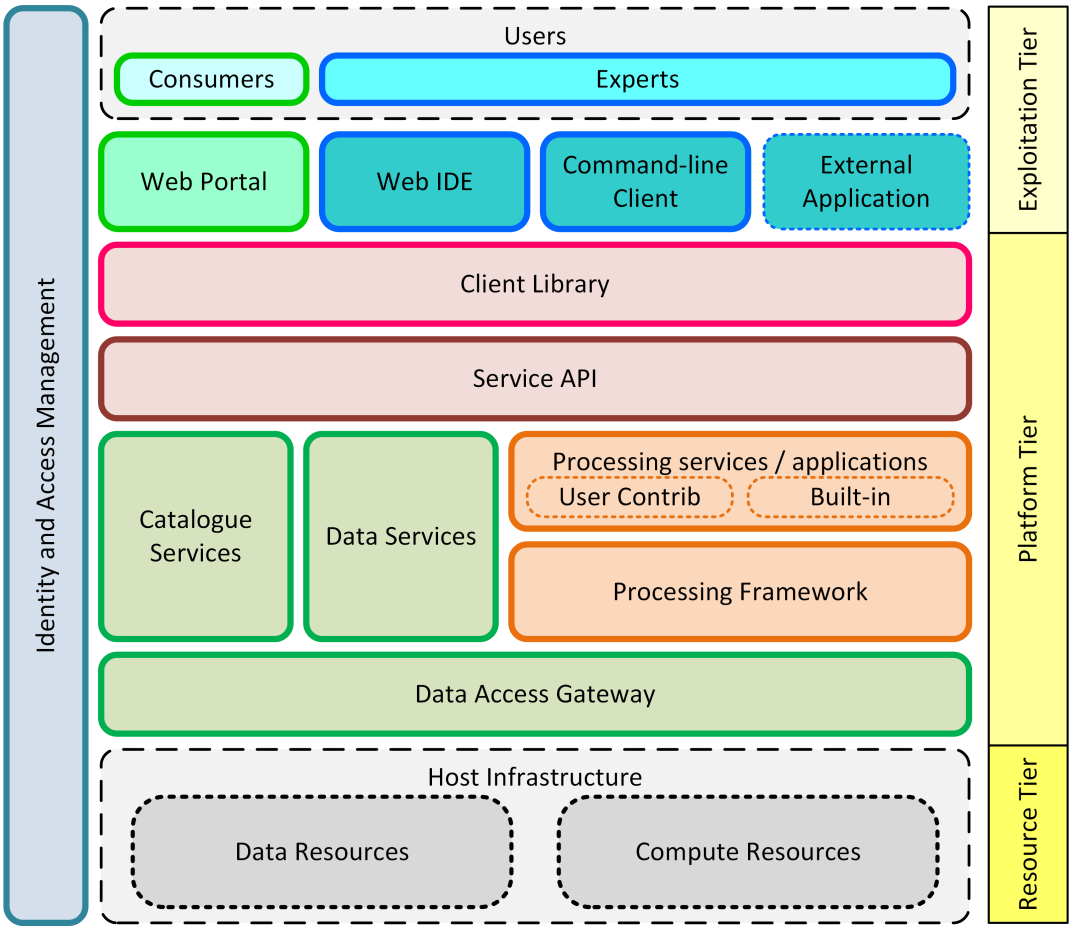

Figure 2. Architecture Layers

Resource Tier

The Resource Tier represents the hosting infrastructure and provides the EO data, storage and compute upon which the exploitation platform is deployed.

Platform Tier

The Platform Tier represents the Exploitation Platform and the services it offers to end-users. The layers comprising the Platform Tier are further described below.

Exploitation Tier

The Exploitation Tier represents the end-users who exploit the services of the platform to perform analysis, or using high-level applications built-in on top of the platform's services.

The Exploitation Platform builds upon the services provided by the hosting infrastructure - specifically accessing its data holding and using its compute resources. The components providing the EP services are deployed within the compute offering, with additional compute resources being provisioned on-demand to support end-user analysis activities.

The EP's services access the data resources through a Data Access Gateway that provides an

abstraction of the data access interface provided by the resource tier. This abstraction provides a 'standard' data access semantic that can be relied upon by other EP services - thus isolating specific data access concerns of the resource tier to a single EP component.

The Processing Framework provides the environment through which processing services and applications are executed in support of end-user analysis activities. It might be envisaged that some built-in (common) processing functions are provided, but the main focus of the processing framework is to support deployment and execution of bespoke end-user processing algorithms, and interactive analysis. Access to the underlying data from the executing processes is marshalled through the Data Access Gateway and its supporting Data Access Library.

The EP provides Catalogue services, so that end-users can discover and browse the resources available in the platform and its federated partners. Thus, end-users can discover available processing services and applications, and search for data available for inclusion in their analysis.

Data Services based upon open standards serve the clients of the Exploitation Platform for data access and data visualisation. Access to the underlying data is made via the Data Access Gateway.

The Service API represents the public service interfaces exposed by the Exploitation Platform for consumption by its clients. Covering all aspects of the EP (authentication, data/processing discovery, processing etc.), these interfaces are based upon open standards and are designed to offer a consistent EP service access semantic within the network of EO resources. Use of the network (HTTP) interfaces of the Service API is facilitated by the Client Library that provides bindings for common languages (Python, R, Javascript). The Client Library is a programmatic representation of the Service API which acts as an abstraction of the Exploitation Platform and so facilitates the development of portable client implementations.

The Exploitation Tier hosts the web clients with which the end-user interacts to conduct their analysis/exploitation activities. These clients would typically utilise the Client Library in their implementation. The Web IDE is an interactive web application that Experts use to perform interactive research and to develop algorithms. The Command-line Client builds upon the Client Library to provide a command-line tool that can be used, for example, to automate EP interactions through scripts.

The Web Portal provides the main user interface for the Exploitation Platform. It would be expected that each platform would provide its own bespoke portal implementation, and so is beyond the scope of the Common Architecture. Nevertheless, the architecture and its service interface must meet the needs anticipated by future exploitation platform implementations. Similarly, the External Application represents web applications (external to the hosting environment of the exploitation platform) that use the services of the EP via its Service API and Client Library.

All user interactions with the services of the EP are executed within the context of a given user and their rights to access resources, with associated resource usage billing. Thus, the Identity and Access Management component covers all tiers in this layered model.

The focus of this design document is the Platform Tier, which is elaborated in subsequent sections of the document:

Section User Management

This section adresses the main concerns of User Management which are user identity, access to resources and billing for resource usage.

Section Processing and Chaining

This section covers application packaging and the Processing Framework through which services/applications can be deployed in federated workflows.

Section Resource Management

This section covers resource discovery through catalogues that act as a marketplace for data, services and applications. Resource Management ensures data is accessible through standard interfaces that serve the processing framework, and public data services to visualise and consume platform data.

Section Platform API

This section provides a consolidated decription of the service interface of the EP and its associated client library, which together present a standard platform interface against which analysis and exploitation activities may be developed, and through which platform services can be federated.

Chapter 4. User Management

In the context of the Common Architecture, User Management covers the following main functional areas:

Identity and Access Management (IAM)

Identification/authentication of users and authorization of access to protected resources (data/services) within the EP.

Accounting and Billing

Maintaining an accounting record of all user accesses to data/compute/services/applications, supported by appropriate systems of credits and billing.

User Profile

Maintenance of details associated to the user that may be needed in support of access management and billing.

These are explored in the following sub-sections.

4.1. Identity and Access Management (IAM)

The solution for IAM is driven by the need for Federated Identity and Authorization in the context of a network of collaborating exploitation platforms and connected services. This federated environment should facilitate an end-user experience in which they can use a single identity across collaborating platforms **(Single Sign-On)**, they can bring their own existing identity to the platforms **('Login With' service)**, and platforms can access the federated services of other platforms on behalf of the end-user **(delegated access and authorization)**.

The goal of IAM is to uniquely identify the user and limit access to protected resources to those having suitable access rights. We assume an Attribute Based Access Control (ABAC) approach in which authorization decisions are made based upon access policies/rules that define attributes required by resources and possessed (as claims) by users. ABAC is is seen as a more flexible approach than Role Base Access Control (RBAC), affording the ability to express more sophisticated authorizations rules beyond the role(s) of the user - and noting the fact that a role-based ruleset could be implemented within an attribute based approach, (i.e. RBAC is a subset/specialisation of ABAC).

In achieving this there are three main concerns:

- Unique user identification
- Determine the access policy applicable to the resource
- From the access policy determine:
  - what attributes are required to access the protected resource
  - whether the user has the required attributes

For the Common Architecture, we establish separation of User Identification from Access Management. User identity is federated and handled external to the platform. Within the Network

of EO Resources, resources held within an exploitation platform are made available to federated partner platforms. Authorization policy is enforced within the platform at point of access, but the access policy can be federated within the network of EO resources, leading to a system of *federated authorization*.

- The identity is provided externally. The external IdP has no association to the exploitation platform, and hence is not the appropriate place to administer attributes that relate to EP resources
- The protected resources are under the custodianship of the exploitation platform and hence the exploitation platform enforces the access policy decision
- The administrative domain for an access policy should should not be tied to an exploitation platform, which facilitates the provision of federation and virtual organisations

Federation of services between exploitation platforms is an important goal of the Common Architecture. Thus, the IAM design must offer an approach through which user access is managed between platforms, ensuring proper enforcement of access controls and billing.

4.1.1. IAM Approach

Figure 3 presents the basic approach. At this stage it does not consider the case in which an exploitation platform accesses resources in another platform on behalf of a user, (for example a workflow step that is invoked on another platform). This is addressed in a later section. Users are authenticated by redirection to an external identity provider, (their 'home' IdP). This returns the authentication decision and some basic user information as required (such as name, email, etc.).

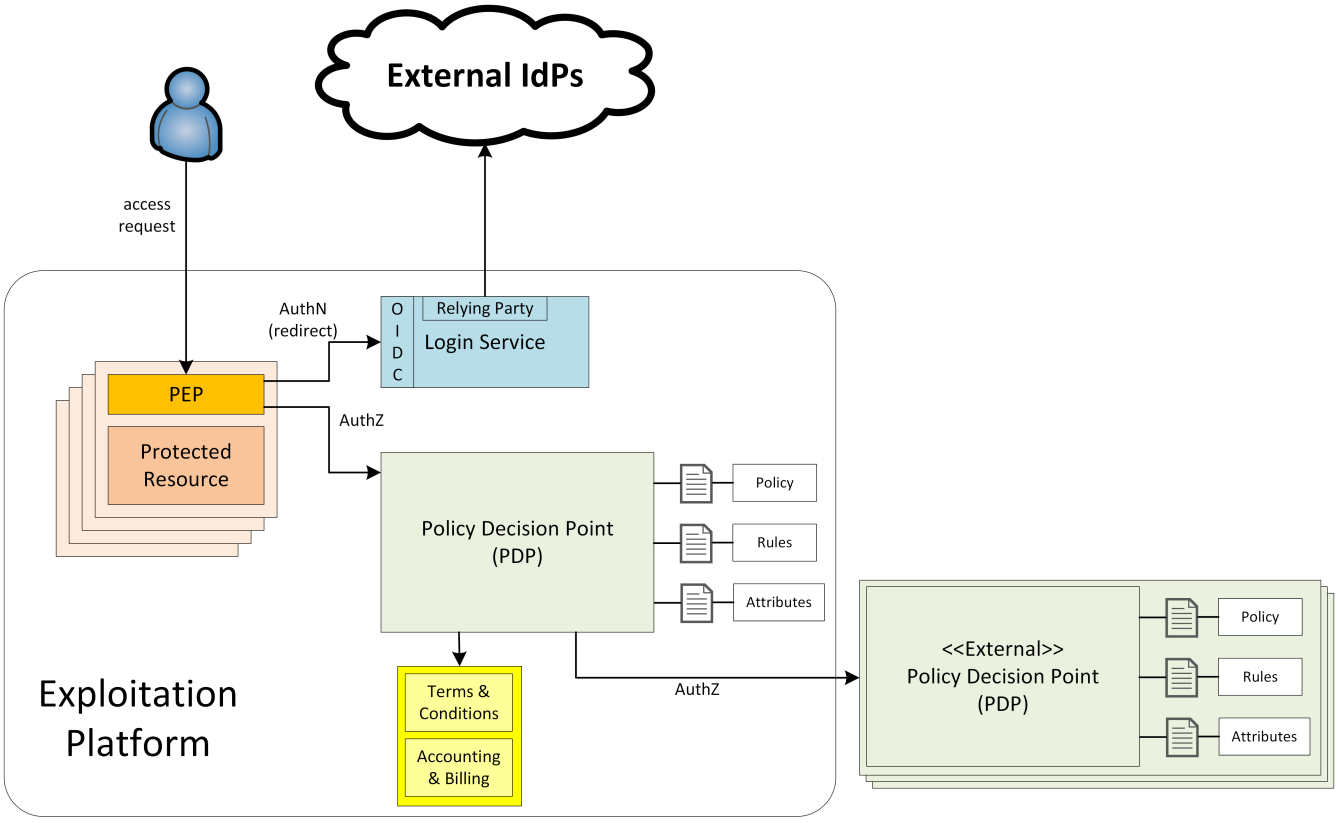

Figure 3. Identity and Access Management Overview

Each protected resource is fronted by its Policy Enforcement Point (PEP), which acts as filter that

will only permit access if the appropriate conditions are met. This decision is made according to a set of rules that are under the control of and configured within the exploitation platform.

The Login Service is provided as a common component that is utilised by each PEP to perform the authentication flow with the external IdPs. In the case of an unauthenticated request that requires authentication, the PEP will initiate the Login Service by redirection of the User's originating request. The successful flow ultimately redirects back to the PEP and so maintains the direct connection between the end-user agent and the resource server. An alternative approach would be the use of an API Gateway to perform the role of the PEP, acting as an intermediary between the end-user agent and the resource server. However, this would have the effect of proxying the connection which can have an impact on data transfer performance, which is of particular importance in the case of significant data volumes being returned to the User.

The PEP interrogates the PDP for an authorization decision. The PEP sends a request that indicates the pertinent details of the attempted access, including:

- Identity of end-user (subject)
- The API (path/version etc.) being accessed (resource)
- The operation (HTTP verb) being performed (action)

The Policy Decision Point (PDP) returns an authorization decision based upon details provided in the request, and the applicable authorization policy. The authorization policy may delegate all or part of the decision to external PDP(s) within the federated network. This represents a Federated Authorization model and facilitates a model of shared resources and virtual organisations.

The authorization policy defines a set of rules and how they should be evaluated to determine the policy decision. The rules are expressed through attributes. The policy is evaluated to determine what attributes are required, and what attributes the user possesses. This evaluation extends through external PDPs according to any federated authorization defined in the policy.

It should be additionally noted that the decision to allow the user access depends upon dynamic 'attributes', such as whether the user has enough credits to 'pay' for their usage, or whether they have accepted the necessary Terms & Conditions for a given dataset or service. Thus, the PDP must interrogate other EP-services such as 'Accounting & Billing' and 'User Profile' to answer such questions.

Figure 4 provides an overview of the IAM Flow, (success case).

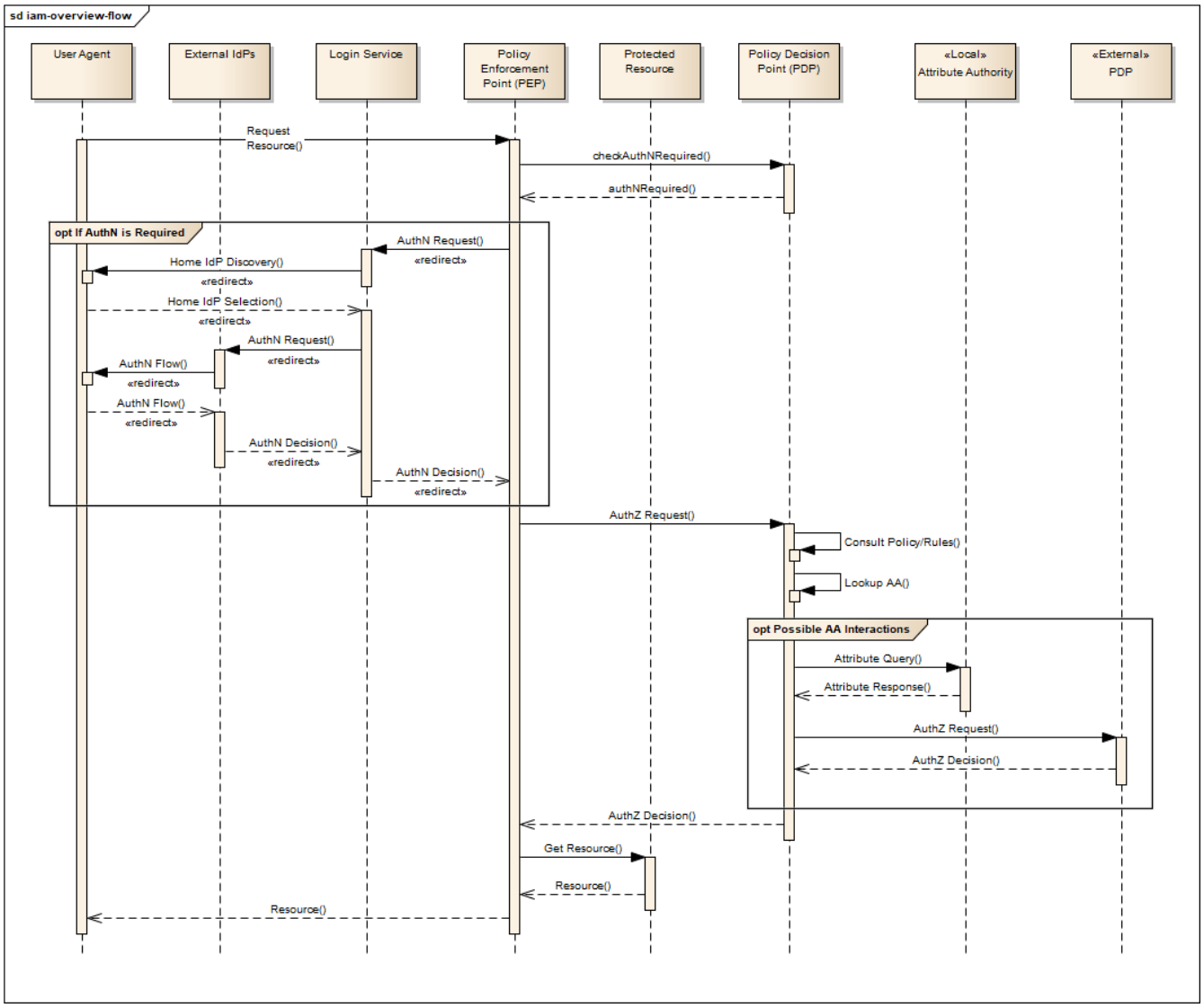

Figure 4. IAM Overview Flow

Flows marked <<redirect>> should be interpreted as flows between services that are made by redirection through the User Agent. For brevity, the interface between the Login Service, the User Agent and the External IdPs is simplified in Figure 4 - they are expanded in section Authenticated Identity. It should also be noted that the flows with the External IdP will vary according to the protocol required by the External IdP, (e.g. OAuth, SAML, etc.).

4.1.2. IAM Top-level Interfaces

Figure 5 illustrates the interfaces of the IAM architecture.

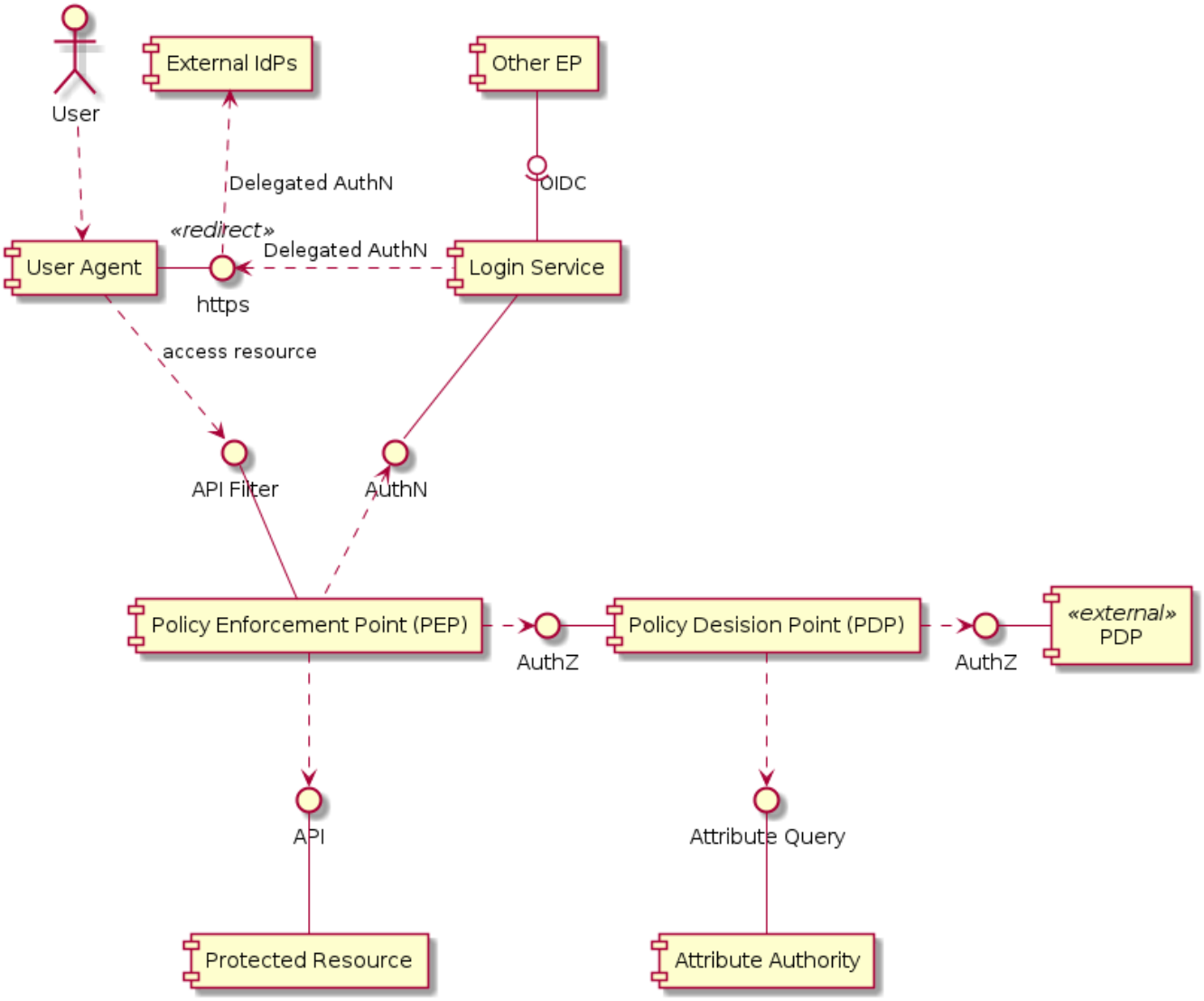

Figure 5. IAM Interfaces

User Agent → PEP

The PEP acts as a HTTP filter on the access request to the API of the Protected Resource. The PEP intercepts the incoming request in order to enforce the authorization policy decision.

PEP → Protected Resource

The Protected Resource exposes a public API for user consumption. The user's access to the protected resource is only granted once the request has passed the authorization decision, which may or may not require user authentication.

PEP → Login Service

The PEP uses a redirect to delegate the authentication flow to the Login Service.

Login Service → User Agent → External IdP

The Login Service and External IdP interface via redirects through the User Agent. In order to support multiple external identity suppliers, the Login Service must act as a client to multiple external IdPs, and so must establish individual trust relationships with each of these. Alternatively, the Login Service can instead interface to a single external IdP Proxy, that interfaces to the external IdPs on behalf of the EP. The IdP Proxy can provide this service to multiple EPs.

PEP → PDP

The PEP defers to the PDP to establish the authorization status of the incoming access attempt.
The request to the PDP carries the user identification, the URI of the resource, and the action requested.
The PDP-response returns the authorization decision.

PDP → Other PDP

This interface represents federated authorization.
It has the same interface characteristics as PEP→PDP.

4.2. Authenticated Identity

The approach to user identity and authentication centres around the use of OpenID Connect. Each Exploitation Platform maintains their own OIDC Provider through which tokens can be issued to permit access to protected resources within the EP. The authentication itself is delegated to external Identity Providers at the preference of the end-user wishing to reuse their existing identity provision.

4.2.1. Overview

The Login Service is an OpenID Connect Provider that provides a 'Login With' service that allows the platform to support multiple external identity providers. The Login Service acts as a Relying Party in its interactions with the external IdPs to establish the authenticated identity of the user through delegated authentication.

The Login Service presents an OIDC Provider interface to its clients, through which the OIDC clients can obtain Access Tokens to resources. The access tokens are presented by the clients in their requests to resource servers (intercepted by PEP). The PEP (acting on behalf of the resource server) relies upon the access token to establish the authenticated identity of the users making the requests. Once the user identity is established, then the PEP can continue with its policy decision (deferred to the PDP).

Thus, clients of the EP must act as OIDC Clients in order to authenticate their users to the platform, before invoking its services. Clients include the web applications that provide the UI of the exploitation platform, as well as other external applications/systems (including other exploitation platforms) wishing to use the services of the EP.

The Login Service must act as client (Relying Party) to each of the External IdPs to be supported and offered as a 'Login With' option. The interface/flow with the External IdP is integrated into the OIDC flow implemented by the Login Service. This includes prompting the user to discover their 'home' Identity provider. The interactions with the external IdP represents the 'user authentication step' within the OIDC flows. Completion of a successful authentication with the external IdP allows the Login Service to issue the requested access tokens (depending on the flow used).

Figure 6 illustrates the basic user access flow, invoked through a web browser.

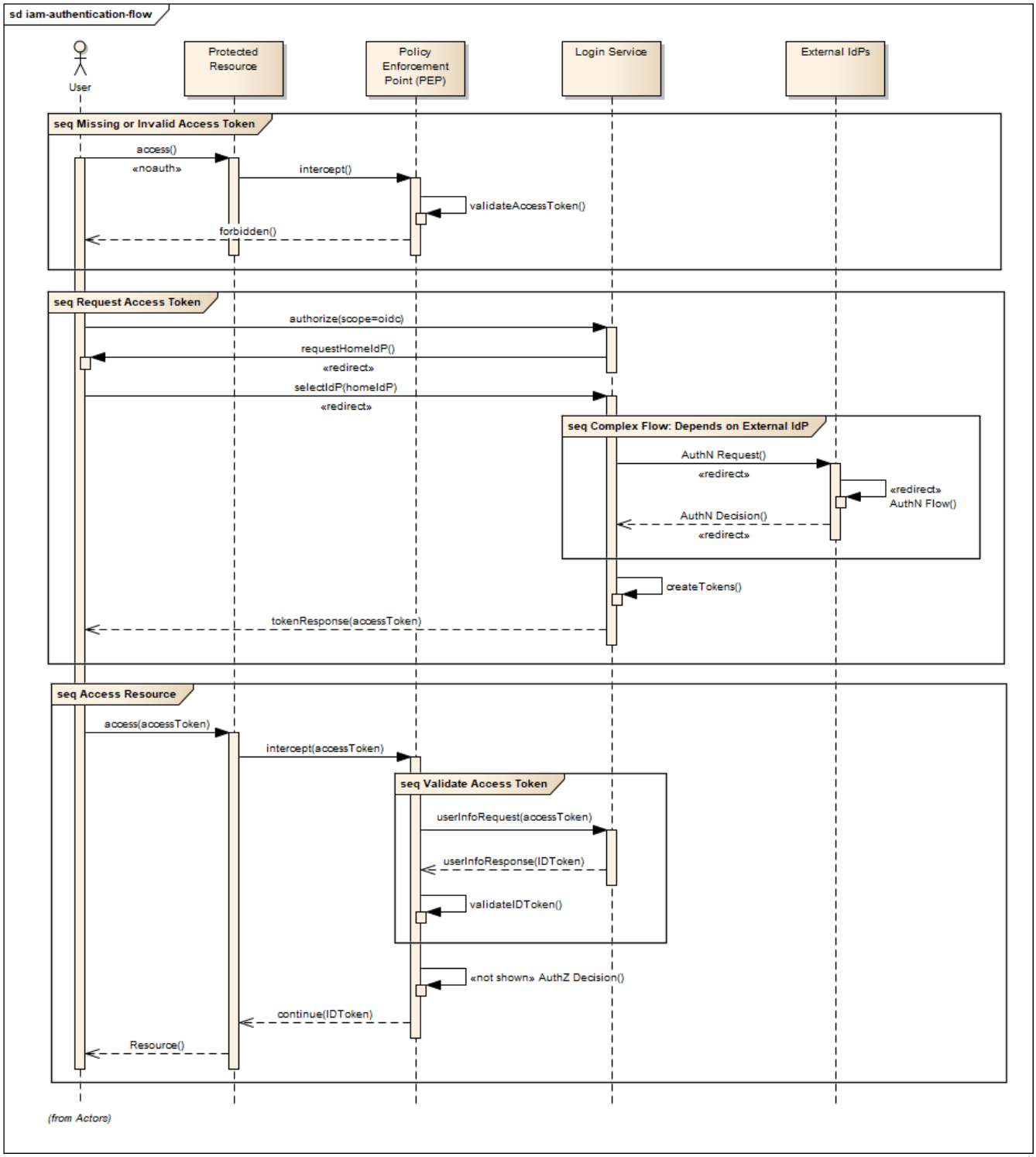

Figure 6. IAM Authentication Flow (Browser)

4.2.2. Login Service

The Login Service is an OIDC Provider that provides a 'Login With' service that allows the end-user to select their Identity Provider for purposes of authentication.

The Login Service is designed to support the onward forwarding of the authentication request through external identity services, which should be expected to include:

- EduGain
- GitHub

- Google
- Twitter
- Facebook
- LinkedIn
- Others (to be defined)

The Login Service must establish itself as a client (Relying Party) of all supported external IdPs, with appropriate trust relationships and support for their authentication flows.

The primary endpoints required to support the OIDC flows are as follows (these endpoints are taken, by example, from OKTA OIDC discovery metadata, https://micah.okta.com/oauth2/aus2yrcz7aMrmDAKZ1t7/.well-known/openid-configuration):

authorization_endpoint (/authorize)

To initiate the authentication, and to return the access tokens / code grant (depending on flow).

token_endpoint (/token)

To exchange the code grant for the access tokens.

userinfo_endpoint (/userinfo)

To obtain the user information ID token in accordance with the scopes requested in the authorization request.

jwks_uri (/keys)

To obtain signing keys for Token validation purposes.

end_session_endpoint (/logout)

To logout the user from the Login Service, i.e. clear session cookies etc. Although, given that the actual IdP is externalised from the Login Service, it would remain the case that any session cookies maintained by the external IdP would still be in place for a future authentication flow.

introspection_endpoint (/introspect)

Used by clients to verify access tokens.

revocation_endpoint (/revoke)

Used for (refresh) token revocation.

As described in section 'Discovery', the following endpoints relate to Discovery:

OIDC Discovery (/.well-known/openid-configuration)

Dynamic discovery of OIDC endpoints by clients.

As described in section 'Client Registration', the following endpoints relate to Dynamic Client Registration:

registration_endpoint (/clients)

Dynamic registration of clients (Authentication Agents).

As described in section ‘Federation’, the following endpoints relate to the establishment of a federation of collaborating Exploitation Platforms through a dynamic trust model:

/.well-known/openid-federation

OIDC Federation API endpoint through which Entity Statements are published about itself and other entities (such as other Exploitation Platforms). See section ‘Federation’.

4.2.3. OIDC ID Token

The ID Token is a JWT that is returned to from the /userinfo endpoint of the Login Service. The returned OIDC ID Token has been signed (JWS) by the Login Service and thus results in a token that asserts a user’s authenticated identity with integrity, and non-repudiation.

4.2.4. OIDC Clients

Clients are Relying Parties that act on behalf of users accessing the services of the Exploitation Platform. They will either pre-emptively obtain their access token for required resources, or will attempt resource access and be redirected by exception to the OIDC Provider authentication flow.

In the case of a web application (browser hosted), the Implicit Flow would be used. In other cases, where possible, the Authorization Code Flow would be preferred.

The OIDC flows are initiated with the appropriate response_type (‘id_token token’ for Implicit Flow, ‘code’ for Authorization Code Flow) and scope of ‘oidc profile’.

At the successful conclusion of the flow the client receives the Access Token and ID Token. The Access Token is then used by the client as a Bearer token in its subsequent calls to access the EP resources.

4.2.5. PEP (Resource Server filter)

The PEP (acting on behalf of the resource server) receives the client request to access the protected resource. In the case that the access requires an authenticated user, then the PEP expects that the request includes a valid access token.

Thus, the PEP follows the logic:

- The PEP checks with the PDP whether an authenticated user is required for access
- If no authenticated user is required then the request can continue (pending authorization) as an 'anonymous' user
- If access requires an authenticated user then
  - If the access token is not present then no user is logged in, so the request should be redirected to the /authorize endpoint (HTTP redirect)
  - If the access token is present, then it should be validated with the Login Service (direct call), as described below
  - If the access token validation completes successfully then the request can continue (pending authorization), with the user identity provided by the ID Token received during token

validation

- If the token is invalid, then the request should be redirected to the /authorize endpoint (HTTP redirect)

4.2.6. Access Token Validation

The PEP validates the access token by using it as a Bearer token in a request to the Login Service's /userinfo endpoint. A successful response has two outcomes:

- Confirms the validity of the access token from the point-of-view of the Login Service that issued it
- Provides an ID Token for the user that provides the information required to uniquely identify the user within the EP and utilise this identity within the subsequent policy decision made by the PDP

The ID Token is a JWT that has been signed by the Login Service. Using the jwks (see section 'OIDC Federation') endpoint of the Login Service, the PEP is able to obtain the necessary keys to validate the signature of the ID Token. This provides the full user context for the resource access.

4.2.7. Federated User Access

Based upon the above authentication model, an EP could access the resources of another EP by obtaining an access token through OIDC flows. However, considering that these EP→EP invocations will typically be Machine-to-machine (M2M), then we need to consider how the end-user (resource owner) is able to complete their consent. Two possibilities are explored in the subsequent sections:

1. The user pre-authorizes the EP→EP access in advance of the operation
2. Use of OIDC JWKS for trusted federation of identity between platforms

4.2.7.1. User Pre-authorization

Using the facilities of the Exploitation Platform, the user (perhaps via their User Profile management console) initiates the authorization flow from one EP to another. The end result is that the originating EP obtains delegated access to another EP on behalf of the user - with the resulting access tokens being maintained within the user's profile on the EP.

At the point where the EP needs to access a resource on another EP, then the access tokens are obtained from the user's profile and used as Bearer token in the resource request to the other EP. Refresh tokens can be used to ensure that authorization is long-lived.

Conversly, the user's profile at a given EP should also provide the ability to manage any inward authrosations they have granted to other EPs, i.e. ability to revoke a previous authorization by invalidating the refresh token. This would invole interface with the Login Service.

4.2.7.2. Possible use of OIDC JWKS Federation

OIDC provides a distributed key-hierarchy that could be used to support federated user access between collaborating exploitation platforms. The concept is explored in this section.

Reference: https://openid.net/specs/openid-connect-federation-1_0.html

OIDC provides a framework in which RPs and OPs can dynamically establish verifiable trust chains, and so share keys to support signing and validation of JWTs.

Dedicated 'federation' endpoints are defined that allow an entity (such as RP or OP) to publish their own Entity Statements, and to obtain Statements for other entities that are issued by trusted third-parties within the federation. The metadata/signatures within the Entity Statements establish a chain of trust that can be followed to known (trusted) Trust Anchors, and so the Entity Statements and the included entity public keys can be trusted.

Thus, through this mechanism public keys can be shared to underpin the signing and validation of JWTs.

Within an EP, when a resource server is executing a user's request, it may need to invoke a resource in another EP with which it is collaborating. The resource access to the other EP must be made on behalf of the originating user.

The nominal solution is for the originating EP to act as an OIDC Client to interface with the Login Service of the other EP, and so obtain the access token required to access the other resource. In this case, we should consider the fact that the resource access may be asynchronous to the end-user request and is not made within the context of the end-user's user agent. Therefore, we should explore possibilities (flows) provided by OIDC/OAuth that support this type of access.

One possibility is to make use of the signed-JWT ID Token that can be carried through the calls into and across resource servers. Through the facilities provided by JSON Web Key Set (JWKS), ID Tokens can be verified and trusted by other platforms operating within the same JWKS key hierarchy.

Thus, using the trusted ID Token, it may be possible follow an OIDC/OAuth flow from one EP to another, in which the user is deemed to have a-priori authorized the third-party access. At this point it is only the user's identity that has been established, with the authorization decision subject to the rules of the PDP/PEP of the remote system. The identified user must have appropriate a-priori permissions (attributes) on the target resources to be granted access, (ref. 'Federated Attributes').

Thus, it is the ID of the user that has been passed machine-to-machine to facilitate the service federation. This effectively achieves cross-EP single sign-on, without relying upon the user agent of the end-user providing cookies to the other EP.

4.2.8. Additional OIDC Capabilities

OpenID Connect provides some additional functionalities that are of interest in the context of the Common Architecture.

4.2.8.1. OIDC Discovery

Reference: https://openid.net/specs/openid-connect-discovery-1_0.html

OpenID Connect makes provision for two types of discovery:

1. Discovery of the OpenID Provider Issuer based upon the user's identifier
2. Discovery of the OpenID Provider Configuration Information

In the case of our usage within the Exploitation Platform, type 1) is not application since the user's ID comes from their 'Home' organisation and is not (necessarily) tied to an OpenID Connect Provider. Instead the Login Service must implement a discovery 'flow' in which the user is able to select the provider of their identity, as one that is supported by the Login Service deployment.

Regarding discovery type 2), the Login Service exposes an OIDC Provider interface, and this should support retrieval of OIDC Provider Configuration Information. Thus, OIDC Clients can utilise the discovery interface of the Login Service to exploit its services.

This is of most interest in the case of access to federated resources in other EPs, where a resource server in one EP may be acting as an OIDC client of the Login Service in another EP – in which case auto-discovery might be more attractive.

4.2.8.2. Client Registration

Reference: https://openid.net/specs/openid-connect-registration-1_0.html

The possibility exists for the OIDC Client (Login Service) to perform auto-registration with the Login Service, using OIDC Client Registration. In doing so the OIDC client obtains its Client ID and Secret.

This may be of interest in a couple of cases:

- The case of access to federated resources in other EPs, where a resource server in one EP may be acting as an OIDC client of the Login Service in another EP – in which case auto-client-registration might be of interest.
- The case where a common Login Service is deployed outside of the context of a given Exploitation Platform, acting as an IdP Proxy. In this case, the local Login Service deployed in each EP would register as an OIDC Client of the IdP Proxy.

4.3. Authorization (Policy Decision)

The Authorization flow is relevant to the following interfaces:

- PEP <→ PDP
  The PEP, acting as a filter for the access attempt on the resource, defers its authorization decision to the PDP.
- PDP <→ Other PDP
  Aspects of the authorization policy can be delegated from one PDP to another, e.g. to govern resource access through an administrative domain represented by a *Virtual Organisation*.

The approach (data model and protocol) for these two interfaces can be aligned - but is not yet defined in the Common Architecture. At this stage the candidate technologies to be investigated are:

- XACML (eXtensible Access Control Markup Language)
- UMA (User-Managed Access)

XACML and UMA can possibly be used together in an overall solution in which **XACML provides the policy language** through which rules are configured and **UMA provides the 'protocol' flows**.

4.3.1. XACML (eXtensible Access Control Markup Language)

XACML defines an architecture in which the access decision is separate from the point of use, and is thus consistent with the high-level IAM approach described in this design document. The XACML architecture framework describes a Request-Response protocol, and a policy language in which access policies are defined as rules comprising attributes. XACML additionally describes the process through which policies are evaluated, for example through *combining algorithms* that mediate competing rules.

4.3.1.1. XACML 3.0

XACML 3.0 includes some additional aspects that are of interest to our proposed IAM approach. Namely:

Administrative delegation

The delegation mechanism is used to support decentralized administration of access policies. It allows an authority (delegator) to delegate all or parts of its own authority, (or someone else's authority), to another user (delegate) without any need to involve modification of the root policy.

JSON Request/Response Profile

JSON bindings for the request/response messages between the PEP and the PDP - as an alternative to the core XML message definitions.

REST Profile

Providing REST semantics for the PEP/PDP interface.

4.3.1.2. GeoXACML

GeoXACML is standardised by the OGC to provide a geo-specific extension to XACML 2.0. GeoXACML provides support for spatial data types and spatial authorization decision functions. Those data types and functions can be used to define additional spatial constraints for XACML based policies.

4.3.1.3. ALFA (Abbreviated Language For Authorization)

Authoring of XACML rules can be facilitated by the use of **ALFA (Abbreviated Language For Authorization)**, which is a pseudocode language used in the formulation of access-control policies. ALFA policies can be directly converted to XACML 3.0 policies. ALFA can support GeoXACML through extensions, that supports definition of custom geometry functions.

4.3.1.4. XACML Related Technologies

The following items represent some interesting existing implementations that may be considered as a possible basis for a reference implementation of the Common Architecture.

AuthzForce PDP

An authorization service providing authorization policy decision evaluation and policy

administration.
Provides an API to get authorization decisions based on authorization policies, and authorization requests from PEPs. The API follows the REST architecture style, and complies with XACML v3.0.

- Project page: https://fimac.m-iti.org/6d.php
- Documentation: https://authzforce-ce-fiware.readthedocs.io/en/latest/

geoPDP

GeoXACML for AuthzForce PDP - extends the AuthZForce PDP implementation with a Geometry data type and related functions as specified in the OGC Implementation Standard GeoXACML 1.0.1.

- GitHub page: https://github.com/securedimensions/authzforce-geoxacml-basic
- geoPDP Docker: https://github.com/securedimensions/geopdp-docker

geoPEP

Related to the geoPDP is the geoPEP that delivers a PEP as an Apache2 reverse proxy - implemented as an Apache2 module - interfacing with the geoPDP for the authorization decisions.

- geoPEP Docker: https://github.com/securedimensions/geopep-apache2-reverse-proxy

4.3.2. UMA (User-Managed Access)

User-Managed Access (UMA) is a profile of OAuth 2.0. UMA defines how resource owners can control protected-resource access by clients operated by arbitrary requesting parties, where the resources reside on any number of resource servers, and where a centralized authorization server governs access based on resource owner policies. Resource owners configure authorization servers with access policies that serve as asynchronous authorization grants.

4.3.2.1. UMA Main Actors

The main UMA actors are briefly summarised as follows:

Resource Owner (RO)

The "user" and the owner of the resource who defines the access authorization - typically the end-user, but it can also be a corporation or other legal person.

Client

An application making protected resource requests with the Resource Owner's authorization and on the Requesting Party's behalf - typically a web or native application.

Resource Server (RS)

The custodian of the resource - typically a service through which access to the resource is gained.

Authorization Server (AS)

A server that issues *Authorization Data* and RPTs to a client and protects resources managed at a resource server. Authorization Data is associated with an RPT to enable some combination of the Authorization Server and Resource Server to determine the correct extent of access to allow to a Client.

Requesting Party (RqP)

The entity that uses a Client to seek access to a protected resource. May be the Resource Owner, or can be some other third party entity requesting access to the resource.

The software components that fill the roles of UMA authorization servers, resource servers, and clients respectively are intended to work in an interoperable fashion when each is operated by an independent party (for example, different organizations).

> In a typical OAuth flow, a human resource owner (RO) operating a client application is redirected to an authorization server (AS) to log in and consent to the issuance of an access token so that the client application can gain access to the resource server (RS) on the RO's behalf in future, likely in a scoped (limited) fashion. The RS and AS are in all likelihood operating within the same security domain, and any communication between them is not standardized by the main OAuth specification.
>
> UMA adds three main concepts and corresponding structures and flows. First, it defines a standardized API at the AS, called the protection API, that the RS speaks to; this enables multiple RS's to communicate with one AS and vice versa, and because the API is itself secured with OAuth, allows for formal trust establishment between each pair. This also allows an AS to present an RO with a centralized user interface. Second, UMA defines a formal notion of a requesting party (RqP) that is autonomous from an RO, enabling party-to-party sharing and delegation of access authorization. An RO need not consent to token issuance at run time but can set policy at an AS, allowing an RqP to attempt access asynchronously. Third, UMA enables access attempts to result in successful issuance of tokens associated with authorization data based on a process of trust elevation in the RqP, for example, gathering identity claims or other claims from them.
>
> — Wikipedia, https://en.wikipedia.org/wiki/User-Managed_Access

4.3.2.2. UMA Phases

UMA describes three phases…

1. **Protect Resource**
   The Resource Owner, who manages online resources at the Resource Server, introduces it to the

Authorization Server so that the latter can begin protecting the resources. To accomplish this, the Authorization Server presents a ***Protection API*** to the resource server. This API is protected by OAuth (or an OAuth-based authentication protocol) and requires a ***Protection API Token (PAT)*** for access. Out of band, the Resource Owner configures the Authorization Server with policies associated with the resource sets that the resource registers for protection.

2. **Get Authorization**
   The Client approaches the Resource Server seeking access to an UMA-protected resource. In order to access it successfully, the Client must first use the Authorization Server's ***Authorization API*** to obtain authorization data and a ***Requesting Party token (RPT)*** on behalf of its Requesting Party, and the Requesting Party may need to undergo a process of trust elevation, for example, supplying identity claims. The API is protected by OAuth (or an OAuth-based authentication protocol) and requires an ***Authorization API Token (AAT)*** for access.

3. **Access Resource**
   The Client successfully presents to the Resource Server an RPT that has sufficient authorization data associated with it, gaining access to the desired resource.

4.3.2.3. UMA Terminology

To elaborate the descriptions of these phases, UMA employs a set of terminology that is summarised below...

TERMS

- **Policy**
  The configuration parameters of an authorization server that effect resource access management - typically defined is terms of "subjects", "verbs" and "objects". Policy configuration takes place between the resource owner and the authorization server.
- **Claim**
  A statement of the value or values of one or more identity attributes of a Requesting Party. A Requesting Party may need to provide claims to an Authorization Server in order to gain permission for access to a protected resource.
- **Resource Set**
  One or more protected resources that a resource server manages as a set - thus, a Resource Set is the "object" being protected.
- **Scope**
  A bounded extent of access that is possible to perform on a Resource Set - a scope is one of the potentially many "verbs" that can logically apply to a Resource Set ("object").
- **Permission**
  A scope of access over a particular Resource Set at a particular Resource Server that is being requested by, or granted to, a Requesting Party. Thus, a Permission is an entitlement that includes a "subject" (Requesting Party), "verbs" (one or more Scopes of access), and an "object" (Resource Set).
- **Permission Ticket**
  A correlation handle that is conveyed from an Authorization Server to a Resource Server, from a Resource Server to a Client, and ultimately from a Client back to an Authorization Server, to enable the Authorization Server to assess the correct policies to apply to a request

for Authorization Data.

- **Authorization Data**
  Data (e.g. a Permission) associated with an RPT that enables some combination of the Authorization Server and Resource Server to determine the correct extent of access to allow to a Client.

TOKENS

- **Requesting Party Token (RPT)**
  An UMA access token associated with a set of Authorization Data, used by the Client to gain access to protected resources at the Resource Server.
- **Authorization API Token (AAT)**
  An OAuth access token with the scope ***uma_authorization***, used by the Client at the Authorization API *'RPT Endpoint'*. An AAT binds a Requesting Party, a Client being used by that party, and an Authorization Server that protects resources this Client is seeking access to on this Requesting Party's behalf. The issuance of an AAT represents the approval of this Requesting Party for this Client to engage with this Authorization Server to supply claims, ask for authorization, and perform any other tasks needed for obtaining authorization for access to resources at all Resource Servers that use this Authorization Server.
- **Protection API Token (PAT)**
  An OAuth access token with the scope ***uma_protection***, used by the Resource Server at the Protection API, consisting of the *'Resource Set Registration'*, *'Permission Registration'*, and *'Token Introspection'* endpoints. A PAT binds a Resource Owner, a Resource Server the owner uses for resource management, and an Authorization Server the owner uses for protection of resources at this Resource Server. The issuance of a PAT represents the approval of the Resource Owner for this Resource Server to use this Authorization Server for protecting some or all of the resources belonging to this Resource Owner.

ENDPOINTS

- **Authorization API**
  Requires access token with *uma_authorization* scope (AAT). An entity that can acquire an access token with this scope is by definition a Client.
  - **RPT Endpoint**
    An endpoint at the Authorization Server that issues RPTs and Authorization Data to the Client.
- **Protection API**
  Requires access token with *uma_protection* scope (PAT). An entity that can acquire an access token with this scope is by definition a Resource Server.
  - **Resource Set Registration Endpoint**
    An endpoint at the Authorization Server that allows the Resource Server to register Resource Sets.
  - **Permission Registration Endpoint**
    An endpoint at the Authorization Server that allows the Resource Server to request Permission Tickets.
  - **Token Introspection Endpoint**
    An endpoint at the Authorization Server that allows the Resource Server to query the

status of an RPT and its associated Authorization Data.

4.3.2.4. Resource Protection

The resource owner, resource server, and authorization server perform the following high-level actions to put resources under protection:

1. The Authorization Server issues client credentials to the resource server, (either dynamically or statically)
2. The Resource Server acquires a PAT from the Authorization Server
3. The Resource Server registers any Resource Sets with the Authorization Server for which it intends to outsource protection, using the Resource Set Registration Endpoint of the Protection API

4.3.2.5. Access Resource

An Authorization Server orchestrates and controls Clients' access on their Requesting Parties' behalf to a Resource Owner's protected resources at a Resource Server, under conditions specified by that Resource Owner through policy.

The process of getting authorization and accessing a resource always begins with the Client attempting access at a protected resource endpoint at the Resource Server.

If the Client's attempt has a valid RPT with sufficient Authorization Data, the Resource Owner's policies have been met for access to the protected resource, and hence access is granted. See Figure 7.

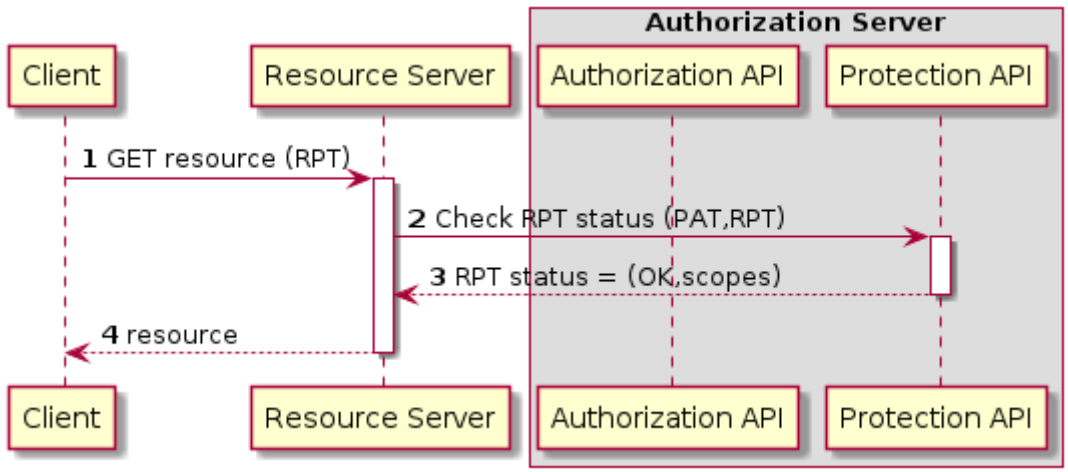

Figure 7. UMA Access Resource (Success)

4.3.2.6. Resource Server Register Permission

If the Client's request at the protected resource has no RPT, or has an invalid RPT or insufficient Authorization Data associated with the RPT as determined through RPT status checking, the Resource Server registers a requested permission with the relevant Authorization Server, and responds to the Client with the resulting Permission Ticket and the Authorization Server's location. The extent of the requested permission MUST suffice for the extent of the Client's access attempt at that resource. The PAT provided in the API request enables the Authorization Server to map the

requested permission to the appropriate Resource Owner. The Authorization Server returns a Permission Ticket in its response for the Resource Server to give to the Client that represents the same extent of requested access that the Resource Server registered. See Figure 8.

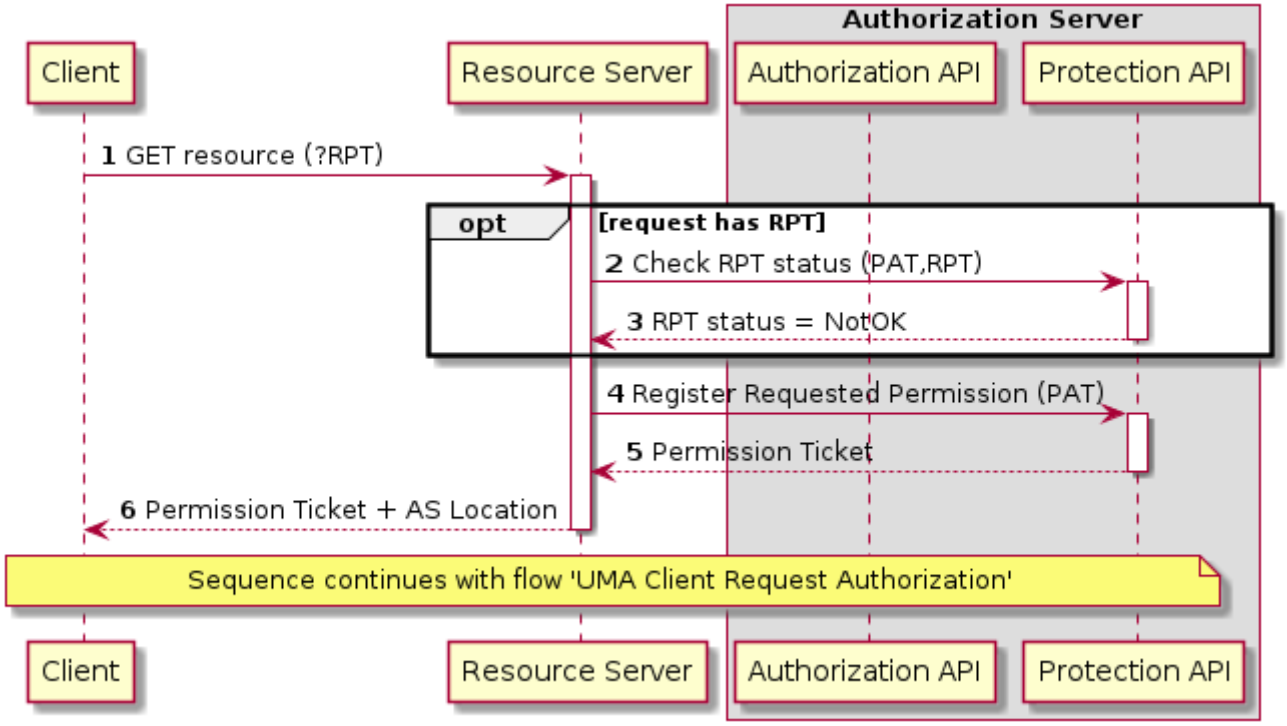

Figure 8. UMA Resource Server Register Permission

4.3.2.7. Client Request Authorization

In order to access a protected resource successfully, a client needs to present a valid RPT with sufficient authorization data. The client uses the authorization API to acquire an RPT and to ask for authorization data, providing the permission ticket it received from the resource server.

The authorization server uses the permission ticket to look up the details of the previously registered requested permission, maps the requested permission to operative resource owner policies based on the resource set identifier and scopes associated with it, potentially requests additional information and receives additional information such as claims, and ultimately responds positively or negatively to the request for authorization data. **The authorization server bases the issuance of authorization data on resource owner policies. Thus, these policies function as authorization that has been granted ahead of time.** See Figure 9.

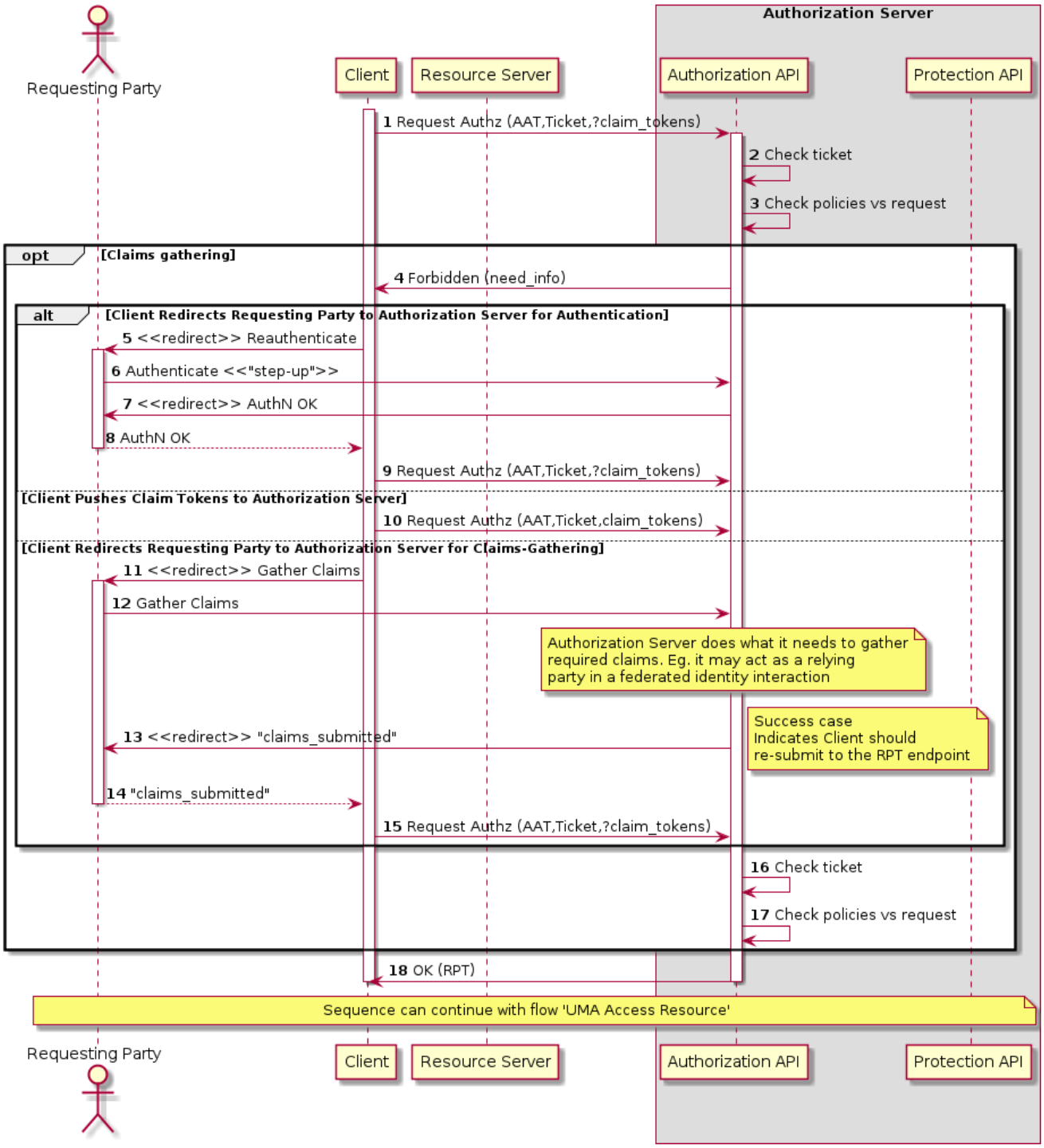

Figure 9. UMA Client Request Authorization

Optionally, the Authorization Server may need additional information in order to determine whether the Client is authorized to have this authorization data. The Authorization Server response indicates "need_info" including one or more sub-properties with hints about the nature of further required information. The Client then has the opportunity to continue seeking authorization by engaging in follow-on flows with the authorization server, either directly or through redirection of an end-user Requesting Party.

4.4. Accounting and Billing

The platform must account for resource use both within the platform and in other platforms via federation. In addition, several inter-platform billing models are supported as defined in the use

cases, [EOEPCA-UC]. A number of principles must first be established:

- Actions are performed within the context of a 'billing identity', which may be different to the user's identity.
- Charges are the result of discrete 'billing events' occuring within a particular 'billing window'. Pricing must consider all events within the window, not events individually (to support, for example, tiered pricing).
- Different platforms may follow completely different pricing and billing models. The architecture and federation messaging cannot assume any particular method of calculation or for describing prices.
- Only the platform hosting it can accurately price the use of a licensed Resource or compute resource.
- Costs may be estimated but the estimate is not required to be binding. Federated access can never rely on binding estimates.
- Debts can only be created where there is a direct contractual relationship and opportunity for credit control. A user can never owe money directly to another platform unless he has an account with it.
- A platform prices in a single currency (but could choose to allow a user to settle a bill with another currency). Different federated platforms may choose different currencies.

4.4.1. Billing Identities

A billing identity is a user identity for a user who has established a billing relationship with the platform. A billing user may delegate chargeable service access to other users within the system, permitting that user to use resources billed to the billing identity.

Individual platforms may choose models with varying complexity. For example, one platform may require that the billing and user identity are always the same, whilst another may permit a user working on multiple cross-organizational projects to choose the billing identity to use. Identities may be related to organizations, projects, etc, for access control and credit control purposes - but these relationships are not required by the architecture.

As required by their purpose, cross-platform messaging will include both the user id and the relevant billing identity.

Both the billing and user identities, and other information such as the location of each one and the type of organization involved, may be relevant to determining prices. This is because the place of supply for VAT purposes must be determined, plus any discounts for, for example, academic use. Note that 'location' means more than 'country' (eg, the Canary Islands have lower VAT than Madrid). Also, some organizations may be treated differently such as international organizations exempt from all tax.

4.4.2. Billing Service

A Billing Service will operate within the platform which receives reports of billing events from

other components. These are recorded against the relevant billing identity. Billing events have arbitrary attributes defined in them, which the billing service does not interpret (but are sufficient for the pricing engine, see below), a transaction ID identifying the original user action which caused it, and enough additional information for display to the user. Some example billing events might be:

- 1 hour of extra-large-vm
- 12 CPU-hours of container execution time
- licence for satellite image x
- execution cost of $x from federated platform y

Individual components decide when to generate billing events - for example, compute cost billing events may be generated every hour. Billing events may have a start and end or a single time - events with a start and end may be split to keep them within a single billing window.

The Billing Service can generate reports for the user. This may involve combining billing events in to line items, such as consolidating VM use in to the number of hours so far this month.

The Billing Service will generate bills for each billing window by pricing complete windows. Fixed prices are assigned and recorded at this point. It may also keep track of and, where supported by the platform, initiate payments.

To support the PDP and other services, the Billing Service may be required to periodically assess the account's standing and make decisions on the acceptibility of resource use. This depends on the billing model in use but could involve checking that credits are not exhausted, checking that a reasonable credit limit has not been reached and the detection of potentially fraudulent behaviour. An account which is no longer in good standing may result in API requests for resource use being denied, or it may result in termination messages being sent in response to billing events.

Where billing events are reported in another currency, as may happen with federated resource use, the Billing Service must determine the time and rate for currency exchange.

4.4.3. Pricing Engine

To maximize reusability price calculation is separated in to a different service (but not necessarily a different address space). Given access to a price database describing current, future and past configured prices, a Pricing Engine is otherwise stateless and can:

- Given a list of billing events or consolidated line items within a pricing window return calculated rates and prices for each one. For some pricing models this may involve multiple charges for each item or may contain blended rates/prices.
- Return estimated prices for estimated resource use.
- Where a platform wished to provide such a service to users, return price information and estimated prices in response to API requests.
- Given a commercial licence billing event, calculate the charge to the user, the credit to the Licenser and the platform fee.

4.4.4. Commercially Licensed Resources

Users may publish Resources which are licensed to others on commercial terms and use the platform to collect payments. There are two types of charges which require support within User Management: time-based and volume-based.

Time-based charges occur when a user requests a licence which costs a fixed price for a fixed time (or is permanent), regardless of the accesses made to the Resource. The Data Access Services and Execution Management Services determine when such a licence is required and the licence manager manages the process for buying one, including emitting a billing event. This typically will happen in advance of a request. The licence manager may give the billing service an opportunity to reject the request, if applicable to the platform's billing model.

Volume-based charges occur as access to a licensed Resource proceeds or completes (for example, on first access to a specific satellite image or for each input image passed to a commercial machine learning model). Again, the licence manager reports these as billing events when a licence requirements check is made.

Pricing is specified by the Licenser (in a particular form supported by the platform) and stored by the pricing engine (quantity to price mapping) and licence manager (method for determining which licences and 'quantity'). The licence manager must emit three billing events when license grants are bought: a charge to the user, a credit to the Licenser and a charge to the Licenser representing the platform fee for handling payment processing.

4.4.5. Budgets

TBD

4.4.6. Inter-platform Payments

Three different models for federated availability of commercial services are supported, two of which require support from the accounting and billing mechanisms of the platforms involved. This support comes in the form of inter-platform payments, allowing users to pay for executions or Resource licences which are located elsewhere in the federation.

Note that *three* platforms may be involved in providing a chargeable federated commercial service:

- The home platform where the user is registered and the action is initiated.
- The host platform where the licenced Resource or chargeable compute resource is located.
- The compute platform where processing occurs.

Consider, for example, a processing chain invoked on the home platform which invokes a processing service running on the compute platform using a software container published by a Licenser registered on the host platform. Frequently, two or more of these platforms are the same. However, even if all three are the same the platform may wish to use the same process where payments to a Licenser are involved.

4.4.6.1. Inter-platform Payment Model and Process

An inter-platform payment supports a User of one platform paying for a service provided by either another platform or by a User of another platform. It's important to repeat that a debt is only ever created between two entities which have a legal relationship and an opportunity for credit control. This requires that inter-platform payments involve two or three separate debts being: one from User to home platform, one from home platform to the host platform and the third from the host platform to the User providing the service (if any). The process must also cope with the price not being known in advance in all cases - processing costs in particular may be unpredictable. To support this, the following stages are involved:

- **Authorization stage**: This provides an opportunity for credit control decisions in advance of debts being incurred. This establishes a maximum amount of debt before a new authorization must be sought or the operation aborted but will not necessarily ever be owed in full. Both home and host platform must agree to authorize an inter-platform payment (the host platform may reject if it doesn't believe the home platform will pay). The home platform may 'hold' some account credit from its user or authorize a credit card payment if appropriate in its billing model.
- **Clearing stage**: This occurs after a debt is legally incurred, such as after (some of) the computation or data access is completed. The platform on which the service is provided, the host platform, reports to the home platform how much debt has actually been incurred. It may happen in stages - for example a large authorization may occur, followed by the clearing of smaller amounts after every hour of compute time. It cannot exceed the amount authorized.
- **Settlement stage**: This involves a batch of multiple payments, such as a day or a month of payments. The platforms with payment processing contracts in place must reconcile their records and calculate a net amount owed (potentially in multiple currencies). They must then settle the net debt by making a payment using the banking system.

Two different commercial models are supported: bilateral clearing and central clearing. In bilateral clearing every platform must negotiate a contract with every other platform (or as far as possible - incomplete coverage will limit what users can do). This has certain commercial downsides, such as a need for every-pair auditing for accurate reporting of resource use and a danger of incumbents excluding new entrants. In central clearing a clearing house must exist and all platforms form a relationship with the clearing house. The clearing house technical functionality is not further explored here, nor is the management of counterparty risk. The messaging and process is intended to be the same in both models.

Where inter-platform payments are used the host platform is acting a subcontractor to the home platform. Should the host platform fail to perform, a dispute resolution process must be used. This is considered out of scope of the architecture, except that payments may be marked as disputed, refunded or charged back. This must be accounted for during reconciliation between platforms.

4.4.7. Federated Commercial Services Without Inter-platform Payments: Direct Payments

If inter-platform payments are not available, for example because two platforms do not have a payment agreement, it may still be possible to provide services across multiple platforms providing the user has an account and billing relationship with each one directly. This requires that both

platforms recognize both the user and the selected billing identity, and that the billing user has delegated access to the user in both platforms.

To handle direct payments the user must authorize the home platform to act on its behalf when submitting requests to the host platform. This is done using OAuth. The home platform must redirect the user to the host platform which then returns an authorization token to the home platform. Federated platforms must run an OAuth endpoint for this purpose and certain restrictions must be put on its functioning (for example on refresh token lifetime).

Other system components must then use an access token when making requests to the host platform. The host platform should still report costs and identifiers to the home platform, which must be passed to the Billing Service to be recorded. This aids dispute resolution and the reporting of total costs for particular requests.

4.4.8. Estimating Inter-platform Costs

TBD

4.4.9. Relationship to System Components

The Billing Service handles inter-platform payments and supports direct payments in response to requests from other components, such as the EMS. The direct payment model is very different to inter-platform payments but knowledge of the distinction and when each should be used should be isolated in the Billing Service as much as possible.

To support this for volume-based charges, interaction between other system components and the Billing Service proceeds as follows:

- Prior to federated resource use, a component must make a request to the Billing Service with the estimated cost (or a fixed value if not available) and the identity of the host platform. It must also include the transaction ID for the user action which resulted in the payment.
- The Billing Service determines what kind of payment handling is available, if any. It returns success or failure and, optionally, an OAuth URL to authorize direct payment.
- The component proceeds with its activity, incurring charges. The activity occurs on the compute platform, which may also be the home or host platform.
- The compute platform seeks authorization from the host platform before charges are incurred. The host platform checks that an authorized payment exist (directly between the home and host platform). If the charge is for compute resources then these are the same platform and may be a no-op, but this may not be the case for computation using licensed data or software.
- If the compute platform seeks access from a host platform which has no authorized payment in place then it must report this to the home platform. The home platform may then request authorization or abort the processing. This may happen if the home platform cannot fully predict the accesses made during computation.
- The compute platform computes, incurring charges. The compute platform may also access the host platform to retrieve data or software but this may also be cached. The resource use is reported by the compute platform to the host platform - for example, a list of images accessed or processed. This happens in multiple chunks when charges are incurred over time.

- The host platform clears pieces of the original inter-platform authorization by sending a clearing request directly to the home platform. Note that only the host platform is considered authoritative for calculating the true cost (which is returned here).
- If the original authorization is exhausted then the home platform may pre-emptively extend it by creating a new payment (with the same transaction ID). Otherwise the host platform must reply to a charge report from the compute platform with a response prohibiting further charges.
- On receiving such a message the compute platform must suspend further processing and forward the response to the home platform. The home platform must then either seek a new authorization or send an abort message to the compute platform.

For time-based licences the flow can be simpler:

- The component requests payment authorization from the Billing Service, specifying an exact price.
- The component communicates with the host platform to acquire the licence.
- The host platform sends a payment clearing message to the home platform Billing Service to clear the entire authorization.

4.4.10. Payment Processing Systems

Payment processing itself, in particular card payment processing, may be initiated by the Billing Service but should be strictly separate from it. [PCI-DSS] imposes many onerous requirements not just on the software and hardware used for payment processing, but also on the wider organization and its processes (for example, for formal change reviews and code reviews, the use of specialist cryptographic hardware security modules, the separation of duties between staff and requirements in recruitment and training). For these reasons some implementers will need to avoid card processing within the system entirely and redirect users to externally hosted payment servers. This may constrain them to an account credit-based model whilst other providers may be able to initiate an authorization or full payment on-demand.

4.5. User Profile

The User Profile is a system resource that maintains a set of data for each user including:

- User details
- Terms and conditions accepted by the user
- Licence keys held by the user
- User API key management

The User Profile for a given user is tied to the unique identifier provided by their Home-IdP through the authentication process.

4.5.1. Licence and T&C Management

A licence manager must determine whether or not licence requirements permit certain actions by a

certain user. For freely available resources simple acceptance of the licence may be necessary. For commercially licensed resources it may be much more complicated. For example, a licence may have been bought for non-educational use by up to 5 users for satellite images with a certain resolution and area, with an extra charge made for images less than 15 degrees off nadir. Alternatively, a commercially licensed processing service may be charged by the CPU-hour or user-month. This is handled by the pricing and billing services, but acceptance of these terms must still be made first.

Some concepts applicable within the licence manager must be established:

- A licence consist of the legal text itself, a name and version, a description of pricing where appropiate and other metadata.
- A licence terms acceptance is the acceptance by a particular user (and organization) of the licence terms and conditions.
- A licence grant grants a users access to particular parts of a resource or for particular purposes. This is only applicable to commercial licensed resources. A licence grant is signed by the licenser.
- The licence manager does not know which resources require which licences. It only knows data about identified licences and about which users have which acceptances and grants.

The licence manager does not formally know how to calculate the price of a commercial licence grant. Instead, it produces an identifier for a particular type of grant and a quantity. The billing engine turns this in to a price, which may involve applying any user-specific or volume-based discounts. The quantity may be in, for example, square kilometres. Alternatively, the licences may be priced at €1/unit, effectively transferring responsibility to the licence manager's configuration.

Note that licencers must ensure that their licences are uniquely identified across the whole federation. That is, if they use the same licence on multiple platforms they must give it the same ID and must not otherwise reuse IDs.

4.5.1.1. Licence Requirement Checks

At the request planning stage the EMS determines the licences required (as far as is possible in advance). This results in a list of licence requirement specifications. These may vary in complexity, from simply identifying a dataset to specifying an AoI, ToI and additional attributes, depending on platform support and on any knowledge the EMS has about which request fields are licence-relevant. The licence manager, however, only performs matching of these against rules or configuration using no or limited knowledge of the specific meaning of fields.

On receiving licence requirement specifications, the licence manager must compare them against the licences and grants possessed by the user and determine what licences, if any, must be obtained by the user before the action is permitted. On failure, the result should contain something the user can act on, such as a URL for viewing and agreeing to dataset terms or for buying licences. On success, the licence manager may return information on which fields were used so that the EMS can avoid repeated checks.

The licence manager should also be able to determine when additional commercial licence grants should be added (and charged for) automatically. The user must have previously agreed to the

license terms and pricing. When a new licence grant is added it should record it and issue a billing event.

Licence grants may also be managed by an external service operated by the licenser. This communication is managed entirely by the licence manager.

Processing may cross platform boundaries within the federation. A platform executing processing or supplying a resource must be able to determine that the processing is running in a context in which any required licences are available. To support this, the context must include enough information to identify the licence manager of the originating platform. When a licence manager receives a licence requirement specification which can't be satisfied locally it should use this endpoint to perform a licence requirement check. The originating platform may fail this request, may accept it based on existing data (returning signed licence grants if appropriate) or may perform an automated licence grant acquisition. The host/compute platform may then store these licence grants against the user ID for use in future checks.

Note that cross-platform executions may involve running, for example, a processing chain initiated from platform A with a component involving a commercially licenced compute service from platform B running on platform C. In these cases platform B may check that the user has accepted its platform acceptable use policy by contacting platform A, fetch the compute service from platform C which will then also directly contact platform A to ensure that licences are available before returning the container ot platform B.

TODO: Diagrams!

4.5.1.2. Licence Acquisition

Unless managed by an external service, users must be able to view and accept terms and purchase licence grants using the licence manager. For licences where no licence grants must be bought this is very simple - for example, the licence manager may provide APIs enabling the UI to fetch licence text and submit acceptance. This can be done from a resource information display page or following a refused request.

Where a licence grant must be bought the flow for the user is managed by other components. A user interface may be used to choose licence attributes or particular subsets of data, for example, or a user may have the option to allow automatic purchases as data is accessed. This licence manager must support this functionality in the following ways:

- A human-readable description of the pricing model and prices is included with the licence metadata. This should be displayed to the user.
- The licence manager can accept a licence requirement specification and turn it in to either a product code and quantity (which the caller can then pass to the pricing engine) or information on which additional fields are required. The field names, types and UI information is supplied by other services as part of the resource definition.
- The licence manager can accept a command to buy a specified licence. It will then emit a billing event. This may happen synchronously or asynchronously depending on the needs of the platform's billing model.

When federated access is involved, such as when a processing chain runs some components on

another platform or when data or processing services are transferred to run locally, a user may need to accept licence terms or acquire a licence grant for a resource which is not published via the home platform. This must always be initiated from the home platform, either in advance of the execution or in response to an event returned by a host platform. For terms acceptance the licence manager must contact the host platform and transfer the necessary T&C data. For a (commercial) licence grant, the licence manager must ask the billing manager to authorize a payment to the host platform and then make a request to the host licence manager to buy the licence (specifying the payment ID). The host licence manager must verify the price before asking its own billing manager to clear the payment. It should then record the licence grant as well as returning it to the home platform.

4.5.1.3. Licence Administration

Resource owners must be able to configure licences. The UIs and APIs allowing them to do this must interact with the licence manager (and the pricing engine) to configure their licences. This includes only the licences themselves - assignment of licence requirements to resources is out of the licence manager’s scope.

4.5.1.4. Porting Licences Within the Federation

In some cases users may have multiple home platforms, initiating some workloads from different locations. To ensure that users can use their licences for workloads initiated across all locations licence 'porting' may be used.

A user 'ports' a licence from one platform to another by using OAuth to authorize the licence manager on the local platform to access his licences on another. This is only permitted if the licences have been marked as 'portable' by the licenser.

This mechanism may be used for two purposes. In the first, a publisher has published his Resource in both platforms (which may be done to permit lower processing latencies, lower payment processing costs or the use of proprietary features). The platform receiving the licence must verify its signature using the licenser’s public key before accepting it. In the second case the Resource is not available on the receiving platform but may still be used in cross-platform workflows (including the case when a processing service is transferred from a remote host platform to execute locally).

Chapter 5. Processing and Chaining

The Processing & Chaining domain area must provide an extensible repository of processing functions, tools and applications (referred here generically as ‘processing services’) that can be discovered by search query, invoked individually, and utilised in workflows.

The Resource Management domain area provides the facilities through which processing services are published in an Application Catalogue that acts as a Marketplace and facilitates their discovery, (see section Application Catalogue). Via the Marketplace users have a single point of access to all processing services that are published across the federated system. In order to invoke processing services and workflows, users must specify the data inputs and parameterisation.

Users must be able to define and execute workflows that chain processing steps, in which the input(s) of a step are provided by the output of preceding step(s). Users can publish workflows as new processing services, and so the possibility of workflow nesting.

A workflow comprises multiple steps (processing service invocations), each of which can be executed on the platform that is closest to the data. Thus, the workflow must be orchestrated to invoke the steps on the appropriate platform and stage in/out the data between platforms along the execution pipeline. Thus, processing services should be relocatable between federated EO platforms, such that they can be deployed and instantiated for execution ‘close to the data’. This implies that applications are packaged in a way that is self-contained, standardised and agnostic of the underlying hosting technology.

Users must be able to develop and integrate their own processing services into the platform. Once integrated the user can publish their processing service so that it is discoverable by search query and available in the federated marketplace – and hence available for exploitation by other users, including use in workflows. In support of this, an integrated development environment should be provided that allows users to develop, test and debug their applications before submission.

The interface between the Processing Framework and the compute resource should be abstract so that the solution is not tied to any particular provider (cloud, DIAS, etc.).

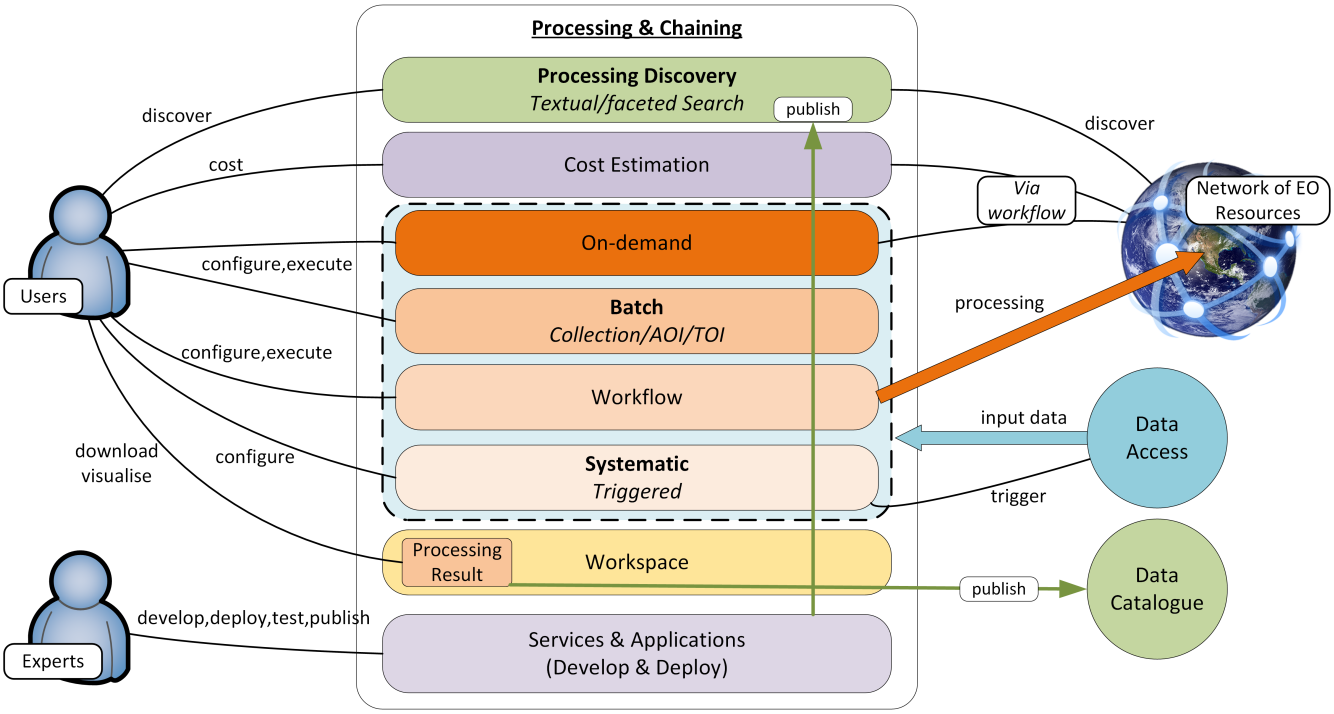

Figure 10. Processing & Chaining Use Case

In meeting these requirements the following key challenges are identified:

- Processing and data interoperability must be established through clear and consistent metadata definitions, to ensure that type mismatches are avoided. This is particularly challenging across federated systems where it becomes more difficult to enforce use of common profiles and vocabularies
- Defining an application packaging approach whose paradigm is easy to work with, whilst providing a rich environment that allows expert users to exploit the full compute capability of the platform
- Federation of processing services across the network of EO resources, such that processing implementations can be made available 'on-demand' amongst federated platforms, to facilitate the movement of the "processing to the data". Use of a common packaging format that is agnostic of underlying host characteristics is key to this challenge
- Enforcement of access controls to processing and data resources through multi-step federated workflows requires the user's 'request context' to be carried through all layers of the request fulfilment. At each point of resource access, the user's identify and access rights must be asserted. The service interface standards, (such as WPS, CSW, WCS, etc.), must be evaluated and necessary enhancements identified to ensure that the user's access envelope is respected

5.1. Solution Overview

The Processing & Chaining solution is based upon the work performed in the OGC Testbeds, described by the following Engineering Reports:

- OGC 17-023 - OGC Testbed-13, EP Application Package ER [TB13-AP]
- OGC 17-024 - OGC Testbed-13, Application Deployment and Execution Service ER [TB13-ADES]
- OGC 18-049r1 – OGC Testbed-14, Application Package Engineering Report [TB14-AP]

- OGC 18-050r1 - ADES & EMS Results and Best Practices Engineering Report [TB14-ADES]

Additionally, the current OGC Testbed-15 Thread-2 Earth Observation Process and Application Discovery (EOPAD).

Processing-services are packaged as Docker images, which can then be deployed as self-contained applications within the Exploitation Platform's processing framework. OGC-WPS provides a standard interface to expose all processing services (and workflows) for invocation by WPS clients.

Each processing service is described by an Application Descriptor, which is a file that accompanies its deployment to the processing framework of the EP. The Application Descriptor provides all the metadata required to accommodate the processor within the WPS service and make it available for execution.

The architecture is defined by the following main components:

Execution Management Service (EMS)

WPS-T (REST/JSON) service that provides an umbrella orchestration service to deploy/invoke processing services within the ADES of the appropriate (close to data) Exploitation Platform. Thus, the EMS is responsible for the orchestration of workflows, including the possibility of steps running on other (remote) platforms, and the on-demand deployment of processors to local/remote ADES as required.

Application Deployment and Execution Service (ADES)

WPS-T (REST/JSON) service that incorporates the Docker execution engine, and is responsible for the execution of the processing service (as a WPS request) within the 'target' Exploitation Platform (i.e. one that is close to the data). The ADES relies upon the EMS to ensure that the processor is deployed as a WPS service before it is invoked.

Application Catalogue (ref. Resource Management domain)

An Application Catalogue provides an inventory of processing services that acts as a Marketplace for the discovery and browse for processing services. The Application Catalogue provides a service that can be searched by facet/keyword and provides supporting metadata and information.

Thus, each platform that supports processing should include an ADES, and each platform that supports workflow orchestration should include an EMS.

Figure 11 illustrates the main architecture components and their interfaces.

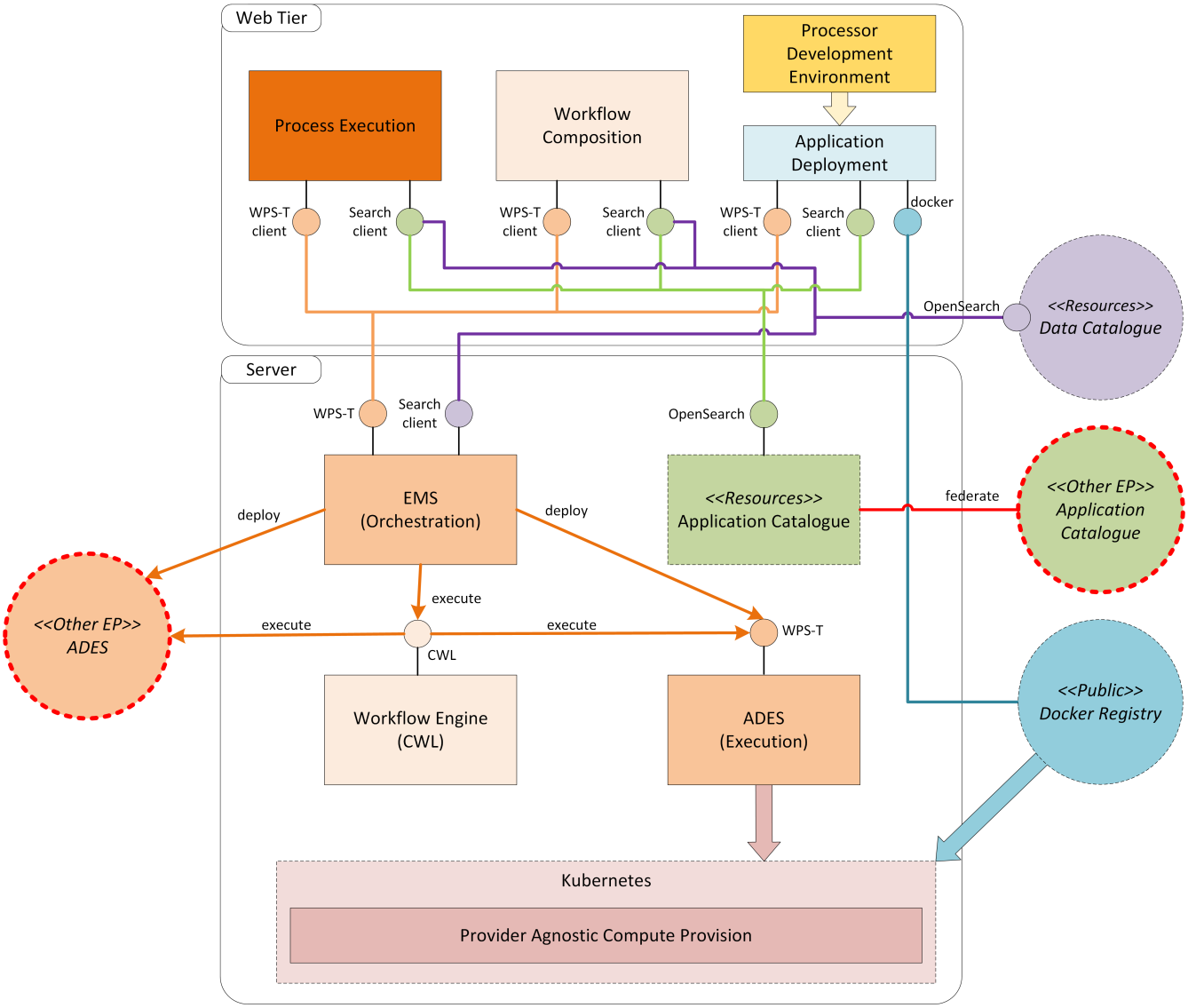

Figure 11. Processing & Chaining Overview

In order for processing services and their data input/outputs to be aligned a formalism is required to describe the data types for M2M consumption. This is required to ensure that a processor is invoked with compatible data inputs, its outputs are understood, and that coherent workflows can be constructed in which the outputs → inputs are aligned. For this purpose, Common Workflow Language (CWL) is used in the Application Package Description to describe the processor input/ouputs.

The Application Catalogue is the subject of the current Testbed-15 through which the Data Model and catalogue Service Interface are being explored.

For the Expert User with a service/application to execute in the EP, we might consider three levels of integration:

- **Importing**
  The service/application is packaged (unchanged) as a black-box.
  Relies upon the stage-in/out of data to the applications existing data access expectations by the Processing Framework.
- **Adapting**
  The service/application is adapted (modified) to use the data access interfaces offered by the

Common Architecture.

- **Porting**
  The service/application is ported to use the services of the EP intrinsically - typically by use of the Client Library defined by the common architecture.

Section Processing Service Data Access provides further discussion regarding the data stage-in/out approach for processing services.

5.2. Resource Layer (Infrastructure) Interface

The Processing & Chaining has significant points of interface with the hosting infrastructure for provision of scalable compute resource and access to data for input/output. The definition of this interface should be agnostic of the infrastructure provider onto which the Exploitation Platform is deployed.

Kubernetes provides an infrastructure abstraction layer that allows the EP to be architected in a way that is agnostic to the underlying hosting infrastructure – the only requirement being the existence of a K8s cluster in which to deploy and run the platform. This abstraction provides points of interface for:

- System deployment
- Access to back-end data
- Execution of processing services and applications

Thus, the Processing & Chaining solution is designed to utilise a Kubernetes Cluster, whose API provides the means to invoke the WPS processing services as docker containers, and also provides the means to support stage-in/out of data for the process execution.

This has particular impact on the ADES, as described in [ADES].

5.3. Application Packaging

The Application Package provides a platform independent and self-contained representation of a software item, providing executable, metadata and dependencies such that it can be deployed to and executed within an Exploitation Platform. Typically, in the context of the exploitation platform, the application is an EO data processing algorithm or a workflow.

The Application Package allows the application to be exchanged in an interoperable way on any platform within the EP ecosystem. Additionally, the developer of the package need only concern themselves with conformance to the package specification and need not concern themselves with the infrastructure details of any particular EP.

The Application Package comprises two main parts:

- Application Descriptor - metadata descriptor file
- Application Artefact – i.e. the ‘software’ component that represents to the execution unit

In accordance with the approach advocated in OGC Testbed-14 (ref. [TB14-ADES]), the Application

Descriptor is encoded in accordance with the WPS-T DeployProcess document defined by WPS-T JSON encodings (ref. [WPS-REST-JSON]). In this way, the Application Descriptor broadly provides the following details:

- A link to the application execution unit
- A description of the application's inputs and outputs
- Other auxiliary information

Currently supported are two types of application execution unit:

1. Docker container
2. Workflow, expressed in CWL

...but the design of the application package should be extensible to support future types.

The Application Descriptor must address the needs of at least two types of users:

Application Developers

Who may not be IT experts (such as scientists), requiring an encoding that is simple enough for them to create for themselves

Machine-To-Machine (M2M)

Requiring all the information to ensure that the application is fully portable and will behave the same on all supporting platforms

Figure 12 provides an illustration of the Application Descriptor structure.

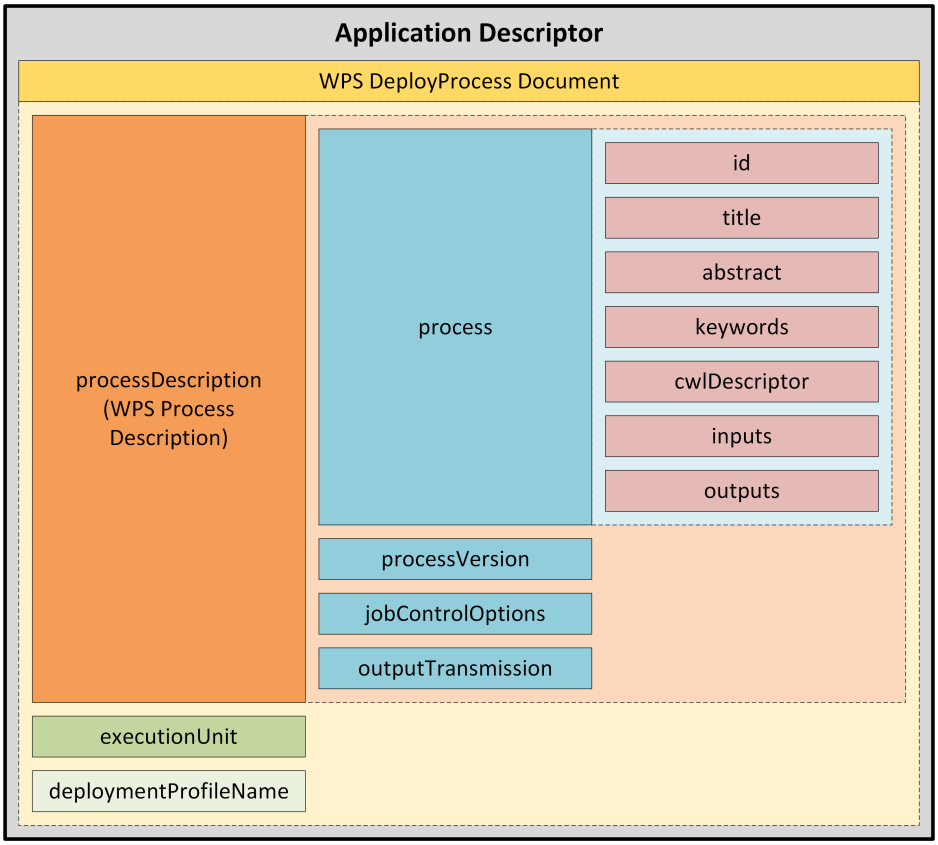

Figure 12. Structure of Application Descriptor data model

Thus, the WPS-T DeployDocument comprises the following parts:

processDescription (WPS Process Description (WPD))

Corresponds to a WPS Process Description document encoded in JSON, including details such as process ID, name, title, etc. as well as options to describe the job invocation and the output handling.
Additional points of note:

- cwlDescriptor
  The cwlDescriptor provides a CWL formatted (YAML) workflow definition of the application. This aids the stage-in/out of data by providing a CWL definition of the input/outputs of the application, and is given in addition to the inputs/outputs included in the body of the WPD. This entry is included as an extension to the WPD via an owsContext offering.
  Note that this is not required for the Execution Unit type of 'workflow' which already carries its CWL file in its executionUnit parameter.
- inputs/outputs
  Specifies the number and types of the data input/outputs. Provided as part of the WPD, and in addition to the contents of cwlDescriptor.
  The inputs can be provided as references to data, accessible through data access service endpoints, or can be specified as query parameters collection/AOI/TOI.

executionUnit

Specifies the 'software' item to be executed, within the context of the deploymentProfileName, as follows:

- dockerizedApplication
  executionUnit specifies the URL of the docker image to run.
- workflow
  executionUnit specfies the URL of the CWL file that defines the workflow.

deploymentProfileName

Enumerates the type of the executionUnit. Currently supported:

- Docker image (http://www.opengis.net/profiles/eoc/dockerizedApplication)
- CWL Workflow (http://www.opengis.net/profiles/eoc/workflow)

Example Application Descriptor

```
{
  "processDescription": {
    "process": {
      "id": "EoepcaProcessor",
      "title": "EOEPCA Processor",
      "owsContext": {
        "offering": {
          "code": "http://www.opengis.net/eoc/applicationContext/cwl",
          "content": {
            "href": "https://eoepca.github.io/processor/cwl/EOEPCAProcessor.cwl"
```

```
          }
        }
      },
      "abstract": "",
      "keywords": [],
      "inputs": [
        {
          "id": "images",
          "title": "Input Images",
          "formats": [
            {
              "mimeType": "application/zip",
              "default": true
            }
          ],
          "minOccurs": 1,
          "maxOccurs": "unbounded",
          "additionalParameters": [
            {
              "role": "http://www.opengis.net/eoc/applicationContext/inputMetadata",
              "parameters": [
                {
                  "name": "EOImage",
                  "values": [
                    "true"
                  ]
                }
              ]
            }
          ]
        }
      ],
      "outputs": [
        {
          "id": "output",
          "title": "Stacked Image",
          "formats": [
            {
              "mimeType": "image/tiff",
              "default": true
            }
          ]
        }
      ]
    },
    "processVersion": "1.0.0",
    "jobControlOptions": [
      "async-execute"
    ],
    "outputTransmission": [
      "reference"
```

```
    ]
  },
  "executionUnit": [
    {
      "href": "hub.docker.com/eoepca/processor:latest"
    }
  ],
  "deploymentProfileName": "http://www.opengis.net/profiles/eoc/dockerizedApplication"
}
```

5.4. Execution Management Service (EMS)

The EMS provides a Transaction WPS 2.0 (WPS-T) interface, with REST/JSON encodings, as described in section WPS-T REST/JSON.

WPS-T extends standard WPS by adding DeployProcess and UndeployProcess operations. Once a process has been deployed to a WPS then the existing wps:Execute operation remains applicable for execution in the standard way.

The EMS provides a WPS-T (REST/JSON) interface that provides an umbrella orchestration service to deploy/invoke processing services within the ADES of the appropriate (close to data) Exploitation Platform. Thus, the EMS is responsible for the orchestration of workflows, including the possibility of steps running on other (remote) platforms, and the on-demand deployment of processors to local/remote ADES as required.

The description in this section refers to the WPS operations: GetCapabilities, DescribeProcess, Execute, GetStatus, GetResult, DeployProcess, UndeployProcess. See WPS-T REST/JSON for a mapping of these operations into the REST/JSON encoding.

The EMS provides the endpoint for the user's web client, through which applications and workflows are deployed to the EMS to make them available for execution.

Figure 13 illustrates the deployment of applications and workflows to the EMS.

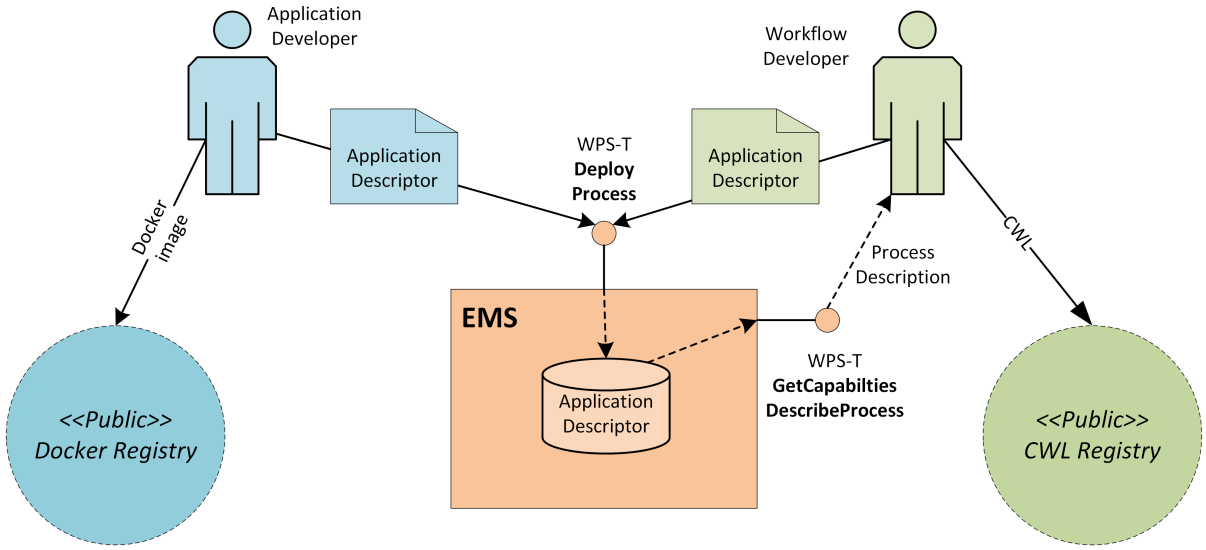

Figure 13. EMS Deployment

Applications are deployed to the EMS so that they are available for inclusion in workflows.

Workflows are deployed to the EMS where the steps of the workflow reference applications that are known to the EMS.

As illustrated in Figure 14, the EMS orchestrates the workflow execution by invoking the steps as subordinate invocations of wps:Execute at the ADES identified at time of task invocation. The EMS uses wps-t:DeployProcess on the target ADES to ensure that the process is registered before execution.

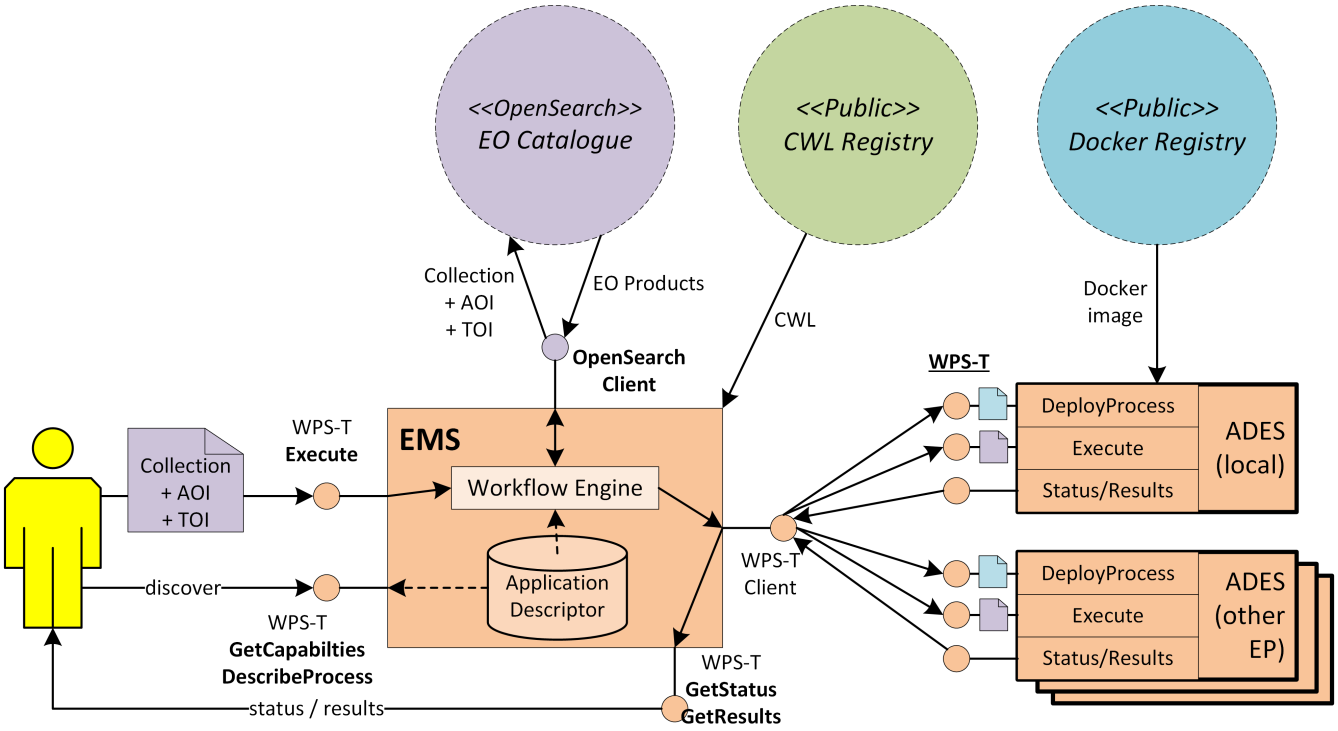

Figure 14. EMS Workflow Execution

At time of `wps:Execute` the input data must be specified by the invoking user. Two possibilities are currently identified, both of which should be supported by EMS:

1. Direct URL references to specific data products, accessible through data access service endpoints (such as WCS, WFS, etc.)
2. OpenSearch query parameters that identify the data characteristics as a combination of Collection/AOI/TOI

In case 1) the EMS can simply pass-through the input arguments to the ADES WPS-T.

In case 2) the EMS must resolve the input data by OpenSearch catalogue queries with the provided parameters. The OpenSearch catalogue end-point can either be defined by the application (in its Application Descriptor), or defined as a parameter of the `wps:Execute`.

In either case, the end result is that the EMS resolves the input specification to a set of data products URLs that can be passed on to the ADES for execution.

The EMS requires a means to determine the target platform (ADES) for the execution, i.e. typically the one closest to the data. In the case of the OGC Testbeds, this determination was made as a one-to-one mapping from the collection identified in the input data specification. If collections are

identified to be globally unique, e.g. with a namespace prefix that identifies the hosting platform, then this assertion can be reliably made and the target ADES can be derived from the collection ID. Otherwise, the `wps:Execute` must be parameterised suitably to identify the target ADES.

Performing the orchestration between steps, the EMS must handle the stage-in and stage-out of data. In the simple case, the result URL returned from a step can be directly used as an input URL for the subsequent step. Use of CWL and the accompanying `cwl-runner` tool should facilitate this orchestration.

The end result of the successful execution is to present the output result to the invoking user. The EMS establishes the location of the results within the storage provision of the Exploitation Platform, and interfaces with the EP Workspace component (Workspace) to register the result in the user's workspace. At this point the WPS execution is complete as reported by the `wps:GetStatus` and `wps:GetResult`.

5.5. Application Deployment and Execution Service (ADES)

The ADES provides a WPS-T (REST/JSON) service that incorporates the Docker execution engine, and is responsible for the execution of the processing service (as a WPS request) within the 'target' Exploitation Platform (i.e. one that is close to the data). The ADES relies upon the EMS to ensure that the processor is deployed as a WPS service before it is invoked.

The main responsibilities of the ADES are:

- Check the user is authorized to access the requested data
- Perform stage-in of data before execution
- Invoke the container from the Docker image in accordance with the ApplicationDescriptor and the `wps:Execute` request
- Monitor the status of the job and obtain the results
- Perform stage-out of results at execution conclusion

Resource Layer (Infrastructure) Interface introduces the use of Kubernetes (K8s) as the provider agnostic interface to the Resource Layer. The ADES has touch-points with the Resource Layer for access to data and compute resource. The following sub-sections elaborate the approach.

The work carried out in the OGC Testbeds 13/14, performed the execution of the 'packaged' processing service by invoking the 'run' of a docker container in the machine that hosts the WPS-T service. The Common Architecture design builds upon this, by instead invoking the container as a K8s Job that is deployed for execution in the K8s cluster.

This approach is consistent with the current Application Package / ADES definition that specifies a docker image for the processing service. As illustrated in Figure 15, the ADES provides a K8s-aware Execution Engine that handles the complexities of constructing the jobs and interfacing with the K8s cluster.

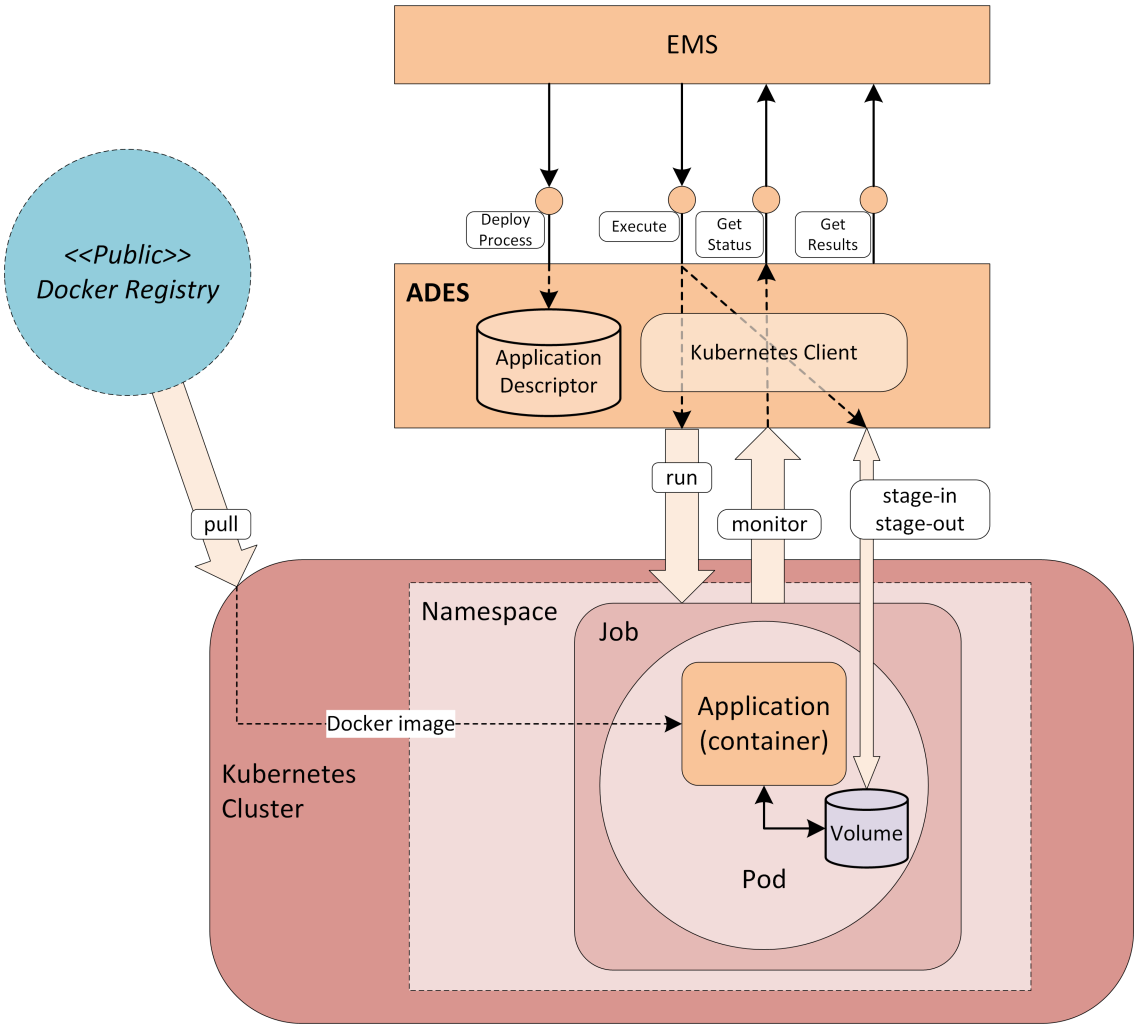

Figure 15. ADES Process Execution

A Kubernetes cluster comprises a set of Nodes. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine. Each Node is managed by the Master. A Node can have multiple pods, and the Kubernetes master automatically handles scheduling of the pods across the Nodes in the cluster. The Master's automatic scheduling takes into account the available resources on each Node.

Pods are the atomic unit on the Kubernetes platform. A Pod is a Kubernetes abstraction that represents a group of one or more application containers. A Pod always runs on a Node. The containers in a Pod share an IP Address and port space, are always co-located and co-scheduled, and run in a shared context on the same Node.

Each WPS processing task will be constructed as a Pod and invoked as a dedicated K8s Job. A Job creates one or more Pods to perform a given task. The Job object takes the responsibility of Pod failures. It makes sure that the given task is completed successfully. Once the task is over, all the Pods are terminated automatically.

Kubernetes provides Namespaces, which are an abstraction that supports multiple virtual clusters on the same physical cluster. It may be interesting to explore the use of K8s Namespaces for the purpose of establishing a sandboxed execution environment for each task execution.

5.6. Processing Service Data Access

The Processing Framework (EMS/ADES) provides the environment through which processing services and workflows access input/output data. The EMS must ensure that the outputs of one step are marshalled to the next, and the ADES must prepare the inputs before job invocation, and collect the outputs at the job conclusion. The processing task is invoked as a Docker container. In doing so, the container execution environment must be provisioned with the input data for the task, and with the means to 'export' its outputs to the processing orchestration. Figure Figure 16 illustrates.

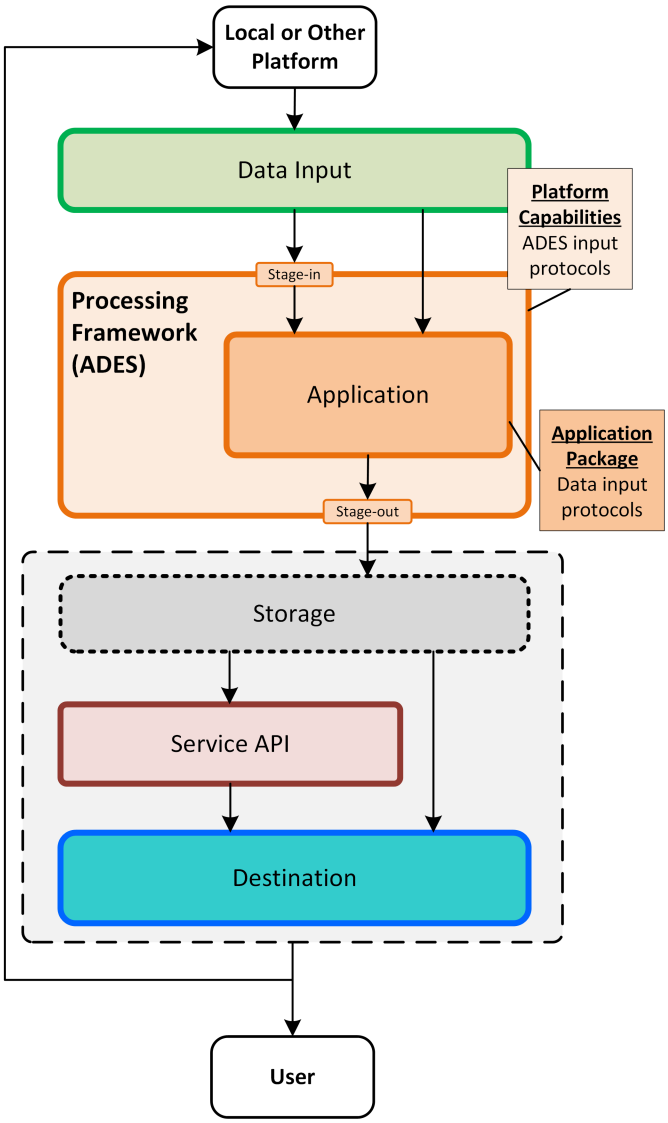

Figure 16. Processing Framework Data Access

Data Input Sources

The data input sources may be the local exploitation platform, or output from a previous workflow step (local or other platform).

Data Input Protocol

The data input protocol may be natively supported by the processing service - otherwise it will need to be staged-in by the Processing Framework.

Data Stage-in

In order to stage-in the data, the Processing Framework must support the data access protocol

through which the input data is provided, and it must know the capabilities of the processing service to be invoked. **This represents a two-way contract between the Platform (on behalf of the Processing Framework) and the service/application being executed.** Options for staging-in the data include local file-system access (e.g. via s3fs-fuse mount), or it may be more optimal for the service to access the data directly via HTTP-based interfaces, e.g. to exploit the efficiencies offered by cloud-optimised file formats.

Data Stage-out

The ADES stages-out the results from the service/application.
The EMS orchestrates the outputs to the next workflow step, or makes available the results to the end-user.

Data Output Protocol

The data output protocol must be supported by the data destination. The retrieval of the output data is facilitated by use of simple approaches that can be encoded in an HTTP-base URL, such as HTTP GET/KVP or Object Store.

Data Output Destination

The data output destination is either the EMS/next-process for a workflow, or the end-user receiving their results. The two-way Platform (Processing Framework) <→ Service/Application contract informs this data flow, with the workflow construction/orchestration taking into account the respective capabilities of the service/application and the platform in which it is being invoked.

The work carried out in the OGC Testbeds 13/14 relied upon use of mounted volumes within the processing task docker container. These mounted volumes present the input data, and receive the output data, as 'local' file system access from the point of view of the running container and the processor running within. Use of mounted volumes is equivalently supported by K8s, and is an approach that is relied upon for many (existing) applications that are capable only of accessing data through POSIX file-system interfaces.

Thus, for Mounted Volumes, the approach is to use standard container volumes that present as well-identified directories within the container. The input data is provided in a read-only input directory that mounts into the container hosting infrastructure. Similarly, an empty writable directory is presented for the processing to write is outputs, to be collected by the Processing Framework.

Nevertheless, we might envisage that access to the underlying data will be provided by the hosting platform through a variety of data access protocols, including: Object Store (S3/Swift), OGC (WMS, WMTS, WFS, WCS, WCPS), OPeNDAP, plus local file-system as mentioned above.

The processing framework must establish an environment in which the data access capabilities of the processing service are matched to the data access offering of the platform. There are two possibilities that need to be handled by the Processing Framework:

1. The processing service / application natively supports the data access protocol, in which case there is no need to stage-in the data. Nevertheless, the Processing Framework must support the pass-through of input data as URL, and the reception of the output(s) as URL. This is facilitated by the utilisation of a Data Access Library as described below. Also, it must be ensured that any

outputs, e.g. to object store, are appropriately directed.

2. The processing service / application does not natively support the data access protocol(s) offered by the underlying platform storage, e.g. the processing service only supports local storage (mounted volume), in which case the Processing Framework must facilitate access to the data on behalf of the processing service.

In cases where the processing service does not natively support the data access protocols offered by the underlying platform, the processing framework must facilitate access to platform data in a form that can be consumed by the processing service.

To support this, the service/application should declare the data access protocol that it requires. This declaration should be made in the Application Descriptor (ref. Application Package), and is selected from a standard set of protocols, including:

- AWS S3 Object Store
- Swift Object Store (OpenStack)
- OGC data access services:
  - Web Map Service (WMS)
  - Web Map Tile Service (WMTS)
  - Web Feature Service (WFS)
  - Web Coverage Service (WCS)
  - Web Coverage Processing Service (WCPS)
- OPeNDAP
- Mounted Volume (local storage)

Building upon the work of OGC Testbed-14 (ref. [TB14-AP]), the Application Descriptor described in section Application Packaging can be enhanced to include additional application capabilities, as illustrated in Figure 17.

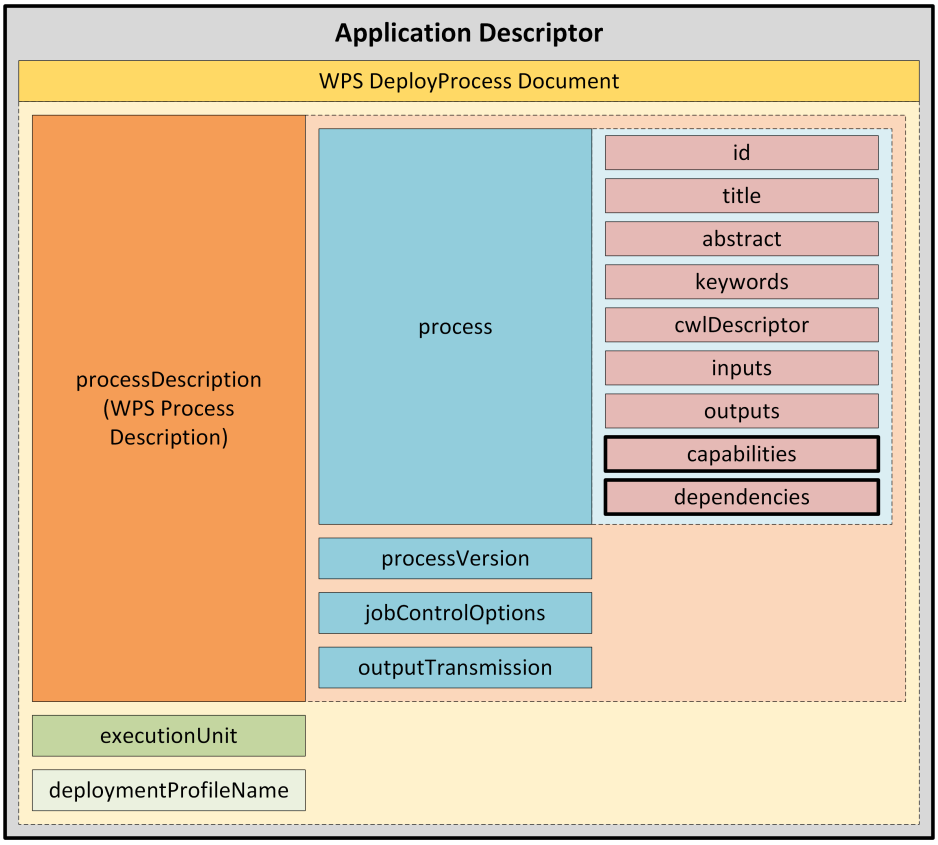

Figure 17. Application Descriptor (Enhanced)

The Application Descriptor is enhanced with additional components:

- **Capabilities**
  Containing, for example, data access protocols supported, (implicitly 'local filesystem' for OGC Testbeds 13/14).
- **Dependencies**
  Specifying, for example, required interfaces to a specific processing, such as dask/Spark etc.

The 'Platform Capabilities' are elaborated in section Platform Capabilities.

The platform implementation must then ensure that the data interface is presented to the processing container in accordance with its descriptor. It is the job of the platform implementation to translate from the back-end data access protocol to that required by the container. There are a number of techniques that can be employed by the processing framework to facilitate this data access mediation:

Local File-system Stage-in/out

The processing framework must perform the data retrieval to stage-in the data for presentation through a mounted volume as local file-system access. At the conclusion of the processing the outputs must be marshalled into the appropriate platform storage for further consumption, e.g. push to object store.
The approach can be facilitated by use of a FUSE file-system driver to achieve the mediation.

Use of Filesystem in Userspace (FUSE)

Access to the back-end data storage (e.g. HTTP-based) is provided through a user-space driver that presents the remote data as if it were a local directory. For example, using s3fs-fuse

(https://github.com/s3fs-fuse/s3fs-fuse), access to data in an S3 object store is provided through a FUSE mounted directory. Thus, from the perspective of the processing task, inputs (read) and outputs (write) are accessed through the local file-system interface – satisfying the constraints of the processor.

Use of Data Access Library

A Data Access Library (DAL) provides an abstraction of the interface to the data. The library provides bindings for common languages (including python, Javascript) and presents a standard programmatic semantic for accessing the data from within the processing service codebase. Specific implementations of the DAL can be made to abstract the data access layer for a given Exploitation Platform. The Processing Framework must support the ability to 'plugin' an alternative implementation of the DAL at processor execution time.
See section Data Access Library.

Data Access Gateway

Access to the underlying platform data is provided through a common service layer that provides standard data access interfaces that are translated (gateway) to those of the underlying data tier.
For example, to satisfy a processing service that requires an S3 interface, but is executed in an environment where the data is available through a POSIX file system. Minio (https://github.com/minio/minio) is an open source object store implementation that overlays an S3 interface over a POSIX file system.
See section Data Access Gateway.

5.6.1. User Authorization Context

The stage-in/out of data must operate within the context of the user's 'account'. Thus, the security context of the user must be passed through all aspects performed by the Processing Framework on behalf of the user. This is necessary to ensure that the user is only able to access data to which they are entitled and accounting & billing considerations are properly maintained.

5.7. WPS-T REST/JSON

This interface specification is used for both the Client <→ EMS, and the EMS <→ ADES interfaces.

WPS-T extends standard WPS by adding *DeployProcess* and *UndeployProcess* operations. Once a process has been deployed to a WPS then the existing `wps:Execute` operation remains applicable for execution in the standard way.

The following table is reproduced from [TB14-ADES].

Resource	HTTP Method	Description	WPS operation
/	GET	The landing page provides links to the API definition, the Conformance statements and the metadata about the processes offered by this API	
/processes	GET	Retrieve available processes	Get Capabilities
/processes	POST	Deploy a process	Deploy Process
/processes/{id}	GET	Retrieve a process description	Describe Process
/processes/{id}	DELETE	Undeploy a process	Undeploy Process
/processes/{id}/jobs	GET	Retrieve the list of jobs for a process	
/processes/{id}/jobs	POST	Execute a process	Execute
/processes/{id}/jobs/{jobID}	GET	Retrieve the status of a job	GetStatus
/processes/{id}/jobs/{jobID}	DELETE	Dismiss a job	
/processes/{id}/jobs/{jobID}/result	GET	Retrieve the result(s) of a job	GetResult
/processes/{id}/quotations	GET	Retrieve the list of quotation ids for a given process	
/processes/{id}/quotations	POST	Request a quotation for a given process	
/processes/{id}/quotations/{quotationID}	GET	Retrieve quotation information	
/processes/{id}/quotations/{quotationID}	POST	Execute a quoted process	
/processes/{id}/visibility	GET	Retrieve the visibility status for a process	
/processes/{id}/visibility	PUT	Change the visibility status for a process	
/quotations	GET	Retrieve the list of all quotation ids	

Resource	HTTP Method	Description	WPS operation
/quotations/{quotationID}	GET	Retrieve quotation information	
/quotations/{quotationID}	POST	Execute a quoted process	
/bills	GET	Retrieve the list of all bill identifiers	
/bills/{billID}	GET	Retrieve bill information	
/conformance	GET	list all requirements classes specified in the standard (WPS REST/JSON Binding Core) that the server conforms to	

5.8. Interactive (Graphical) Applications

The work carried out in the OGC Testbeds focused on non-graphical applications, i.e. non-interactive processing functions executing algorithms without intervention. It is also noted that WPS does not facilitate the invocation of GUI-based interactive applications which offer a synchronous experience to the end-user.

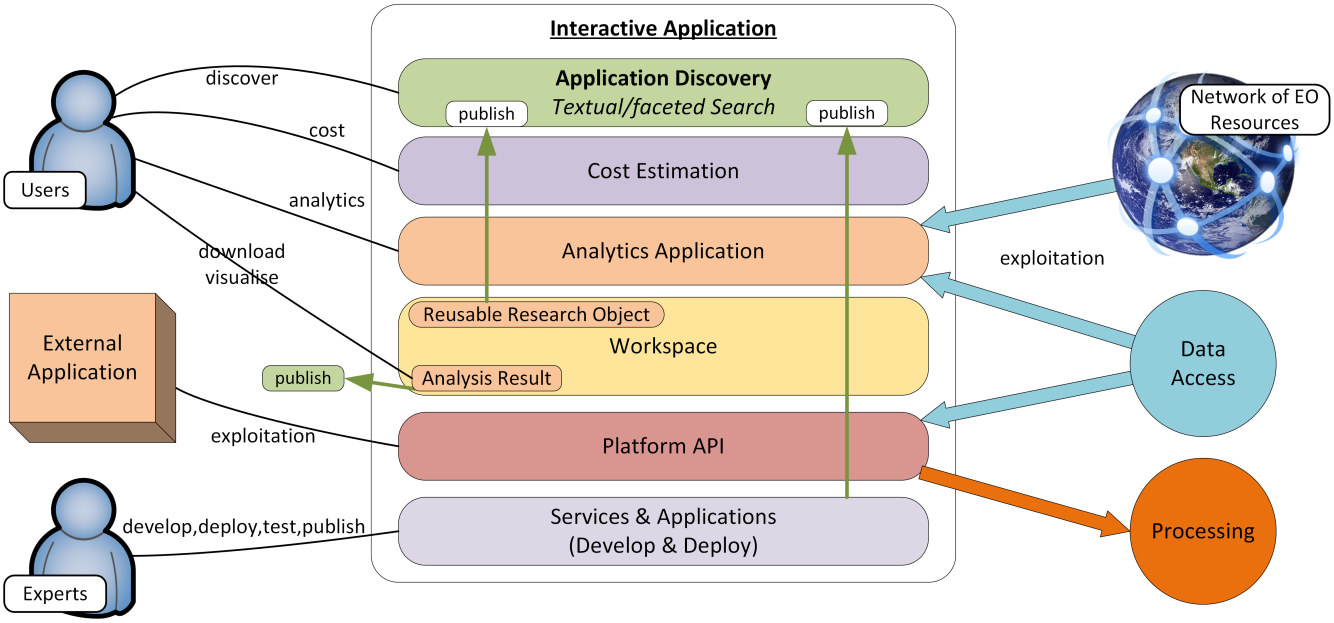

Figure 18. Interactive Applications Use Case

That said, the approach to application packaging undertaken in the testbeds does lend itself to the packaging of GUI-based applications, which can be packaged, deployed and executed as docker containers, including:

Native applications

A remote desktop (RDP) approach is used to present the interface to the user, typically rendered through a web page presented in the user's browser

Web applications

The web application is delivered through the portal interface of the hosting exploitation platform.

In both cases, docker containers offer a good solution to package and deploy the application. At execution time it is necessary to ensure that the appropriate ports are exposed from the running container.

The Application Descriptor needs to be extended to:

- Introduce additional `deploymentProfileNames` and `executionUnit` types
- Provide parameterisation to support the delivery of the GUI to the end-user
- Ensure that data access is presented within the container in a way that is compatible with the GUI application (the mechanisms provided for non-interactive applications may be sufficient).

5.9. Parallel Processing

The OGC Testbeds 13/14 only consider serial processing jobs running in a single Docker container. Here we consider how this approach can be extended to accommodate job requiring parallelisation.

One possible approach, is to invoke the processing task as a docker container (as described above), but then within the implementation of this task it makes subordinate invocations that exploit some specific data processing clustering infrastructure available within the platform. For example, the invoked process executes some Python code that then invokes a dask or SLURM cluster to perform the processing work.

In this case, the processing task would have a dependency that the Exploitation Platform provides the required data processing technology. In order to resolve this capability dependency the following approach can be made:

- The processing service declares within its Application Deployment Package, that it ‘requires’ a particular service
- The Exploitation Platform declares within the capabilities document output from its WPS endpoint, that it ‘provides’ particular services
- The EMS must ensure that the target EP provides the required service of the processing task to be invoked
- The parameterisation for the ‘required’ service are passed to the processing task at invocation

A consistent vocabulary of services must be defined to unambiguously express the ‘required’ and ‘provides’ declarations.

5.10. Processor Development Environment (PDE)

The Processor Development Environment provides a rich, interactive environment in which processing algorithms and services can be developed, tested, debugged and ultimately packaged so that they can be deployed to the platform and published via the marketplace.

The PDE supports the packaging of the user's application in accordance with the Application Packaging format that is suitable for deployment at the EMS/ADES. It provides a sandboxed environment in which the user can test the deployment and execution of their packaged application, with access to suitable test data to perform the validation.

The PDE provides the tools for the developer to fully specify the metadata for their validated application and then add it as a tool in their workspace and/or publish it to the Resource Catalogue for wider consumption.

5.11. Interactive Analysis Tool

The Interactive Analysis Tool presents a hosted coding environment through which expert users can interact directly with the data and services of the platform. It should provide support for a variety of coding languages, including those most popular in the community – Python, R, Julia, Javascript.

For example, the Interactive Analysis Tool can be provided as a Jupyter Notebook instance that is provisioned in the user's context with a 'platform integration' layer that provides simple access to platform resources, including resources held within the user's Workspace.

It must be integrated with the platforms authentication and authorization scheme in accordance with the IAM approach described by the User Management domain.

The Interactive Analysis Tool should support the user to save their interactive analysis sessions for future resumption and sharing with others for purposes of collaboration.

Chapter 6. Resource Management

The role of the Resource Management domain is the storage, discovery and access to resources in the Exploitation Platform. In this context, resources primarily refers to data and processing assets.

Storage is largely taken care of by the Resource Tier upon which the Exploitation Platform is hosted. The role of the Exploitation Platform is to ensure that the data can be accessed through common data access protocols based upon open standards.

This is important for:

- the end-user wishing to access data directly, and accessing their results after processing
- processing services accessing data for input/output
- processing workflow steps accessing intermediate outputs from prior steps
- other federated Exploitation Platforms accessing each other's data and services through well understood interfaces

To exploit the services of the Platform, users need to discover available resources and obtain detailed resource information. For example, a user's data discovery workflow should include the ability to view collection/product information and visualise the data in the platform - this applies to data held within the platform, data added by end-users and data produced as the result of processing operations within the platform.

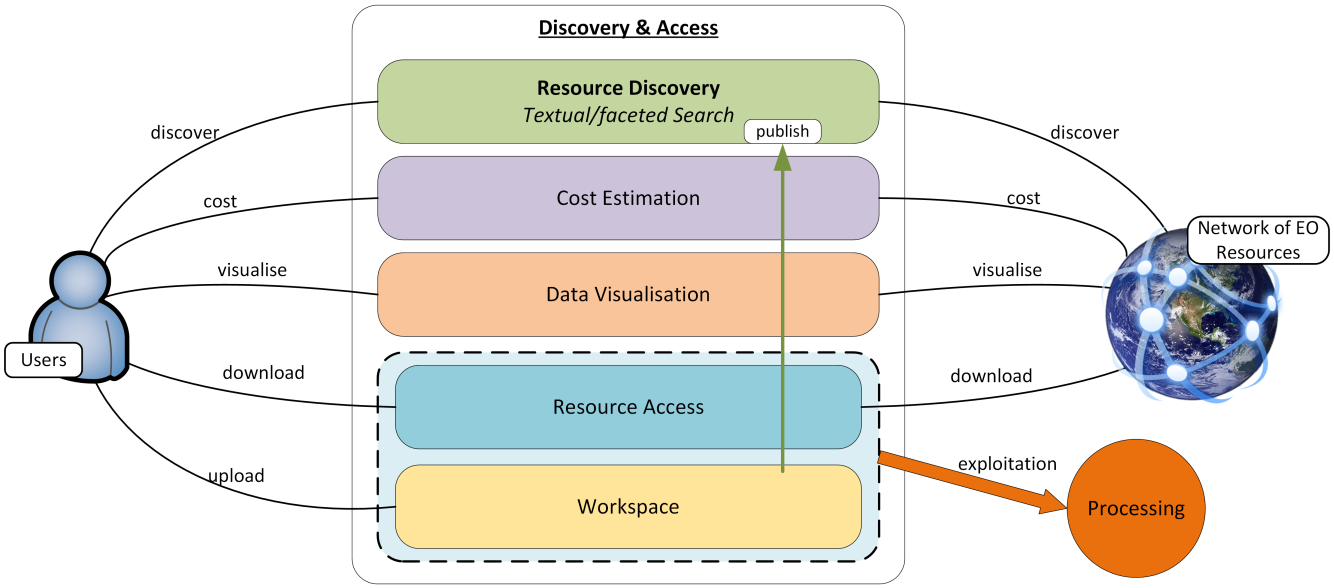

Figure 19. Resource Management Use Case

Processing services and applications are also platform resources that are stored in artefact repositories and must be discoverable by users, including the information required by users to exploit the service. It is assumed that users will store their software artefacts in external public repositories such as DockerHub, GitHub, etc. In the future, it may be necessary for an Exploitation Platform to provide such repository services to its users. Discovery of processing services and applications is met though the provision of an Application Catalogue.

The inventory and presentation of resources to users must be organised in such a way as to facilitate the discovery and usage of resources in other federated Exploitation Platforms. For

example, users must be able to discover data and services in other EPs in order to construct and execute workflows that span multiple federated EPs. **Thus a Resource Catalogue provides the inventory of data, processing services, applications in such a way as to create a Marketplace for resource discovery, and provide a launchpad for their use within the exploitation platform**.

Access to resources must be controlled according to the privileges afforded to the logged in user, and appropriate hooks must be established into the EPs accounting and billing subsystems. Thus, the Resource Management services must be implemented according to the approach defined by User Management for authorization, accounting and billing.

In addition to the resource holding of the underlying resource tier, the EP maintains a User Workspace in which each user is able to maintain specific data/services of interest to them, and also provides a place to hold results of processing operations. The User Workspace should be provided as a building block of the system that provides this personal inventory. Moreover, the concept can be extended to define Group Workspaces to create a place for sharing and collaboration.

A Data Ingestion component abstracts the interface to the underlying Resource Tier storage, ensures that incoming data is formatted in accordance with defined standards, is supported by appropriate metadata and directed towards the appropriate dataset collection.

The main components comprising the Resource Management domain are illustrated in Figure 20:

- Resource Catalogue
- Data Access Services
- Data Access Gateway
- Data Access Library
- Data Ingestion
- Workspace

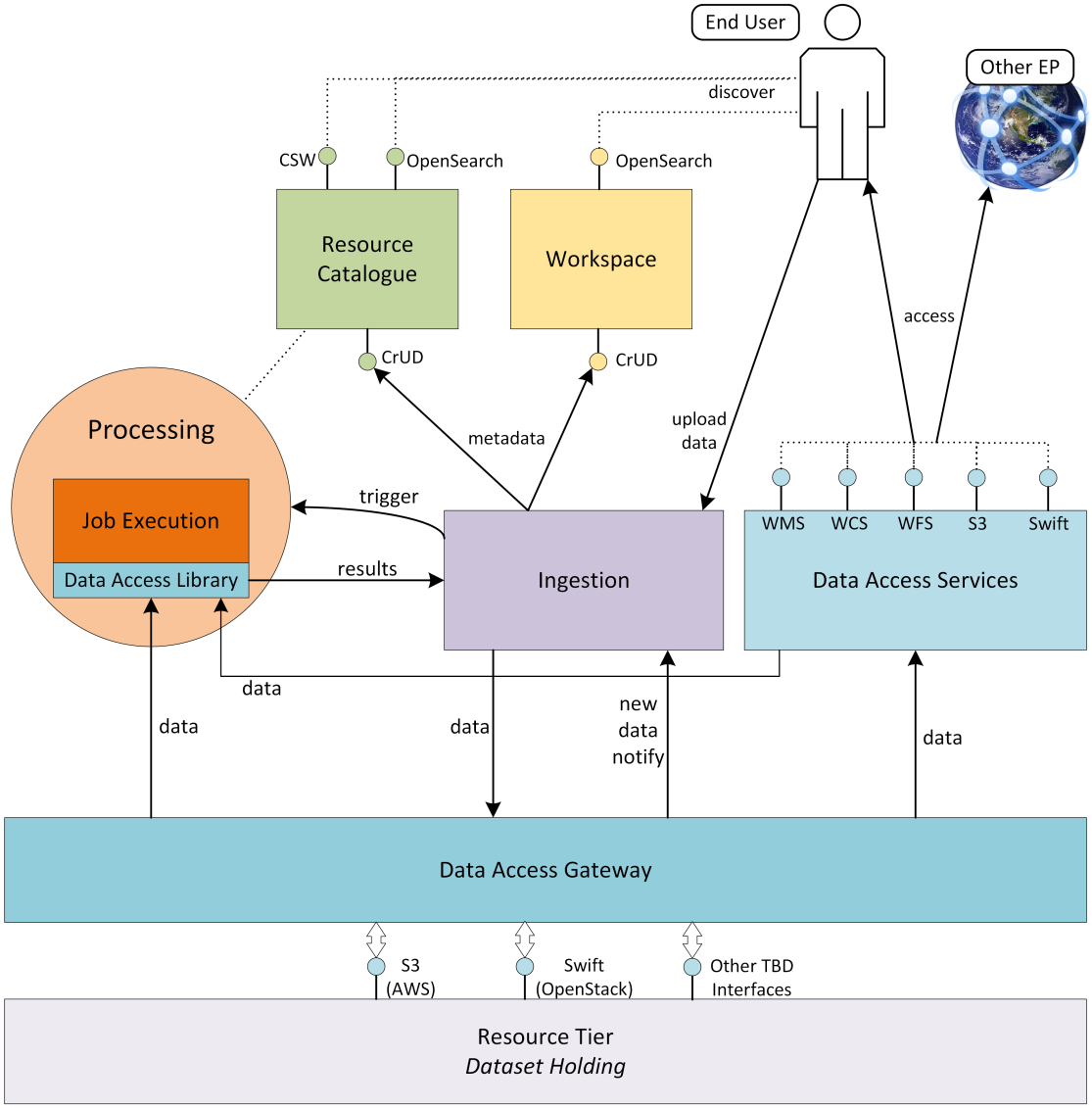

Figure 20. Resource Management Overview

To some degree, the role of these components is to provide an integration of the Exploitation Platform to the Resource Tier, by providing public services that bridge to the underlying data supply.

6.1. Resource Catalogue

The Catalogue provides the user the capability to discover resources, (including data/products/services/applications), by browse/search, and to obtain details on specific resources discovered. Resources of different type can be catalogued and delivered through the same service architecture and catalogue service interfaces - provided as a consolidated catalogue service, or through discrete services for each resource type. In each case the catalogue provides an inventory of resources that can be presented as a *Marketplace* for users to discover and browse.

Perhaps the most challenging aspect of this is that the Catalogues for both Data and Processing-Services must facilitate the proper construction of processing tasks, to ensure there is a correct match of the data types expected as input to the processing. This extends into the construction of workflows where the data types output by a processing task must match the supported inputs of the next task in the chain. The Catalogue must have a rich and consistent metadata model for both Data and Processing-Services in order to achieve these goals.

6.1.1. CEOS OpenSearch Best Practise

The Common Architecture advocates standardisation on the use of OpenSearch based-upon the CEOS OpenSearch Best Practise [CEOS-OS-BP] which provides a blueprint for catalogue search and discovery. Within this context, the following OGC extensions and recommendations are applicable:

- OpenSearch GEO: OpenSearch Geo and Time Extensions [OS-GEO-TIME]
- OpenSearch EO: OGC OpenSearch Extension for Earth Observation [OS-EO]

In addition, the possibility to use the JSON-LD processing model might be considered (further analysis required) through application of:

- OGC EO Dataset Metadata GeoJSON(-LD) Encoding Standard [GEOJSON-LD]
- OGC OpenSearch-EO GeoJSON(-LD) Response Encoding Standard [GEOJSON-LD-RESP]

6.1.2. Application Catalogue

Processing services are published in an Application Catalogue that acts as a Marketplace and facilitates their discovery. Via the Marketplace users have a single point of access to all processing services that are published across the federated system. In order to invoke processing services and workflows, users must specify the data inputs and parameterisation. The metadata for each application record describes what data an application can be applied to, and how it can be chained in a workflow.

The Application Catalogue is the subject of the current OGC Testbed-15 EOPAD Thread, through which the Data Model and catalogue Service Interface are being explored. Thus, in addition to the best practise identified above, the outcomes of the TB15 EOPAD thread should be taken into consideration:

- **OGC 17-084 (GeoJSON(-LD) metadata encoding for EO collections) [GEOJSON-LD]**
  Explore the capabilities of OGC 17-084 to encode application metadata
- **OGC 17-047 (OGC OpenSearch-EO GeoJSON(-LD) Response Encoding Standard) [GEOJSON-LD-RESP]**
  Explore the capabilities of OGC 17-047 to encode OpenSearch responses in GeoJSON(-LD)
  Use of multi-step discovery and faceted search
- **Registration**
  Explore transactional extension to OGC-CSW for application registration.

It is anticipated that the outcome of the OGC Testbed-15 (EOPAD) will further inform the design of the Application Catalogue.

6.1.3. Data Catalogue

The Catalogue provides the user the capability to discover data/products by browse/search, and to obtain details on specific data/products discovered. The Marketplace concept can be extended to embrace the discovery and access to data.

6.1.3.1. Metadata Organisation

The data is organised into Collections, typically representing a dataset. Each collection is composed of multiple granules as files. The catalogue metadata follows a similar organisation and allows the user to discover the data in natural sympathy with this data organisation. Hence, the metadata is presented at the following levels:

Browse Metadata (collection)

Browse metadata is defined at the collection/dataset level. It typically uses ISO19115 records to describe the high-level collection information, such as title, description, spatial/temporal coverage, list of variables available, access rights, T&Cs, etc.
(For collections, the spatial coverage is often full-earth).

Discovery Metadata (product)

Discovery metadata is defined for each granule (file) comprising the collection. This typically includes information such as file-type(s), spatial/temporal coverage, variable, data access (download) method(s). Much of this information can be obtained from the headers of the individual files – depending on file-type. Thus, the Discovery metadata can in-part be populated automatically from the underlying files.

Archive Metadata (file)

Archive metadata refers to the information that is available in the file header. As described above this can be extracted and published into the Discovery metadata of the catalogue.

6.1.3.2. Example Usage with OpenSearch

This metadata model can be exploited, for example, using OpenSearch:

- Initial search is made at the collection level to discover collections/dataset of interest.
- Subsequent OpenSearch requests can then be made to drill-down into a specific collection to discover and obtain details regarding the granules.
- Once discovered, the granules can then be exploited by the user, for example as input to a processing request, or downloaded.
- Facets can be applied to both the Browse and Discovery metadata, to supported facetted search at both levels.

6.1.3.3. Data Access

There is a direct link between the way the data is described in the Catalogue and how it is accessed by the consumers of the data. This links to the Data Access Services (e.g. WMS. WCS, WFS, etc.) provided by the EP, and the way in which the access links are encoded into the Catalogue. These links must be usable by the data consumers which could be processing services, or users downloading the data.

Hence the contents of the Catalogue reflects the data services offered by the platform, including the underlying resource tier services. Each data Collection is presented in the Catalogue as accessible through one or more data access services, as applicable to the specific data. The Catalogue must present the data access URLs in such a way that the URL resolves correctly to the underlying data

via the providing data access service.

6.1.3.4. Catalogue Composition/Aggregation

The Exploitation Platform is designed to be hosted in a compute environment that is close to the data of interest. This means that the typical deployment is made to the likes of DIAS, Public Cloud (such as AWS), or National Research Infrastructure (such as CEDA/JASMIN) – that provide the Resources-tier/infrastructure upon which the EP relies. The Resources-tier provides virtual ICT-infrastructure and data. It is common that the Resources-tier provides their own Catalogue to support the data hosted within.

In order to ensure a coherent link between data discovery and access, the Exploitation Platform provides its own Catalogue that presents the data holding to be accessed through the available data access services. In doing so it must aggregate the catalogue records of the underlying resource tier, the records of other 'federated' platforms, and the value-added data that is contributed through the actions of users on the EP. Thus the EP provides a Catalogue that is tailored to its service offering to ensure a consistent data access interface that can be relied upon by other EP services, in particular by the executing user analysis functions running within the Processing & Chaining context.

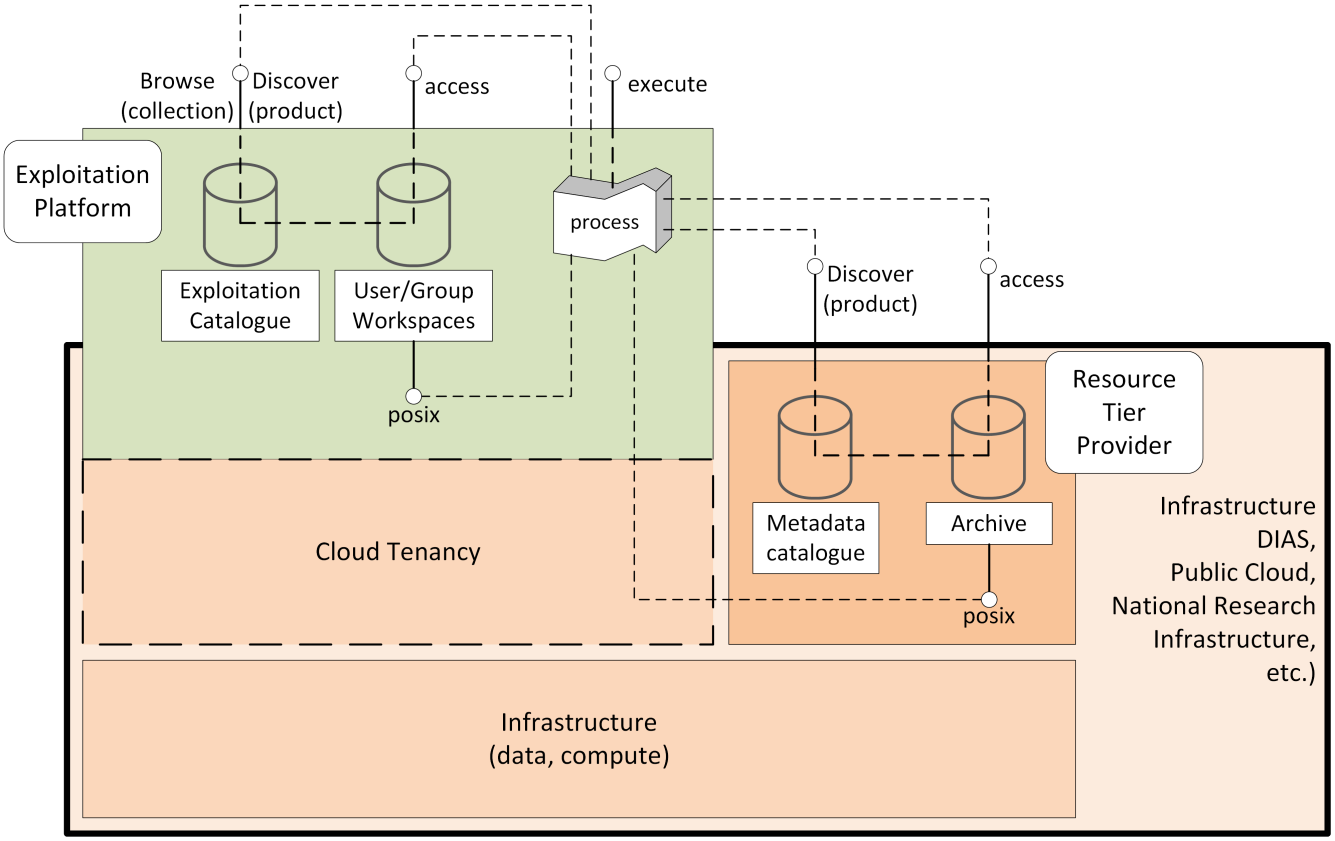

Figure 21. Catalogue Aggregation

We wish the exploitation platform to expose a public catalogue that provides both the Browse (collection) and Discovery (product) views:

- In the case where the Resource-tier provides these in a way that is conformant with the architecture then these can be relied upon directly for the exploitation platform
- In the case where the Resource-tier provides only a suitable Product catalogue, then the Collection catalogue must be provided by the EP, with the granule queries being directed to the back-end catalogue. Alternatively, this could be achieved by harvesting the Resource-tier

product catalogue into the EP catalogue.

- Alternatively, the EP may provide a Catalogue-shim to ensure that an existing Resource-tier catalogue conforms to the interface demands of the open architecture
- Otherwise, the EP must provide all catalogue aspects.

The important point is to ensure that the EP presents interfaces that conform to its defined open standards, and is able to take measures to ensure this is the case. From the perspective of the user of the Exploitation Platform a single Data Catalogue end-point is most desirable. The EP web interface can present a consolidated user view in the case of multiple catalogue end-points. A similar consolidation approach can be applied by the EP programmatic API, which can present a single end-point on behalf of the back-end data catalogues.

6.1.4. Federated Discovery

In order that a user is able to discover data/services of interest in a federated network of Exploitation Platforms, an approach to Catalogue federation must be established between collaborating platforms.

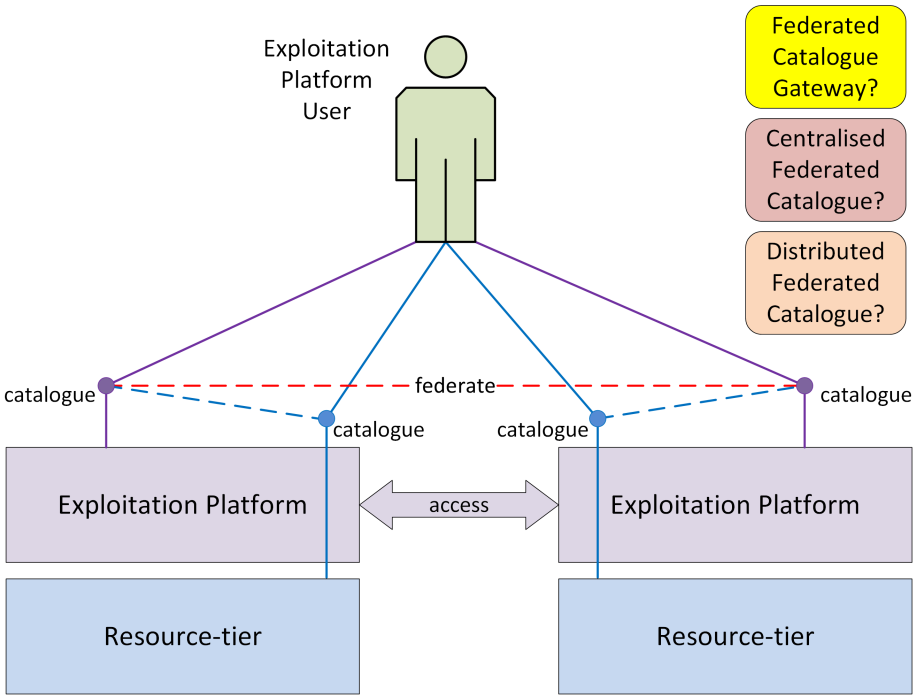

Figure 22. Catalogue Federation

As illustrated in Figure 22 there are a number of possible approaches:

- Gateway – A central proxy
- Centralised – Central mirror
- Distributed - Catalogues mirror each other

Further analysis is required to understand these options, their applicability and impact on the Common Architecture.

6.2. Data Access Services

The Exploitation Platform provides access to data through public services based upon Open Standards, for the consumption by end-users and other federated platforms.

The primary services provided by an Exploitation Platform should include:

- OGC Web Map Service (WMS)
- OGC Web Map Tile Service (WMTS)
- OGC Web Feature Service (WFS)
- OGC Web Coverage Service (WCS)
- OGC Web Coverage Processing Service (WCPS)
- Services provided by Resource Tier:
  - AWS S3 Object Store
  - Swift Object Store (OpenStack)

Other services that may also be considered include:

- WebDAV
- FTP
- CDMI

Integration of these data access services with the data-layer of the hosting Resource Tier relies upon the Data Access Gateway providing an infrastructure agnostic interface for accessing the underlying data holding.

6.3. Data Access Gateway

The EO datasets are stored according to the underlying storage technology of the infrastructure Resource Tier. The storage interface presented is not under the control of the Exploitation Platform.

The role of the Data Access Gateway is to provide an abstraction layer on top of the underlying storage to present a well-defined storage interface to the other components of the Exploitation Platform.

The main EP components that require data access are:

- Processing services and applications: stage-in/out of data/results
- Platform Data Access Services (WMS,WCS,etc.): access to datasets
- Ingestion: storage of ingested data

In the EP system design, these services are designed to be deployed as containers through Kubernetes. This presents the possibility that some aspects of the Data Access Gateway can be met by the facilities offered by Kubernetes volumes. Access to underlying data is provided through volumes that are mounted into the container. Kubernetes volumes have native support for a

number of common storage technologies (such as AWS EBS, Cinder), however these tend to be block rather than object storage.

The Gateway must provide a data bridge between the EP components and the Resource Tier. It fills the gap in the data access capabilities of a given a given service/application, and provides a common data access interface that such components can target in their implementation. We might regard the lowest-common-denominator for data access to be a combination of:

- Local filesystem access
- AWS S3 Object Store

Through docker/kubernetes we can use mounted 'volumes' to present data through a local filesystem interface.
Through s3fs-fuse we can establish local filesystem mount points to S3 object stores.
The Processing Framework makes use of these capabilities to ensure that data is presented to processing services/applications in a form that they can consume.

Thus, the Data Access Gateway presents an S3 interface as an internal data access abstraction, whilst implementing the data access interface to the infrastructure Resource Tier storage.

6.4. Data Access Library (DAL)

In addition to the Data Access Gateway, which operates as an internal service, the Data Access Library (DAL) is provided specifically as a point of integration for processing services and applications. The Data Access Library provides an abstraction of the interface to the data, with bindings for common languages (including python. R, Javascript) and presents a standard programmatic semantic for accessing the data from within the processing service codebase.

The Data Access Library can be seen as a subset of the facilities offered by the Client Library.

The Data Access Library can provide an abstraction at two levels:

Protocol abstraction

Standard programmatic semantics are provided for accessing the data (i.e. CRUD operations on data granules), that is agnostic of the underlying platform storage data access protocols. This is a lower level interface that should be applicable to all use cases.

Data Model abstraction

A common object model is defined with programmatic semantics, which provides a higher-level abstraction of the data that hides the details of the underlying storage, files and file-formats. The abstraction accesses and parses the underlying data to present data structure representations within the language bindings. Such an object model would likely be applicable to some, but not all, use cases. In cases where this approach is not applicable, then protocol abstraction provides the fall-back option.

Thus, processing services and applications can be implemented in a ‘portable’ way that is agnostic to the platform resource-tier storage technology.

Specific implementations of the DAL can be made to abstract the data access layer for a given

Exploitation Platform. The library offered to the processing service at runtime must implement the specific data access interface to the resource-tier storage. Hence, the library should not be 'hard-coded' into the processor application package (Docker image). The Processing Framework must support the ability to 'plugin' an alternative (platform-specific) implementation of the DAL dynamically at processor execution time. It may be possible to develop a 'generic' Data Access Library by implementation against the standard (internal) interface provided by the Data Access Gateway. In this case, the platform-specifics regarding data access are borne entirely by the Data Access Gateway.

See also section Processing Service Data Access which provides a discussion of data access approaches for processing services and the stage-in/out of data.

6.5. Data Ingestion

Data Ingestion presents a standard interface to the EP components, whilst transparently interfacing with the infrastructure Resource Tier.

During data ingestion the following steps may be performed:

- Authorization check
- Quota check
- Metadata extraction
- Preview generation
- Format conversion
- Storage PUT
- Catalogue PUT
- Trigger notifications

Ingestion raises notifications for the following events:

- Raise indicators to users (visual, emails, etc.)
- Trigger systematic actions in other EP services (e.g. systematic processing)

6.6. Workspace

The Workspace provides a service to users through which they can organise data/processing-services that are of current interest to them, they are currently working on, and to organise results of processing executed, Research Objects, etc.

This concept can be extended to create a Group Workspace for sharing and collaboration.

It may be possible to model the Workspace as a Catalogue, in which the browse/discover access privilege is limited to either an individual user (personal workspace) or a group of collaborating users (group workspace):

- READ access: OpenSearch should provide a good fit for this interface

- CREATE/UPDATE/DELETE: Transactional extension to OGC-CSW (to be explored)

Chapter 7. Platform API

The Platform API defines standard interfaces at both service and programmatic levels, with the goal of encouraging interoperation between platforms and providing a consistent and portable programming paradigm for expert users. The Service API and its associated Client Library together present a standard platform interface against which analysis and exploitation activities may be developed, and through which platform services can be federated.

The Service API represents the public service interfaces exposed by the Exploitation Platform for consumption by its clients. Covering all aspects of the EP (authentication, data/processing discovery, processing etc.), these interfaces are based upon open standards and are designed to offer a consistent EP service access semantic within the network of EO resources. Use of the network (HTTP) interfaces of the Service API is facilitated by the Client Library that provides bindings for common languages (Python, R, Javascript). The Client Library is a programmatic representation of the Service API which acts as an abstraction of the Exploitation Platform and so facilitates the development of portable client implementations.

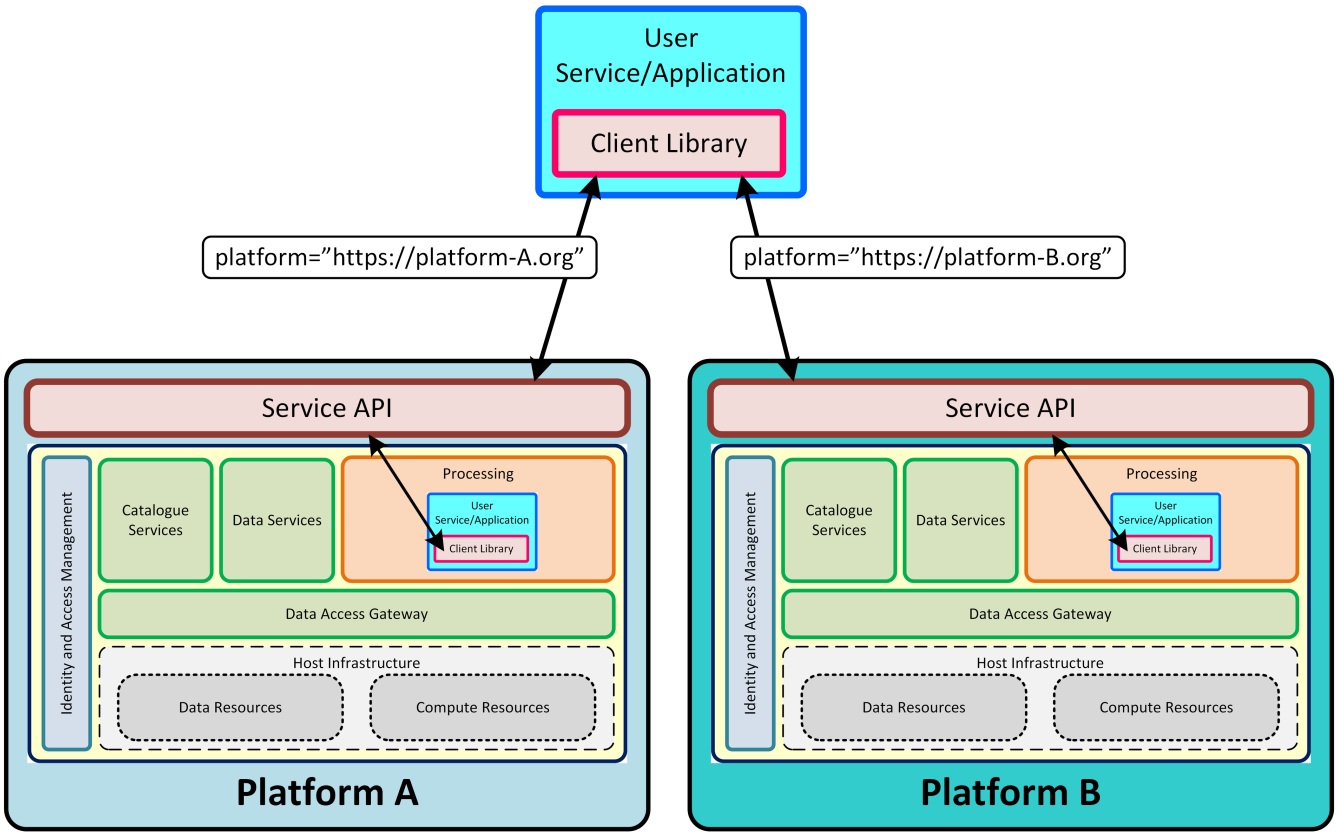

Figure 23. Client Portability

As illustrated in Figure 23, code implemented against the Client Library is not tied to a particular Exploitation Platform, but instead can be initialised and executed against any EP that supports the corresponding Service API. The User Service/Application shown in the figure can either be a process running external to the platform (e.g. on the users local platform), or running as a deployed process within the Processing Framework of the platform. **It should be noted that the use of the Client Library is not mandatory - instead the application can be developed against the Service API directly.**

7.1. Service API

The Service API presented by the Exploitation Platform is largely defined and met by the fundamental data/processing services it offers. There are some additional meta-services that support the clients in the discovery and usage of these core services. The Service API defined in this section seeks to define a standard set of interfaces against which the Client Library can be developed and that can be relied upon for platform-to-platform interoperability.

7.1.1. Platform Capabilities

Provides a single well-known 'bootstrap' URL through which the service capabilities and their endpoints can be discovered:

- Declare platform capabilities
- Discover service endpoints
- Core platform data/processing services
- Additional meta-services to support clients
- Used by Client Library to initialise its platform interfaces
- Used by EMS to match service/application dependencies with platform capabilities
- Used by other platforms to establish interoperability

/.well-known/eoepca-platform

Example Platform Capabilities

```
{
    "services": [
        { "type": "oidc", "role": "authentication", "path": "/connect" },
        { "type": "<tbd>", "role": "billing", "path": "/billing" },
        { "type": "opensearch", "role": "data_search", "path": "/search" },
        { "type": "csw", "role": "data_catalogue", "path": "/catalogue" },
        { "type": "opensearch", "role": "app_search", "path": "/applications" },
        { "type": "wms", "role": "map", "path": "/map" },
        { "type": "wmts", "role": "tile", "path": "/tile" },
        { "type": "wfs", "role": "feature", "path": "/feature" },
        { "type": "wcs", "role": "coverage", "path": "/coverage" },
        { "type": "wcps", "role": "datacube", "path": "/datacube" },
        { "type": "s3", "role": "object_store", "path": "/storage" },
        { "type": "wps-t", "role": "ems", "path": "/ems" },
        { "type": "wps-t", "role": "ades", "path": "/ades" },
        { "type": "<tbd>", "role": "workspace", "path": "/workspace" },
    ],
    "extended_capabilities": [
        { "type": "dask", "role": "cluster", "details": {} },
    ]
}
```

The endpoints referenced in eoepca-platform are described in the following sections.

7.1.2. authentication

The URL of the OpenID Connect Provider that implements the Login Service for user authentication.

/connect (example)

Implements the OpenID Connect protocol as described in [OIDC].

7.1.3. billing

The URL of the endpoint of the Billing service in the platform.

/billing (example)

The Billing service provides a cenrtralised point of contact within the platform responsible for tracking and billing for usage of resources and services. The approach to billing is currently not defined, but will be documented in section Accounting and Billing.

7.1.4. data_search

The URL of the OpenSearch interface to the Data Catalogue.

/search (example)

Implements an OpenSearch interface in accordance with section Resource Catalogue.

7.1.5. data_catalogue

The URL of the OGC CSW (Catalogue Services for the Web) interface to the Data Catalogue.

/catalogue (example)

Implements an OGC CSW catalogue in accordance with standard **OGC Catalogue Services 3.0 Specification - HTTP Protocol Binding** as defined in [OGC-CSW].

7.1.6. app_search

The URL of the OpenSearch interface to the Application Catalogue.

/applications (example)

Implements an OpenSearch interface in accordance with section Application Catalogue.

7.1.7. map

The URL of the OGC WMS (Web Map Service) interface that supports the data maintained in the Data Catalogue.

/**map** (example)

Implements an OGC WMS service in accordance with standard **OGC Web Map Server Implementation Specification** as defined in [OGC-WMS].

7.1.8. tile

The URL of the OGC WMTS (Web Map Tile Service) interface that supports the data maintained in the Data Catalogue.

/**tile** (example)

Implements an OGC WMTS service in accordance with standard **OGC Web Map Tile Service Implementation Standard** as defined in [OGC-WMTS].

7.1.9. feature

The URL of the OGC WFS (Web Feature Service) interface that provides a *Feature-oriented* access to the underlying data holding of the platform.

/**feature** (example)

Implements an OGC WFS service in accordance with standard **OGC Web Feature Service 2.0 Interface Standard – With Corrigendum** as defined in [OGC-WFS].

7.1.10. coverage

The URL of the OGC WCS (Web Coverage Service) interface that provides a *Coverage-oriented* access to the underlying data holding of the platform.

/**coverage** (example)

Implements an OGC WCS service in accordance with standard **OGC Web Coverage Service (WCS) 2.1 Interface Standard - Core** as defined in [OGC-WCS].

7.1.11. datacube

The URL of the OGC WCPS (Web Coverage Processing Service) interface that provides a queryable 'data cube' interface to multi-dimensional coverage data.

/**datacube** (example)

Implements an OGC WCPS service in accordance with standard **Web Coverage Processing Service (WCPS) Language Interface Standard** as defined in [OGC-WCPS].

7.1.12. object_store

The URL of the *Amazon S3* interface that provides object storage access to the underlying data holding of the platform.

/**object_store** (example)

Implements an Amazon S3 service in accordance with **Amazon Simple Storage Service REST API** as defined in [AWS-S3].

Amazon S3 Compatibility Subset

For the purposes of this interface, a subset of the full Amazon S3 REST API will be defined as mandatory. The motivation is to define a consistent interface that is supported by third-party object storage implementations that provide an S3-compatible API, such as:

- Ceph (https://ceph.com/ceph-storage/)
  - CEPH OBJECT GATEWAY S3 API (http://docs.ceph.com/docs/mimic/radosgw/s3/)
- OpenStack Swift (https://docs.openstack.org/swift)
  - S3/Swift REST API Comparison Matrix (https://docs.openstack.org/swift/latest/s3_compat.html)

7.1.13. ems

The URL of the Execution Management Service service.

/ems (example)

Implements an **EMS service** as described in section Execution Management Service.

7.1.14. ades

The URL of the Application Deployment and Execution Service service.

/ades (example)

Implements an **ADES service** as described in section Application Deployment and Execution Service.

7.1.15. workspace

The URL of the 'user workspace' service as described in section Workspace.

/workspace (example)

The interface for the workspace is currently TBD.

7.2. Client Library

The Service API and its associated Client Library together present a standard platform interface against which analysis and exploitation activities may be developed, and through which platform services can be federated.

The Client Library is a programmatic representation of the Service API which acts as an abstraction

of the Exploitation Platform and so facilitates the development of portable client implementations.

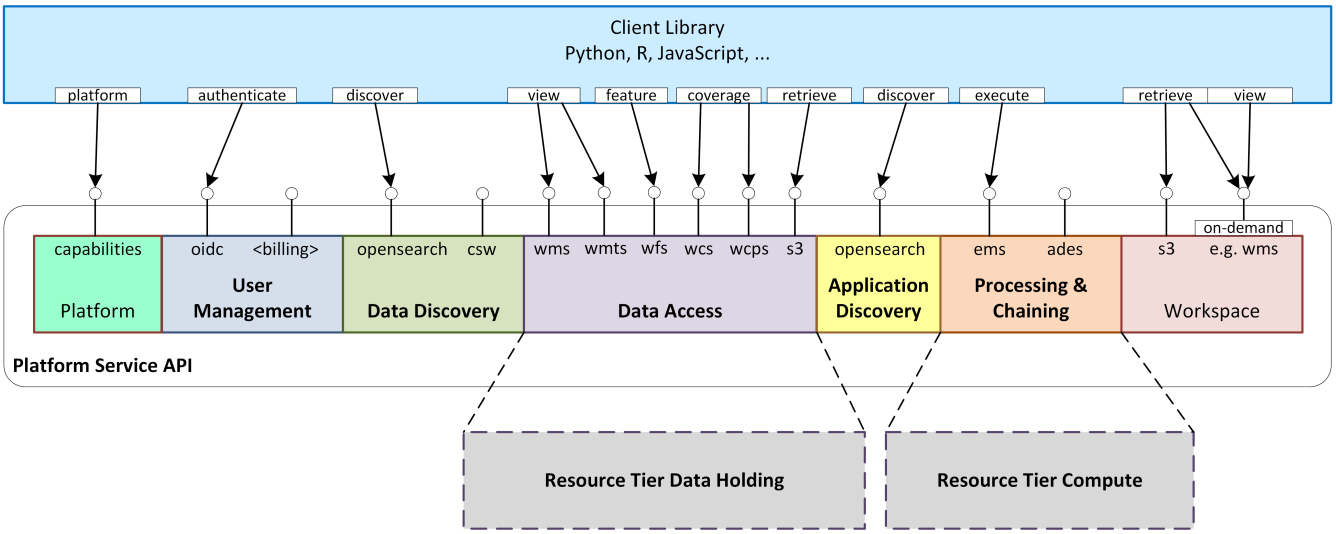

Figure 24. Client Library Service Interfaces

As illustrated in Figure 24, the Client Library provides bindings for common languages (Python, R, Javascript) that utilise the network (HTTP) interfaces of the Service API, covering all aspects of the Exploitation Platform functionality (authentication, data/processing discovery, processing etc.).

7.2.1. Client Library Concept Illustration

The design of the Client Library is not yet established. To illustrate its concept we present an example based upon the workflow scenario that was used to demonstrate the EMS/ADES best practice in OGC Testbed-14 ([TB14-ADES]). The scenario is shown in Figure 25.

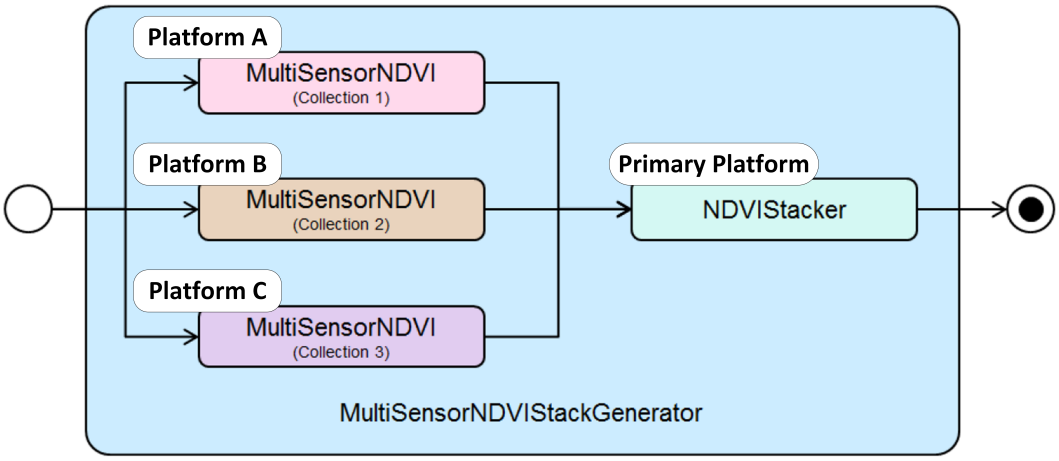

Figure 25. OGC Testbed-14 Workflow Scenario (NDVI Stacker)

Four platforms are involved in the scenario:

Platforms A/B/C

These platforms perform the local execution (ADES) of the MultiSensorNDVI processor on a collection in their local data holding. These executions are managed under the orchestration (EMS) of the Primary Platform.

Primary Platform

The Primary Platform is in charge of the workflow. Thus, the orchestration (EMS) is conducted from this platform, which interfaces to the 'subordinate' platforms (A/B/C) for the execution of the steps 'close-to-the-data', and then completes the workflow by execution (ADES) of the NDVIStacker processor to produce the final result.

In response to this scenario we might envisage a client application implemented against the Client Library as follows...

Example python program using Client Library

```
import eoepca

# Connect to primary platform - e.g. when running on own desktop
primaryPlatform = eoepca.platform("http://primary.platform.eo") ①
primaryPlatform = primaryPlatform.authenticate("bob@home.org", "<<MY-API-KEY>>") ②
# Or, use the 'local' hosting platform - e.g. when running in 'cloud' platform
primaryPlatform = eoepca.platform().authenticate("bob@home.org", "<<MY-API-KEY>>") ②
③

# Init supporting platforms
platA = eoepca.platform("http://platform-a.eo") ① ④
platB = eoepca.platform("http://platform-b.eo") ① ④
platC = eoepca.platform("http://platform-c.eo") ① ④

# Specify extent
extent = { "bbox": [ -0.489, 51.28, 0.236, 51.686 ], "time": [ "2018-01-01", "2018-12-
31" ] }

# Specify processes
coverage1 = platA.collection("PLAT_A_DATA").coverage(extent) ⑤
proc1 = coverage1.process("MultiSensorNDVI") ⑥
proc2 = platB.collection("PLAT_B_DATA").coverage(extent).process("MultiSensorNDVI") ⑤
⑥
proc3 = platC.collection("PLAT_C_DATA").coverage(extent).process("MultiSensorNDVI") ⑤
⑥

# Specify workflow
workflow = primaryPlatform.parallel([proc1, proc2, proc3]).process("NDVIStacker") ⑦

# Get result - initiates 'lazy' execution
result = workflow.retrieve(format="geotiff", options={}) ⑧
print(result)
```

① When each platform object is initialised, its endpoint `${platform-url}/.well-known/eoepca-platform` is interrogated, to understand its capabilities and learn its service endpoints.

② The user must authenticate to the primary platform (that 'executes' the workflow), using their API key.

③ In the case where the client is **running on the primary platform** then the platform URL is not

required in the initialisation (implying 'local' platform). Examples of this case include: code running in a hosted Jupyter notebook, or a deployed processing service that chooses to use the Client Library.

④ Subordinate platforms are initialised (for capabilities) without authentication, on the basis that the primary platform authentication can be carried through the workflow 'call-stack' through delegated/federated IAM solution - ref. section Identity and Access Management.

⑤ At each platform the collection is selected through its unique collection identifier, and a data coverage subset is specified through a definition of the required spatial/temporal extent.

⑥ At each platform the processing task to be exectued against the selected coverage is specified.

⑦ The workflow is defined by requesting the parallel execution of the `MultiSensorNDVI` processor on each of the three platforms, with these results providing input to the `NDVIStacker` process executed on the primary platform.

⑧ The call to `retrieve` the outcome of the workflow initiates its **'lazy' execution**. Prior to this point the Client Library has been operating on **'proxy'** objects that record the **specification of the workflow** requested by the code. At this point the Client Library converts the workflow specification into a CWL definition suitable for deployment and execution at the EMS of the primary platform. In response the EMS will interface with the ADES of each subordinate platform to ensure the `MultiSensorNDVI` is deployed and executed against the requested coverage. Subsequently the EMS interfaces with the ADES of the local platform to ensure the `NDVIStacker` is deployed and executed against the outputs of the three `MultiSensorNDVI` process executions.

Chapter 8. Web Portal

The Web Portal represents the browser-based user interface through which the user interacts with the EO Exploitation Platform.

The Web Portal is not a domain area in its own right – it is contributed to by the collaborative developments of the defined domain areas. The user's view of the platform is consolidated through the browser-based user interface, and hence it is convenient to present all aspects of this view together. Thus, the Web Portal provides the user facing 'front' of the system and interfaces to the services provided by the domain areas identified in the system design.

The Web Portal must provide a consistent and cohesive user experience that aggregates data and processing services of the EO Exploitation Platform. In doing so it must provide the following main functionalities:

- User login
- Marketplace, that provides a federated system search for discovery of data and processing capabilities
- User workspace to support scientific analysis and collaboration
- Data discovery and download of data
- In-browser visualisation of data and processing/analysis results
- Discovery, execution and monitoring of processing jobs
- Definition of workflows from discovered data/processing resources
- Hosting of user defined applications with interactive user interfaces
- Hosting of rich media content that is linked to catalogued resources. Such content ranges from documents & manuals to tutorials and instructional media.
- Hosting of community and collaboration tools such as Wikis, FAQs and forums

These user-facing web components form part of other domain areas that together present a rich integrated user experience. Figure 26 presents these functional areas, organised within their respective domain areas.

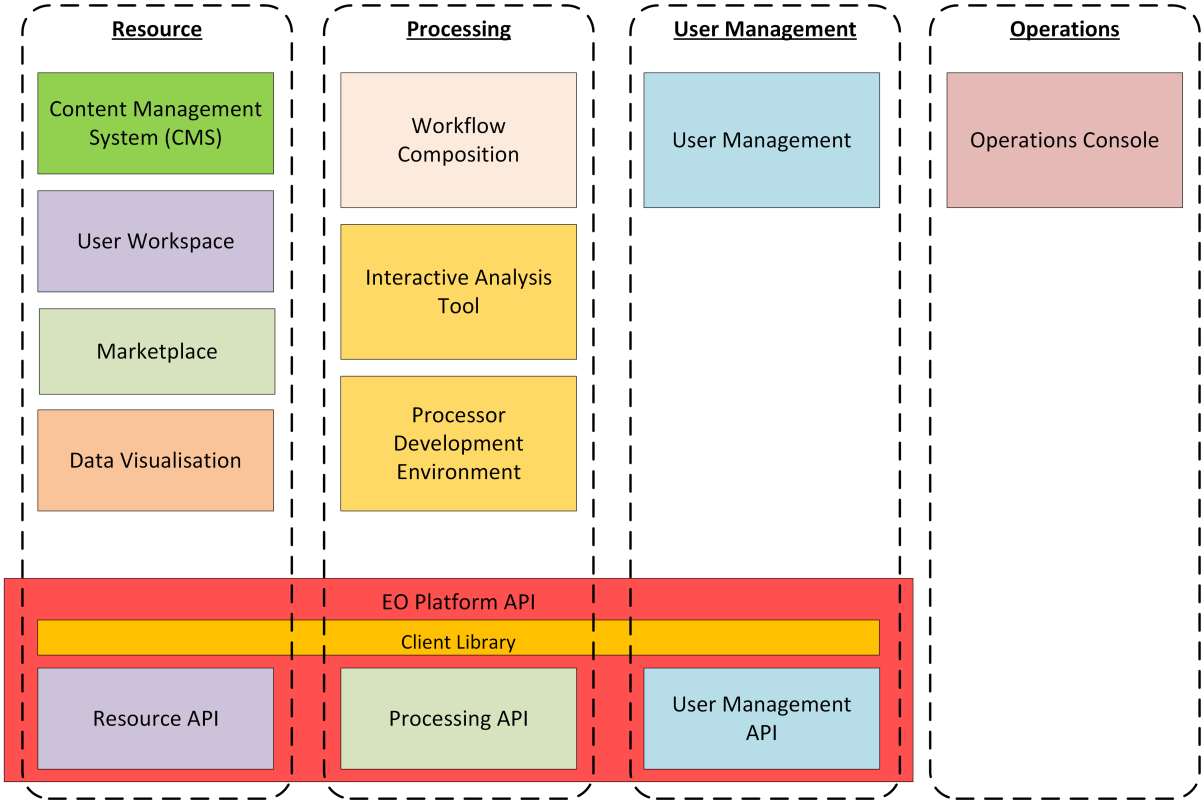

Figure 26. Web Portal: Overview

The platform should support provision of limited access to unauthenticated (guest) users, in which they can search the marketplace to discover the services and data available, and browse supporting materials. Access to the full capabilities of the platform requires registered users to identify and authenticate.

Optionally, a **Content Management System (CMS)** can provide a framework within which the platform's web presence is hosted. It facilitates the creation of user content that can be linked to data and processing resources in the Resource Catalogue. In addition, the CMS provides out-of-the-box facilities for Wikis, FAQs, forums etc.

The **Marketplace** builds a user experience on top of the Application & Data Catalogues that provides a consolidated inventory of all services, applications and data published within the federated system. The user is presented with the ability to browse and to perform rich search queries to discover items of specific interest. The Marketplace content for a data item can include interactive Data Visualisation, such as providing a WMS viewer that exploits the WMS service provided with the platform's resource service. This **Data Visualisation** component is re-usable such that it can be used elsewhere in the user experience, for example from the user's workspace to visualise some processing results.

The **User Workspace** provides the environment where users are able to organise data and processing they are interested in, and to manage asynchronous 'tasks' they have submitted into the platform. Thus, they are able to monitor data retrieval and processing requests and obtain the outputs at completion. The facility is also provided for them to publish derived 'added-value' outcomes from their workspace into the Resource Catalogue, and so present them in the marketplace.

Experts use the **Workflow Composition** interface to chain and combine multiple processing functions and input data into reusable workflows. The interface allows them to select these

resources discovered via the Marketplace, architect and execute their workflow, and ultimately publish it as a reusable processing function that is available to others in the Marketplace.

Experts are provided with an **Interactive Analysis Tool** that presents a hosted coding environment through which they can interact directly with the data and services of the platform. Additionally, Experts are able to develop and submit to the EO Exploitation Platform their own custom processing algorithms, tools and applications. The **Processor Development Environment** provides a rich, interactive environment in which processing algorithms and services can be developed, tested, debugged and ultimately packaged so that they can be deployed to the platform and published via the marketplace.

User Management provides the functionality associated with user profiles. New users will have the ability to self-register and then manage all aspects of their profile interactively - noting that the intention in the Common Architecture is to delegate User Identity management to external IdPs.

Operators will have access to management interfaces for system monitoring and administration.

<< End of Document >>