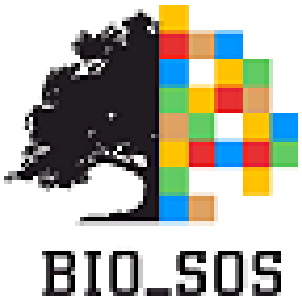


Project Title:	**BIO_SOS Biodiversity Multisource Monitoring System: from Space TO Species**
Contract No:	FP7-SPA-2010-1-263435
Instrument:	Collaborative Project
Thematic Priority:	FP7-SPACE-2010-1
Start of project:	1 December 2010
Duration:	36 months

Deliverable No: D3.2

Architecture Design Document

Due date of deliverable:	31/07/2012
Actual submission date:	03/12/2012

Version:1st version of D3.1

Main Authors: Jens Stutte (PKI), Dimitrios Karachalios (PKH).

and contribution by: Richard Lucas, Pete Bounting (P11), Jordi Inglada (P16), Vasiliki Kosmidou (P2), Maria Adamo (P1), Palma Blonda (P1)

Project ref. number	**263435**
Project title	**BIO_SOS: Biodiversity Multisource Monitoring System: from Space to Species**

Deliverable title	ADD – Architecture Design Document
iDeliverable number	TBC
Deliverable version	v1
Previous version(s)	
Contractual date of delivery	31/07/2012
Actual date of delivery	03/12/2012
Deliverable filename	pkt291-20-1.1_D3.2_ADD_Architecture_Design_Document_v1
Nature of deliverable	R = Report
Dissemination level	PU = Public
Number of pages	62
Workpackage	WP 3
Partner responsible	PKI
Author(s)	Jens Stutte (PKI)
	Diomede Illuzzi (PKI)
	Dimitrios Karachalios (PKH)
Editor	Jens Stutte (PKI)
EC Project Officer	Florence Beroud

Abstract	The ADD describes the architectural concept of the system. It will be updated over time
Keywords	System, architecture, workflow

Signatures

Written by	**Responsibility- Company**	**Date**	**Signature**
Jens Stutte	WP 3 Leader (PKI)	29/11/2012	
Dimitrios Karachalios	Development (PKH)	03/12/2012	
Verified by			
Jens Stutte	WP 3 Leader (PKI)	03/12/2012	
Jordi Inglada	Involved in WP5 (UPS)	30/11/2012	
Approved by			
Palma Blonda	Project Coordinator, CNR	03/12/2012	
Fifamè Koudogbo	Quality Team, AI	03/12/2012	

Table of Contents

1. Executive summary

The EODHaM system is an operational prototype of an ecological modelling system for effectively and multi-annual monitoring of NATURA 2000 sites (and other ecologically sensitive sites) and their surrounding areas.

This Architecture Design Document (ADD) has the aim of describing the architectural issues of the EODHaM system, taking in account different viewpoints.

In particular:

1. to define the use cases modeled by the system
2. to provide a system architecture
3. to provide a generalized processor wrapper design

The general design approach is characterized by the modularization into components and the separation of different levels of abstraction for the use of the processors. This is driven by the defined use cases and their different actors, in particular the distinction between the Chain Manager and the Processor Provider, which have to act at different levels of both defining and using the system.

The workflows are defined within and executed by a BPEL engine that engages the single processors through a standardized SOAP interface, which enables a loose and remote coupling of the system's components.

2. Introduction

2.1 General Context

The main objective of BIO_SOS is the development of an operational ecological modeling system suitable for effective and timely multi-annual monitoring of NATURA 2000 sites and their surroundings in areas particularly exposed to different and combined types of pressure.

Due to several national and regional differences in policies/ funding and the lack of a centralised management of biodiversity data, even at the same regional-local level, a noticeable effort is required in order for a continuous, operational and quasi real-time monitoring of ecologically sensitive areas to be initiated. In search of such a monitoring system, the starting point should be to know the main 'actors' requirements and expectations. According to them, it is expected that such a system should:

- function at fine spatial scales (1:5,000 or finer) where habitats ought to be represented,
- be user-oriented efficient and reliable,
- be sensitive to changes in the input datasets and the user-defined parameters
- minimize the time between data acquisition and product delivery
- minimize the involved costs (e.g., by reducing manpower, exploiting open source software solutions, etc.)

Related to the aforementioned pre-operational system requirements, BIO_SOS project will propose and develop the so-called three-stage EODHaM system, which intends to be a pre-operational ecological modelling system for effectively and multi-annual monitoring of NATURA 2000 sites (and other ecologically sensitive areas) and their surrounding areas.

2.2 A word on terms, assumptions and choices

2.2.1 Prior knowledge, ancillary data and metadata

The DOW and other project related documents refer as input to several processing modules “prior knowledge”, “pre-existing data” and “ancillary data”. We assume that:

1. “prior knowledge” is information or rules that are incorporated into the system, thus they do not adapt to data and do not change with time. Prior knowledge is acquired by a supervisor or expert (human) based on background domain knowledge, expertise and evidence from data observation before (prior to) the data processing system starts looking at the remote sensing data available. In other words, prior knowledge is acquired by the supervisor and, next, taught by the supervisor to a deductive expert system (equivalent to a deductive machine teaching-by-rule paradigm) rather than being learned from data by an inductive information processing system (according to an inductive machine learning-from-data paradigm). Thus “prior knowledge” is not part of any input data/information flows that we describe in this document.
2. ''pre-existing data” and “ancillary data” are synonyms for data not directly derived from the main EO data input, but that may change from one processing to the other and thus must be considered in our workflow design. We use here the term “pre-existing data”.
3. “Metadata” is intended here as the set of parameters and fields that describe the content of Earth Observation or ancillary data in a ''searchable'' way. WP 4 and in particular D4.1 and D4.5, which are due to be delivered in the following months, will provide the guidelines for the metadata collection, harmonization and availability to the system.

2.2.2 Partner responsibility referred to SW development

Whenever, in this document, we indicate partners as responsible for a module, only the directly responsible partners for the development of the SW itself are cited (as kind of contact point for the system WP). This does not exclude that other partners will also contribute to such modules with their research or other activities.

2.2.3 Quality Assurance Framework for Earth Observation (QA4EO)

The international Quality Assurance Framework for Earth Observation (QA4EO), led by the Committee of Earth Observations (CEOS) Working Group on Calibration and Validation (WGCV) considers mandatory:

1. An appropriate coordinated program of calibration and validation (Cal/Val) activities throughout all stages of a spaceborne mission, from sensor building to end-of-life. This ensures the harmonization and interoperability of multi-sources observational data and derived products.
2. Metrological / statistically-based quality indicators (QIs), provided with a degree of uncertainty in measurements, to be established for all sensor derived data products

3. EODHaM System Definition

The EODHaM system is defined as a complete automatic system of work-flow administration. The system will serve in efficiency the aim of the project for a multi-annual monitoring of NATURA 2000 site and their surrounding areas, as other ecologically sensitive sites. The EODHaM system will use different technologies, all well-structured between them, with final aim the processing, production, cataloguing and distribution of the BIO_SOS products.

3.1 EODHaM introduction

The EODHaM system will be composed by different distinct BIO_SOS processing modules and shall aim at the coordinated execution of the individual tasks and activities, with organized method having as final result the collaborative processing and BIO_SOS product's production.

Each BIO_SOS processor module forms a distinct unit of the EODHaM system with instruction of executions as derived from the built processing algorithm and shall be exposed to the system as web-based services of those structured activities.

The EODHaM system shall be built with a Service-Oriented Architecture (SOA). The Service Oriented Architecture allows to develop the system in the form of interoperable web-based services. SOA defines how to integrate widely disparate services in a Web-based environment and defines the interface in terms of protocols and functionality.

For that purpose the EODHaM system shall make use of the *Web Services Business Process Execution Language* (WS-BPEL), a standard executable XML-based language for specifying actions within business processes with web service. The WS-BPEL serves also as standardization of interaction between business processes in a distributed environment. A BPEL business process, for the exportation and importation of information, uses exclusively web service interfaces.

An EODHaM business process is a collection of related, structured activities or tasks in a sequence order of execution that produce a specific service or product.

An EODHaM processor module, from the system point of view, is a unit of instruction execution, and is the building block of a *process*. Each processor *activity* operates on a piece of shared data (context) to fulfill part of the overall goal of the *process*.

The EODHaM Business Process Work-flow Management module shall be able to orchestrate the BIO_SOS processors in a full automatic business process in terms of interoperability, tasks coordination, and automatic processes execution. The business process shall be configurable, extensible and easy to manage and maintain.

The execution of the EODHaM Business Process Work-flow shall permit the interaction of all processor units and their related structured activities, with the objective of realising the BIO_SOS items production, publication, cataloging and dispersion including the relative metadata of the products.

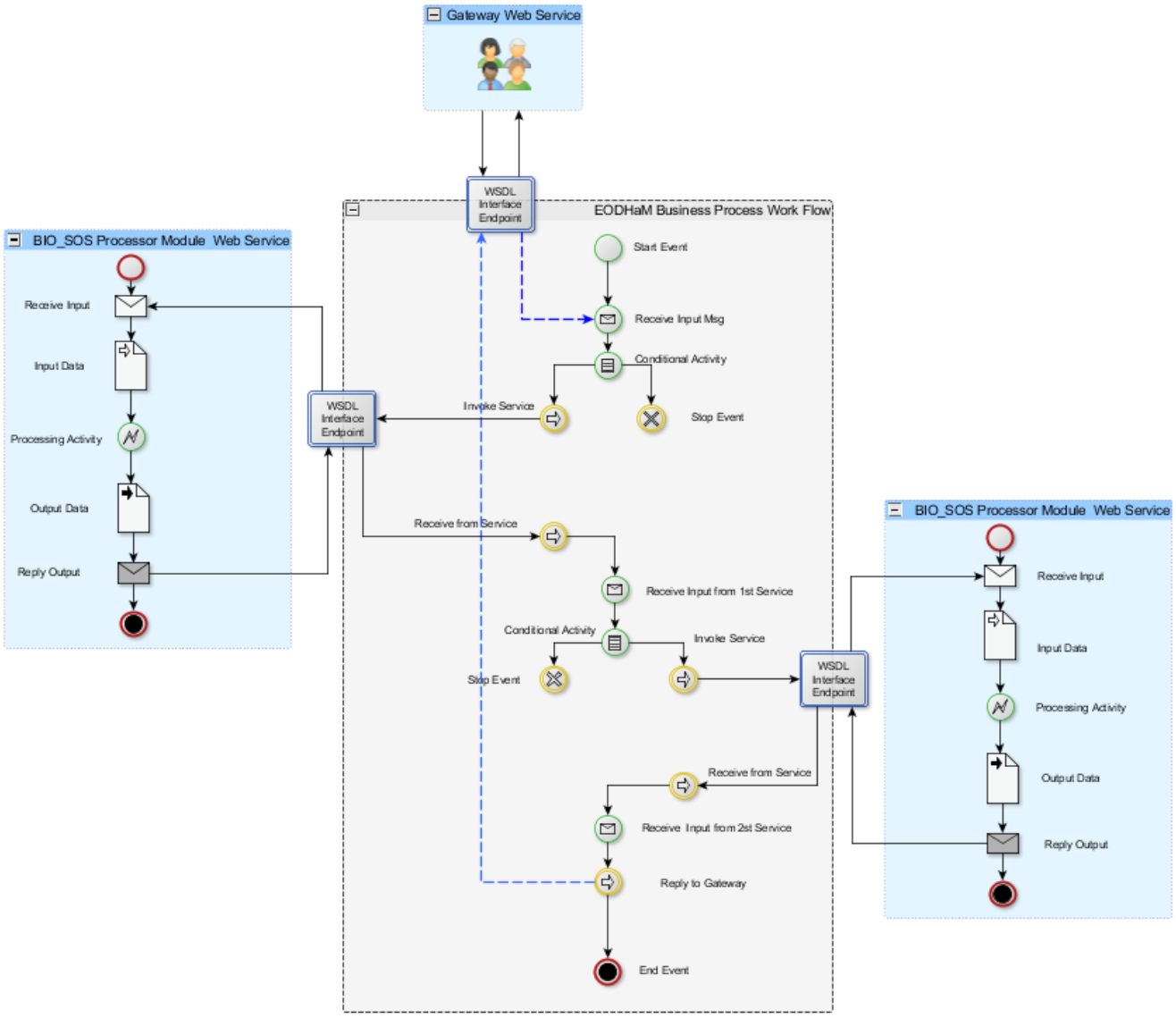


Figure 1: Generic schema of Business Process Work – Flow concept.

In the generic schema, we can observe the principal meanings of a BPEL Business Process work-flow. The EODHaM Business Process Work Flow is the “logical container” of the work-flow, where an orchestration of sequence activities is made, by performing the process interaction and communication between web services.

Every web service that is involved with the interaction of the EODHaM Business Process has to have a WSDL description file that describes how the EODHaM system can communicate with this web service.

3.2 EODHaM component units as Web services

In the EODHaM system each component (e.g., a BIO_SOS processor module) maintains the integrity of the internal and autonomous functionality and represents a software application, which when it is invoked, executes an internal processing of the input data given and produces a specific output according on criteria-based on the type of the input data and the executable processing instructions that were applied.

All the EODHaM software applications must be developed as Web Services. W3C defines a Web Service as a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other

systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

Web services can have the ability to be synchronous or asynchronous. Synchronicity refers to the binding of the client to the execution of the service. In synchronous invocations, the client blocks and waits for the service to complete its operation before continuing. Asynchronous operations allow a client to invoke a service and then execute other functions. Asynchronous clients retrieve their result at a later point in time, while synchronous clients receive their result when the service has completed. Asynchronous capability is a key factor in enabling loosely coupled systems. The EODHaM system and the compound web services shall provide Asynchronous operations.

3.2.1 EODHaM types of web services

In the EODHaM system, we can recognize three principal types of web service.

3.2.1.1 BIO_SOS pre-processing chain web service

It consists in a group of activities that usually take place at the beginning of the execution of a business process. It groups structural activities of data pre- processing activities that shall apply on the input data and shall provide capable data for further processing activities.

Such activities are the Geocoding Orthorectification, the Absolute Calibration (TOA Radiance), the TOARD to TOA Reflectance and Atmospheric Correction (TOARF to Surface RF).

Those pre-processing activities are considered as preparative steps for the sequent BIO_SOS processing activities.

3.2.1.2 BIO_SOS Processing web service

It refers to each BIO_SOS processing module, exposed to the EODHaM system as web service. Each web service BIO_SOS processing has to have unique name and end-point of the exposed service. The interface development, in terms of WSDL and the type of SOAP messages, must be conform with the EODHaM standardized interface as will be defined in the following chapters. The web service-Oriented development of each BIO_SOS processing module with the EODHaM standard interface is crucial requirement for a homogeneous orchestration of business process. Further explanation of the system requirements will be described in the following chapters.

3.2.1.3 BIO_SOS Metadata Catalog web service:

The web service BIO_SOS metadata catalog shall allow the BIO_SOS product registration and messages interchange between the BIO_SOS processing web services and the catalog itself during business process execution.

3.2.2 EODHaM system and Business Process Execution Language concepts

For a better conceptual definition of the EODHaM system, an introduction to the most significant BPEL concepts is necessary (Figure 2).

All the options offered by these attributes and elements are not discussed here; more information on these points can be found in the BPEL specification.

WS-BPEL provides a language for the specification of executable business processes. This language is an XML based programming language to describe high level business processes. More detailed, a BPEL *process* declares a number of *partners* to communicate with, an *activity* which defines the logic behind the interaction with its partners and *variables* that are required to model a state full interaction. Activities are the building blocks of the behavior of a BPEL process. They are used to describe control flow and data flow. An activity may be either basic or structured.

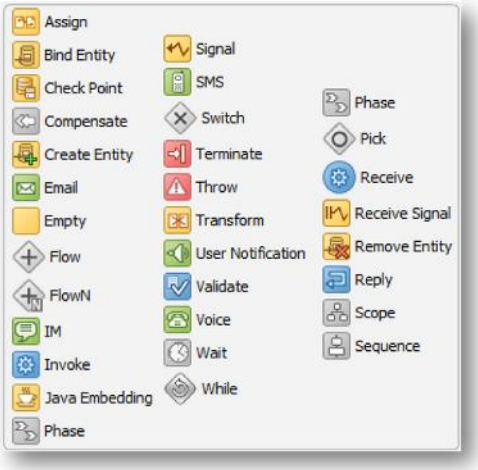


Figure 2: BPEL activities

Partners

An important part of the use of BPEL is the description of business process interaction between partners communicating with Web Services. To connect these partners, it is important that relationships can be specified between these partners. Every partner that is involved with the interaction of the BPEL process should have a WSDL description file that describes how EODHaM system can communicate with this partner.

In the EODHaM system each BIO_SOS processor web service module is considered as Partner and is exposed to the business process as PartnerLink.

Each BIO_SOS processor web service will implement the same common interface WSDL. That will create standardization of the used interfaces, and will facilitate the implementation and orchestration of a BPEL business process. Having common WSDL means that each BIO_SOS processor web service follows the same communication model with the system, offering homogeneity on the expected behavior of the interchangeable messages.

PartnerLink

Description: In BPEL a Web Service that is involved in the process is always modeled as a *partnerLink*. Every partnerLink is characterized by a *partnerLinkType* which is defined in the WSDL definition. A *partnerLinkType* specifies the role and the type of a partner.

Every partnerLink has to have a unique name that can be used to identify the *partnerLink*. The role of the process is specified by the attribute *myRole* and the role of the partner is specified by the attribute *partnerRole*. When a *partnerLinkType* is used with only one role, then one of these attributes can be discarded.

Code example: <partnerLinks>

```
  <partnerLink name="ncname" partnerLinkType="qname"
        myrole="ncname" partnerRole="ncname">
  </partnerLink>
</partnerLinks>
```

Data manipulation

Description: In BPEL it is possible to use variables. A variable is always connected to a message from a WSDL. When the message in a WSDL file is structured like the message in following code example:

Code example:

```
<message name="creditInfo">
  <part name="firstname" type="xsd:string"/>
  <part name="surname" type="xsd:string"/>
  <part name="credit" type="xsd:string"/>
</message>
```

Then this message can be used in BPEL in the following way:

```
<variable name="creditInformation" messageType="creditInfo"/>
```

Variables

Description: Variables offer the possibility to store messages that hold the state of the process. The messages that get stored are most of the time either coming from partners or going to partners. Variables also offer the possibility to store data that is only state based and never send to partners.

Code example:

```
<variables>
  <variable name="ncname"
        messageType="qname"
        type="qname"
        element="qname"/>
<variables>
```

Assignment

Description: Copying the data from one variable to the other is something that will happen very often in a business process. Copying data can be achieved with the *assign* activity. This activity can also be used to copy new data into a variable.

Code example:

```
<assign>
  <copy>
    <from variable="ncname" part="ncname"/>
    <to variable="ncname" part="ncname"/>
  </copy>
</assign>
```

Basic activities

Every basic activity has several standard attributes and elements than can be used to specify certain properties.

Invoke

Description: With an *invoke* activity a process can call another Web Service that has been defined as a partner. The *invoke* can be either asynchronous or synchronous.

Code example:
```
<invoke partnerLink="ncname"
        portType="qname"
        operation="ncname"
        inputVariable="ncname"
        outputVariable="ncname">
</invoke>
```

Receive

Description: A business process offers services to its partners by *receive* and matching *reply* activities. A *receive* activity specifies a *partnerLink*, a *portType* and an *operation* that can be invoked. There is also an attribute in which a variable can be specified. The *receive* activity plays an important role in the life cycle of a business process. One of the ways to initiate a process is by way of a *receive* activity with the attribute *createInstance* set to *yes*.

Code example:
```
<receive partnerLink="ncname"
         portType="qname"
         operation="ncname"
         variable="ncname"
         createInstance="yes|no">
</receive>
```

Reply

Description: A *reply* activity is used for sending a response after a *receive* activity has been called. The *reply* is only useful in a synchronous interaction. A asynchronous reaction always has to be given by use of an *invoke*. With the *reply* activity it's also possible to transfer data by specifying a variable. A *reply* activity can only be placed after a *receive* or a *onMessage* activity with the same *partnerLink*, *portType* and *operation*.

Code example:
```
<reply partnerLink="ncname"
       portType="qname"
       operation="ncname"
       variable="ncname"
       faultName="qname">
</reply>
```

Signaling faults

Description: When the process wants to report an internal fault, the *throw* activity can be used. Every fault must have a globally unique *qname*. In the *fault* activity this name must be defined together with a *faultVariable*. The *faultVariable* contains further information

about the fault.

Code example: <throw faultName="qname" faultVariable="ncname" />

Waiting

Description: The *wait* activity offers the possibility to build in a certain time to sleep or wait till a specified deadline has passed.

Code example: <wait (for="duration-expr" | until="deadline-expr") />

Empty activity

Description: There are situations where it's necessary that a process does nothing. This could happen when you want to catch a fault, but then want to suppress it. In this case you can use the *empty* activity.

Code example: <empty />

Structured activities

Structured activities offer a way to structure a BPEL process into a specific order. They describe the flow of a process by structuring basic activities. In this way control patterns, data flow, fault handling and coordination of messages can be achieved.

The structured activities of BPEL are:

- Basic sequence control between activities, offered by the *sequence*, *switch* and *while* activities.
- Synchronization and concurrency of activities, offered by the *flow* activity.
- A choice based on information from the outside, offered by the *pick* activity.

Structured activities can be used recursive and it is important to see that structured activities can be used in anyway to create process flow. On this document we will refer to both basic and structured activities with the word *activity*.

Sequence

Description: A sequence activity has one or more activities that are executed sequential in the order they are placed within the *sequence* element. The *sequence* activity stops when all activities within it are done.

Code example:
```
<sequence>
  activities
</sequence>
```

Switch

Description: The *switch* activity makes it possible to specify conditional behavior. This activity consists out of an ordered list of conditional branches. Every branch is specified by a case element followed by one optional otherwise element. The *case* elements in a *switch* are looked at in the order in which they are placed. The activities specified in the *case* are executed when the condition of the *case* is *true*. When none of the cases are *true*, the activities in the *otherwise* element are executed. The *switch* activity is done when all the activities of one of the branches are completed.

Code example: <switch>

```
  <case condition="bool-expr">
    activity
  </case>
  <otherwise>
    activity
  </otherwise>
</switch>
```

While

Description: The *while* activity offers the possibility to walk through certain activities in an iterative way. The activities in the *while* activity are executed as long as the *boolean expression* in the condition attribute is *true*.

Code example:

```
<while condition="bool-expr">
  activity
</while>
```

Pick

Description: The *pick* activity waits till one event in a set of events occurs. When an event occurs the activities associated with that event are executed. When more that one event occurs only the event that occurs first is being processed. Possible events are the arrival of a message or a timer. The *pick* activity is done when all the activities in the started event are completed.

Code example:

```
<pick createInstance="yes|no">
  <onMessage partnerLink="ncname"
        portType="qname"
        operation="ncname"
        variable="ncname">
    activity
  </onMessage>
  <onAlarm (for="duration-expr" | until="deadline-expr")>
    activity
  </onAlarm>
</pick>
```

Flow

Description: The *flow* activity makes it possible to execute several activities parallel. A *flow* activity is done when all the activities in it are done. One of the possibilities offered by a *flow* activity is the synchronization of activities within the flow.

Code example:

```
<flow>
  activities
</flow>
```

Correlation – Instance of Process

A BPEL process is executed in instances. The declaration of the elements of the process serves as a template for each instance. An instance is always created due to an inbound message for which no running instance exists yet. The instances are clearly separated by their states which are (among others) constituted by the state of the control flow and the values of the declared variables. Since a BPEL process is a stateless WSDL service from the point of view of its environment, inbound messages are associated to the correct process instance by the help of BPEL's genuine correlation handling mechanism and the use of endpoint reference.

3.2.3 EODHaM BPEL engine

A business process needs a BPEL engine to be deployed and executed. For the EODHaM system the WS-BPEL 2.0 engine Riftsaw will be used.

Riftsaw is based on Apache ODE (Orchestration Director Engine) and supports the following features:

- WS-BPEL 2.0 OASIS standard and the legacy BPEL4WS 1.1 vendor specification.
- JBossWS Native and CXF Web Service stack support.
- UDDI registration of BPEL endpoints, and Runtime UDDI Endpoint lookup as preview feature.
- Enterprise quality GWT based BPM console to manage process definitions and instances.
- High level API to the engine that allows you to integrate the core with virtually any communication layer.
- JBoss deployment architecture, enabling hot deployment.
- Compiled approach to BPEL that provides detailed analysis and validation at the command line or at deployment.
- Short-lived and long-running process executions.
- Process persistence & recovery.
- Process versioning.
- Ant-based deployment.
- Integrated with the JBoss ESB.
- Eclipse-based BPEL designer and deployment, supported through JBoss Tools.
- Runs in JBoss Cluster.

3.3 EODHaM Business Process Orchestration

The EODHaM system shall be the 'system of systems' in terms of integration and interoperability. The EODHaM Business Process Manager module using BPEL orchestration shall compose Web Services into Business Processes. BPEL is the widely accepted standard for combining, coordinating and controlling the work-flow of web services into an end-to-end business process.

The BPEL orchestration of services can be defined as behavior resulting from a central conductor, co-coordinating the behaviors of individual entities (PartnersLink) and performing tasks independent of each other. In BPEL orchestration, a central process (in Bpel called Business process) takes control of the involved Web services and coordinates the execution of different operations on the Web services involved in the operation. The involved Web services do not acknowledge that they are involved in a composition process and that they are taking part in a higher-level business process. Only the central coordinator of the orchestration is aware of this goal, so the orchestration is centralized with explicit definitions of operations and the order of invocation of Web services.

The goal of Business Process Orchestration in the EODHaM system is to organize and coordinate various processing activities in a common operating line of BIO_SOS products. The imported data, will be driven to the various processes, organized as structural chain and all the derivative data, will be given as output products.

During the business process execution, each derivative product as result of a processing activity is considered valid BIO_SOS product. The same product can be characterized as final or intermediary, depending on the processing activities. The intermediary derivative products could be considered as exported output data of one activity and as imported input data of the following activity.

Each processing web services shall be able to produce a BIO_SOS product with the appositive metadata. The metadata shall be used for the registration on the metadata catalog, describing the product itself. The BIO_SOS product metadata are data providing information about one or more aspects of the data.

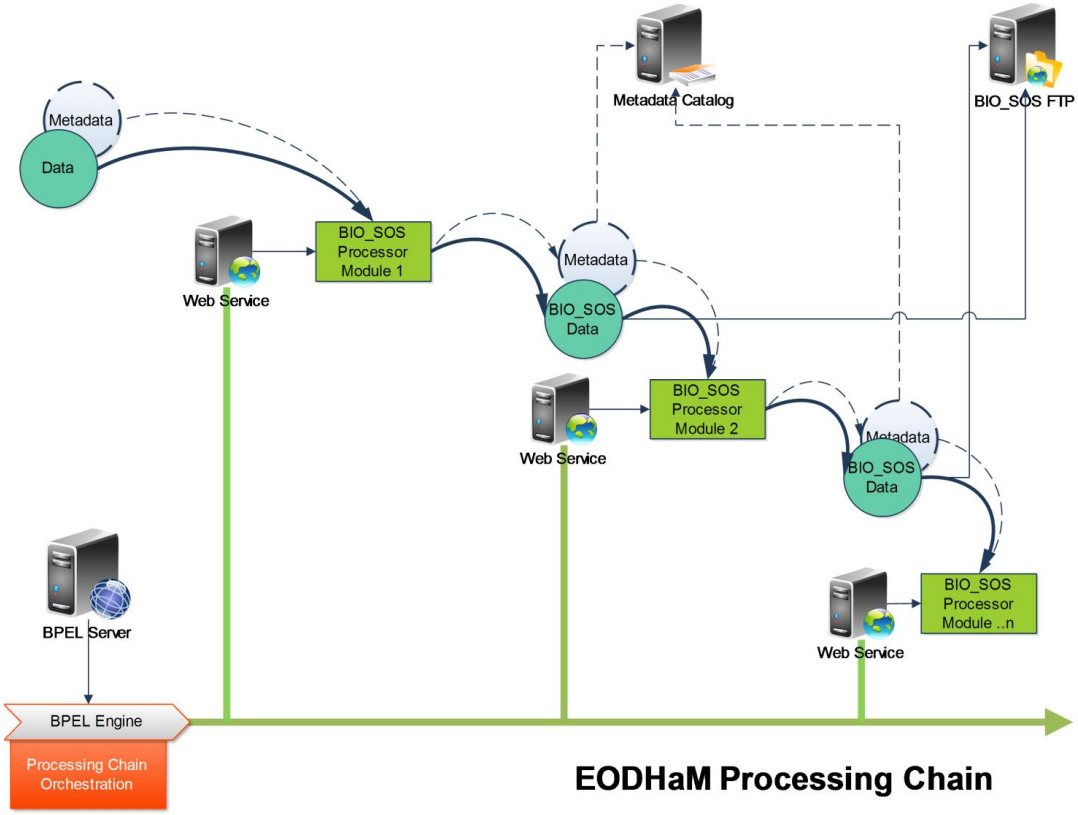


Figure 3: EODHaM Processing Chain

3.3.1 Describe processing chain based on metadata.

The EODHaM system, apart from the standard procedure, where the user imports the data and in the end of the work flow receives the products, shall be able to give a response to hypothetical requests like 'what products can be produced if I insert such input data corresponding to that specific metadata' providing as response a valid estimation of possibly products that can be obtained. That functionality of the EODHaM system shall be based only on the metadata information. The metadata information will be elaborated through the business process and each BIO_SOS processing component will be able to provide an estimation. This functionality servers the aim to describe effectively the capabilities of the processing chain.

3.4 EODHaM System Requirements

To successfully perform the EODHaM system interoperability and the web services interaction, each component has to respect the EODHaM requirements with specially emphasis on the interface development of each participator (as web service) in the business process.

3.4.1 EODHaM orchestration requirements

General Requirements:

1. Every component must be exposed to the system as web service.
2. Each web service interface must be well defined in terms of WSDL and SOAP messages.
3. Each web service could be invoked in asynchronous mode.
4. Each web service must be able to be invoked (in BPEL *invoke activity* is the activity that permits to call a web service) in automatic way and return a response.
5. The invocation of each web service is the start event for the internal processing activities.
6. Each web service must be able to receive input data, to execute the internal processing activities in the received data and produce output data.
7. Each web service, processing component, must be able to handle also only metadata as input data and provides an estimation of possibly products, operating in a 'virtual processing modality'.
8. In each step, of invoke, reply or fault, the system must provide additional email notifications to the administrators and log messages.

3.4.2 *EODHaM BIO_SOS processor - web service application*

It was previously mentioned that each BIO_SOS processor has an important part in the business process of the EODHaM system.

Although the processing algorithm core of the processor is distinct for each processor, a common infrastructural classification must be done, with aim of reducing complexity through abstraction and separation of concepts. The development of each processor must be conform to the common architectural rules, specially in terms of interfaces and types of soap messages interchangeable.

The processor classification is considered as high level of the EODHaM requirement in terms of homogeneity of the web services in the orchestration of the business process.

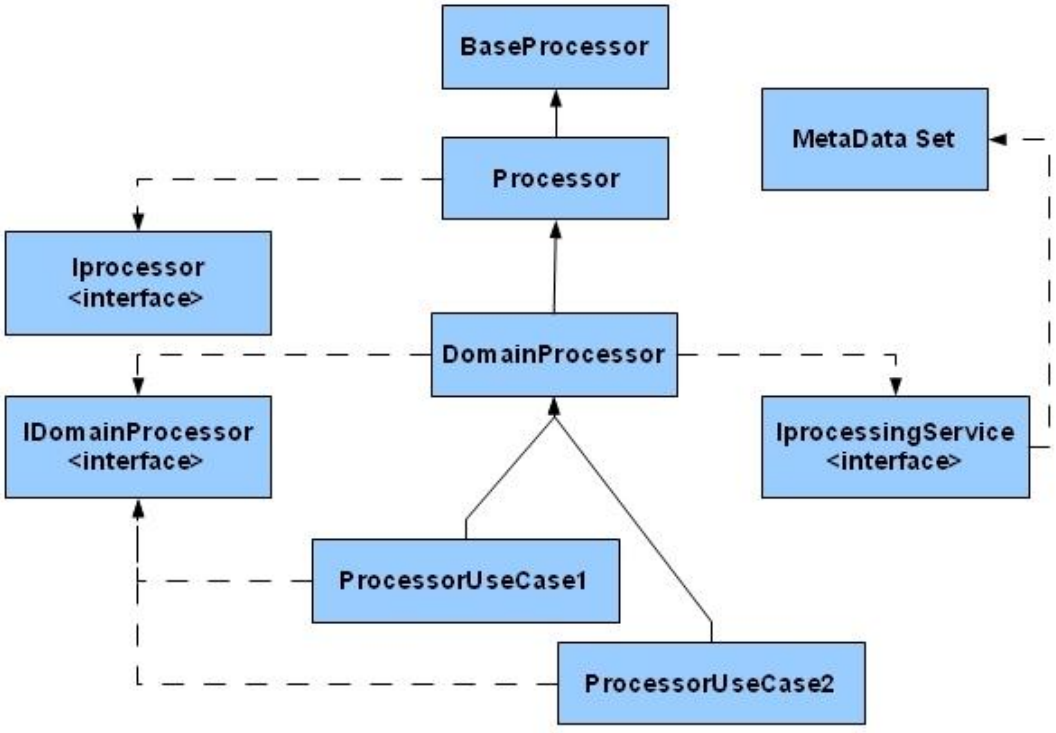


Figure 4: Processor wrapper classification

The Interface **IProcessor** is used to define constraints on the processor generic behavior. Similarly, the Interface **IDomainProcessor** defines constraints for the prediction loop.

The **IprocessingService** interface (as **DomainProcessor** and the classes which extend it) can behave as WSDL endpoint.

The Class **BaseProcessor**: It's the base class that includes utility methods used by the different processor at every level for input/output data transfer and metadata parsing/manipulation.

The Class **Processor**: Implements methods for the processor invocation. Every node in the chain has to implement a class like this.

The Class **DomainProcessor** instantiates a new domain processor, creates the processor configuration, returning the description of the concrete processing and generate the Prediction Metadata.

The Class **DefaultMetaData** is the class binding the ISO 19139 XML implementation schema for ISO 19115.It is used to parse, validate, and exchange geospatial metadata in the processing chain.

4. Actors

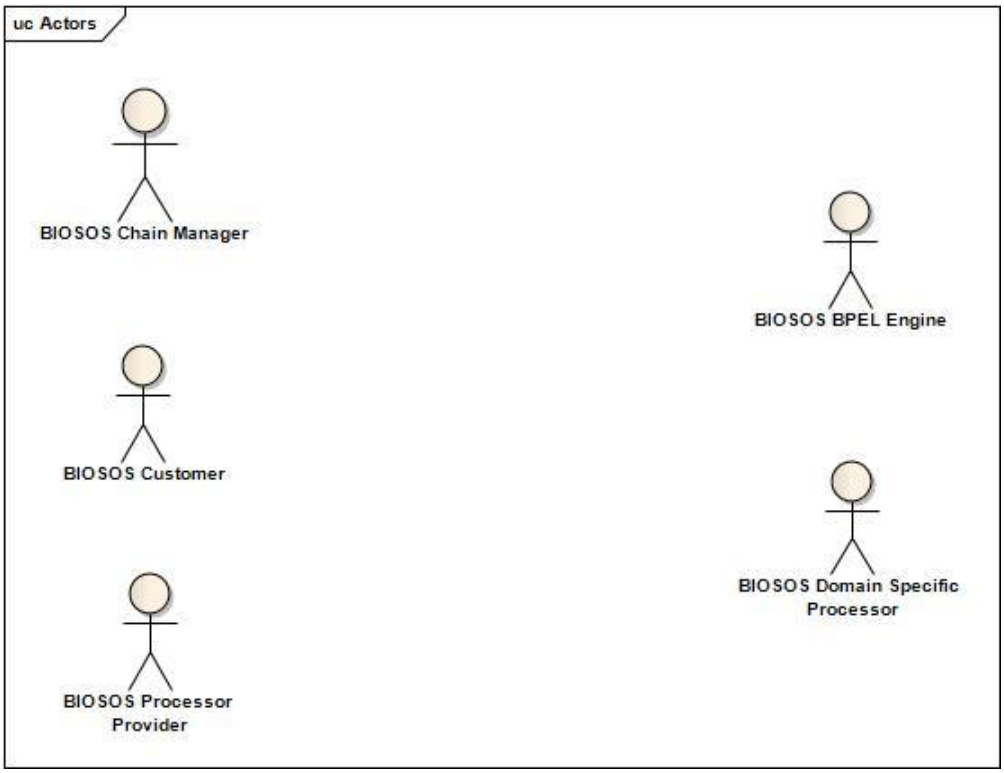


Figure 5: Principal actors of EODHaM system Use cases

4.1 BIO_SOS Customer

The BIO_SOS Customer can make a BIO_SOS request for specific BIO_SOS services or BIO_SOS products. He can also request a monitoring activity for a specific area. The collaboration terms are subject of the agreements made by the customer with the BIO_SOS consortium.

4.2 BIO_SOS Chain Manager

The BIO_SOS Chain Manager is the administrator of the EODHaM BPEL system and has multiple responsibilities and tasks to accomplish.

- Is the receiver of the customer requests.
- Is responsible for the tasks of selection, collection and validation of the input data for the processing.
- Has administration access to the EODHaM BPEL Engine.
- Can define, modify and deploy a BPEL “business process”. In other terms, orchestrates the work flow of the processing activities between the processor web services.
- Can execute a processing chain, an instance of the “business process”.
- Receives and publishes the products to the BIO_SOS metadata catalog and to the customer after a successful execution of a processing chain.

4.3 BIO_SOS BPEL Engine

The BIO_SOS BPEL Engine is the system core of the processor chain. It constitutes the logical container of the “business process” that defines the sequential order of the work flow of the processing

chain. It is the central point of all interchange activities between the EODHaM web services and has for final objective the organization of the overall processing activities in an automatic execution.

4.4 BIO_SOS Domain Specific Processor

The BIO_SOS Domain Specific Processor is the responsible component for the exposure of the processor functionality of elaboration as web service to the EODHaM system. The modules implementation is made around the processor as a wrapper and has the scope of creating the necessary interface in which the EODHaM system can communicate with the processor web processor.

The BIO_SOS Domain Specific Processor is responsible:

- To implement the WSDL interface that describes the communication rules between the EODHaM system and the processor web service.
- To implement the describeProcessor and ExecuteProcesor methods for the task of processor capabilities request and processor execution.
- To receive the input data, to analyze, execute the processing, create the product metadata and store the produced dataset.

4.5 BIO_SOS Processor Provider

The BIO_SOS Processor Provider is the administrator of the processor web service.

The BIO_SOS Processor Provider tasks are:

- Definition of a new Domain specific Processor. The BIO_SOS Processor Provider may have the necessity to create a new configuration profile with different processing parameters for the same core of algorithm processor.
- Deploy of a new Domain Specific Processor. The deployment of the domain permits the implementation of a new instance of the same processor but with different parameter set of configuration. Each new instance, specific domain, corresponds to a new unique WSDL interface endpoint.

5. Use cases

The present section describes all the significant use cases of the system.

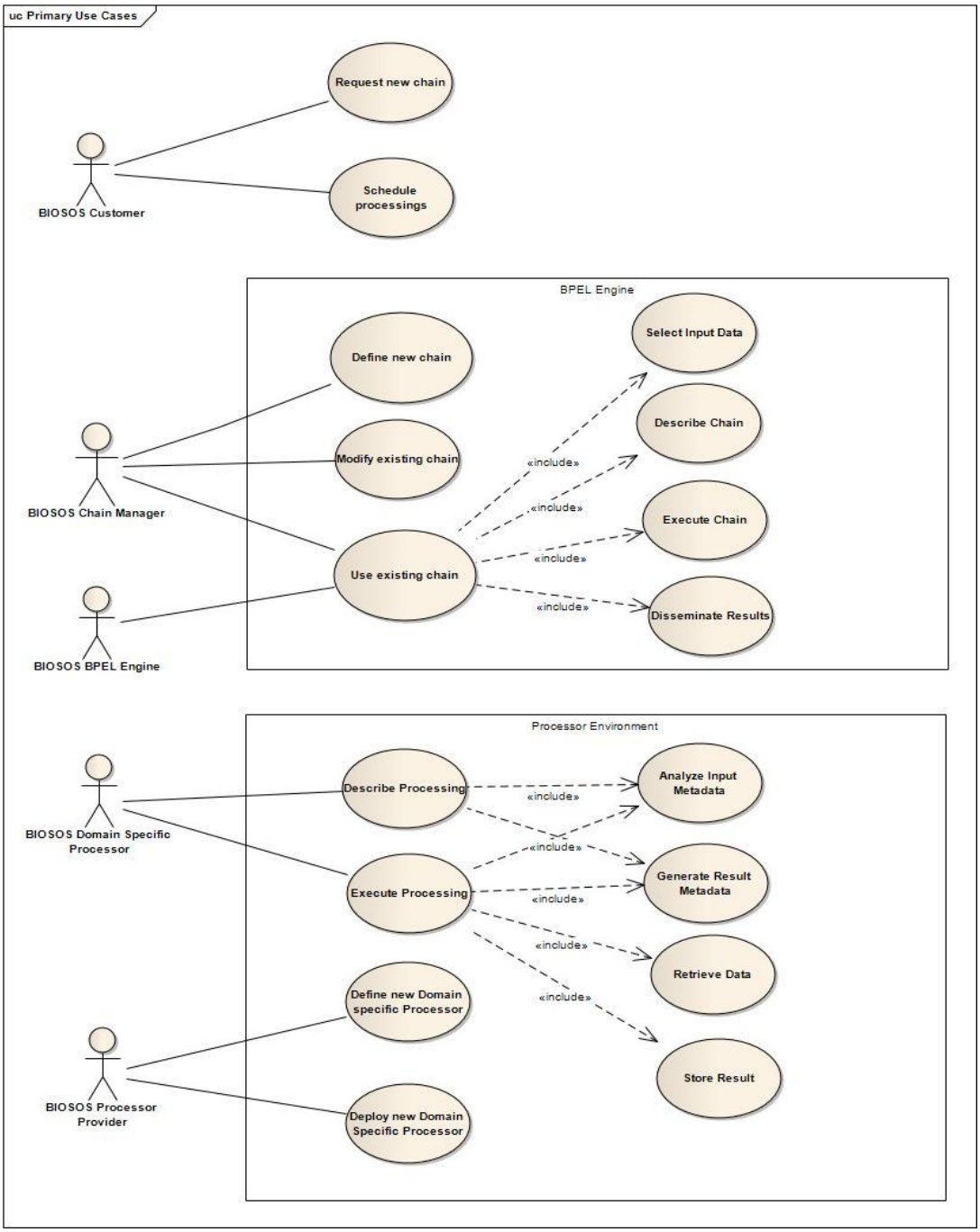


Figure 6: EODHaM Use Cases

5.1 Nomenclature for Use Case

Nomenclature used for the description of the uses cases:

- **UC – BC – xx.xx – xx**: Use case, BIO_SOS Customer, number of case, .(dot)sub case, number of step.
- **UC- BE – xx.xx– xx**: Use case, BPEL Engine,number of case,.(dot)sub case, number of step.

- **UC- PE – xx.xx – xx**: Use case, Processing Chain,number of case.,(dot)sub case, number of step.

5.2 Use Case Definitions

5.2.1 UC-BC-01: Request of a new BIO_SOS processing chain

USE CASE : UC – BC – 01		**Request of a new BIO_SOS processing chain.**
Goal in Context		Satisfy a request made by a BIO_SOS customer, for new BIO_SOS processing chain. **Note:***The procedure of this use case can have ulterior modifications and redefinitions, subject to strategic decisions that have to be taken by the partners .*
Preconditions		The BIO_SOS customer has the right credentials, to make a BIO_SOS service request.
Actors		BIO_SOS Customer, BIO_SOS Chain Manager
Trigger		E-mail communication to the BIO_SOS Chain Manager from BIO_SOS Customer
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS customer has read and understood the terms of the SLA of the BIO_SOS services.
	2	The BIO_SOS customer compiles a form document, that defines the requirements of the request.
	3	The BIO_SOS customer sends the email communication to the BIO_SOS chain manager.
	4	The BIO_SOS chain manager examines the request and “orchestrates” the new chain of the BIO_SOS processing tasks that are necessary for the satisfaction of the request.
	5	The BIO_SOS chain manager tests the new processing chain capabilities.
	6	The BIO_SOS chain manager search and selects the input data from the BIO_SOS metadata catalog. If not available on the catalog, alternative research in the different providers will be done.
	7	The BIO_SOS chain manager executes the new “business process” involving the individual BIO_SOS web services , in a work flow.
	8	After the successful execution of the “business process”, the BIO_SOS chain manager sends an email to the BIO_SOS customer with the results and the available products, with instructions to make them available to the customer.

	9	The BIO_SOS customer following the instructions received by the BIO_SOS chain manager receives the desired BIO_SOS products. (Usually by Ftp download or in some cases by receiving via post a DVD that contains the products.)

5.2.2 UC- BC – 02 Schedule processing

USE CASE : UC- BC – 02		**Schedule processing**
Goal in Context		Establish a schedule processing, executable at given intervals, for purpose of monitoring.
Preconditions		The BIO_SOS customer has the right credentials, to make a BIO_SOS service request.
Actors		BIO_SOS Customer, BIO_SOS Chain Manager.
Trigger		E-mail communication to the BIO_SOS Chain Manager from BIO_SOS Customer.
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS customer has read and understood the terms of the SLA of the BIO_SOS services.
	2	The BIO_SOS customer compile a form document, that defines the requirements of the request, indicating the time interval in which the BIO_SOS processes must be repeated and the Area of Interest.
	3	The BIO_SOS customer sends the email communication to the BIO_SOS chain manager.
	4	The BIO_SOS chain manager examines the request and “orchestrates” the new chain of the BIO_SOS processing tasks, if necessary, or uses one existing processing chain.
	5	The BIO_SOS chain manager search and selects the input data from the BIO_SOS metadata catalog. If not available on the catalog, alternative search in the different providers will be done.
	6	The BIO_SOS chain manager executes the new “business process” involving the individual BIO_SOS web services , in a work flow.
	7	The BIO_SOS chain manager repeats the execution of the “business process” in time intervals as are defined for the purpose of monitoring.
	8	After the successful execution of each repeated “business process”, the BIO_SOS chain manager sends an email to the BIO_SOS customer with the results and the available products, wit instructions to make them available to the customer.
	9	The BIO_SOS customer following the instructions received by the BIO_SOS chain manager receives the desired BIO_SOS products.

		(Usually by Ftp download or in some cases by receiving via post a DVD that contains the products.)

USE CASES		**BPEL Engine use cases**
Goal in Context		Verify the BPEL Engine operations: Definition, Modification and Execution of a processing chain.
Preconditions		The EODHaM system status is operational. All EODHaM web services are available and functional.
Actors		BIO_SOS Chain Manager, BIO_SOS BPEL Engine, EODHaM web services processors.
Trigger		Whenever the need of a new processing chain creation, or to modify an existing processing chain, or to execute an existing processing chain appears.
Description	**Step**	Action (Branching Action)
	1	UC- BE- 01 - Define new processing chain
	2	UC- BE-0 2 - Modify existing processing chain
	3	UC- BE- 03 - Use existing processing chain

5.2.3 UC- BE- 01 Define new processing chain

USE CASE : UC- BE- 01		**Define new processing chain**
Goal in Context		Create an EODHaM “business process”, using a collection of related, structured processing activities in a sequence order of execution that produces a specific BIO_SOS product.
Preconditions		The EODHaM system status is operational. All EODHaM web services are available and functional.
Actors		BIO_SOS Chain Manager, BIO_SOS BPEL Engine, EODHaM web services processors.
Trigger		Whenever the need of a new processing chain creation appears.
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS Chain manager examines the BIO_SOS web services capabilities in terms of type of the processing and the data specifications that each processor can elaborate (in input and in output).For the data, the specifications are extracted from their metadata.
	2	The BIO_SOS Chain manager creates the work flow logic of the processing chains, coordinated in such way, so the data of output of

		the first processing activity can be used as input data in the second selected processing activity and so on. The generated work flow is criteria-based on the automatic production (as result of the cooperation of the processors web services) of a specific BIO_SOS product starting the process from specific input data.
	3	The BIO_SOS Chain manager has to create a detailed documentation that describes each step of the "business process" concerning the processing activities in the work flow and also the production of the BIO_SOS products (including each intermediate output data that has been produced , apart the initial and final data). Also an identification name and a version number of the "business process" have to be provided.
	4	The BIO_SOS Chain manager creates the BPEL "business process". After an authentication access to the BIO_SOS BPEL Engine Console, makes the deployment of the "business process" in the BIO_SOS BPEL Engine.
	5	After the successful deployment of the BPEL "business process" , the BIO_SOS Chain manager activates the running process of the selected "business process".
	6	The BIO_SOS Chain manager executes successfully an instance of the "business process". During execution, each involved BIO_SOS web service is invoked in a sequential execution, receives an input data and produces the expected output data. The final aim (the final expected product) of the processing chain is accomplished successfully.

5.2.4 UC- BE- 02 Modify existing processing chain

Use Case : UC- BE- 02		**Modify existing processing chain**
Goal in Context		Modification of an existing deployed EODHaM "business process" to the BIO_SOS BPEL Engine.
Preconditions		The EODHaM system status is operational. All EODHaM web services are available and functional.
Actors		BIO_SOS Chain Manager, BIO_SOS BPEL Engine, EODHaM web services processors.
Trigger		Whenever the need of a processing chain modification appears.
Description	**Step**	Action (Branching Action)
	1	After an authentication access to the BIO_SOS BPEL Engine Console, the BIO_SOS Chain Manager stops the running process of the selected "business process".
	2	The BIO_SOS Chain Manager opens the BPEL "business process" in

		an editor and makes the necessary modifications. Saves the modified files as a new version of the existing one and creates a technical note of the modifications.
	3	From the BIO_SOS BPEL Engine Console, the BIO_SOS Chain Manager makes an undeploy operation of the existing “business process”.
	4	From the BIO_SOS BPEL Engine Console, the BIO_SOS Chain Manager makes the deployment of the new modified “business process” and activates the running process of the selected “business process”. Expected result is that all the process are running successfully.
	5	The BIO_SOS Chain manager executes successfully an instance of the “business process”. During execution, each involved BIO_SOS web service is invoked in a sequential execution, receives an input data and produce the expected output data.

5.2.5 UC- BE-03 Use existing processing chain

USE CASE : UC- BE-03		**Use existing processing chain**
Goal in Context		Prepare and Execute an instance of the processing chain and obtain the desired results.
Preconditions		The EODHaM system status is operational. All EODHaM web services are available and functional. A “business process” is already successfully deployed to the BIO_SOS BPEL engine. All the BPEL process are active.
Actors		BIO_SOS Chain Manager, BIO_SOS BPEL Engine, EODHaM web services processors.
Trigger		Whenever the need of a processing chain execution appears.
Description	**Step**	Action (Branching Action)
	1	UC- BE- 3.1 - Select Input Data
	2	UC- BE- 3.2 - Describe chain
	3	UC- BE- 3.3 - Execute chain
	4	UC- BE-3.4 - Dissemination Results

5.2.6 UC- BE- 3.1 Use existing processing chain - Select Input Data

USE CASE : UC- BE- 3.1	**Use existing processing chain - Select Input Data**
Goal in Context	Selection of Input data for the execution of a processing chain.

Preconditions		The EODHaM system status is operational. All EODHaM web services are available and functional. A “business process” is already successfully deployed to the BIO_SOS BPEL engine. All the BPEL process are active.
Actors		BIO_SOS Chain Manager, BIO_SOS BPEL Engine, EODHaM web services processors.
Trigger		The BIO_SOS Chain Manager receives a communication from the BIO_SOS Customer with a BIO_SOS request to execute a BIO_SOS processing chain.
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS Chain Manager receives a communication for a request to execute a BIO_SOS processing chain, capable to produce the desired BIO_SOS products.
	2	The BIO_SOS Chain Manager examines the parameters of the request.
	3	The BIO_SOS Chain Manager makes a search on the BIO_SOS metadata catalog for the necessary input data for the requested Area Of Request and time coverage. If nothing is found, the BIO_SOS Chain Manager makes alternative searches in different providers for the provision of the necessary data.
	4	The BIO_SOS Chain Manager makes a validation of the inputs data, and examines that the input data and their metadata are conformed with the requirements of the selected processing chain. If necessary the use of auxiliary data, it would be included together with the input data.
	5	The BIO_SOS Chain Manager selects and places the necessary input data in a repository (usually a FTP repository) with authenticated access and assures that the EODHaM system can have access to that repository for upload and download activities.

5.2.7 UC- BE- 3.2 Use existing processing chain - Describe chain

USE CASE : UC- BE- 3.2	**Use existing processing chain - Describe chain**
Goal in Context	The BIO_SOS Chain Manager based only on the input metadata information, executes a “processing chain capabilities”, receiving the processing chain capabilities. **Note:** *The mode “processing chain capabilities” creates a description of the processing chain. Given as input a metadata, the system executes an instance of “business process” that takes under consideration, as input and output, only metadata. The result is an estimation of eventual BIO_SOS products (as metadata results) that can be produced if the complete (image and metadata) input data*

		corresponding to the initial insert metadata is given for processing.
Preconditions		The EODHaM system status is operational. All EODHaM web services are available and functional. A “business process” is already successfully deployed to the BIO_SOS BPEL engine. All the BPEL processes are active. The necessary input data are selected and ready to be processed from the repository.
Actors		BIO_SOS Chain Manager, BIO_SOS BPEL Engine, EODHaM web services processors.
Trigger		Whenever it appears the need of a request for a processing capabilities
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS Chain Manager, after authenticated access to the BPEL console, selects the desired “business process” and executes an instance of processing chain. The given input is only metadata and in each step of the processing chain, from each processor services, is expected only production of metadata that describes the possible product that each involved processor can produce.
	2	The processing chain execution begins with the invocation of the first web service (processor module) by the BPEL Engine. The web service receives the metadata and possibly auxiliary data if needed and proceeds to the internal cycle of BIO_SOS processor elaborations.
	3	In the end of the internal elaborations, the web service generates a metadata that describes the specifications of product that can be produced if the input data as described in the input metadata is given for processing.
	4	The generated metadata of the web service processor unit is returned as reply to the BPEL Engine “business process”. Then the BPEL Engine invokes the second web service (processor unit), in the order of execution as the “business process” declares. **Note :** The produced metadata can be characterized as follows: • output metadata of the first involved web service (processor unit) • input metadata for the second involved web service (processor unit), part of the processing chain. • intermediate product of the global processing chain.
	5	The BPEL Engine invokes then the second involved web service (processor unit) that receives the metadata (from the first web service) and continues with the activation of his internal cycle of processing elaborations and generates a new metadata and so on..
	6	After the completion of the processing chain, the final result is returned as reply to the BIO_SOS BPEL Engine. The instance of the BPEL “business process” execution is terminated successfully.

	7	The BIO_SOS Chain Manager has received from each involved web service, the produced metadata and the final one. Those results offering the knowledge of the processing chain capabilities.

5.2.8 UC- BE- 3.3 Use Existing processing chain - Execute processing chain

USE CASE : UC- BE- 3.3		**Use Existing processing chain - Execute processing chain**
Goal in Context		The BIO_SOS Chain Manager executes a existing processing chain, giving a input data and receives the BIO_SOS products.
Preconditions		The EODHaM system status is operational. All EODHaM web services are available and functional. A “business process” is already successfully deployed to the BIO_SOS BPEL engine. All the BPEL process are active. The necessary input data are selected and ready to be processed from the repository.
Actors		BIO_SOS Chain Manager, BIO_SOS BPEL Engine, EODHaM web services processors.
Trigger		Whenever the need of a processing chain execution appears.
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS Chain Manager, after authenticated access to the BPEL console, selects the desired “business process” and executes an instance of processing chain. The given input is the complete data set (data plus metadata and possible auxiliary data).
	2	The processing chain execution begins with the invocation of the first web service (processor module).
	3	The web service (processor unit), downloads the input data set from the repository in a temporary locale storage for further processing activities.
	4	The web service proceeds to the internal cycle of BIO_SOS processor elaborations.
	5	In the end of the internal processing activities, the web service produce a BIO_SOS output set (the data itself with the metadata) that is returned to the BPEL Engine as reply. Also provides information for the repository used for the product storage.
	6	The BPEL Engine invokes the next web service (processor unit) in the declared order of the “business process” and so on (steps 2,3,4)
	7	After the completion of the processing chain, the final result is returned as reply to the BIO_SOS BPEL Engine. The instance of the BPEL “business process” execution is terminated successfully.
	8	The BIO_SOS Chain Manager has received from each involved web

		service, the produced BIO_SOS data and the final one.

5.2.9 UC- BE- 3.4 Use existing processing chain - Dissemination Results

USE CASE : UC- BE- 3.4		**Use existing processing chain - Dissemination Results**
Goal in Context		The BIO_SOS Chain Manager updates the metadata catalog with the new BIO_SOS products.
Preconditions		The EODHaM system status is operational. A processing chain is already executed successfully.
Actors		BIO_SOS Chain Manager, EODHaM web services processors.
Trigger		Whenever the need to possess a BIO_SOS product, result of the processing chain appears.
Description	**Step**	Action (Branching Action)
	1	During processing chain, the BPEL invokes the involved web services and receives their reply, that contains the produced BIO_SOS metadata (corresponds to each produced output) of each web service.
	2	The BIO_SOS Chain Manager collects that information. Also the BIO_SOS Chain Manager verifies the storage location of each product and the availability of them.
	3	The BIO_SOS Chain Manager after successful authentication on the metadata catalog registers the new products to the catalog.
	4	The BIO_SOS Chain Manager updates the metadata catalog with the new available products and their download location.

USE CASE	**Processing Environment Use Cases**
Goal in Context	Verify the Processing Environment operations : Describe and execute a processing, define new Domain and Deploy new Domain specific Processor.
Preconditions	The EODHaM Processor web service status is operational.
Actors	BIO_SOS Domain Specific Processor, BIO_SOS BPEL Engine, BIO_SOS Processor Provider.
Trigger	Whenever the need to describe or execute a processing, to define or deploy a new specific Processor appears.

Description	**Step**	Action (Branching Action)
	1	UC-PE-01 Describe Processing
	2	UC-PE-02 Execute Processing
	3	UC-PE-03 Define new Domain specific Processor.
	4	UC-PE-04 Deploy new Domain specific Processor.

5.2.10 UC- PE- 01 Describe Processing

USE CASE : UC- PE-01		**Describe Processing**
Goal in Context		Receive a request for BIO_SOS product generation and return a reply.
Preconditions		The EODHaM system status is operational. All EODHaM web services are available and functional. A “business process” is already successfully deployed to the BIO_SOS BPEL engine. All the BPEL processes are active. The necessary input data are selected and ready to be processed from the repository.
Actors		BIO_SOS Domain Specific Processor, BIO_SOS BPEL Engine
Trigger		Whenever the web service processor is invoked with a valid request.
Description	**Step**	Action (Branching Action)
	1	UC-PE-1.1 Analyze input Metadata
	2	UC – PE-1.2 Generate Result Metadata

5.2.11 UC-PE-1.1 Analyze input Metadata

USE CASE : UC-PE-1.1		**Analyse input Metadata**
Goal in Context		Receive a request and analyze the metadata content.
Preconditions		The EODHaM system status is operational. The EODHaM web services (corresponding to the BIO_SOS Domain Specific Processor) are available and functional. The request is in valid format.
Actors		BIO_SOS Domain Specific Processor, BIO_SOS BPEL Engine,
Trigger		Whenever the web service processor is invoked with a valid request.
Description	**Step**	Action (Branching Action)
	1	The web service processor is invoked by the BIO_SOS BPEL Engine.

	2	The web service interface of the processor module (IDomainProcessor) receives the request. The content of the request is a soap message that includes the input metadata which have to be processed by the processor.
	3	The BIO_SOS Domain Specific Processor extracts the input metadata.
	4	The BIO_SOS Domain Specific Processor also extracts the File Info about the storage location of the input data set with the internal method extractFileInfoFromMetaData .
	5	The BIO_SOS Domain Specific Processor proceeds with the elaboration of the metadata and calculates the possible product that can be produce.

5.2.12 UC-PE-1.2 Generate Result Metadata

USE CASE : UC-PE-1.2		**Generate Result Metadata**
Goal in Context		Provide the processor capability. Generate result metadata about the product that can be obtain by the processor.
Preconditions		The EODHaM system status is operational. The EODHaM web services (corresponding to the BIO_SOS Domain Specific Processor) are available and functional. The request is in valid format. The input Metadata has been extracted.
Actors		BIO_SOS Domain Specific Processor, BIO_SOS BPEL Engine,
Trigger		Whenever the web service processor is invoked with a valid request.
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS Domain Specific Processor activates the internal method processInputMetadata and elaborates the metadata.
	2	The BIO_SOS Domain Specific Processor activates the internal method describeProcessing and executes a processing activity based only on those metadata, as a processing simulation of the complete input data set.
	3	The BIO_SOS Domain Specific Processor generates the metadata of the possible product that can be produced. In other terms, the generated metadata is the reply of the processor capability, to the initial request of the input metadata.
	4	The produced metadata is sent as reply to the BPEL Engine.

5.2.13 UC-PE-02 Execute Processing

USE CASE : UC-PE-02		Execute Processing
Goal in Context		Execute Processing, generating the new BIO_SOS product.
Preconditions		The EODHaM system status is operational. The EODHaM web services (corresponding to the BIO_SOS Domain Specific Processor) are available and functional. The request is in valid format. The input Metadata has been extracted.
Actors		BIO_SOS Domain Specific Processor, BIO_SOS BPEL Engine
Trigger		Whenever the web service processor is invoked with a valid request.
Description	**Step**	Action (Branching Action)
	1	UC-PE-1.1 Analyze input Metadata
	2	UC-PE-1.2 Generate Result Metadata
	3	UC-PE-2.1 Retrieve Data
	4	UC-PE-2.2 Store Result

5.2.14 UC-PE-2.1 Retrieve Data for Processing

USE CASE : UC-PE-2.1		Retrieve Data for Processing
Goal in Context		The BIO_SOS Domain Specific Processor, retrieves the input data set and proceeds with the processing step.
Preconditions		The EODHaM system status is operational. The EODHaM web services (corresponding to the BIO_SOS Domain Specific Processor) are available and functional. The request is in valid format. The input Metadata has been extracted and analyzed. The BIO_SOS Domain Specific Processor has already produced the metadata product.
Actors		BIO_SOS Domain Specific Processor, BIO_SOS BPEL Engine
Trigger		Whenever the web service processor is invoked with a valid request for BIO_SOS production.
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS Domain Specific Processor uses the storage location information for the input data set that has already been extracted during the execution of the UC-PE-1.1(4) step and downloads them in a temporary local storage.
	2	The BIO_SOS Domain Specific Processor, with the use of the executeProcessing method, performs the processing activity to the data items that belongs to the input data set.

	3	After the production of the new data, result of the processing, the BIO_SOS Domain Specific Processor, creates the output data set, including the necessary metadata of the produced data.
	4	The output produced data are ready to be stored and disseminated.

5.2.15 UC-PE-2.2 Store Result

USE CASE : UC-PE-2.2		**Store Result**
Goal in Context		The BIO_SOS Domain Specific Processor, saves and publishes the produced data.
Preconditions		The EODHaM system status is operational. The EODHaM web services (corresponding to the BIO_SOS Domain Specific Processor) are available and functional. The request is in valid format. The input Metadata has been extracted and analyzed. The BIO_SOS Domain Specific Processor has already produced the data and the metadata product.
Actors		BIO_SOS Domain Specific Processor, BIO_SOS BPEL Engine
Trigger		Whenever the web service processor is invoked with a valid request for BIO_SOS production.
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS Domain Specific Processor saves the produced data on a repository (usually Ftp repository) and provides authenticated access for download.
	2	The BIO_SOS Domain Specific Processor includes the information of the product location in the product metadata.

5.2.16 UC- PE-03 Define New Domain Specific Processor

USE CASE : UC- PE-03		**Define New Domain Specific Processor**
Goal in Context		Create a new configuration set of parameters that determinate a new Processor profile, creating a New Domain specific Processor.
Preconditions		N/a
Actors		BIO_SOS Processor Provider, BIO_SOS Domain Specific Processor
Trigger		Whenever is required a new Domain specific Processor
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS Processor Provider creates a new configurable set of

		Processing parameters that influence the Processor behavior through the method DomainProcessor. The new configurable parameters profile provides to the processor algorithm new functionalities, without change the algorithm itself.
	2	The BIO_SOS Processor Provider implements a new instance of the Domain specific Processor.

5.2.17 UC-PE-04 Deploy New Domain Specific Processor

USE CASE : UC-PE-04		**Deploy New Domain Specific Processor**
Goal in Context		Deploy a new configuration set of parameters that determinate a new Processor profile, creating a New Domain specific Processor.
Preconditions		The definition of the New Domain specific Processor is already done.
Actors		BIO_SOS Processor Provider, BIO_SOS Domain Specific Processor
Trigger		Whenever is required a new Domain specific Processor
Description	**Step**	Action (Branching Action)
	1	The BIO_SOS Processor Provider implements a new DomainProcessor instance (and the classes which expands it) and creates a new WSDL endpoint.
	2	The new web service WSDL interface creates a new instance of the same Processor algorithm core with different processing behavior that corresponds to the different configuration parameters.

6. Logical view

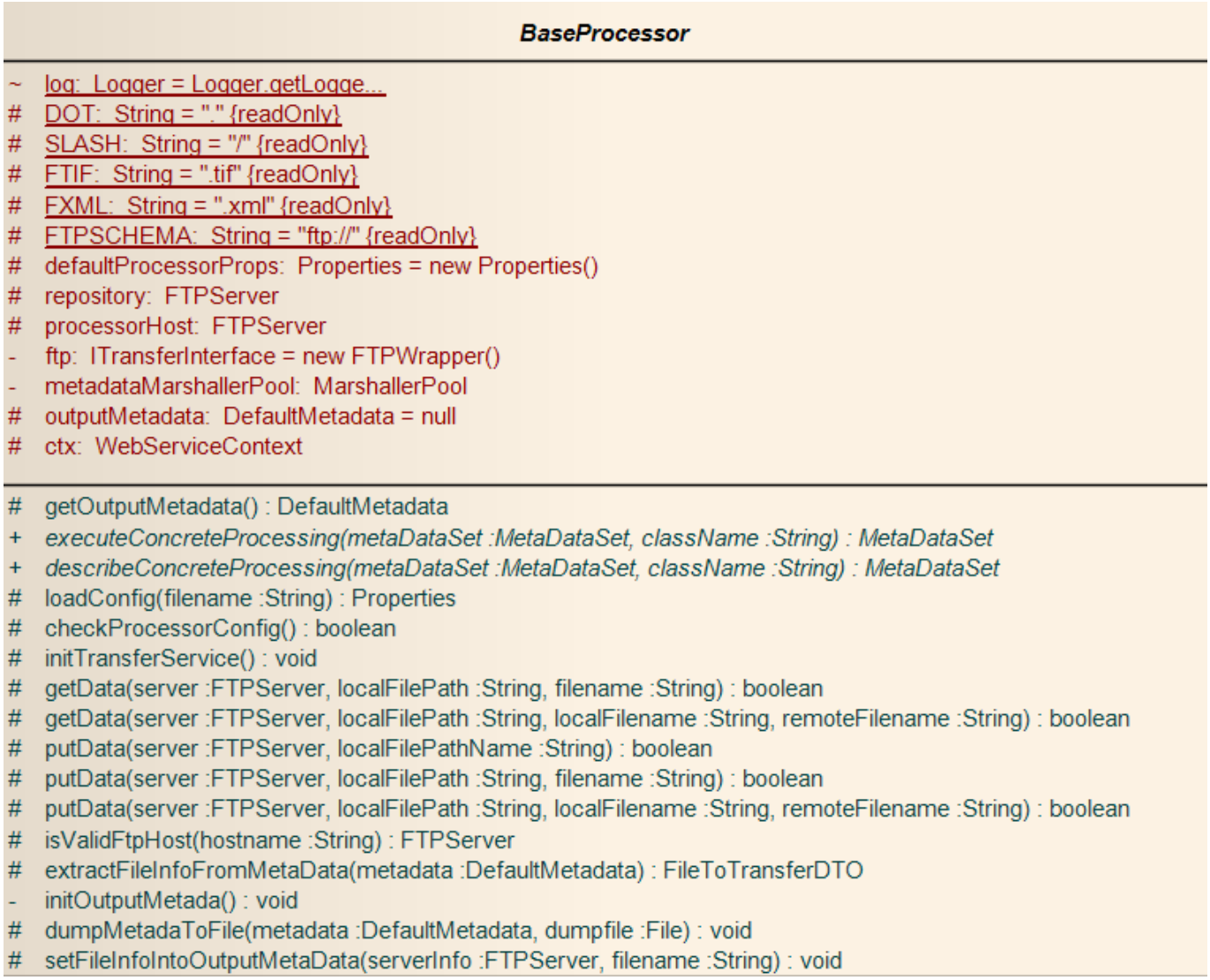
This section describes the logical view level of the system. There will be taken in account:

1. the system main decomposition in subcomponents
2. the interaction between the users and the system
3. the internal and external interfaces of the system

6.1 Processor Environment Components

6.1.1 BaseProcessor

It is the base class that includes utility methods used by the different processor at every level for input/output data transfer and metadata parsing/manipulation.



The BaseProcessor implements the following methods:

- **loadconfig**

Load config. - returns the properties

- **checkProcessorConfig**

Check processor config – return true, if successful

- **isValidFtpHost**

Checks against configuration if the host is a valid ftp server – return the FTP server]

- **InitTransferService**

Inits the transfer service.

- **getData**

Gets the file from the remote server to be used if local file name has to be the same as remote file - return true, if successful

- **extractFileInfoFromMetaData**

Extract file info from meta data. The file to process has to be fetched from a remote address specified inside the metadata; the assumption here is that it is stored in the "dataSetURI" element

- **executeConcreteProcessing**

Execute concrete processing and return the meta data set

- **describeConcreteProcessing**

Describe concrete processing -return the meta data set

- **initOutputMetadata**

Inits the output metadata

- **setFileInfoIntoOutputMetaData**

Sets the file info into meta data; in general to be used before returning metadata to BPEL processor

- **dumpMetadataToFile**

Dump metadata to file

- **getOutputMetadata**

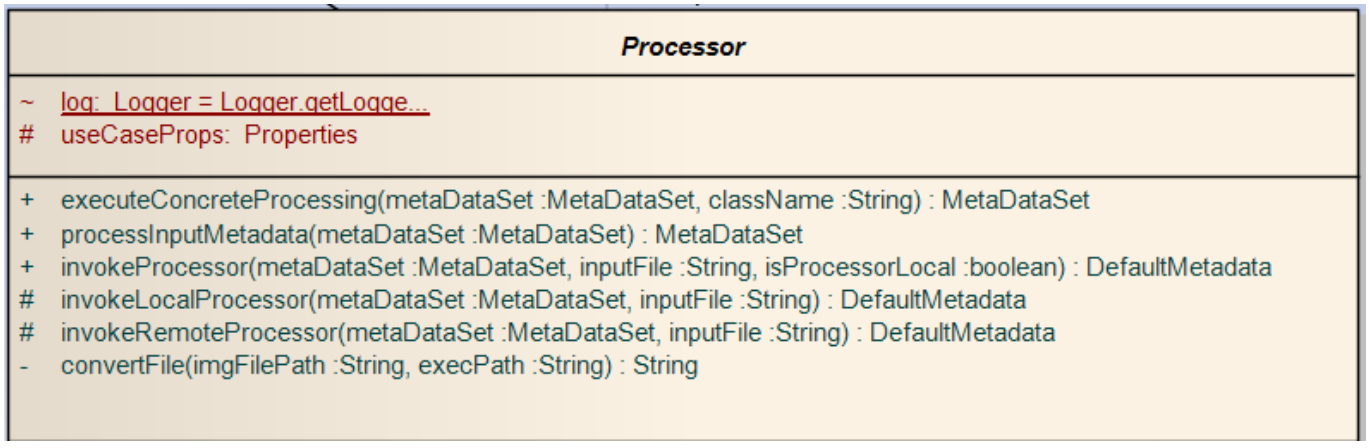
Gets the output metadata -returns the output metadata

- **putData**

Puts the file onto the remote directory to be used if local and remote file name can be the same – return true, if successful.

6.1.2 Processor

The Class Processor implements methods for the processor invocation. Every node in the chain has to implement a class like this



The class Processor implements the following methods:

- **invokeProcessor**

This implementation creates a switch between local processing and remote processing, then the two methods make same thing, with the 'remote' that performs a file transfer first; in general a processor won't have to implement both 'invokeLocalProcessor' and 'invokeRemoteProcessor', just one would suffice according to the deployment schema. - return the default metadata

- **invokeLocalProcessor**

Invoke local processor. as an example here we call a locally installed software for image format conversion; the dummy metadata record is generated from a file template, with just the dataSetURI filled with the location of the generated file, maybe after transfer to a remote host
-return DefaultMetadata record referred to the output generated by the processor

- **invokeRemoteProcessor**

Invoke remote processor. Same as invokeLocalProcessor, but the input data file has to be transferred to remote processor node and the processing executable called has to be a remote shell execution ('ssh -x....'**)**
- return the default metadata

- **processInputMetadata**

In this sample implementation, the URI location of the dataset processed by previous processor is extracted from last metadata record, and the processor is invoked locally or remotely according to the configuration.
- return the metadata set after processing]

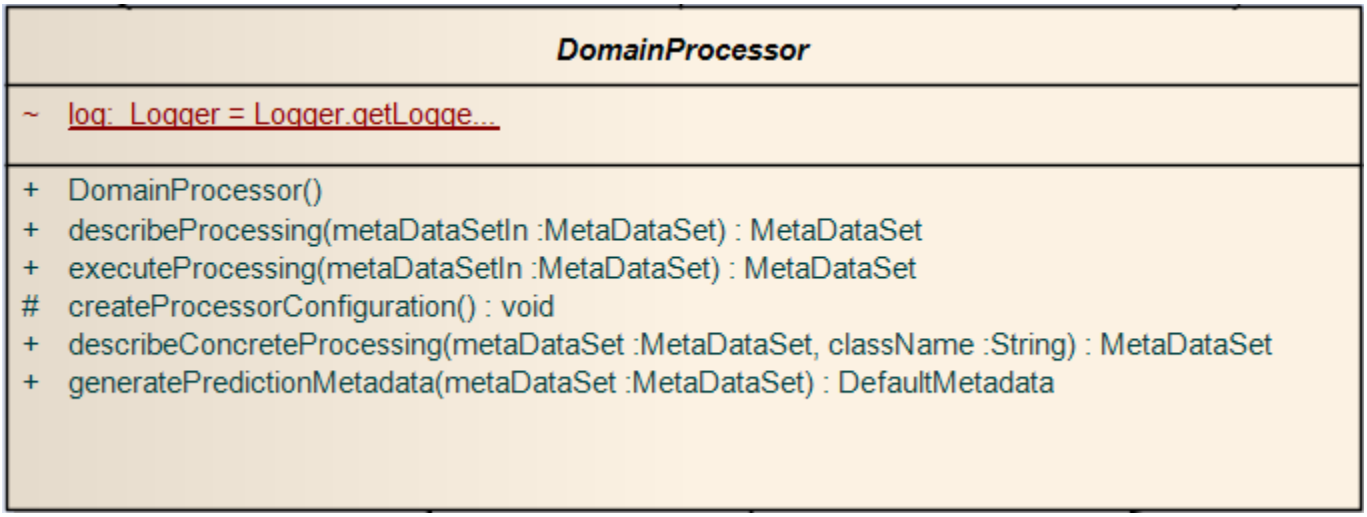
- **executeConcreteProcessing**

Execute concrete processing and return the metadata set.

- **ConvertFile**

Convert file. - return the full path of the converted file.

6.1.3 DomainProcessor



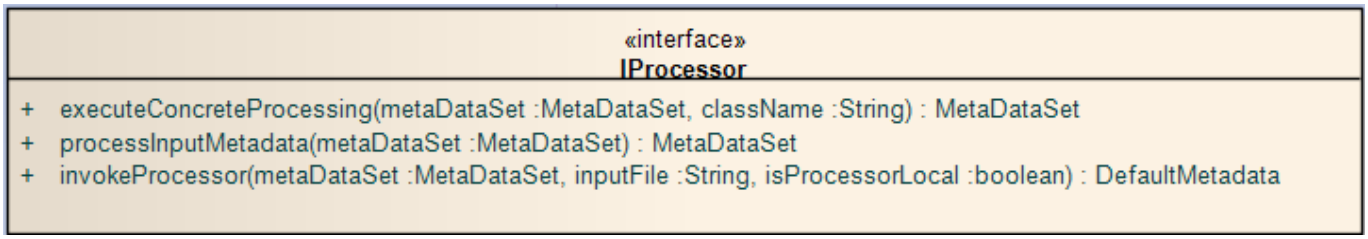
The class DomainProcessor implements the following methods:

- **DomainProcessor**
  Instantiates a new domain processor
- **createProcessorConfiguration**
  Creates the processor configuration.
- **describeConcreteProcessing**
  Describe concrete processing -return the meta data set
- **generatePredictionMetadata**
  Invoke prediction – param: inputFile – return the default metadata

describeProcessing

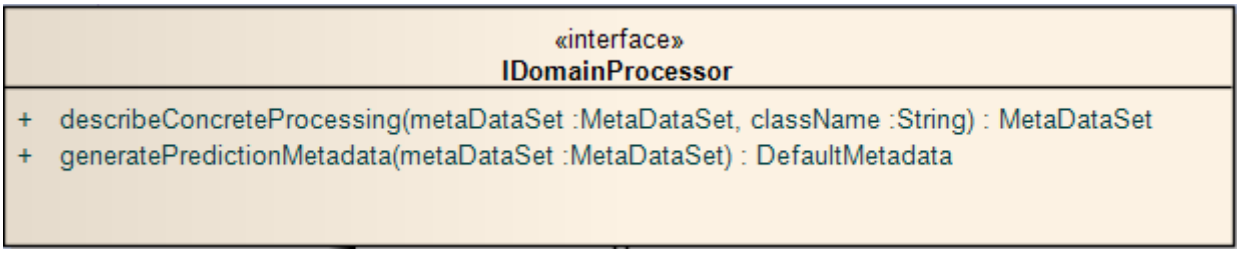
- **executeProcessing**

6.1.4 IProcessor



6.1.5 IDomainProcessor

The Interface IProcessor is used to define constraints on the processor generic behavior. Similarly, the Interface IDomainProcessor defines constraints for the prediction loop. Different classes (one per use case) have to implement it



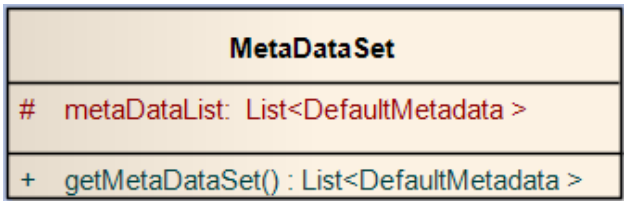
This class implements the following methods:

- **describeConcreteProcessing**
  Describe concrete processing. It's the web service method, invoked by the BPEL processor, that has to be implemented by every processor in the chain for the prediction loop - return the output meta data set
- **generatePredictionMetadata**
  Generates prediction metadata. This method has to be customized for every processor, which must parse one or more input metadata record to retrieve useful processing information, invoke the prediction algorithm, format a new metadata record with the defined prediction information and return the generated metadata record - return the output metadata record generated.

6.1.6 IProcessingService

The classes implementing the IProcessingService interface (as DomainProcessor and the classes which extend it) can behave as WS endpoints:

6.1.7 MetaDataset



It is the class binding the ISO 19139 XML implementation schema for ISO 19115; it is used to parse, validate, and exchange geospatial metadata in the processing chain. Represents the set of metadata records flowing in the processing chain

6.2 Workflow Environment Components

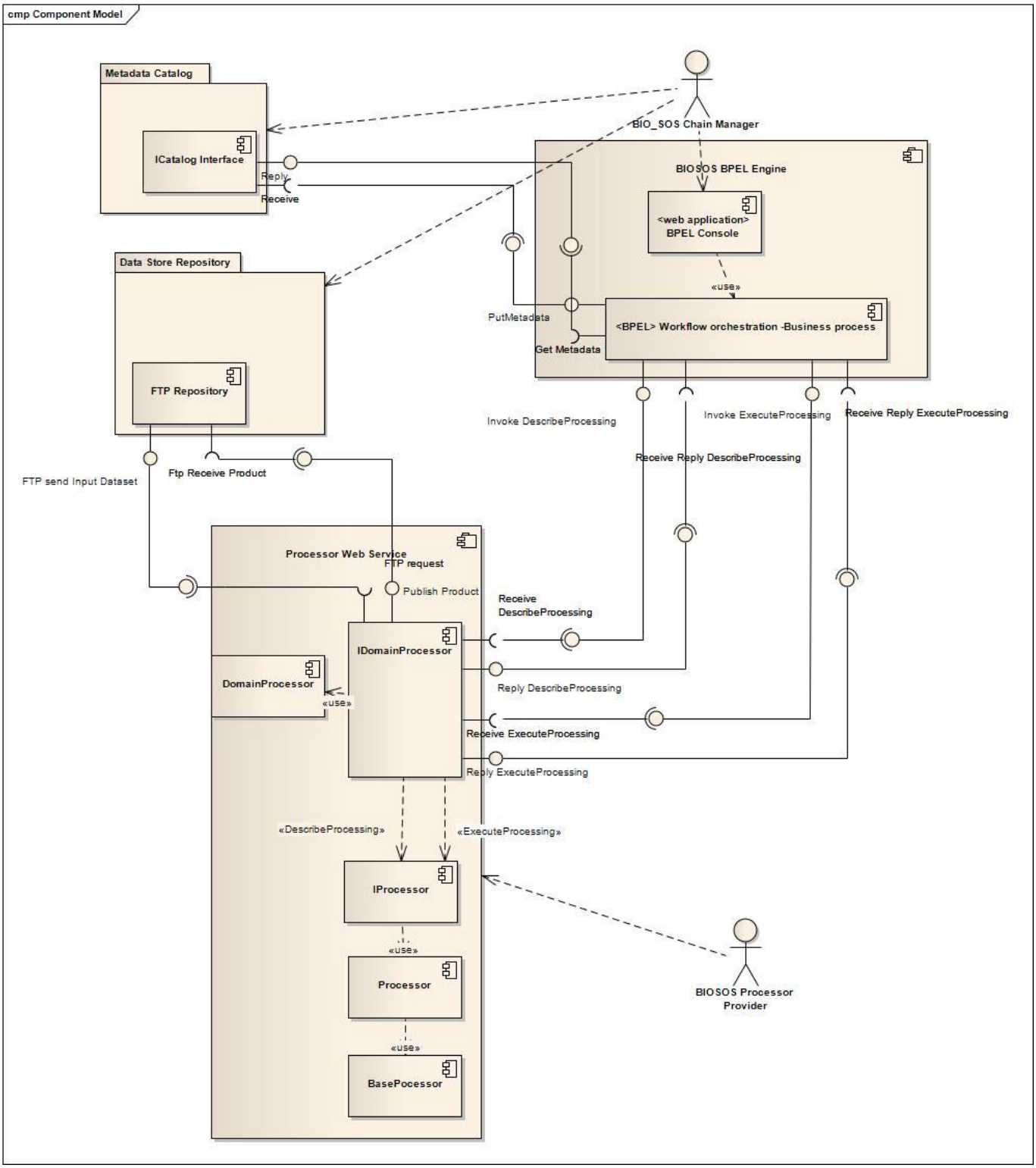


Figure 7: EODHaM system workflow environment components

6.2.1 BIOSOS BPEL Engine

The EODHaM system part, responsible for the work flow, is the BIOSOS BPEL Engine. It is used the Riftsaw WS-BPEL 2.0 engine that is optimized for the Jboss Application Server container. Riftsaw BPEL Engine is based on Apache ODE.

Apache ODE (Orchestration Director Engine) executes business processes written following the WS-BPEL standard. It talks to web services, sending and receiving messages, handling data manipulation and error recovery as described by your process definition. It supports both long and short living process executions to orchestrate all the services.

The BPEL Engine permits the deployment of different “business processes”. Each business process represents the defined sequential order of task execution. Each different “business process” corresponds on different operational scenarios and can involve different web services and in different order.

6.2.2 BIOSOS BPEL Console

The BIO_SOS Chain Manager can administrate the BPEL Engine with the BPEL Console. The console consists to a user interface web application in which the administrator can execute commons tasks about the business process as deployment, modification, undeployment, execution, monitoring the status, receive fault exceptions and other control operations of the business process.

6.2.3 BIOSOS Web services

The BPEL Engine is in direct dependence of the involved web services. Each Processor web service has to implement the interface and the extended components as described on the Processor Environment Components paragraph 6.1

7. Implementation Details

7.1 Hierarchical order of BIO_SOS operational components

The EODHaM system follows the following operational component subdivision.

EODHaM stage: Is the sequence order of EODHaM tasks. Each stage is composed by BIO_SOS Processor Modules that performs EODHaM sub tasks.

The EODHaM stage is an organized plan of EODHaM tasks subdivision. The processing macro task, are subdivided in four classified stages. Each stage contains different BIO_SOS Processor Modules that perform a specific processing activity.

BIO_SOS Processor Module: Is a distinct program unit that maintains the integrity of the internal and autonomous functionality and represents a software application, which when it is invoked, executes an internal processing of the input data given and produces a specific output according on criteria-based on the type of the input data and the executable processing instructions that were applied.

BIO_SOS Processor Functions: Each function (also termed procedure, routine, method, subroutine, or subprogram) is a part of source code within a software application, in our case the BIO_SOS Processor Module. A BIO_SOS Processor Module can have multiple Functions. Each Function expresses a distinct functionality and is connected with the other arranged functions, completing the operation of the Processor. The identity of one Processor Module is characterized by his included Functions.

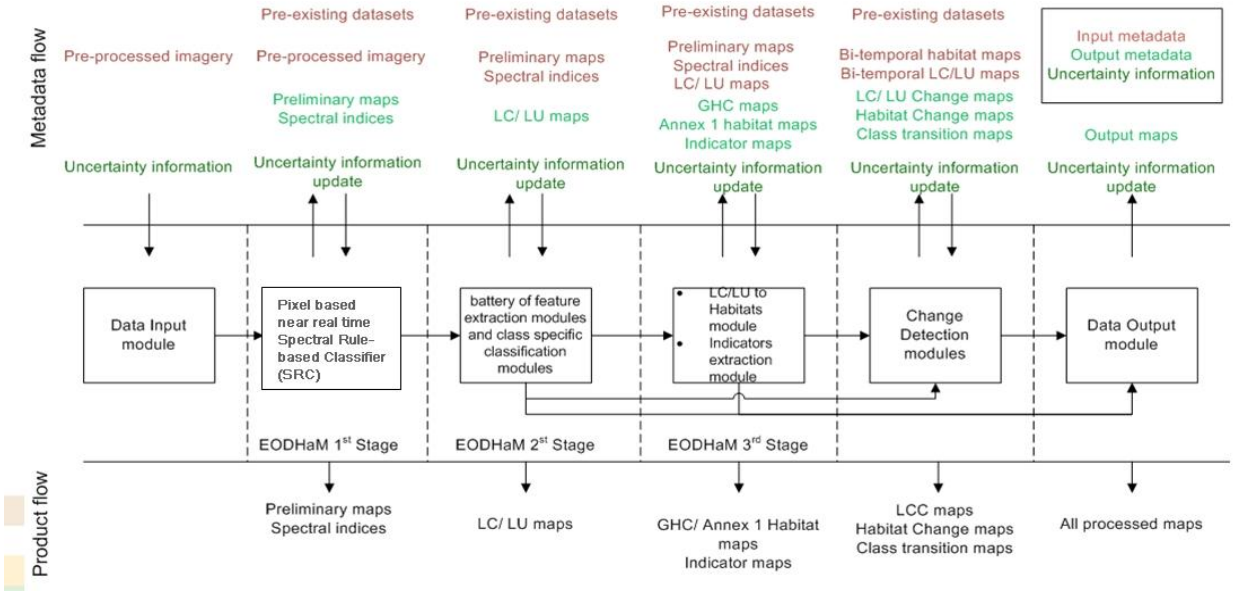


Figure 8: EODHaM processing tasks workflow

7.2 Hierarchical List

The following list, describes the organized plan of the EODHaM system components as

1 Stage

1.1 BIO_SOS Processor Module

1.1.1 BIO_SOS Processor Function

EODHaM system:

1 Data Input Module

1.1 **BIO_SOS Processor Module 1 - Data preparation**

1.1.1 Automated download (Reference DEM, Ground control points, Reference image for matching)

1.1.2

1.2 **BIO_SOS Processor Module 2 - Preprocessing**

1.2.1 Orthorectification

1.2.2 Radiometric correction

1.2.3 Surface reflectance (ratio of red/blue for optical thickness)

1.2.4 Topographic correction

2 EODHaM 1st Stage

2.1 **BIO_SOS Processor Module 3 - Derived products**

2.1.1 Index calculations

2.1.2 First order texture feature calculation

2.1.3 Optional pre - processing (Vegetation height estimation through first-order texture analysis under LiDAR unavailability)

2.2 **BIO_SOS Processor Module 4 – Segmentation**

2.2.1 Image boundaries detection

2.2.2 Large objects detection

2.2.3 Small objects detection

2.2.4 Post – processing (Combine segments, Shape calculations for objects)

2.3 **BIO_SOS Processor Module 5 – Preliminary spectral map extraction**

2.3.1 Preliminary spectral map extraction based on: FAO-LCCS classification: level 1 to level 2

3 EODHaM 2st Stage

3.1 **BIO_SOS Processor Module 6 – LC/LU final classification**

3.1.1 Context-features extraction

3.1.2 Classification of LC/LU classes based on FAO-LCCS scheme: LCCS Level 3 and beyond

3.1.3 Dempster – Shafer fuzzy classifications.

3.1.4 Conversion between raster and vector formats

3.1.5 Colour-map creation

3.2 **BIO_SOS Processor Module 7 – Land Cover/Use Change detection (LCC)**

3.2.1 Object change

3.2.2 Within object change (dilation or erosion / Invasion or invaded)

3.2.3 Class transition and trends

4 EODHaM 3st Stage

4.1 **BIO_SOS Processor Module 8 – LC to GHC Translation**

4.1.1 Pre – processing: detection of linear and points elements (area lower than 400sqm) at LCCS level

4.1.2 LC/LU to GHC translation (and labeling of all the elements, including point elements)

4.1.3 Dempster – Shafer fuzzy classification

4.1.4 Conversion between raster and vectors formats

4.1.5 Refined GHC classifications: Merging point elements to create areal ones, at the GHC level, if needed.

4.1.6 Colour-map creation

4.2 **BIO_SOS Processor Module 9 – GHC to Annex I translation**

4.2.1 Combining GHC classes, ancillary information (e.g. DEM) and GHC environmental qualifiers (e.g. Moisture regimes, Ellenberg values), to classify Annex I habitats

4.2.2 Colour-map creation

4.3 **BIO_SOS Processor Module 10 – LC to Annex I translation**

4.3.1 Combining LC/LU classes and environmental LCCS attributes (e.g. lithology, soil aspect, water salinity) information to classify Annex I habitats

4.3.2 Colour-map creation

4.4 **BIO_SOS Processor Module 11 - Habitat Change detection**

4.4.1 Object change (new or deleted)

4.4.2 Object change (split or merge)

4.4.3 Object or pixel attribute change (index)

4.4.4 Within object change (ditation or erosion / Invasion or invaded)

4.4.5 Change attribution

4.4.6 Temporal change, within daily to decadal

4.5 **BIO_SOS Processor Module 12 – Accuracy and uncertainty assessment**

4.5.1 Confusion matrices

4.5.2 Dempster Shafer

4.6 BIO_SOS Processor Module 13 -Ecological modeling at landscape level

4.7 **BIO_SOS Processor Module 14 – Biodiversity Indicators extraction**

4.7.1 Biodiversity indicators extraction

4.8 **BIO_SOS Processor Module 15 – Biodiversity Indicators change detection**

4.8.1 Biodiversity indicators change detection

4.8.2 Biodiversity indicators trend estimation

4.8.3 Warning signal

7.2.1 Diagram Presentation

The same list representation of the EODHaM components is described in Figure 9 as a diagram illustrating the correlation of each BIO_SOS Processor Module.

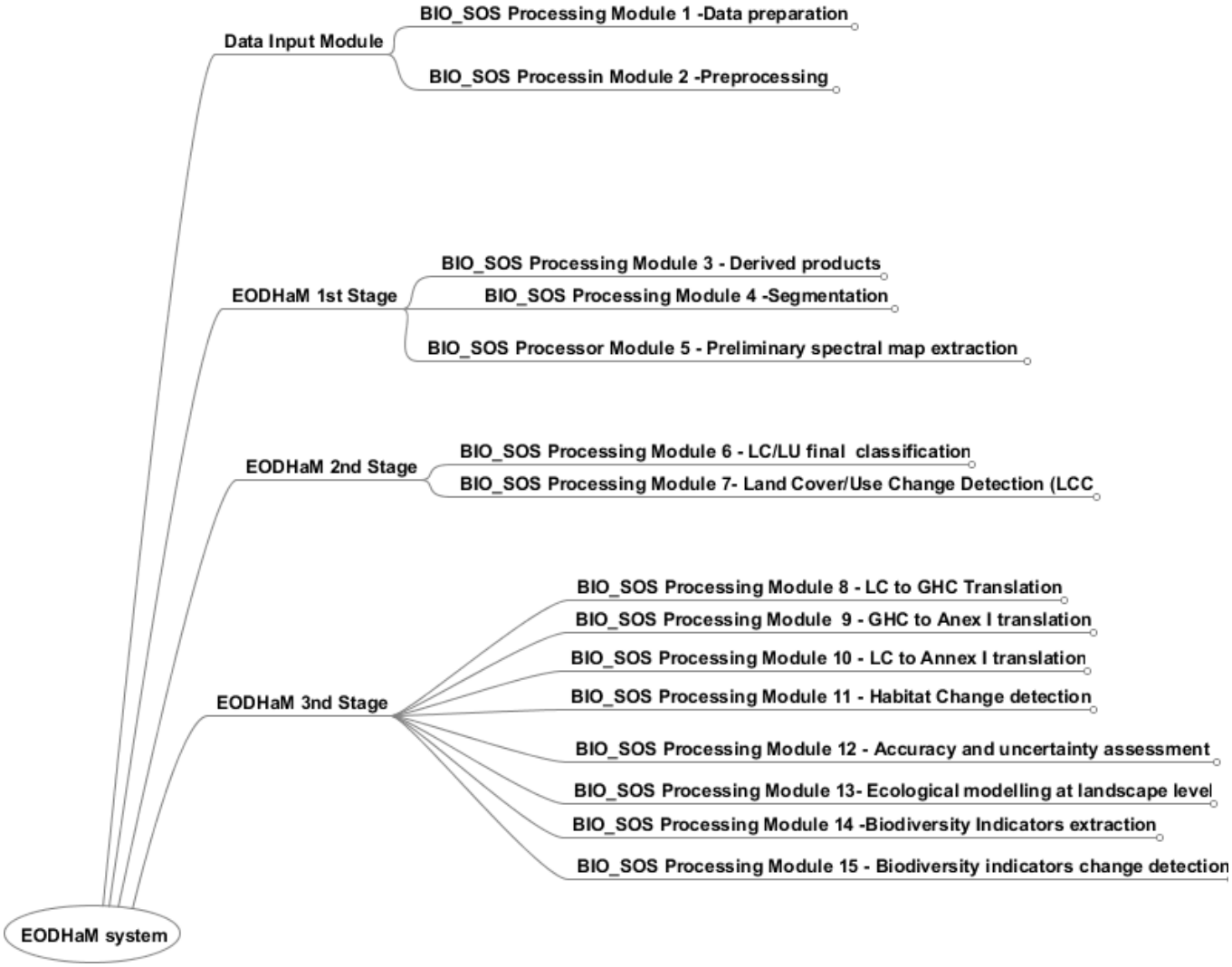


Figure 9: EODHaM system – Bio_sos Processing Modules

Data Input Module :

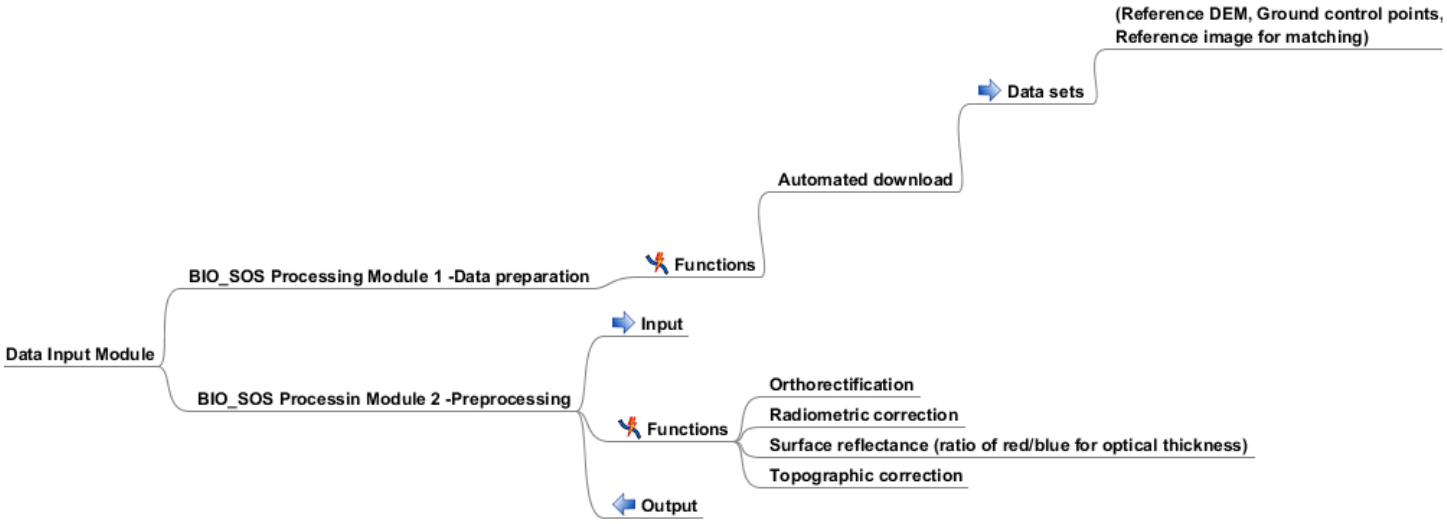


Figure 10: Data Input Module

EODHaM 1st Stage:

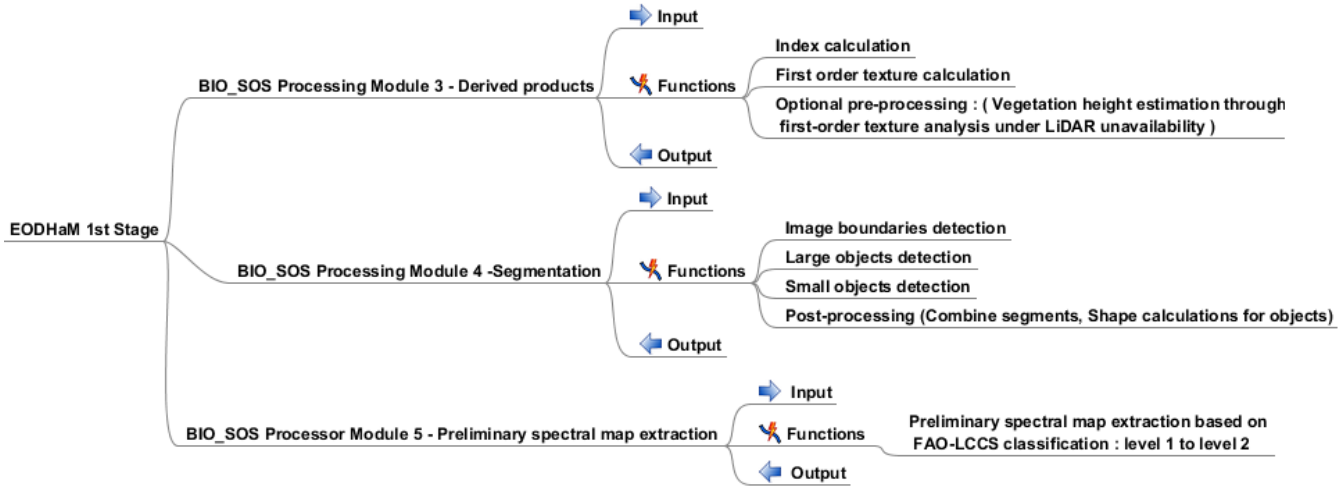


Figure 11: Bio_sos Processor Modules of the 1st Stage

EODHaM 2nd Stage:

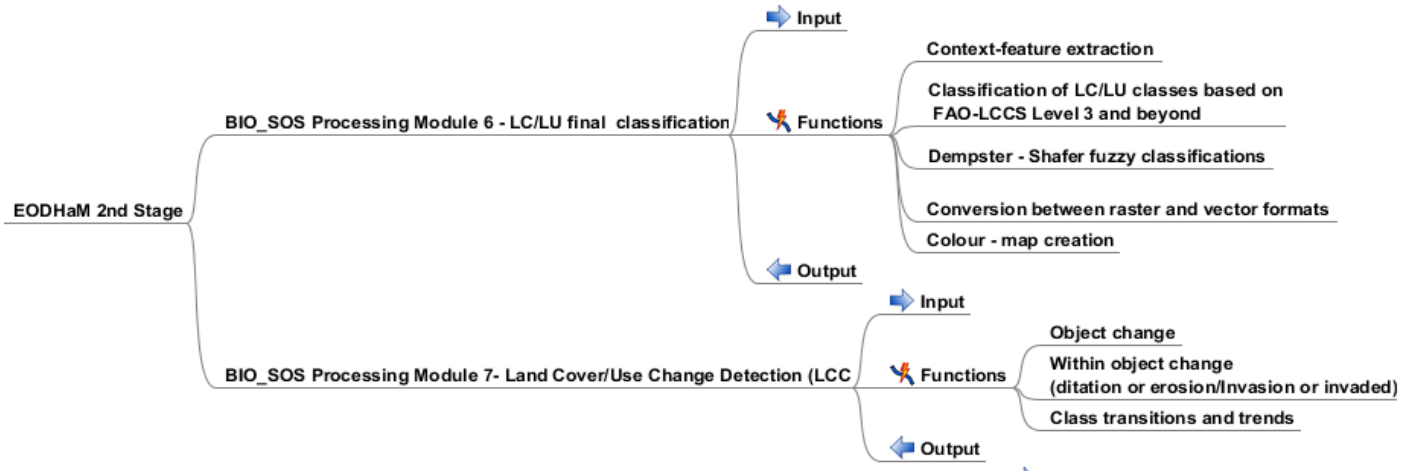
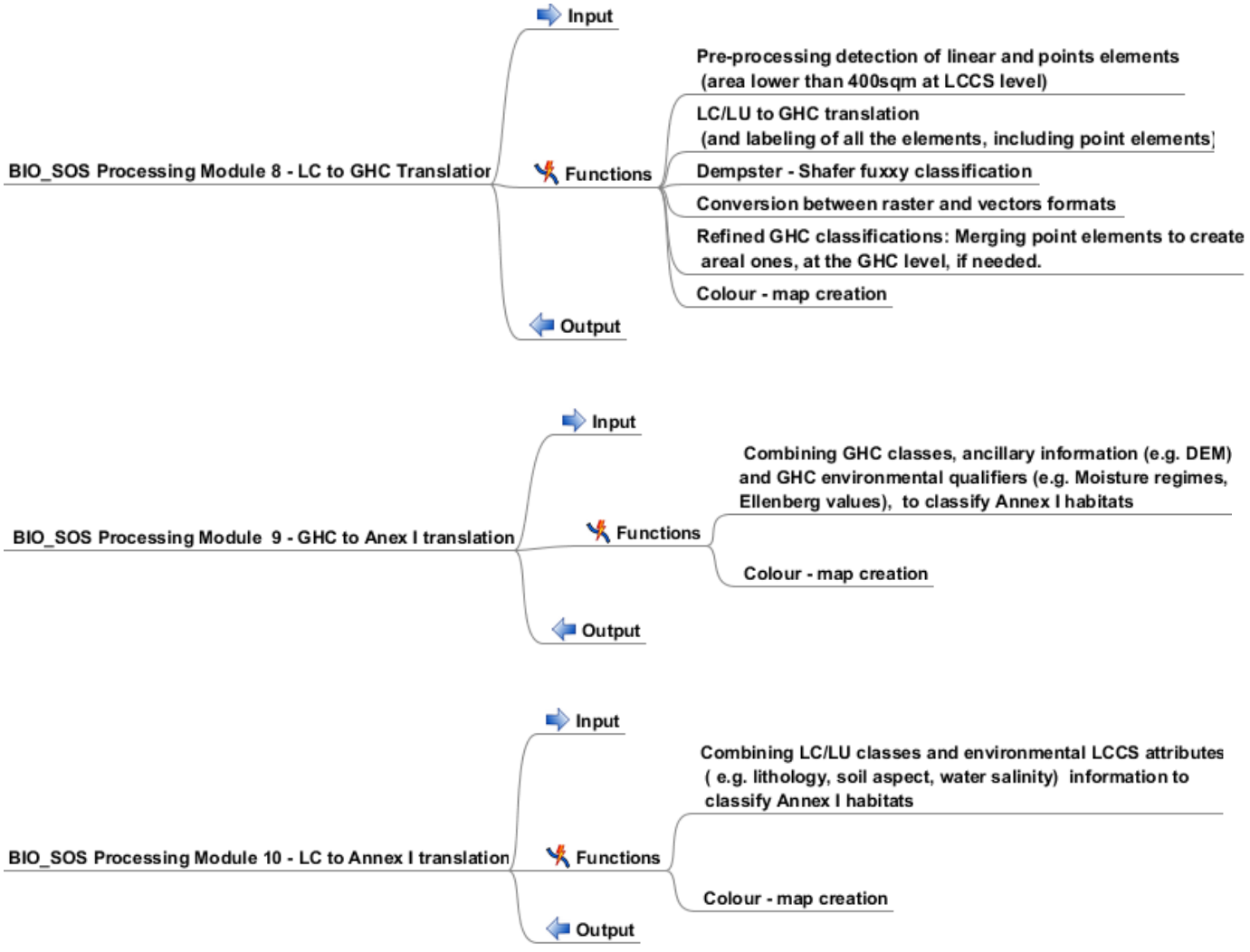


Figure 12: Bio_sos Processor Modules of the 2nd Stage

EODHaM 3rd Stage:



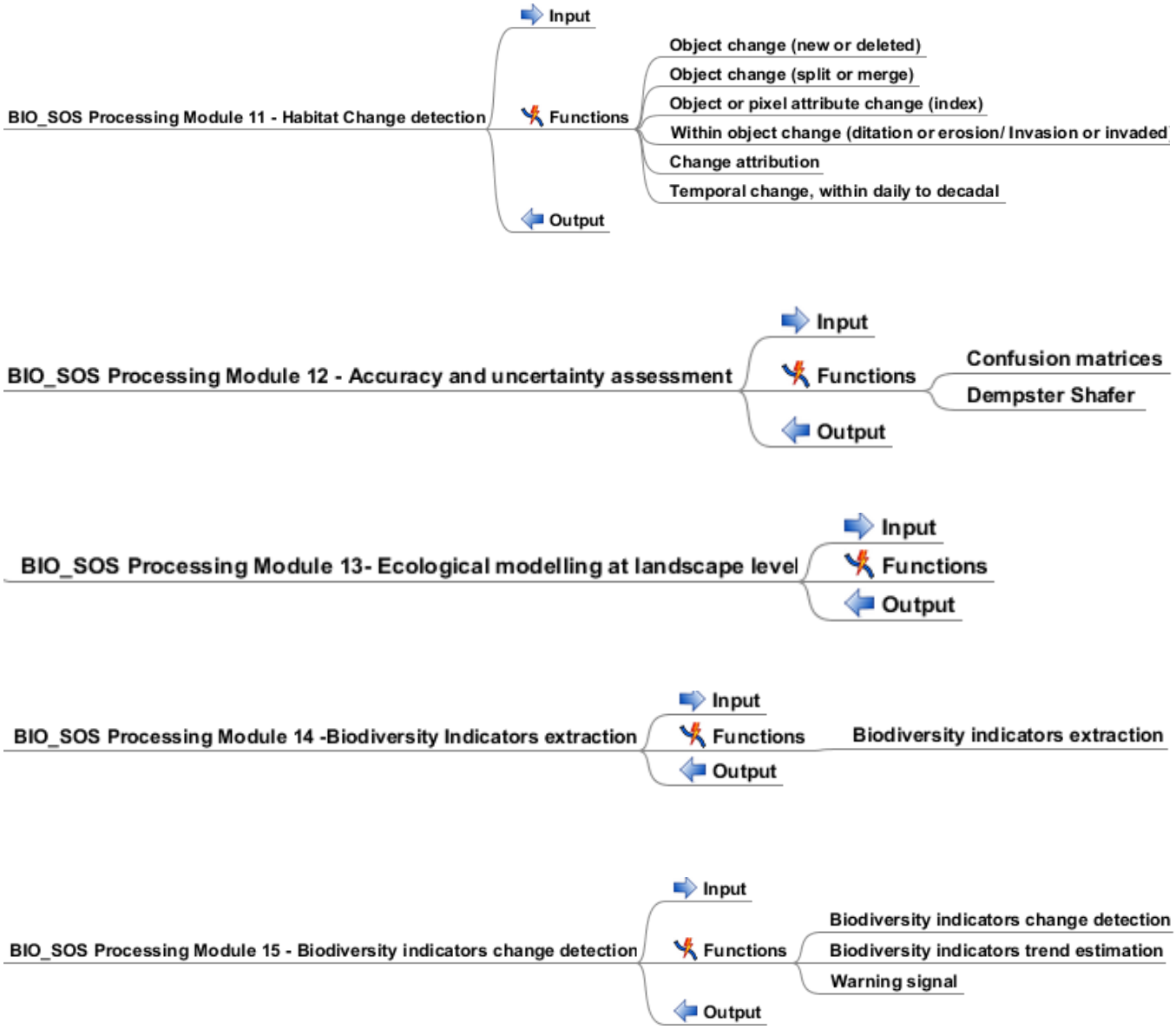


Figure 13: Bio_sos Processor Modules of the 3rd Stage

7.3 Technical Infrastructure of the EODHaM system and Cloud computing

In information technology and on the Internet, infrastructure is the interconnecting hardware and software, including devices that control transmission paths and supports the flow and processing of information.

7.3.1 Cloud computing

Cloud computing is a type of computing that relies on sharing computing resources rather than having local servers or personal devices to handle applications. In cloud computing, the word "cloud" (also phrased as "the cloud") is used as a metaphor for "the Internet," so the phrase cloud computing is used to mean a type of Internet-based computing, where different services -- such as servers, storage and applications -- are delivered to an organization's computers and devices through the Internet. Cloud computing uses networks of large groups of servers typically running on low-cost consumer PC technology with specialized connections to spread data-processing chores across them. This shared IT

infrastructure contains large pools of systems that are linked together. Often, virtualization techniques are used to maximize the power of cloud computing.

The cloud computing technique is more and more acceptable as gives the flexibility to use high computing resources without the financial cost to purchase, deploy and maintain an infrastructure. The common pay-as-you-go subscription model is designed to let end users easily add or remove services and typically the payments are only for what really used.

A leader company that offers cloud computing platform solutions as web Services, is Amazon.com. Amazon Web Services (abbreviated AWS) is a collection of remote computing services (also called web services) that together make up a cloud computing platform, offered over the Internet by Amazon.com.

For the EODHaM infrastructure the following Amazon Web Services, shall be used.

7.3.2 Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud (EC2) allows administrators to rent virtual computers on which to run their own computer applications. EC2 allows scalable deployment of applications by providing a Web service through which a user can boot an Amazon Machine Image to create a virtual machine, which Amazon calls an "instance", containing any software desired. An administrator can create, launch, and terminate server instances as needed, paying by the hour for active servers, hence the term "elastic". EC2 provides administration with control over the geographical location of instances that allows for latency optimization and high levels of redundancy. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing the user to quickly scale capacity, both up and down. Amazon EC2 changes the economics of computing by allowing the user to pay only for capacity that actually used.

7.3.3 Amazon EC2 Instance Store

Each Amazon EC2 instance, unless it's a micro instance, can access storage from disks that are physically attached to the host computer. This disk storage is referred to as *instance store*. Instance store provides temporary block-level storage for use with an Amazon EC2 instance. An instance store is dedicated to a particular instance; however, the disk subsystem is shared among instances on a host computer, as shown in Figure 14.

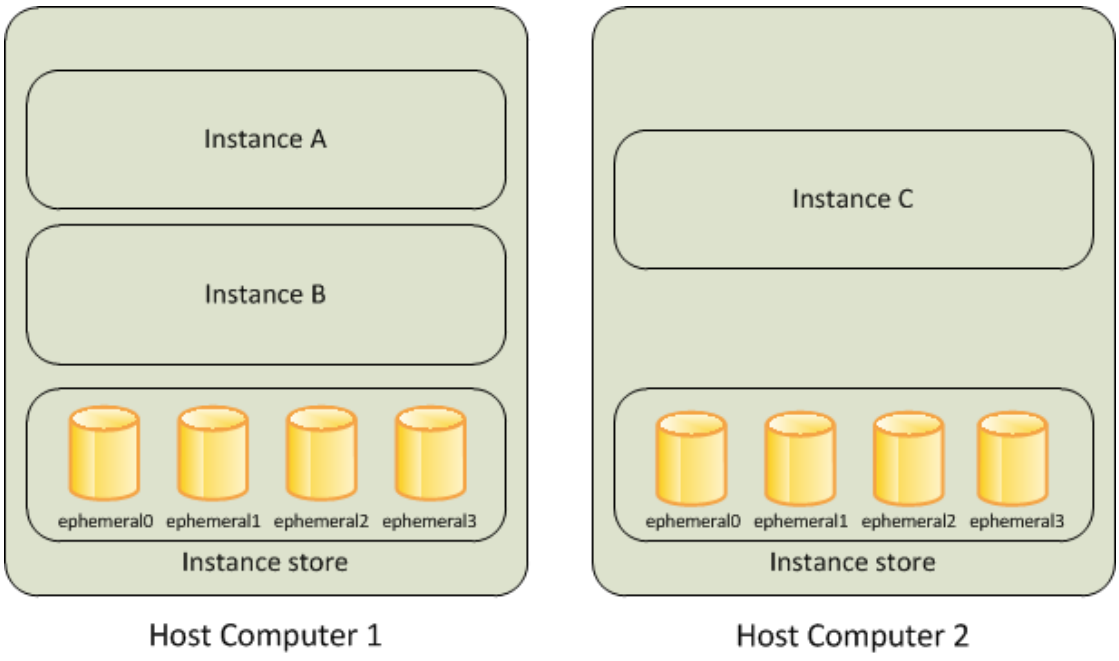


Figure 14: Amazon EC2 common Instance Store

7.3.4 Amazon Elastic Block Store (EBS)

Amazon Elastic Block Store (EBS) provides block level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes are network-attached, and persist independently from the life of an instance. Amazon EBS provides highly available, highly reliable, predictable storage volumes that can be attached to a running Amazon EC2 instance and exposed as a device within the instance. Amazon EBS is particularly suited for applications that require a database, file system, or access to raw block level storage.

Those two Amazon web services can be organized in an infrastructure of Virtual Private Network using the Amazon Virtual Private Cloud (Amazon VPC) web service.

7.3.5 Amazon Virtual Private Cloud (Amazon VPC)

A **virtual private network (VPN)** extends a private network and the resources contained in the network across public networks like the internet. It enables a host computer to send and receive data across shared or public networks by emulating the properties of the private network such as shares, server access, and printers by establishing and maintaining the security and management policies of the private network. This is done by establishing a virtual point-to-point connection through the use of either a dedicated connection or through encryption, or a combination of both.

In the same manner, **Amazon Virtual Private Cloud (Amazon VPC)** lets user provision a private, isolated section of the Amazon Web Services (AWS) Cloud where the user can launch AWS resources in a virtual network according the user definition.

Amazon VPC provides advanced security features such as security groups and network access control lists to enable inbound and outbound filtering at the instance level and subnet level. In addition, the user can store data in Amazon S3 and restrict access so that it's only accessible from instances in the VPC.

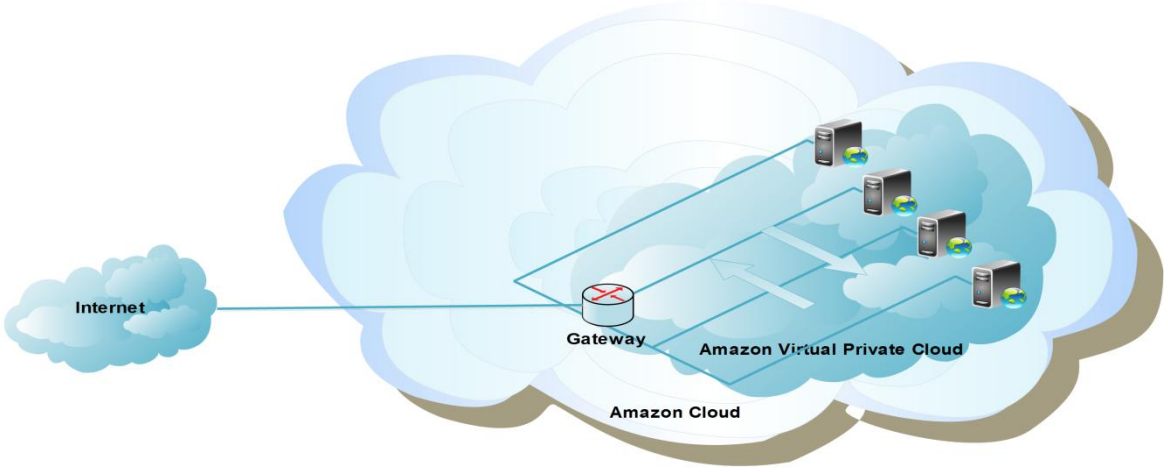


Figure 15: Amazon Virtual Private Cloud

A dedicated Amazon VPC *(VPN) connection* that links an external network with the Amazon VPC can be obtained with the use of a virtual private gateway.

7.3.6 Use of Virtual Machine, a facilitation of the deployment infrastructure

As the project evolves, current and future hardware needs, can change. The Amazon Elastic Compute Cloud, offers as said, the ability to control and in flexible way change the resources of the virtual machines, permitting an adaptable management for the EODHaM system resources. The benefit of that

flexibility, concern not only the level storage for the BIO_SOS products, also the further development of BIO_SOS Processor Modules with extended capabilities.

A **virtual machine (VM)** is a software implementation of a machine that executes programs like a physical machine. A system virtual machine provides a complete system platform which supports the execution of a complete operating system (OS).

One physical machine can be converted in virtual machine Image without any loss of data. That renders the use of the Virtual Machine an optimal tool , as permits the transfer of all the environment, (operating system, BIO_SOS Processor Module, software dependencies and specific configurations), guaranteeing the transfer of an operational status, from the local physical machine to another remote point.

In other case, a recreation of the infrastructure environment from the scratch would be needed, demanding more effort and cost.

For the purpose of the EODHaM system, a base edition of a Virtual Machine Image will be created, with the objective to offer a stable and secure environment of operating system, that includes all the necessary software dependencies for the further deployment of the BIO_SOS Processor Modules.

That EODHaM Virtual Machine Image occurs to be created one time and can be used a replication of the Instance, creating with less effort all the EODHaM machines, guaranties a stable functionality.

In any case if the need of a specific BIO_SOS Processor Module needs a different environment (for example different operating system), a Virtual Machine Image can easily be created and transferred to the EODHaM VPC.

Amazon VM Import/Export tool enables the user to easily import virtual machine images from the user existing environment to Amazon EC2 instances and export them back to the user on-premise environment.

7.3.7 Amazon CloudWatch

Amazon CloudWatch provides monitoring for AWS cloud resources and the applications customers run on AWS. Developers and system administrators can use it to collect and track metrics, gain insight, and react immediately to keep their applications and businesses running smoothly. Amazon CloudWatch monitors AWS resources such as Amazon EC2 and Amazon RDS DB instances, and can also monitor custom metrics generated by a customer's applications and services. With Amazon CloudWatch, the administrator gain system-wide visibility into resource utilization, application performance, and operational health. For Amazon EC2 instances, Amazon CloudWatch **Basic Monitoring** collects and reports metrics for CPU utilization, data transfer, and disk usage activity from each Amazon EC2 instance at a five-minute frequency. Monitoring data is retained for two weeks, even if the AWS resources have been terminated. This enables the administrator to quickly look back at the metrics preceding an event of interest. Basic Monitoring is already enabled automatically for all Amazon EC2 instances, and the administrator can access these metrics immediately in either the Amazon EC2 tab or the Amazon CloudWatch tab of the AWS Management Console.

7.3.8 Amazon Auto Scaling

Auto Scaling allows the administrator to scale the Amazon EC2 capacity, up or down automatically according to user defined conditions. With Auto Scaling, the administrator can ensure that the number of Amazon EC2 instances in use, increases during demand spikes to maintain performance and decreases automatically during demand lulls to minimize costs. Auto Scaling is particularly well suited for applications that experience hourly, daily, or weekly variability in usage.

Auto Scaling enables the administrator to closely follow the demand curve for his applications, reducing the need to provision Amazon EC2 capacity in advance. For example, the administrator can set a condition to add new Amazon EC2 instances in increments of 3 instances to the Auto Scaling Group when the average CPU utilization of the user Amazon EC2 fleet goes above 70 per cent; and similarly, he can set a condition to remove Amazon EC2 instances in the same increments when CPU Utilization

falls below 10 per cent. Often, the administrator may want more time to allow the fleet to stabilize before Auto Scaling adds or removes more Amazon EC2 instances. The administrator can configure a cool-down period for his Auto Scaling Group, which tells Auto Scaling to wait for some time after taking an action before it evaluates the conditions again. Auto Scaling enables the administrator to run the Amazon EC2 fleet at optimal utilization.

7.4 The EODHaM system in the Amazon Virtual Private Cloud

Amazon web services offers different options and choices for the development of a dynamic infrastructure, adaptable to the need of the system. During the processing chain, the processing workload is variable from time interval of peak-load to time interval of inactivity. Amazon offers adaptable solutions to the real needs of each processing time frame.

The EODHaM system shall be deployed as virtual private network through the Amazon Virtual Private Cloud. It is worth noticing that the number of the virtual machine corresponds to current estimation for the EODHaM system. As the BIOSOS project - research evolves and new BIO_SOS Processor Modules will be created, the number of virtual machine as resources can easily be adapted to the EODHaM system needs at any time. A schema presentation of the EODHaM Cloud infrastructure is the following diagram.

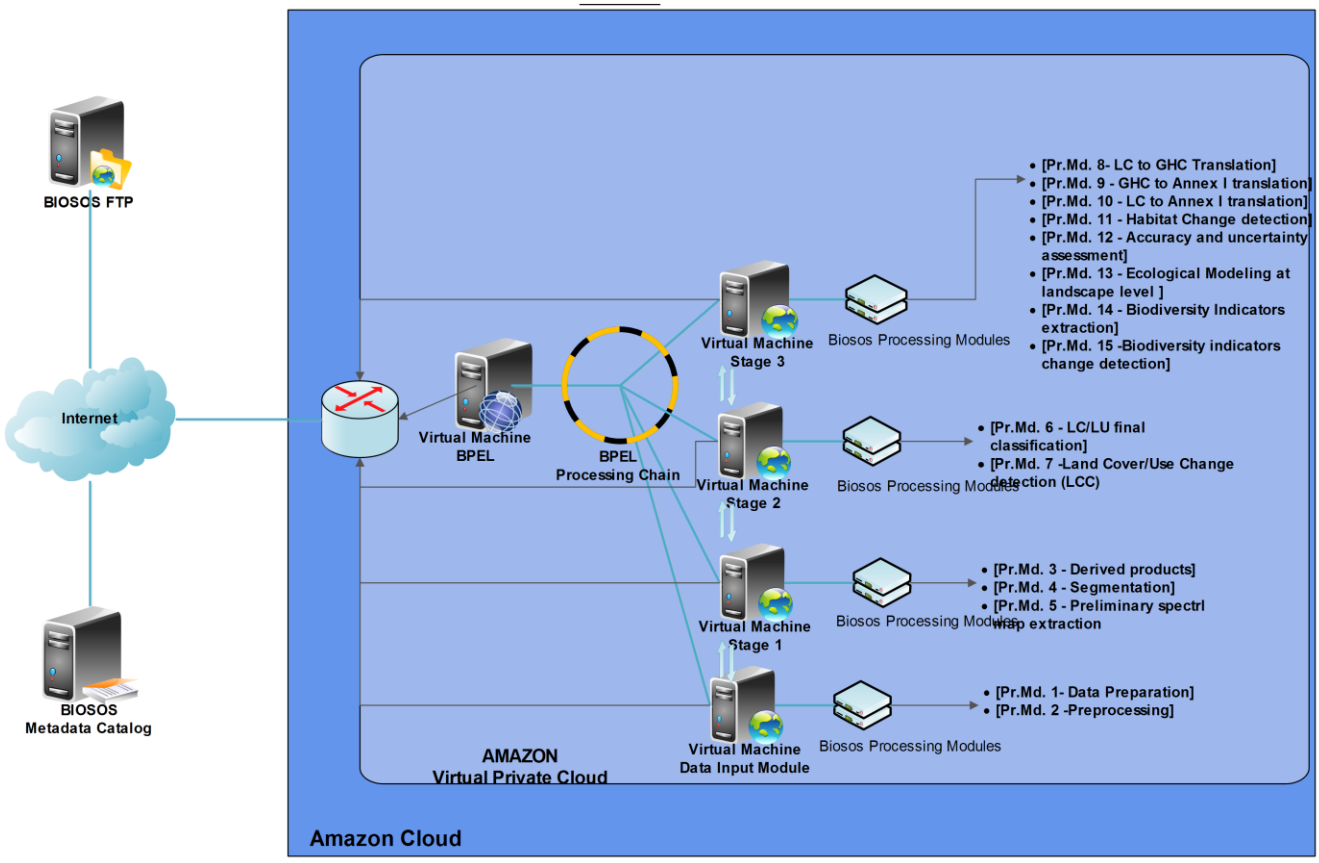


Figure 16: EODHaM system in Amazon Virtual Private Cloud

The information interchange capability that offer the connectivity of the same private network (VPC) can facilitate the transfer of the input and output data between the different BIOSOS Processor Modules, during the execution of the EODHaM processing chain.

For the EODHaM system at least five (5) virtual machines are needed. The infrastructure shall follow the same hierarchical subdivision as has been described previously. Each operational stage shall be a virtual machine server that hosts the dependent **BIO_SOS Processor Modules.** All the BIO_SOS Processors Modules shall be exposed as web services in interaction with the BPEL processing chain controlled by the BPEL virtual machine server.

7.4.1 The EODHaM Virtual Private Cloud

The EODHaM system administrator shall configure the Amazon VPC, creating the first infrastructure that will collect the EODHaM servers. The private network under the Amazon Cloud shall be configured to guaranties security and connectivity with external networks. The use of the private network shall permit a easier and faster transfer communication of BIO_SOS data between the different BIO_SOS Processor Modules, during the execution of the Processing Chain. That will benefit the time execution of the EODHaM processing chain. Indeed the transfer behavior of the network, between EODHaM system components will be similar to a local private network.

7.4.2 The “stage” virtual Machine Image Server

As already mentioned, a Basic Edition of Server shall be built and configured as Virtual Machine Image; its aim consist in setting up the basic environment for the deployment of the BIO_SOS Processor Modules. The operating system shall be open source linux server of 64 bit and will contain all the software dependencies and necessary configurations for the BIO_SOS Processor Module publication as web service (example : Apache Tomcat) .

Will include, already configured, all the basic software packages that could be used for the functionality of the BIOSOS Processor Module like java environment or python environment.

With the Amazon tool VM Import/Export the EODHaM system administrator shall replicate the basic Virtual Machine Image in consideration of the operational EODHaM stage. That procedure is named cloning the Virtual Machine Image, which consists in creating a copy of the same stable basic environment. The aim is to organize the infrastructure following the same EODHaM operational plan and guaranties a stable functional status.

Each Virtual Machine shall be controlled with remote control consoles, permitting the further installation and deployment of the BIO_SOS Processor Modules as web services.

Although the basic model of deployment as described is good enough to cover the technical needs of environment for the BIO_SOS Processor Modules, it is important to be said, that the use of the Cloud infrastructure permits ulterior flexibility and adaptions.

In case for example, of explicit different dependencies of one BIO_SOS Processor Modules, (for example a BIO_SOS Processor Module has operating system restriction to Microsoft Windows), the EODHaM system administrator can add another Virtual Machine Image that will satisfy the requested restriction. In the EODHaM system, the new Virtual Machine shall be considered as sub-stage and the EODHaM system will have the same integrity and functionality as if was deployed under the same "stage" Virtual Machine.

Each “stage” Virtual Machine can have access to the same disk subsystem “Instance storage” and is shared among instances on a host computer. The Amazon EC2 Instance storage is a convenient easy way to administrate the data flow of BIO_SOS products between the BIO_SOS Processor Modules. See paragraph Amazon EC2 Instance Store for more details.

The system administrator can monitor the system behaviour in continuous time with the Amazon tool CloudWatch and can adapt the resources of the VPC and of each virtual Image instance in real time. That offers a maximum flexibility not only in dependency of the BIO_SOS Processor Module requirements, but also in dependence of the time execution of each Processor Module. During BIOSOS Processor Module execution, the administration can increase the CPU of the host VMI for best performance. After the end of the processing activity the administrator can decrease the unnecessary

resources. In that way, the administrator can balance the cost of the Amazon resource, in directly dependency with the real needs of the EODHaM system.

7.4.3 The BPEL Server

The same procedure than for the Virtual Machine Image shall be followed for the BPEL Server. The dedicated Virtual Machine shall contain all the necessary software dependencies like java environment and the BPEL Engine for jboss (riftsaw). The BPEL Engine shall have the rule of the central controller for the orchestration of the processing chain.

The use of VMI permits also an easy backup of them, to be protected by system failures.

8. Appendices

Appendix I. Acronym and Abbreviation List

ATREE	Ashoka Trust for Research in Ecology and the Environment – India
ADD	Architecture Design Document
Amazon EC2	Amazon Elastic Compute Cloud
API	Application Programming Interface
AWS	Amazon Web Service
BIO_SOS	BIOdiversity multi-SOurce monitoring System: from Space TO Species
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execition Language for Web Services
BPM	Business Process Management
BIO_SOS	Biodiversity Multi-Source Monitoring System: From Space To Species
Cal/Val	Calibration and Validation
CEOS	Committee of Earth Observations
CERTH	Informatics And Telematics Institute Of The Centre For Research And Technology – Greece
CIBIO	Research Center in Biodiversity and Genetic Resources (Portugal)
CNR	Consiglio Nazionale delle Ricerche
CPU	Central Processing Unit
DEM	Digital Elevation Model
DN	Digital Number
DVD	Digital Versatile Disk
EBS	Elastic Block Store
EBONE	European Biodiversity Observation Network
EO	Earth Observation
EODHaM	EO Data for Habitat Monitoring
ESB	Enterprise Service Bus
EU	European Union
FAO	Food and Agriculture Organization
FAO-LCCS	FAO - Land Cover Classification System
FP7	Seventh Framework Program
FTP	File Transfer Protocol
GCP	Ground Control Point

GEOSS	Global Earth Observation System of Systems
GHC	General Habitat Categories
GLC	Global Land Cover
GMES	Global Monitoring for Environment and Security
GWT	Google Web Toolkit
HR	High Resolution
INSPIRE	Infrastructure for Spatial Information in Europe
ISO	International Organization for Standardization
JBoss	JavaBeans Open Source Software Application Server
LC	Land Cover
LCC	Land Cover Change
LCCS	Land Cover Classification System
LU	Land Use
MS	Multispectral
ODE	Orchestration Director Engine
PKH	Planetek Hellas
PKI	Planetek Italia
QA4EO	Quality Assurance Framework for Earth Observation
QAP	Quality Assurance Plan
QI	Quality Indicator
RDS	Relational Database Service
RS	Remote Sensing
RS-IUS	Remote Sensing Image Understanding System
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SDD	Service Design Document
SLA	Service Level Agreement
SRC	Spectral Rule-based Classifier
SURF	Surface Reflectance
TOA	Top Of Atmosphere
TOARF	Top Of Atmosphere Reflectance
UDDI	Universal Description Discovery and Integration
VHR	Very High Resolution
VM	Virtual Machine
VMI	Virtual Machine Image

VPC	Virtual Private Cloud
VPN	Virtual Private Network
WGCV	CEOS Working Group on Calibration and Validation
WP	Work Package
WS	Web Service
WSDL	Web Service Description Language
XML	eXtensible Markup Language

9. References

1. Web Service,
   - http://en.wikipedia.org/wiki/Web_service
   - http://www.w3schools.com/webservices/ws_intro.asp
2. Web Services Business Process Execution Language (WS-BPEL) ,
   - http://docs.oasis-open.org/wsbpel/2.0/varprop
   - http://www.information-management.com/infodirect/20060602/1055720-1.html?zkPrintable=1&nopagination=1
3. Business Process Execution Language for Web Services (BPEL4WS),
   - http://xml.coverpages.org/bpel4ws.html
4. Web Services Description Language (WSDL) 1.1 ,
   - http://www.w3.org/TR/wsdl
   - http://www.w3.org/TR/ws-desc-usecases/
   - http://www.w3.org/TR/ws-arch/
5. Web Services Description Language (WSDL) Version 2.0,
   - http://www.w3.org/TR/wsdl20/
6. Simple Object Access Protocol,
   - http://en.wikipedia.org/wiki/SOAP
   - http://www.w3schools.com/soap/soap_intro.asp
   - http://www.w3.org/TR/ws-addr-soap/
7. Service-Oriented Architecture,
   - http://en.wikipedia.org/wiki/Service-oriented_architecture
   - http://msdn.microsoft.com/en-us/library/aa480021.aspx
8. Jboss Application Server,Web service framework,
   - http://www.jboss.org/jbossws
9. Apache ODE, BPEL, Language Guide, WS-BPEL 2.0 Specification Compliance ,
   - http://ode.apache.org/ws-bpel-20-specification-compliance.html
10. Riftsaw 2.3.0.Final,Getting Started Guide,
    - http://docs.jboss.org/riftsaw/releases/2.3.0.Final/gettingstartedguide/pdf/GettingStartedGuide.pdf
11. Riftsaw 2.3.0. Final, User Guide,
    - http://docs.jboss.org/riftsaw/releases/2.3.0.Final/userguide/pdf/UserGuide.pdf
12. Amazon Elastic Compute Cloud (Amazon EC2),
    - http://aws.amazon.com/ec2/
13. Amazon Virtual Private Cloud
    - http://ode.apache.org/ws-bpel-20-specification-compliance.html

14. Amazon Elastic Block Store (EBS),
    - http://aws.amazon.com/ebs/
15. Amazon CloudWatch,
    - http://aws.amazon.com/cloudwatch/
16. Amazon VM Import/Export,
    - http://aws.amazon.com/ec2/vmimport/
17. Virtual Machine,
    - http://en.wikipedia.org/wiki/Virtual_machine