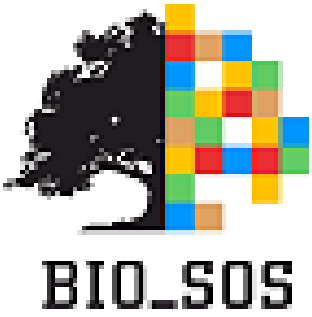

Project Title:	**BIO_SOS Biodiversity Multisource Monitoring System: from Space TO Species**
Contract No:	FP7-SPA-2010-1-263435
Instrument:	Collaborative Project
Thematic Priority:	FP7-SPACE-2010-1
Start of project:	1 December 2010
Duration:	36 months

Deliverable No: D3.3

Design Justification File

Due date of deliverable:	31/10/2012
Actual submission date:	04/12/2012

Version:1st version of D3.3

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

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

Deliverable title	DJF – Design Justification File
Deliverable number	TBC
Deliverable version	v1
Previous version(s)	
Contractual date of delivery	TBC
Actual date of delivery	04/12/2012
Deliverable filename	BIO_SOS_D3.3_DJF_Design_Justification_File_v1
Nature of deliverable	R = Report
Dissemination level	PU = Public
Number of pages	14
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	D3.3 describes the reasons and the justification of that technological choice as the most appropriate technical solution for the EODHaM system development
Keywords	System, architecture, workflow

Signatures

Written by	**Responsibility- Company**	**Date**	**Signature**
Jens Stutte	WP 3 Leader (PKI)	30/12/2012	
Dimitrios Karachalios	Development (PKH)	04/12/2012	
Verified by		TBC	
Jens Stutte	WP 3 Leader (PKI)	30/12/2012	
Approved by			
Palma Blonda	Project Coordinator, CNR	04/12/2012	
Fifamè Koudogbo	Quality Team, AI	03/12/2012	

Table of Contents

1. Executive summary

A Design Justification File is an explicit documentation of the reasons behind decisions made when designing the EODHaM system

The document is in direct relation with the concepts as described on the Architecture Design Document.

The aim of this document is to explain the reasons and the benefits of the technological choices made for the realization of the EODHaM system. Each technological choice has been studied to fit the requirements of the system, offering maximum benefit to the system. The Service Oriented Architecture applied with Business Process Execution Language is a winner combination that offers system interoperability and functionality. The applied concept of Modular programming also provide the necessary development flexibility of the BIO_SOS Processor Modules. The choice of cloud computing , provides a flexible environment for implementation and cooperation. The result of that choices is to create a persistent stable and functional system that will permit the efficient monitoring and processing activities of the BIO_SOS project.

2. Introduction

2.1. General Context

The EODHaM system, is defined as a pre-operational system of work-flow administration that can serves the aim of the project for a multi-annual monitoring of NATURA 2000 site and their surrounding areas, as other ecologically sensitive sites.

The architectural design of the system is based to the Service-Oriented Architecture (SOA) with 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. In the following chapters will be described the reasons and the justification of that technological choice as the most appropriate technical solution for the system development.

2.2. A word on terms, assumptions and choices

2.2.1. Prior knowledge, ancillary data and meta-data

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, they thus do not adapt to data and do not change with time. Prior knowledge is acquired by a supervisor or expert (human) based on intuition, 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. “Meta-data” 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, will provide the guidelines for the meta-data 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 Processing Chain and SOA

The principal goal of the EODHaM system is to administrate the complex sequence of processing tasks executed by the individual BIO_SOS Processors. The arrangement of the processing tasks results as a unique processing chain.

The processing chain can be characterized by dynamic type in terms of decision differentiable during the work flow execution based on dependencies to different parameters, as the operational status of the participative components and the nature of their derivative.

For the successfully interoperability of the Processing chain, the interchange capabilities of the participant components have to be ensured at maximum level of interaction.

Service oriented architecture (SOA) is an architecture where independent systems and applications communicate with each other by exposing and using web services. Web Services are defined using open standards, making inter-communication much easier to implement, and less dependent on proprietary communication protocols.

The Service Oriented Architecture permits system developing 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.

SOA offers a fast, always available, secure and affordable way to integrate, and provide visibility and business insight across many different platforms, data sources and applications.

3.1. BPEL Orchestration

A BPEL orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer.

BPEL orchestration plays a key role in delivering service-oriented architecture (SOA) benefits, contributing to easy service reuse for lower SOA costs and to quick process change for business agility. Easier to use than Java or C+, BPEL (Business Process Execution Language) is the widely accepted standard for combining, coordinating, and controlling the work-flow of Web services into an end-to-end business process.

Specifically, with BPEL orchestration capabilities, organizations can:

- Compose processes out of existing services and processes.
- Correlate events within and across a running process.
- Control complex flows such as conditionals, loops, delays, and scoped state.
- Compensate for completed activities in the event of failure.
- Manage concurrent, long-running service interactions.

3.1.1. BPEL Design Goal

There were ten original design goals associated with BPEL:

1. Define business processes that interact with external entities through web service operations defined using WSDL1.1, and that manifest themselves as Web services defined using WSDL 1.1. The interactions are “abstract” in the sense that the dependence is on portType definitions, not on port definitions.
2. Define business processes using an XML-based language. Do not define a graphical representation of processes or provide any particular design methodology for processes.
3. Define a set of Web service orchestration concepts that are meant to be used by both the external (abstract) and internal (executable) views of a business process. Such a business

process defines the behavior of a single autonomous entity, typically operating in interaction with other similar peer entities. It is recognized that each usage pattern (i.e., abstract view and executable view) will require a few specialized extensions, but these extensions are to be kept to a minimum and tested against requirements such as import/export and conformance checking that link the two usage patterns.

4. Provide both hierarchical and graph-like control regimes, and allow their use to be blended as seamlessly as possible. This should reduce the fragmentation of the process modeling space.
5. Provide data manipulation functions for the simple manipulation of data needed to define process data and control flow.
6. Support an identification mechanism for process instances that allows the definition of instance identifiers at the application message level. Instance identifiers should be defined by partners and may change.
7. Support the implicit creation and termination of process instances as the basic lifecycle mechanism. Advanced lifecycle operations such as "suspend" and "resume" may be added in future releases for enhanced lifecycle management.
8. Define a long-running transaction model that is based on proven techniques like compensation actions and scoping to support failure recovery for parts of long-running business processes.
9. Use Web Services as the model for process decomposition and assembly.
10. Build on Web services standards (approved and proposed) as much as possible in a composable and modular manner.

3.1.2. Asynchronous Process BPEL management

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.

One of the BPEL benefits is that it permits the management of Asynchronous process. It is a notable capability, giving to the EODHaM system the flexibility to manage different invocations of the BIO_SOS web services Processor Modules.

The BPEL Asynchronous management mechanism reproduces real life behavior to the EODHaM system. During the execution of a BPEL processing chain, the web services are invoked in asynchronous way. That means that the EODHaM system for each request creates an instance with three statuses of the execution. (Invoke, Standby, Reply)

Invoke the web services: The EODHaM system send the request to the web service - BIO_SOS Processor Module.

Standby status of the instance: The instance of the request remains to a standby status to receive a reply from the web service or until the defined temporal time of the standby section expires (for example after 48 hours).

Reply of the web service: The web service – BIO_SOS Processor Module has finished the processing activity and replies the result to the EODHaM system. In case of error, the response is an error message.

In that way the administration mechanism of the workflow of the processing chain is not blocked until the final execution of each processing activity. Instead different instances of requests can be managed in parallel mode.

3.2. BIO_SOS Processor Module as Web Service

A Web service is a method of communication between two electronic devices over the World Wide Web.

The 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 Web Services Description Language, known by the acronym 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.

Developing the BIO_SOS Processor Modules as Web Services, we obtain the benefit of maximum interoperable BIO_SOS Processor Module -to- BIO_SOS Processor Module interaction over a network.

Here are the benefits of using Web Services

- **Exposing the existing function on to network:**

A Web service is a unit of managed code that can be remotely invoked using HTTP, that is, it can be activated using HTTP requests. So, Web Services allows you to expose the functionality of your existing code over the network. Once it is exposed on the network, other application can use the functionality of your program.

- **Connecting Different Applications and Interoperability:**

Web Services allows different applications to talk to each other and share data and services among themselves. Other applications can also use the services of the web services. For example VB or .NET application can talk to java web services and vice versa. So, Web Services are used to make the application platform and technology independent.

- **Standardized Protocol:**

Web Services use standardized industry standard protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description and Service Discovery layers) use the well-defined protocol in the Web Services protocol stack. This standardization of protocol stack gives the business many advantages such as wide range of choices, reduction in the cost due to competition and increase in the quality.

For the EODHaM system the WSDL standard is selected:

WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet. A client program connecting to a web service can read the WSDL to determine what functions are available on the server. Any special data-types used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the functions listed in the WSDL.

- **Low Cost of communication:**

Web Services uses SOAP over HTTP protocol for the communication, so you can use existing low cost internet for implementing Web Services. This solution is much less costly compared to proprietary solutions like EDI/B2B. Beside SOAP over HTTP, Web Services can also be implemented on other reliable transport mechanisms like FTP etc.

3.3. BIO_SOS Processor and Modular Programming concept

Each BIO_SOS Processor Module has different functions and processing activities and results a distinct component unit, well incorporated to the system.

One key solution to managing complexity of large software is modular programming: the code is composed of many different code modules that are developed separately. This allows different developers to take on discrete pieces of the system and design and implement them, separately, but always following a right logic of module interconnection and functional system cooperation.

Modular programming (also called "top-down design" and "stepwise refinement") is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable

modules (Figure 1), such that each contains everything necessary to execute only one aspect of the desired functionality. Conceptually, modules represent a separation of concerns, and improve maintainability by enforcing logical boundaries between components. Modules are typically incorporated into the program through interfaces. A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface.

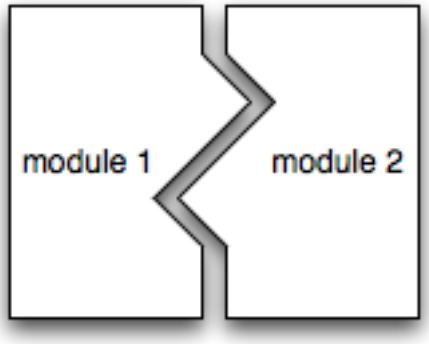

Figure 1: Modular Programming Concept

For the BIO_SOS Processor Module (Figure 2), the applied Modular programming has permitted the separation of concepts.

The module “Processing core” has all that functions that permits the processing concrete activity of the Processor. The expected implemented functions of that module are conformed to the concept of receiving input data, elaborating them with algorithmic sub routines and extracting the output data.

The module “Processor wrapper” defines a module abstraction to specific classes and interfaces capable to expand the “BIO_SOS Processor core” to a web service and permits the management of the Processor functions with the specified methods.

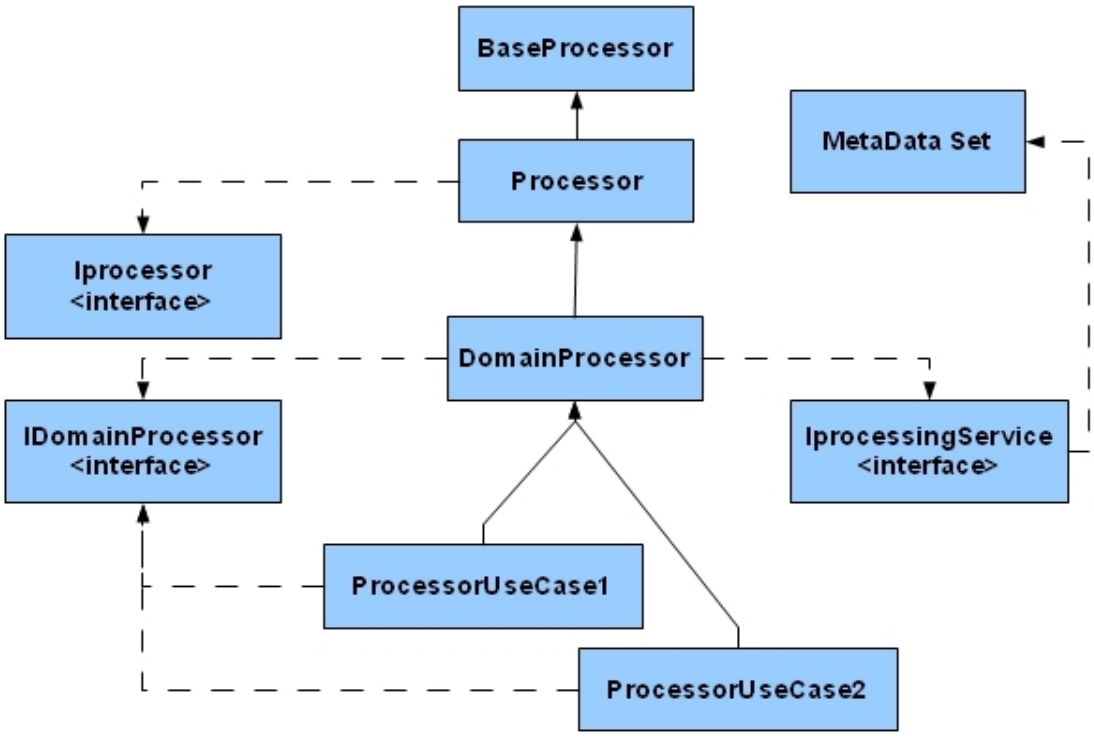

Figure 2: Bio_SOS Processor Wrapper classification

The most important are the Processor, Base Processor and Domain Processor classes with the corresponding interfaces.

Processor:

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

Base Processor:

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 meta-data parsing/manipulation.

Domain Processor:

The Class **DomainProcessor** instantiates a new domain processor, creates the processor configuration, returning the description of the concrete processing and generates the Prediction Meta-data.

Adopting the same modular programming concept for all the BIO_SOS Processors we obtain common implementation methods that arrange the behavior of each BIO_SOS Processor in common lines of functionality but also permit in the same time different configurations of parameters for each of them.

In such way, we have positive impact to the development and to the maintainability of the BIO_SOS Processors as EODHaM components.

For example, during BIO_SOS Processor "Processor core module" upgrading operations , in terms of code modification that have to do with the concrete processing functions, the Processor wrapper does not have to be changed as keeps separately those functions that expand the interaction of the Processor as web service with the rest of the functions of the concrete "core" processing.

4. Cloud

4.1. Hardware resources management

The EODHaM system has variable usage of hardware resources. That consideration has two aspects.

The implementation aspect:

As the system evolves, new BIO_SOS Processor Modules can be added to the system, enriches the system with new functionalities and processing capabilities. In the point of view of hardware resources, that means that the system has to be adaptable to the new requirements and that can allow expansion with hardware addition or hardware updates.

The hardware resources usage:

During execution of processing chain, the hardware resources of CPU, RAM, disk that are used present a spiky diagram of resource usage between inactivity (time interval of standby status) and intensive use of the machine hardware resources for elaborated calculation of the process (Figure 3).

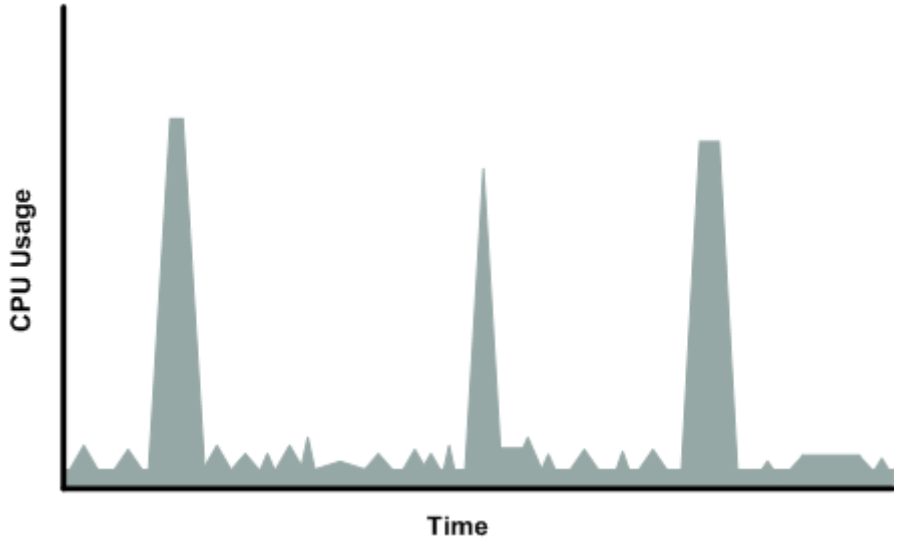

Figure 3: Expected CPU usage profile of BIO_SOS Processors Modules

The decision to adopt the cloud computing offers the right balance between costs of the needed machine hardware resources, costs of the hardware maintenance and performances of the system with flexibility to have hardware expansions whenever it is required.

The pricing policy of the Amazon (one of the leader on cloud computing web services), which is created around the concept “pay only what you use”, permits to keep the development costs at reasonable levels. Indeed the EODHaM system has a profile of variable usage in which the costs are in direct correspondence with the real usage of intensive activity for each time interval of the processing chain.

The benefit of the cloud computing is not only limited to that two aspects. Techniques as the remote control of the machine over the internet facilitates the development in collaboration of the team as different developers can access the same machine and collaborate, without limitations that are usually encountered when the machine is situated under local private networks.

The usage of Virtual Machine Images also gives flexibility and is an optimal tool for the backup, not only of the component, but of the entire Processor operating environment.

5. Annex 1: Acronyms and Abbreviations

B2B	Business-to-business
BIO_SOS	BIOdiversity multi-SOurce monitoring System: from Space TO Species
BPEL	Business Process Execution Language
Cal/Val	Calibration & Validation
CEOS	Committee of Earth Observations
CPU	Central Processing Unit
DJF	Design Justification File
DOW	Descrption of Work
EDI	Electronic Data Interchange
EODHaM	EO Data for Habitat Monitoring
HTTP	Hypertext Transfer Protocol
QA4EO	Quality Assurance Framework for Earth Observation
QI	Quality Indicator
RAM	Random Access Memory
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
W3C	World Wide Web Consortium
WGCV	Working Group on Calibration and Validation
WP	Workpackage
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Services Description Language
XML	Extensible Markup Language