NIVA – NEW IACS VISION IN ACTION

WP2 - Large-Scale Pilot

D2.4 Use Case Component Development

Deliverable Lead: TRAGSA

Deliverable due date: 31/12/2021

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 842009.

Document Control Page

Document Control Page	
Title	D2.4 Use Case Component Development
Creator	Jesús Estrada, Miriano Esposito
Description	This documents explains the ICT components and their source code as implemented across the nine NIVA Use Cases and released under the EU PL license.
Publisher	"NIVA - New IACS Vision in Action" Consortium
Contributors	xx
Date of delivery	dd/mm/yyy
Type	Text
Language	EN-GB
Rights	Copyright "NIVA - New IACS Vision in Action"
Audience	☒ Public ☐ Confidential ☐ Classified
Status	☒ In Progress ☐ For Review ☐ For Approval ☐ Approved

Revision History			
Version	**Date**	**Modified by**	**Comments**
0.1	05/10/2020	Miriano Esposito, AGEA Tommaso Iannucci, AGEA	Initial version, Table of Contents
0.2	16/12/2020	Miriano Esposito, AGEA Tommaso Iannucci, AGEA Ada Mancuso AGEA	Removed subparagraph "Integration interfaces" and "Web service inbound and outbound". Changed label from "Communication" to "Communication protocol"
0.3	13/01/2021	Miriano Esposito, AGEA Tommaso Iannucci, AGEA Ada Mancuso AGEA	Added UC4a
0.4	26/01/2021	Miriano Esposito, AGEA Tommaso Iannucci, AGEA Ada Mancuso AGEA	Added UC5a
0.5	05/02/2021	Miriano Esposito, AGEA Tommaso Iannucci, AGEA Ada Mancuso AGEA	Filled chapters 2 and 7, and finalized.

Copyright © NIVA Project Consortium

Disclaimer

This document is issued within the frame and for the purpose of the NIVA project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 842009. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

This document and its content are the property of the NIVA Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the NIVA Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the NIVA Partners.

Each NIVA Partner may use this document in conformity with the NIVA Consortium Grant Agreement provisions.

Copyright © NIVA Project Consortium

Table of Contents

Document Control Page 2
Table of Contents 4
1. Purpose 9
2. Acronyms and Definitions 10
3. Introduction 11
4. Executive Summary 12
5. NIVA Reference Architecture 13
5.1 Catalog of NIVA requirements & user needs addressed 14
5.2 Components developed & Interfaces 16
5.3 Data Harmonization. WP3, WP5 support 20
5.3.1 OpenEO API layer 20
5.3.2 Sen4CAP 20
5.3.3 CEF Components 22
5.4 Common Components and WP4 support 23
5.5 Expected components from subcontracting and Open Call 23
6. Use cases contributions to NIVA components (eTools) 25
6.1 UC1a - Earth Observation Monitoring and Traffic Lights Text 25
6.1.1 Earth Observation Monitoring and Traffic Lights DSS 25
6.1.1.1 Contact information (author, technical reference) 25
6.1.1.2 Features description 25
6.1.1.3 System requirements 26
6.1.1.3.1 Hardware 26
6.1.1.3.2 Software 26
6.1.1.3.3 Communication protocol 27
6.1.1.3.4 Repository access (link and othes information) 27
6.1.1.3.5 Dependencies 27
6.1.1.3.6 Other modules 27
6.1.1.3.7 Integration interfaces 27
6.1.1.3.8 Web services inbound and outbound 27
6.2 UC1b - Agro-environmental monitoring 28

Copyright © NIVA Project Consortium

6.2.1 Component 1: Carbon Indicator TIER1 28
6.2.1.1 Contact information (author, technical reference) 28
6.2.1.2 Features description 28
6.2.1.3 Architecture description and technology stack 28
6.2.1.4 System requirements 28
6.2.1.4.1 Hardware 28
6.2.1.4.2 Software 28
6.2.1.5 Communication protocol 28
6.2.1.6 Repository access (link and other information) 29
6.2.1.7 Libraries 29
6.3 UC1c - Farmer Performance 30
6.3.1 Component "IACS-FMIS data sharing API" 30
6.3.1.1 Contact information (author, technical reference) 30
6.3.1.2 Features description 30
6.3.1.3 Architecture description and technology stack 31
6.3.1.4 System requirements 32
6.3.1.4.1 Hardware 32
6.3.1.4.2 Software 32
6.3.1.5 Communication protocol 32
6.3.1.6 Repository access (link and other information) 34
6.4 UC2 - Prefilled application, GSAA/Land link 35
6.4.1 Component "Robot Data Harvesting Tool" (also known as "Robot Process Automation" (RPA) or "Robot Framework") 35
6.4.1.1 Contact information (author, technical reference) 35
6.4.1.2 Features description 35
6.4.1.3 Architecture description and technology stack 36
6.4.1.4 System requirements 36
6.4.1.4.1 Hardware 36
6.4.1.4.2 Software 37
6.4.1.5 Communication protocol 37
6.4.1.6 Repository access (link and other information) 38
6.4.1.7 Libraries 39
6.4.2 Component Preliminary parcel boundary automatic delineation 40

Copyright © NIVA Project Consortium

6.4.2.1 Contact information (author, technical reference) 40
6.4.2.2 Features description 40
6.4.2.3 Architecture description and technology stack 40
6.4.2.4 System requirements 41
6.4.2.4.1 Hardware 41
6.4.2.4.2 Software 41
6.4.2.5 Repository access (link and other information) 41
6.4.2.6 Dependencies 41
6.4.2.7 Libraries 42
6.4.3 Component Sen4CAP system results as preliminary data 42
6.4.3.1 Contact information (author, technical reference) 42
6.4.3.2 Features description 42
6.4.3.3 Architecture description and technology stack 42
6.4.3.4 System requirements 42
6.4.3.4.1 Hardware 43
6.4.3.4.2 Software 43
6.4.3.5 Repository access (link and other information) 43
6.4.3.6 Dependencies 43
6.4.3.7 Other modules 43
6.5 UC3 - Farm Registry 44
6.5.1 Component Farm Registry 44
6.5.1.1 Contact information (author, technical reference) 44
6.5.1.2 Features description 44
6.5.1.3 Architecture description and technology stack 44
6.5.1.4 System requirements 46
6.5.1.4.1 Hardware 46
6.5.1.4.2 Software 46
6.5.1.5 Communication protocol 46
6.5.1.6 Repository access (link and other information) 47
6.5.1.7 Dependencies 47
6.6 UC4a - Geotagged photos 48
6.6.1 Contact information (author, technical reference) 48

Copyright © NIVA Project Consortium

6.6.1.1 Features description 48
6.6.1.2 Architecture description and technology stack 48
6.6.1.3 System requirements 50
6.6.1.3.1 Hardware 50
6.6.1.3.2 Software 50
6.6.1.4 Communication protocol 50
6.6.1.5 Repository access (link and other information) 52
6.6.1.6 Dependencies 53
6.6.1.7 Libraries 53
6.7 UC4b - Machine data in Geo-spatial 'on-line' aid application GSAA as added value data 56
6.7.1 Component machine data 56
6.7.1.1 Contact information (author, technical reference) 56
6.7.1.2 Features description 56
6.7.1.3 Architecture description and technology stack 56
6.7.1.4 System requirements 58
6.7.1.4.1.1 Hardware 58
6.7.1.4.1.2 Software 58
6.7.1.5 Communication protocol 59
6.7.1.6 Repository access (link and other information) 59
6.7.1.7 Dependencies 59
6.7.1.8 Other modules 60
6.8 UC5a - Land Parcel Identification System LPIS update ,& change detection 61
6.8.1 Contact information (author, technical reference) 61
6.8.2 Features description 61
6.8.3 Architecture description and technology stack 62
6.8.4 System requirements 62
6.8.4.1 Hardware 62
6.8.4.2 Software 63
6.8.5 Repository access (link and other information) 63
6.8.6 Dependencies 63
6.9 UC5b - Scheme Eligibility and Payment Eligibility: Click-and-Pay 69
6.9.1 Component 1 “(Simulate Payment Entitlements (EU-wide)” 69

Copyright © NIVA Project Consortium

6.9.1.1 Contact information (author, technical reference) 69
6.9.1.2 Features description 69
6.9.1.3 Architecture description and technology stack 69
6.9.1.4 Communication protocol 69
6.9.1.5 Repository access (link and other information) 69
6.9.2 Component "(UC5b.I – Attribute Farm typology in Farm Dossier)" 70
6.9.2.1 Contact information (author, technical reference) 70
6.9.2.2 Features description 70
6.9.2.3 Architecture description and technology stack 70
6.9.2.4 Communication protocol 70
6.9.2.5 Repository access (link and other information) 71
6.9.3 Component 3 "(Click&Pay)" 72
6.9.3.1 Contact information (author, technical reference) 72
6.9.3.2 Features description 72
6.9.3.3 Architecture description and technology stack 73
6.9.3.4 System requirements 73
6.9.3.4.1 Hardware 73
6.9.3.4.2 Software 73
6.9.3.5 Communication protocol 73
6.9.3.6 Libraries 74
6.9.3.7 Other modules 74
6.9.4 Component 4 "(NIDAS)" 75
6.9.4.1 Contact information (author, technical reference) 75
6.9.4.2 Architecture description and technology stack 75
6.9.4.3 System requirements 76
6.9.4.3.1 Hardware 76
6.9.4.3.2 Software 76
6.9.4.4 Libraries 76
6.9.4.5 Other modules 77
7. Conclusions 78

Copyright © NIVA Project Consortium

1. Purpose

This document will list the UCs and their links to the GitLab repositories, along with a brief description of the component and technical requiriements and especifications. Besides this, the second section provides a description of the general NIVA architecture and how the components fit within it.

Copyright © NIVA Project Consortium

2. Acronyms and Definitions

IACS: Integrated Administration and Control System

ICT: Information Communication Technology

LPIS: Land Parcel Identification System

NIVA: New IACS Vision in Action

PA: (CAP) Payment Agency

CAP: Common Agricultural Policy

GSAA: Geospatial Aid Application

Copyright © NIVA Project Consortium

3. Introduction

This document has been estructured by WP2 Manager to cover the following points on the technical description of the Comonents of each NIVA WP2 Use Cases. Nevertheless, tipically the UCs has provided just a general component per Use Case.

Component name:

1. Contact informations (author, technical reference) for each component (eTools)
2. Features description & Componets involved
3. Architecture description and technology stack
4. System requirements per component
    a. Hardware
    b. Software
5. Communication
6. Repository access (link and others informations)
7. Dependencies per component
8. Libraries
9. Others modules
10. Integration interfaces per component (D.2.2 templates, D.2.3)
11. Web services inboud and outbound
12. User expereinces and lessons learned

Also, the section on NIVA Simplified Architecture will show the links between the different components described and their place within the general New IACS Vision in Action approach.

Copyright © NIVA Project Consortium

4. Executive Summary

This documents explains the ICT components and their source code as implemented across the nine NIVA Use Cases and released under the EU PL license.

Copyright © NIVA Project Consortium

5. NIVA Reference Architecture

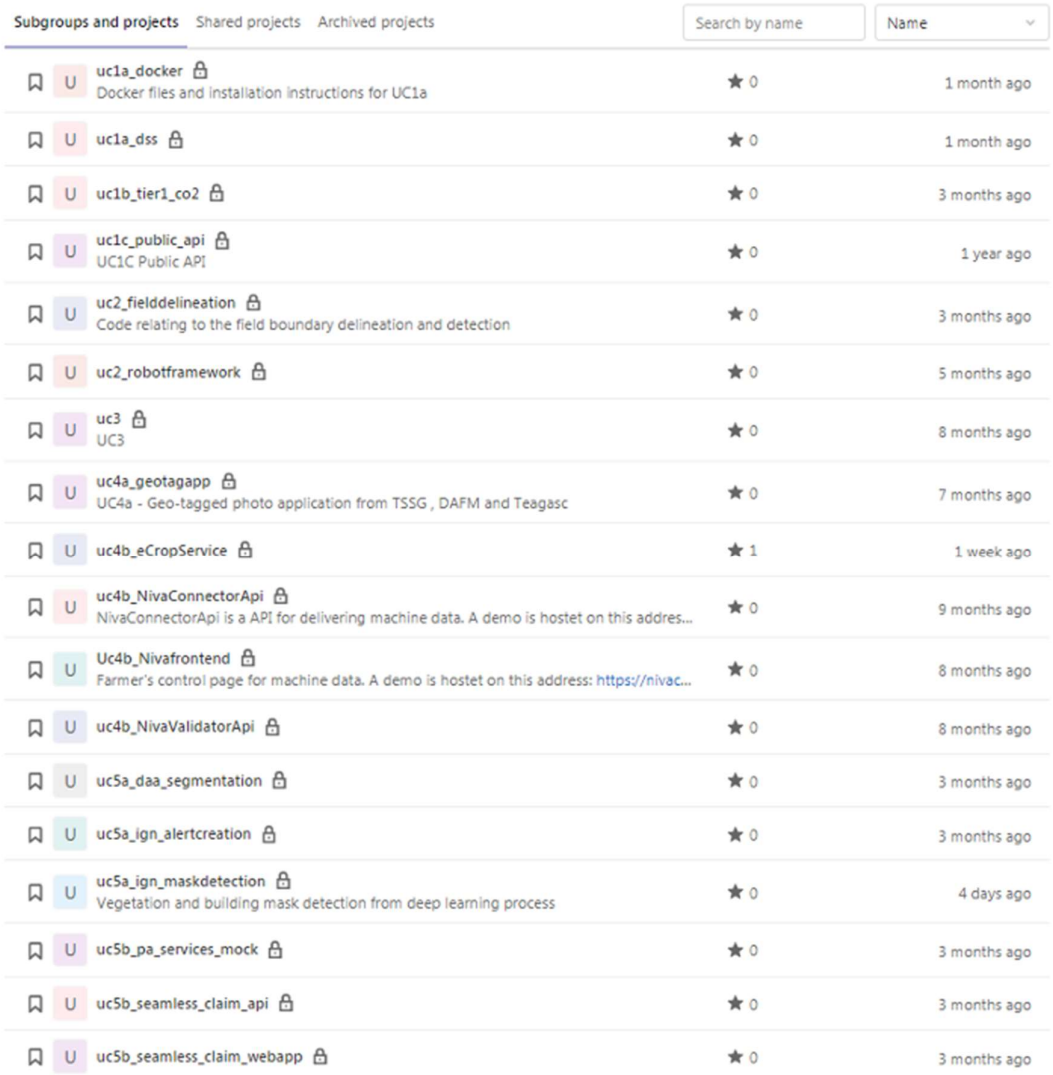
A software reference architecture highlights the basic building blocks and architectural desing decisions of systems in a certain doman. From a software architecture perspective, a reference architecture represents a reusable knowledge in form of generic modules and building blocks that, in our case, will form the different NIVA Use Cases. Therefore, NIVA reference architecture helps simplify the development of the UCs and facilitates standardization and interoperability of the UCs components.

Preliminary IACS Reference Architecture will be presented in following sections. However, this Architecture is a living artefact that will be improved until the end of the proyect and it will have a first completed version in M24 (May 2021)

Copyright © NIVA Project Consortium

5.1 Catalog of NIVA requirements & user needs addressed

Currently, more than 34 components, interfaces or APIs have been depicted in several NIVA documents and all the UCs are available at NIVA GitLab:



All these components, processes and APIs are already available for the Testing phase that will be adequately described in the NIVA D2.5 document. However, it should be noted that the Testing phase itself is a living process that is in a continuous phase of definition and clarification. The NIVA components described can be broadly classified into 14 API programming interfaces, 4 user interfaces and 16 information processes, applied to various data sources. It should also be noted that these information processes are directly related to the data selection and harmonization processes implemented within the WP3 work package.

Copyright © NIVA Project Consortium

Along with these WP2 NIVA components, the Project is also working on the integration of use cases with the following internal WP4 components: OpenEO, Open IAC Upload and Open IACS Dashboard. In addition, work is being done on the possible use of the CEF eID and eSignature components.

Copyright © NIVA Project Consortium

5.2 Components developed & Interfaces

Currently, all the components developed and defined in the NIVA D2.3 document are summarized in the following table. As can be seen, not all the components described in a theoretical way are not yet effectively developed and uploaded to the GitLab repository.

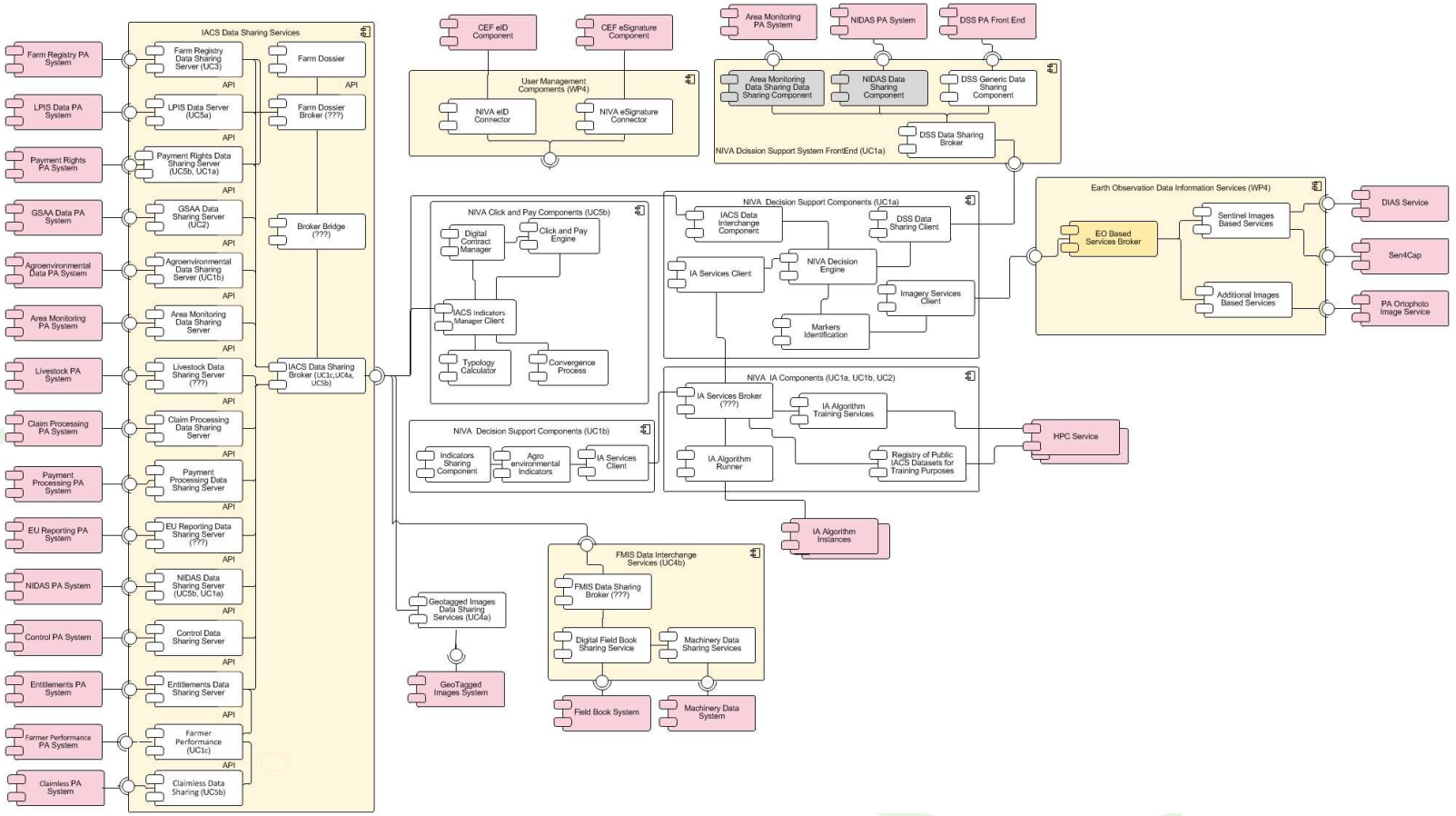
SOURCE	COMPONENTS	NAME	Link to Github
UC1a	Description	Data Import	https://gitlab.com/nivaeu/uc1a_dss
UC1a	Description	Business Rule Engine	https://gitlab.com/nivaeu/uc1a_dss
UC1a	Description	Decision Engine	https://gitlab.com/nivaeu/uc1a_dss
UC1a	Description	Field Map	https://gitlab.com/nivaeu/uc1a_dss
UC1a	Description	Data Export	https://gitlab.com/nivaeu/uc1a_dss
UC1b	Description	Computer of carbon indicator Tier 1	https://gitlab.com/nivaeu/uc1b_tier1_co2
UC1b	Description	Computer of carbon indicator Tier 2	
UC1b	Description	Computer of carbon indicator Tier 3	
UC1b	Description	Computer of nitrate leaching indicator Tier 1	
UC1b	Description	Computer of nitrate leaching indicator Tier 2	
UC1b	Description	Computer of Biodiversity indicator Tier 1	
UC1b	Description	Computer of Biodiversity indicator Tier 2	
UC1c	Description	IACS-FMIS data sharing API	https://gitlab.com/nivaeu/uc1c-public-api
UC2	Description	Preliminary parcel boundary automatic delineation	https://gitlab.com/nivaeu/uc2_fielddelineation
UC2	Description	Robot data harvesting tool	https://gitlab.com/nivaeu/uc2_robotframework
UC3	Description	LoadSingleApplication	
UC3	Description	LoadWork	
UC3	Description	LoadActivity	
UC3	Description	LoadFertilizers	

Copyright © NIVA Project Consortium

SOURCE	COMPONENTS	NAME	Link to Github
UC3	Description	LoadPhytosanitaries	
UC3	Description	LoadLPIS	
UC3	Description	LoadFaST	
UC3	Description	QueryFarmRegistry	
UC4a	Description	AgriSnap (GeoTag Photo App)	https://gitlab.com/nivaeu/uc4a_geotagapp
UC4b	Description	UC4b architecture	
UC4b	Description	Messagebook	
UC4b	Description	NIVA Connector Frontend Farmer	https://gitlab.com/nivaeu/uc4b_nivafrontend
UC4b	Description	NIVA Connector Backend	https://gitlab.com/nivaeu/uc4b_nivaconnectorapi
UC4b	Description	NIVA Validator	https://gitlab.com/nivaeu/uc4b_nivavalidatorapi
UC4b	Description	Webservice PA (WUR)	https://gitlab.com/nivaeu/uc4b_ecropservice
UC4b	Description	Database scripts PA JSON message	https://gitlab.com/nivaeu/uc4b_ecropservice_(database)
UC5a	Description	LPIS automatic change detection	
UC5a	Description	Change detection of high non-agricultural areas	
UC5b	Description	Click and Pay (Claimless system) App and API	https://gitlab.com/nivaeu/uc5b_seamless-claim-webapp https://gitlab.com/nivaeu/uc5b_seamless-claim-api https://gitlab.com/nivaeu/uc5b_pa-services-mock
UC5b	Description	Farm classifier	
UC5b	Description	Simulation tool	
UC5b	Description	Niva Data Supervisor (NIDAS)	
UC2	Description	NIVA - Sen4CAP link	

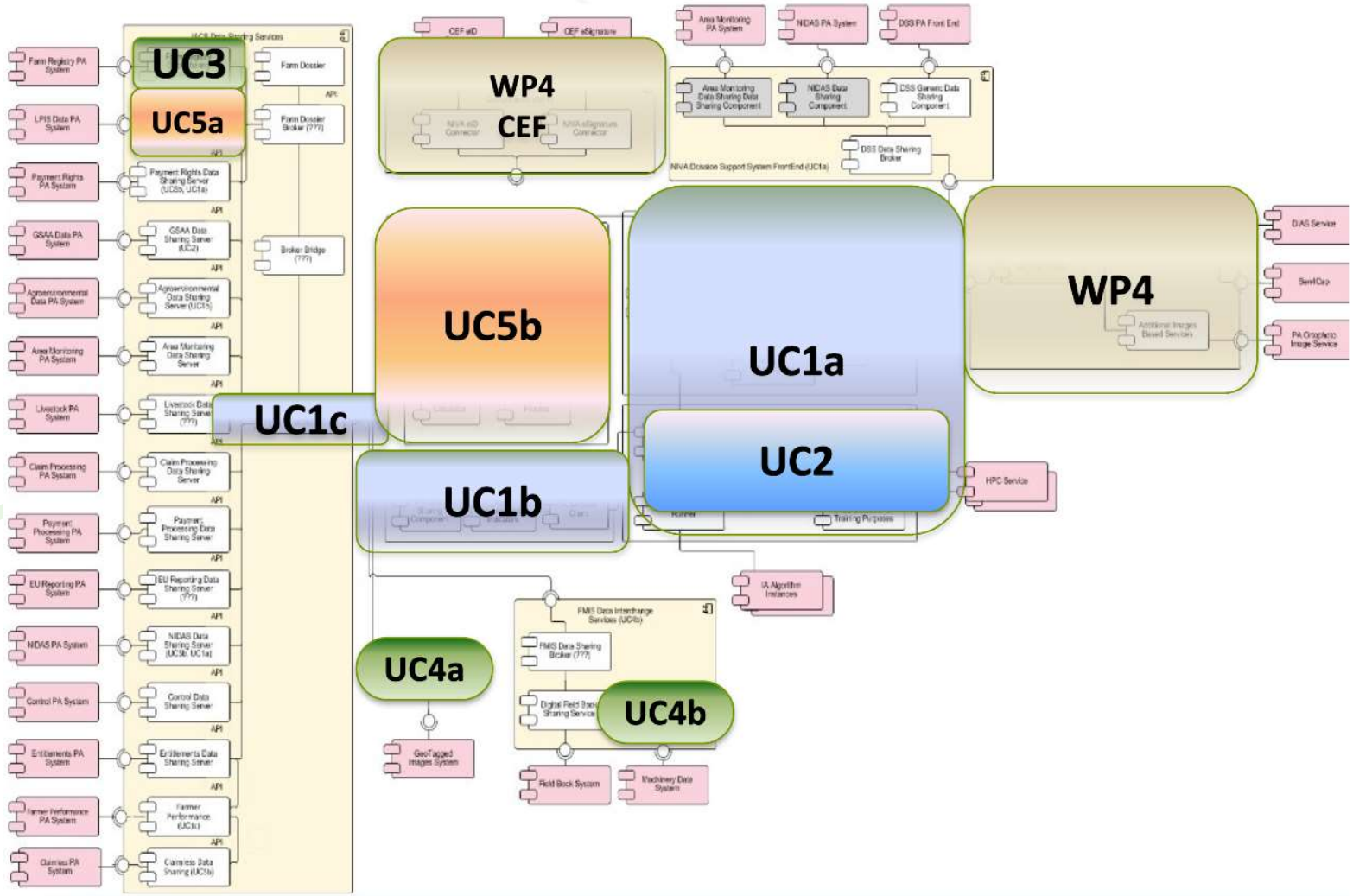
Copyright © NIVA Project Consortium

Based on the components described and the project's own theoretical conceptual needs, part of WP2's efforts have been dedicated to designing a generic NIVA architecture that meets all the needs of the Use Cases. It is important to note that not all the components they are completely (and may not be developed by NIVA) but they do try to cover all the technical needs of the future CAP.



Copyright © NIVA Project Consortium

The components that are being developed by the different UCs are highlighted in the following figure:



Copyright © NIVA Project Consortium

5.3 Data Harmonization. WP3, WP5 support

This section will provide some generic information of componentes developed by other Work Packages than WP2 or provided by third parties or external proviers.

5.3.1 OpenEO API layer

The openEO API defines a HTTP API that lets cloud back-ends with large Earth observation datasets communicate with front end analysis applications in an interoperable way.
As an overview, the openEO API specifies how to

- discover which Earth observation data and processes are available at cloud back-ends,
- execute (chained) processes on back-ends,
- run user-defined functions (UDFs) on back-ends where UDFs can be exposed to the data in different ways,
- download (intermediate) results, and
- manage user content including billing.

The advantages of using OpenEO for the provision of thematic services are:

- Specification is open to contribution and it still support several use cases, here is the link to the open specification and to the
- Data catalogue is done using STAC (Spatio-Temporal Asset Catalogue) specification which is generic enough to include also non-EO data (e.g. Geotagged Photos)
- It embraces the micro-service paradigm allowing a sustainable development of functions and the composition of functions into pipelines. This is quite important for two reasons:
    - A pipeline can benefit of improvement in a function (e.g. Cloud Masking) or from a new function
    - A pipeline can be applied to different data coming from similar sensors (e.g. NDVI applied to both Sentinel 2 and Landsat 8)
- Decouple the UI from the back-end and provide a common interface to different platforms (there are already 5 DIAS and many other EO platform). This is an important point as it allows to evolve the back-end services over the years (e.g. new Deep Learning algorithms, new Satellite sensors and data) while maintaining a consistent API in front of these services without the need to re-develop the clients and the UI by the CAP user

5.3.2 Sen4CAP

The Common Agricultural Policy (CAP) of the European Union aims at improving the European agricultural productivity in a sustainable manner while ensuring a decent standard of living of farmers within the EU. With an annual budget of roughly €59 billion, the CAP strengthens the competitiveness and sustainability of agriculture in Europe by a range of measures including direct payments, market measures and rural development. The largest part of the CAP budget is managed and controlled through its Integrated Administration and Control System (IACS) aiming to safeguard the CAP financials and supporting the farmers to submit their declarations. The IACS is implemented at national level through Paying Agencies of each EU member state.

Copyright © NIVA Project Consortium

The current legal framework of the CAP from 2013 will be reformed by 2020 in order to modernize and simplify the CAP. In this upcoming CAP reform **satellite Earth Observation** (EO) is seen to take an increasing role for improving the IACS and making it more cost efficient.

The **Sentinels for Common Agricultural Policy - Sen4CAP** project aims at providing to the European and national stakeholders of the CAP validated algorithms, products, workflows and best practices for agriculture monitoring relevant for the management of the CAP. The project will pay particular attention to provide evidence how Sentinel derived information can support the modernization and simplification of the CAP in the post 2020 timeframe.

The overall approach proposed for the Sen4CAP project is completely **user-oriented** to ensure responding very concrete user needs and requirements. A Sen4CAP user group includes **Paying Agencies** (PA) from 6 selected Pilot Countries, as well as a set of Champion Users belonging to the Conference of Directors of EU PAs, the Panta Rhei Platform and the Copernicus Agriculture Expert Group. Furthermore, the Steering Committee composed of representatives of the DG-Agri, DG-JRC and DG-Grow and chaired by ESA will provide guidance throughout the course of the project.

The project started in July 2017 and will last 30 months, organized in two distinct phases. The first phase of 14 months aims at delivering a **Proof of Concept**. It will include varying activities starting with the user requirements consolidation and the acquisition and pre-processing of all necessary EO data (both Sentinels 1 & 2) and of IACS data coming from national Paying Agencies (LPIS, declarations, on-the-spot controls, etc.). Fast-track prototype products will then be generated by spring 2018 with the objective of providing early evidence to the Steering group of the benefits of the Sentinels for improving the CAP management in preparation of the CAP 2020 reform. These prototype products will be demonstrated through fast-track use cases and assessed during a Pilot Review. In parallel, the open-source processing system will be developed and validated.

During the second phase (16 months), the full value of the developed EO products and processing system will be assessed through a **national-scale demonstration** during the years 2018 and 2019. This demonstration will be carried out in the operational environment of the engaged Paying Agencies. This phase will also include the necessary training and capacity building activities for the transfer of the developed EO products and services to the Paying Agencies.

The SEN4CAP main objectives are:

- identify and specify **EO products and services** suitable to increase the efficiency, traceability as well as reducing the costs of the IACS,
- develop Algorithm Theoretical Basis Documents along with **open source code** for agricultural EO products based on Sentinel-1 & -2 responding to the user requirements,
- **demonstrate and validate** the developed agricultural EO products up to **national scale,**
- provide evidence for the **utility of Sentinel products within IACS procedures** at EU and national level for a range of national and regional Paying Agencies representative for the heterogeneous agricultural practices, landscape and climate within the EU, as well as to identify the associated limits and conditions of applications,

Copyright © NIVA Project Consortium

- **prepare and facilitate the transfer** of developed EO products and services to the PAs including capacity building and demonstrating cloud computing capabilities.

5.3.3 CEF Components

eID[1] is a set of services provided by the European Commission to enable the mutual recognition of national electronic identification schemes (eID) across borders. It allows European citizens to use their national eIDs when accessing online services from other European countries.

The eID Building Block ensures interoperability on 4 fronts: legal, organisational, semantic and technical interoperability, providing a powerful enabler of digital operations that require cross-border identity recognition.

The high levels of assurance, possible with eIDAS eIDs, result in a reduced risk of identity theft and misuse of personal information. CEF eID ensures the legal validity of cross-borders transactions, providing the same legal status as traditional paper based processes.

The solution drastically reduces the administrative burden, in terms of time, expenses and effort, associated with the use of foreign public services — a necessity for many in an increasingly globalised world.

eSignature[2] is a set of free standards, tools and services that help public administrations and businesses accelerate the creation and verification of electronic signatures that are legally valid in all European Member States.

By removing the need for paper to obtain a signature or seal, the solution facilitates the digitalisation of business processes, eliminating the time, costs and risks of dealing with paper formats.

eSignature is compliant with the eIDAS Regulation, meaning that it ensures the legal recognition and cross-border interoperability of the electronic signatures and seals produced by solutions based on this building block.

With eSignature, you reduce risk of document duplication or alteration. The signer's identity is bound to each specific document, ensuring that signatures and seals are verified, authentic and legitimate for that document alone.

[1] https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eID
[2] https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eSignature

Copyright © NIVA Project Consortium

5.4 Common Components and WP4 support

Technical specifications for software components to be subcontracted will be based on inputs on required components as needed in the large scale pilots of WP2, interoperability trails in WP3 or infrastructural components with WP4, especially for the software components. These software components are a response to needs for the executions of the Large Scale Pilots in WP2, in identifying software development requirements and components that were not foreseen at the initial project set up during the proposal phase.

OPEKEPE will sub-contract the development of new components to support the Large Scale Pilots. Four to five of these components are foreseen based on the specifications set in Task 6.2. The following components were defined during the proposal phase to be sub-contracting:

- Click-and-pay: Software component to authorize payment if certain pre-conditions are met according to objective/verifiable data sources (for example remote sensing).
- LPIS update: A software component used to automatically update the Land Parcel Information System with year to year changes in parcels (e.g. farmers splitting parcels in two, merging parcels, different field borders due to road construction, etc).
- Traffic Light control: This component abstracts the idea of a traffic light for red (definitely required to check), orange (could be needed to check) to green (good to proceed without check), with a generic set up and specification, so that it can easily be deployed across software systems in IACS.

Environmental performance component at farm level: This component abstracts the idea of a traffic light for red (definitely required to check), orange (could be needed to check) to green (good to proceed without check), with a generic set up and specification, so that it can easily be deployed across software systems in IACS.

5.5 Expected components from subcontracting and Open Call

Technical specifications for software components to be subcontracted will be based on inputs on required components as needed in the large scale pilots of WP2, interoperability trails in WP3 or infrastructural components with WP4, especially for the software components. These software components are a response to needs for the executions of the Large Scale Pilots in WP2, in identifying software development requirements and components that were not foreseen at the initial project set up during the proposal phase.

OPEKEPE will sub-contract the development of new components to support the Large Scale Pilots. Four to five of these components are foreseen based on the specifications set in Task 6.2. The following components were defined during the proposal phase to be sub-contracting:

- Click-and-pay: Software component to authorize payment if certain pre-conditions are met according to objective/verifiable data sources (for example remote sensing).
- LPIS update: A software component used to automatically update the Land Parcel Information System with year to year changes in parcels (e.g. farmers splitting parcels in two, merging parcels, different field borders due to road construction, etc).

Copyright © NIVA Project Consortium

- Traffic Light control: This component abstracts the idea of a traffic light for red (definitely required to check), orange (could be needed to check) to green (good to proceed without check), with a generic set up and specification, so that it can easily be deployed across software systems in IACS.
- Environmental performance component at farm level: This component abstracts the idea of a traffic light for red (definitely required to check), orange (could be needed to check) to green (good to proceed without check), with a generic set up and specification, so that it can easily be deployed across software systems in IACS.

Copyright © NIVA Project Consortium

6. Use cases contributions to NIVA components (eTools)

6.1 UC1a - Earth Observation Monitoring and Traffic Lights Text

6.1.1 Earth Observation Monitoring and Traffic Lights DSS

6.1.1.1 Contact information (author, technical reference)

NIVA OPEKEPE Team, UC1a leader, niva@opekepe.gr

- Non-technical questions regarding functionality
- Technical Questions regarding docker installation

NEUROPUBLIC, Yorgos Efstathiou, y_efstathiou@neuropublic.gr

- UC1a implementor
- Technical issues regarding DSS components

6.1.1.2 Features description

The Earth Observation Monitoring and Traffic Lights Decision Support System is an information system that supports determinations and decision-making activities for applying the monitoring approach for cross compliance checks. The DSS application includes 5 modules (or system (sub)components) namely Data Import, Business Rule Engine, Decision Engine, Fieldmap and Data Export. All these multiple modules or components are connected to the main application (DSS) sharing the same system characteristics regarding software & hardware requirements. The architecture diagram is shown in the figure below.

The Data Import component provides the required methods for importing the data file from EO classification engine. The component reads a shapefile containing the classification results imported as a zip file and other crop-field related information (all the necessary attributes) and inserts and updates the data into the according tables of the DSS database system (create and run the DB prepared statements for tables' insertions). Still missing specification for importing secondary data as a web service (under development RESTful APIs →Geotagged Photos Data Import & Farm Calendar Data Import)

The Business Rule Engine component provides the required methods to produce Eligibility Criteria coded as algorithms for Basic Payment Scheme, for Crop Specific payment for Cotton and for certain Voluntary Coupled Schemes. The application combines the 1st and/or 2nd prediction, the 1st and/or 2nd confidence level and/or the sum of the 1st and 2nd confidence levels by using logical statements.

The Decision Engine component provides the required methods for performing decision making in the parcels classification. For each selected classification dataset and defined BRE criteria the Decision Engine component runs the decision making process for all the parcels, locks the UI and

Copyright © NIVA Project Consortium

returns the results and warnings (conclude in the eligibility status at parcel level). Based on predetermined threshold levels each score will qualify to a traffic light colour for each eligibility criterion that will be the final outcome of the UC1a-DSS. The calculated results are can be exported in CSV format.

Additional information from secondary sources (e.g. FMIS, Geo-tagged photos) can be inserted directly in the decision engine and/or employed for the re-training of the EO crop classification engine. The calculated results can be exported in CSV format.

The Field Map component will provide a user-friendly GUI that will enable visualisation of the outcomes of the Decision Engine. Automatically reads the result from the Decision Engine runs displaying the output in a web-GIS environment. It uses a search functionality, which allows filtering results to certain field attributes. When choosing from the result-list the user can see both a crop list and a map.

Within the list the user can further search the "Decision Making" result by parcel code, farmer code and identifier. The user has access also to the parcel code, the farmer code and the classification probabilities, predictions, identifier and declared crop type.

The Data Export component provides access to the system's data. The user can acquire the relevant information blocks via an ogc compliant webservice api. The calculated results will be available to be received by a specific endpoint of the application. The exact format is under specification. The UC1a exports results either via a REST service (under development) or as a static file.

6.1.1.3 System requirements

6.1.1.3.1 Hardware

Here we define two sets of system requirements: minimum and recommended.

Component	Minimum	Recommended
RAM	4GB	12GB
Processor	Any 2core intel/AMD processor	6core processor
Storage space	250GB	1TB + 2TB for archiving
Network	200mb/s dedicated network speed (upload/download)	1000mb/s dedicated network speed (upload/download)
GPU	No minimum requirement	1GB Memory

6.1.1.3.2 Software

Sofware Requirements
CSS3
Bootsrap
HTML5
JavaScript

Copyright © NIVA Project Consortium

JBoss server 7.1.1
jquery
AngularJS
Java 7
Python
Postgres –postgis 12
Centos 7 operating system

6.1.1.3.3 Communication protocol

Representational stateless transfer (REST) protocol over secure HTTP (HTTPS).

The system employs back-mid-front end architecture where each request is sent asynchronously (via standard POST/GET request) to the mid-end. The mid-end communicates with the back-end, generates the content and sends the response to the front-end.

Communication with the 3rd Parties

For viewing and inspecting the agricultural parcels the application uses Google, Bing, Openstreetmap background layers.

6.1.1.3.4 Repository access (link and othes information)

- Issues related to UC1a are registered at https://gitlab.com/nivaeu/uc1a_dss/-/issues
- Issues related to the installation procedure (docker files) are registered at https://gitlab.com/nivaeu/uc1a_docker/-/issues

6.1.1.3.5 Dependencies

Open Street Maps: This is used to provide a map interface for the end user, it is open source.

Bing maps/Google maps: the application supports commercial satellite background layers (key is needed).

6.1.1.3.6 Other modules

No other modules

6.1.1.3.7 Integration interfaces

Data export API (REST) providing decision making results to 3rd parties

6.1.1.3.8 Web services inbound and outbound

Still missing specification for importing secondary data as a web service (under development RESTful APIs →Geotagged Photos Data Import & Farm Calendar Data Import)

Copyright © NIVA Project Consortium

6.2UC1b - Agro-environmental monitoring

6.2.1 Component 1: Carbon Indicator TIER1

6.2.1.1 Contact information (author, technical reference)

- Ludovic Arnaud (ludovic.arnaud@asp-public.fr): IT developer in charge of the code.
- Guillaume Marchand (guillaume.marchand@ign.fr): UC1b contact point and project manager in charge of the technical coordination.

6.2.1.2 Features description

The CO2 tool evaluates carbon fluxes at parcel level from NDVI time series. As input, it takes SEN4CAP compatible csv file that contains time series of NDVI statistics (parcel average value and standard deviation). After selecting this file, the value of the NDVI threshold and the time period, the tool outputs the carbon indicator per parcel in a xlsx file.

6.2.1.3 Architecture description and technology stack

The tool that evaluates the Tiers 1 carbon indicator is constructed around the three following Python codes:

process.py: Contains the code relating to the calculation of the annual net CO2 flux, as well as the calculation of other parameters: number of data gaps, number of vegetation cycles, NDVI statistics...

period_cut.py: Contains the code related to the choice of period. The user needs to select the start and the end of the period over which he/she wants the calculation to be performed.

interface.py: Contains the code for the user interface. This allow for users to select the input file, the threshold value, and the period and launch the process. It also links the previous codes together.

6.2.1.4 System requirements

6.2.1.4.1 Hardware

Minimal configuration:

- Linux or Windows OS. The tools has not been tested on Mac OS. However, it is expected to work similarly to Linux OS (same installation, same execution)
- 2GB of RAM

6.2.1.4.2 Software

The tool is based on Python 3.5 or above.

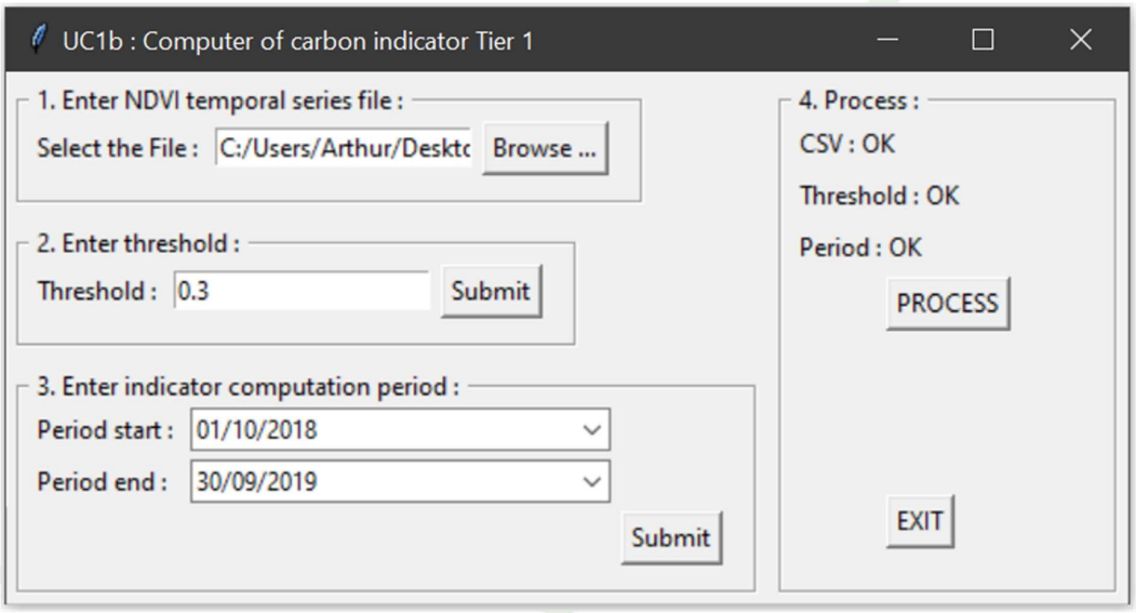
Dependencies are described in the next section

On windows, the tool is run thanks to the **CO2.exe** executable file. On Linux, it is run thanks to the **CO2** file.

6.2.1.5 Communication protocol

The interface facilitates the use of the tool. It allows the selection of the input data file, the setting of the NDVI threshold and the time period.

Copyright © NIVA Project Consortium



No web services are included in this component of the Carbon indicator TIER 1.

6.2.1.6 Repository access (link and other information)

This component is available with a user guide on NIVA Gitlab:
https://gitlab.com/nivagroup/uc1b/-/tree/dev

6.2.1.7 Libraries

The component is based on EUPL compatible and open source libraries as follows:
Csv, copy, datetime, numpy, os, pandas, pyinstaller, threading, tkcalendar, tkinter, time,

Copyright © NIVA Project Consortium

6.3UC1c - Farmer Performance

6.3.1 Component "IACS-FMIS data sharing API"

6.3.1.1 Contact information (author, technical reference)

Component (prototype of a microservice) has been developed by the Estonian Agricultural Registers and Information Board (ARIB). Contact person for UC1c is ARIB project manager Jane Jäger (Jane.Jager@pria.ee), ARIB general contact is pria@pria.ee.

6.3.1.2 Features description

In its essence, product is a microservice to convert a limited set of IACS and FMIS (Farm Management Information System) sample data to eCROP format. Prototype of service is country specific for Estonian IACS, reusable components are recommended standards and data structure of APIs to exchange data between different systems.

Main features of the component are:

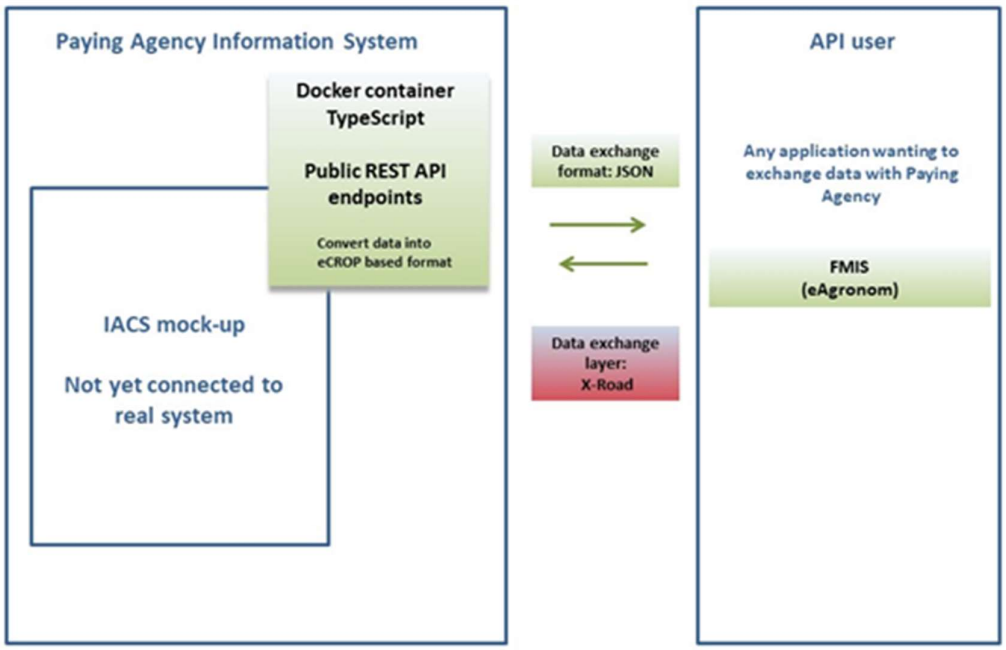
- Converting IACS data into eCROP based format;
- Making IACS data available to FMIS by returning the data as a response to a request sent by FMIS;
- Validating FMIS data compliance with eCROP based format;
- Accepting compliant data (Important note: prototype solution is not actually saving and storing data);
- Rejecting non-compliant data.

Description of the component:

- Microservice for exchanging data between Paying Agency and Farm Management Information System (FMIS) type of software.
- Data exchange protocol is based on eCROP standard, with some modifications.
- Purpose of the prototype is to test suitability of the eCROP standard and possible issues.
- No authentication component – authentication is very country specific and main purpose is to test suitability of data exchange model.
- REST API with 2 endpoints:
    - Requesting data from IACS about all agricultural parcels of a farmer/beneficiary. Requests made based on personal ID code or business registry code.
    - Sending data about agricultural parcel from FMIS to IACS, requests made based on agricultural parcel ID. Parcel ID originates from IACS, for sending data FMIS needs to first obtain parcel ID from IACS.
- Node.js and Typescript application, run in Docker. User interface – Swagger.
- Prototype validates dataset received from FMIS but does not actually save the accepted dataset.

Copyright © NIVA Project Consortium

6.3.1.3 Architecture description and technology stack



Prototype of IACS-FMIS data sharing API is an additional component of the IACS used by Estonian PA, an REST API component enabling IACS to receive data from FMIS and send data to FMIS. FMIS application used as an example in developing the prototype is a product offered by the Estonian company eAgronom. Although the prototype is focusing on IACS data exchange with an FMIS application, eventually the developed component (green box on Architecture diagram) can be used for exchanging data with any application (API user on Architecture diagram) which wants to exchange data with IACS.

UC1c IACS-FMIS data sharing API has two endpoints:

1. GET /api/v1/ecrop/crop-plot – to request data from IACS (FMIS can request data about agricultural parcels from IACS);
2. POST /api/v1/ecrop/crop-plot – to receive data from FMIS (FMIS can send data about agricultural parcels to IACS).

Copyright © NIVA Project Consortium

6.3.1.4 System requirements

6.3.1.4.1 Hardware

The prototype has no hardware interfaces. Constraint on hardware architecture: AMD64 Architecture.

6.3.1.4.2 Software

- Node.js and Typescript application, run in Docker.
- User interface – Swagger.

No constraints on operating system.

Development Framework	Nest.js https://nestjs.com/
Development tool	WebStorm https://www.jetbrains.com/webstorm/
Continuous Integration	GitLab CI
Release management	GitLab https://gitlab.com/nivaeu/uc1c-public-api
Testing Framework	Jest https://jestjs.io/
Docker	https://www.docker.com/

6.3.1.5 Communication protocol

Microservice is providing REST API to exchange data, using encrypted HTTP connections (at least TLS v1.2).

For **GET requests** IACS-FMIS data sharing API retrieves data from IACS: the request parameter specifies personal ID code or business registry code of a farmer (beneficiary), as a response a set of data about agricultural parcel(s) related to this particular beneficiary is returned.

This dataset consists of following data:

- Identification of parties exchanging information – ID's of the Paying Agency and FMIS software. For prototype agreed values are being used, for real life deployment in the future it is advised to use business registry codes.
- Beneficiary identification – personal ID code for a natural person or business registry code for a legal person.
- Agricultural parcel (or parcels, request response returns data about all parcels related to particular beneficiary) number, area measure of parcel in hectares, related reference parcel numbers and cadastral unit number, where the parcel is located. In case of the UC1c prototype which uses data from development environment of Estonian IACS, there is also an additional attribute, object type code. This attribute, which is rather specific to the Estonian IACS, specifies type of the data object – drawn parcel, drawn meadow, claimed parcel and claimed meadow. Such classification originates from the procedure how beneficiaries submit

Copyright © NIVA Project Consortium

applications. First the parcel boundaries are being drawn in ARIB e-service, creating object type „drawn parcel“. Thereafter this parcel can be submitted and object type „claimed parcel“ is created. In the IACS of any other Member State such data object classification may not be applicable.

- Type codes indicating whether this parcel is used for producing agricultural grass seed and whether it is organic land.
- Data about coordinate reference system and parcel geometry (polygons).
- Data about crop/crop type grown on the parcel, according to crop type list used the Paying Agency, plus land use and purpose code (e.g. for seed or forage etc).

For **POST requests** IACS-FMIS data sharing API validates and accepts or rejects an agreed set of FMIS data. The prototype does not actually save the FMIS data in IACS, because it is highly specific to particular IACS system where and in what format to save this data.

This dataset consists of following data:

- Identification of parties exchanging information – ID's of the Paying Agency and FMIS software. For prototype agreed values are being used, for real life deployment in the future it is advised to use business registry codes.
- Beneficiary identification – personal ID code for a natural person or business registry code for a legal person.
- Agricultural parcel number and related reference parcel number. In case of UC1c prototype there is also an additional attribute, object type code. This attribute, which is rather specific to the Estonian IACS, specifies type of the data object – drawn parcel, drawn meadow, claimed parcel and claimed meadow. Such classification originates from the procedure how beneficiaries submit applications. First the parcel boundaries are being drawn in ARIB e-service, creating object type „drawn parcel“. Thereafter this parcel can be submitted and object type „claimed parcel“ is created. In the IACS of any other Member State such data object classification may not be applicable.
- Data about coordinate reference system and parcel geometry (polygons).
- Data about crop/crop type grown on the parcel, according to crop type list used the Paying Agency.
- Data about an agricultural activity (work) performed in the parcel (type of work, start and end dates).
- Data about any product used in connection with an activity performed on an agricultural parcel (like seed, fertilizer, plant protection product etc) - product name, type, amount used.

All requests return HTTP response code:

- 200 OK for GET requests;
- 201 created for POST requests creating a new resource;
- 404 parcel not found for GET requests;
- 400 error message for POST requests, in case eCROP message body has wrong format.

Copyright © NIVA Project Consortium

6.3.1.6 Repository access (link and other information)

Requirements specification and other documentation is available in NIVA Sharepoint UC1c folder. Source code and readme-file with instructions are available in NIVA GitLab repository.

Resource	Link	Comments
NIVA UC1c Public API (IACS-FMIS data sharing API)	https://gitlab.com/nivaeu/uc1c-public-api	Source code, readme
UC1c IACS-FMIS data sharing API Requirements Specification	NIVA Sharepoint	Software Requirements Specification document
UC1c additional documentation	NIVA Sharepoint	UC1c folder in Sharepoint, containing additional documentation

Copyright © NIVA Project Consortium

6.4UC2 - Prefilled application, GSAA/Land link

6.4.1 Component “Robot Data Harvesting Tool” (also known as “Robot Process Automation” (RPA) or “Robot Framework”)

6.4.1.1 Contact information (author, technical reference)

Component (Robot Data Harvesting Tool) has been developed by the iTree. Contact person for this component in iTree is Saulius Baltrėnas (saulius.baltrenas@itreegroup.eu), general iTree contact is info@itreegroup.eu.

6.4.1.2 Features description

Robot Framework is an open-source RPA solution for acceptance testing, acceptance test-driven development (ATDD), and used to automate business processes. It is open and extensible, therefore it can be integrated with any other tool in order to create even more powerful and flexible RPA solutions.

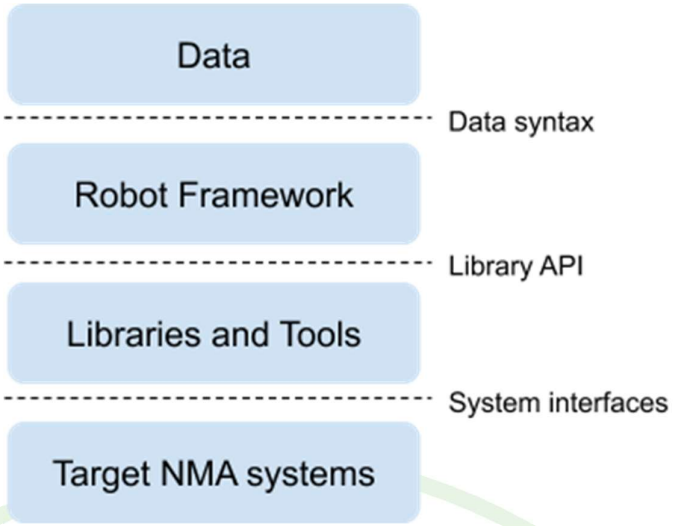
The core of Robot Framework, the generic automation platform, is very popular and has an active community of developers and users in many industry-leading companies, which use it in their software development. This leads to active and continuous support and improvements. The community works actively toward additional functionality, like optical image recognition, HTTP APIs, database access, remote execution, and iOS and Android application support, while orchestration functionality in commercial projects is mostly available through Jenkins plugins.

The Robot Framework has a simple plain text syntax, which can be extended by libraries using Java or Python. This enables any user to integrate new functionality into the Robot Framework.

Because Robot Framework is open source, it is free to use without any licensing costs and users can run any number of Robot Framework RPA processes in parallel. This enables even small enterprises to automate some of their tasks with limited costs.

Copyright © NIVA Project Consortium

6.4.1.3 Architecture description and technology stack



The Robot Framework utilizes a human-readable syntax which is based on keywords. Keywords are composable, meaning you can define new keywords that use pre-existing keywords provided by the libraries to interact with the target system. Libraries can communicate with the system either directly or using other tools as drivers.

The Robot Framework was intended to apply in a test case for the Lithuanian Payment Agency (NMA). NMA is dealing with farmers' applications, which are managed within the internal NMA system. During this process, experts define certain parameters and manually check and compare various data between different systems, external registers, and other data storage. For demo and test purposes, following tasks for Robot Framework were addressed:

- To edit data of certified fields in the PPAIS system (Lithuanian application's administration system in NMA) depending on certificate data of Ekoagros website. The robot has to check the certificate data in Ekoagros website (an external data repository) and based on that, edit specific fields in the internal system.
- To save certain certificates in HTML format in a specific directory in the hard drive.
- Based on certification information, change farmers' applications' attributes if needed.

The code of this demo scenario is saved in the **Nemiga.robot** file and can be executed by Robot Framework.

6.4.1.4 System requirements

Robot Framework is an operating system and application independent.

6.4.1.4.1 Hardware

The prototype has no hardware interfaces and has no constraints on hardware architecture.

Copyright © NIVA Project Consortium

6.4.1.4.2 Software

In order to perform demo tasks, these software configurations were used:

- Python 3.8.2;
- Robot Framework 3.1.2;
- RIDE - Standalone Robot Framework test data editor.

6.4.1.5 Communication protocol

Robot Framework can be integrated in any solution in order to simulate user activities using graphic user interface. Below is an example of a reading, analyzing and altering data.

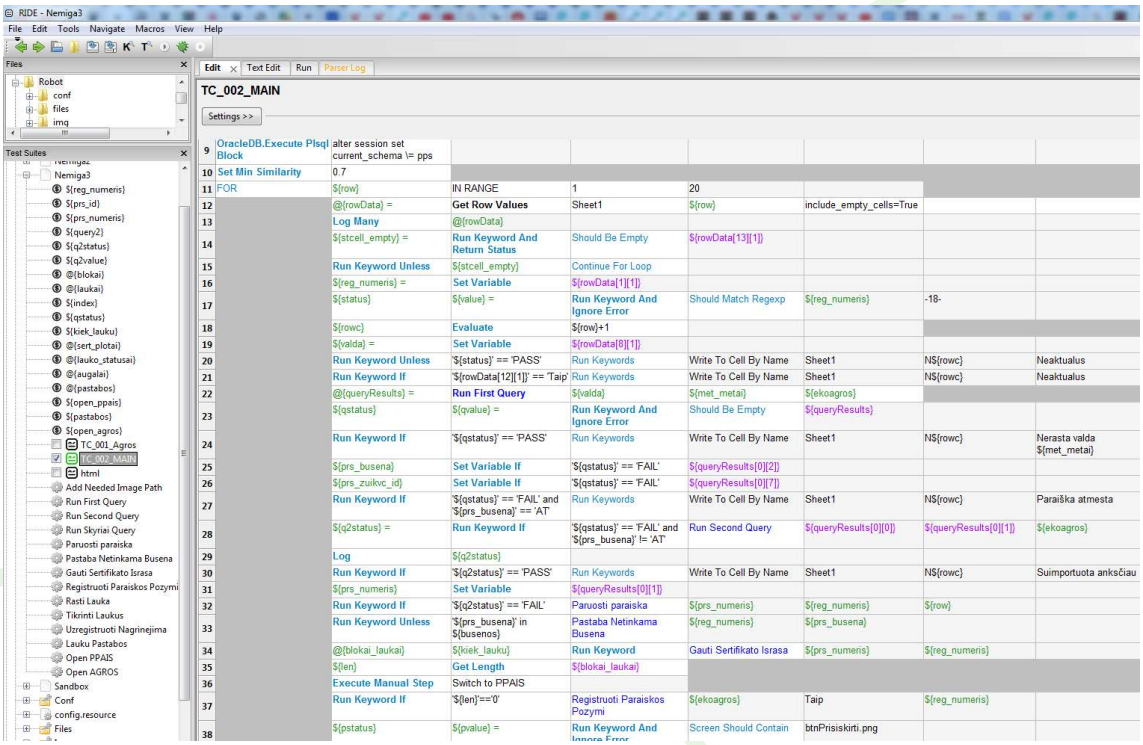
Step-by-step actions of automated tasks, performed by Robot Framework:

1. Read data from XLS (MS Office, Excel) file row by row.
2. Analyze data of every row and run operations depending on analysis results:
   - Check specific cells in the row (reg. number, status);
   - Execute database queries based on checked cell data in the rows;
   - Depending on query results, open Oracle Forms and edit application data, if needed;
   - Open Web Application, find required information by application number, save it to HTML file and read needed information to array;
   - Run database query based on saved array data;
   - Return to Oracle Forms and save check result for each application;
   - Save results for all checked rows in the XLS file.

Robot Framework as input takes the data from XLS file, compares it with the data in the database, if needed edits other data in the repository, searches for other data in the web application, saves in HTML and array, runs a query again, checks the result and saves again in the XLS file.

A mixture of default keywords from different libraries and user-created keywords (figure below). You can combine preset keywords and create your own readable keywords in order to have clear and structured test cases. For example, in the image below (figure x) you can see the keyword “Paruošti paraišką” (in Engl.: “Prepare the application”) which contains all steps needed to be completed to prepare the application in focus for checking.

Copyright © NIVA Project Consortium



9	OracleDB.Execute Plsql Block	alter session set current_schema \= pps						
10	Set Min Similarity	0.7						
11	FOR	${row}	IN RANGE	1	20			
12		@{rowData} =	Get Row Values	Sheet1	${row}	include_empty_cells=True		
13		Log Many	@{rowData}					
14		${stcell_empty} =	Run Keyword And Return Status	Should Be Empty	${rowData[13][1]}			
15		Run Keyword Unless	${stcell_empty}	Continue For Loop				
16		${reg_numeris} =	Set Variable	${rowData[1][1]}				
17		${status}	${value} =	Run Keyword And Ignore Error	Should Match Regexp	${reg_numeris}	-18-	
18		${rowc}	Evaluate	${row}+1				
19		${valda} =	Set Variable	${rowData[8][1]}				
20		Run Keyword Unless	'${status}' == 'PASS'	Run Keywords	Write To Cell By Name	Sheet1	N${rowc}	Neaktualus
21		Run Keyword If	'${rowData[12][1]}' == 'Taip'	Run Keywords	Write To Cell By Name	Sheet1	N${rowc}	Neaktualus
22		@{queryResults} =	Run First Query	${valda}	${met_metai}	${ekoagros}		
23		${qstatus}	${qvalue} =	Run Keyword And Ignore Error	Should Be Empty	${queryResults}		
24		Run Keyword If	'${qstatus}' == 'PASS'	Run Keywords	Write To Cell By Name	Sheet1	N${rowc}	Nerasta valda ${met_metai}
25		${prs_busena}	Set Variable If	'${qstatus}' == 'FAIL'	${queryResults[0][2]}			
26		${prs_zuikvc_id}	Set Variable If	'${qstatus}' == 'FAIL'	${queryResults[0][7]}			
27		Run Keyword If	'${qstatus}' == 'FAIL' and '${prs_busena}' == 'AT'	Run Keywords	Write To Cell By Name	Sheet1	N${rowc}	Paraiška atmesta
28		${q2status} =	Run Keyword If	'${qstatus}' == 'FAIL' and '${prs_busena}' != 'AT'	Run Second Query	${queryResults[0][0]}	${queryResults[0][1]}	${ekoagros}
29		Log	${q2status}					
30		Run Keyword If	'${q2status}' == 'PASS'	Run Keywords	Write To Cell By Name	Sheet1	N${rowc}	Suimportuota anksčiau
31		${prs_numeris}	Set Variable	${queryResults[0][1]}				
32		Run Keyword If	'${q2status}' == 'FAIL'	Paruosti paraiska	${prs_numeris}	${reg_numeris}	${row}	
33		Run Keyword Unless	'${prs_busena}' in ${busenos}	Pastaba Netinkama Busena	${reg_numeris}	${prs_busena}		
34		@{blokai_laukai}	${kiek_lauku}	Run Keyword	Gauti Sertifikato Israsa	${prs_numeris}	${reg_numeris}	
35		${len}	Get Length	${blokai_laukai}				
36		Execute Manual Step	Switch to PPAIS					
37		Run Keyword If	'${len}'=='0'	Registruoti Paraiskos Pozymi	${ekoagros}	Taip	${reg_numeris}	
38		${pstatus}	${pvalue} =	Run Keyword And Ignore Error	Screen Should Contain	btnPrisiskirti.png		

Using code, the user can type in what should be done if certain conditions appear. For instance, after specific operations, if the defined outcome was received, then Robot Framework has to open a browser or a selected system, if Robot acquires a certain file or reaches specific order, then a file should be renamed and moved, or a specific button has to be pressed. This shows that Robot Framework is fully customizable and by having some scripting skills, the tasks can be automated.

6.4.1.6 Repository access (link and other information)

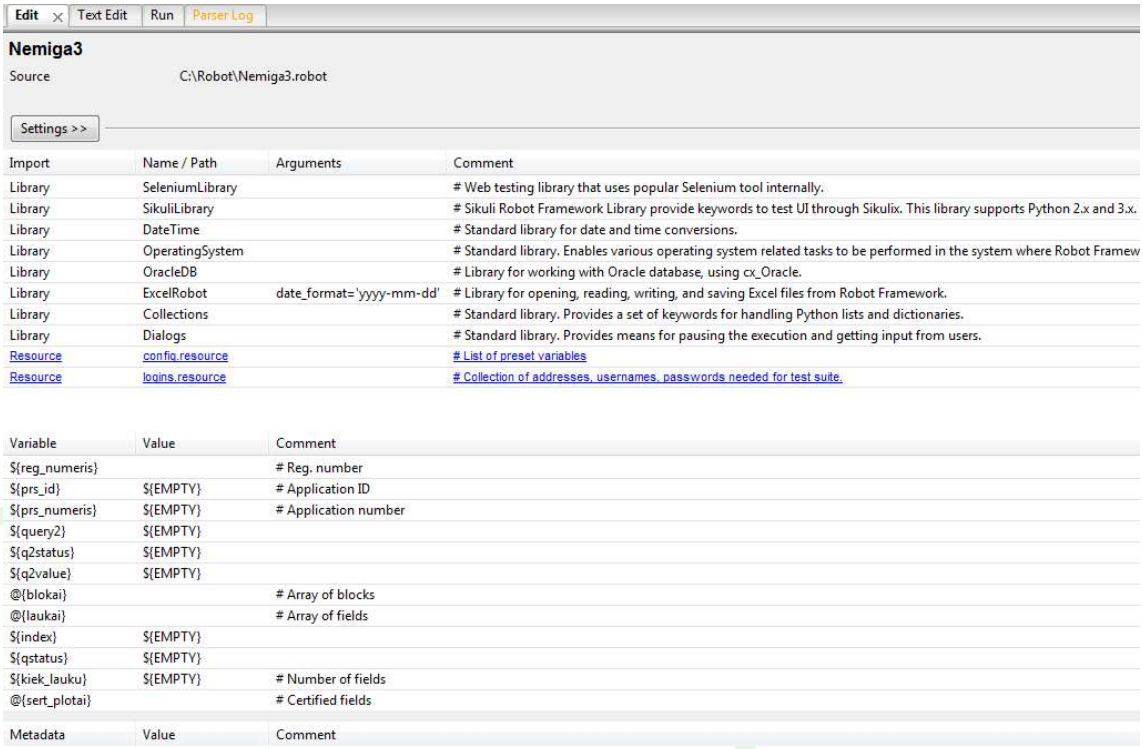
A white paper, detailed specifications of component's interfaces and testing phase results can be found in NIVA Sharepoint UC2 folder. Meanwhile a demo video, demonstrating how the robot functions, environment (required python version and other requirements) and robot prototype itself can be found in the Gitlab depository.

Resources:

- NIVA Sharepoint UC2 folder
- Component's Gitlab folder

Copyright © NIVA Project Consortium

6.4.1.7 Libraries



Edit | Text Edit | Run | Parser Log

Nemiga3

Source C:\Robot\Nemiga3.robot

Settings >>

Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		# Web testing library that uses popular Selenium tool internally.
Library	SikuliLibrary		# Sikuli Robot Framework Library provide keywords to test UI through Sikulix. This library supports Python 2.x and 3.x.
Library	DateTime		# Standard library for date and time conversions.
Library	OperatingSystem		# Standard library. Enables various operating system related tasks to be performed in the system where Robot Framew
Library	OracleDB		# Library for working with Oracle database, using cx_Oracle.
Library	ExcelRobot	date_format='yyyy-mm-dd'	# Library for opening, reading, writing, and saving Excel files from Robot Framework.
Library	Collections		# Standard library. Provides a set of keywords for handling Python lists and dictionaries.
Library	Dialogs		# Standard library. Provides means for pausing the execution and getting input from users.
Resource	config.resource		# List of preset variables
Resource	logins.resource		# Collection of addresses, usernames, passwords needed for test suite.

Variable	Value	Comment
${reg_numeris}		# Reg. number
${prs_id}	${EMPTY}	# Application ID
${prs_numeris}	${EMPTY}	# Application number
${query2}	${EMPTY}	
${q2status}	${EMPTY}	
${q2value}	${EMPTY}	
@{blokai}		# Array of blocks
@{laukai}		# Array of fields
${index}	${EMPTY}	
${qstatus}	${EMPTY}	
${kiek_lauku}	${EMPTY}	# Number of fields
@{sert_plotai}		# Certified fields

Metadata	Value	Comment

Additional external open source Robot Framework libraries were employed:

- SeleniumLibrary - a web testing library that internally uses a popular Selenium tool. Its purpose was to get needed data from certificates on Ekoagros website;
- SikuliLibrary - it provides keywords to test user interface (UI) through Sikulix (uses image recognition powered by OpenCV to identify GUI components). This library was used to interact with PPAIS application based on Oracle Forms;
- OracleDB - library for working with Oracle database;
- ExcelRobot - library for handling Excel files from Robot Framework.

Copyright © NIVA Project Consortium

6.4.2 Component Preliminary parcel boundary automatic delineation

6.4.2.1 Contact information (author, technical reference)

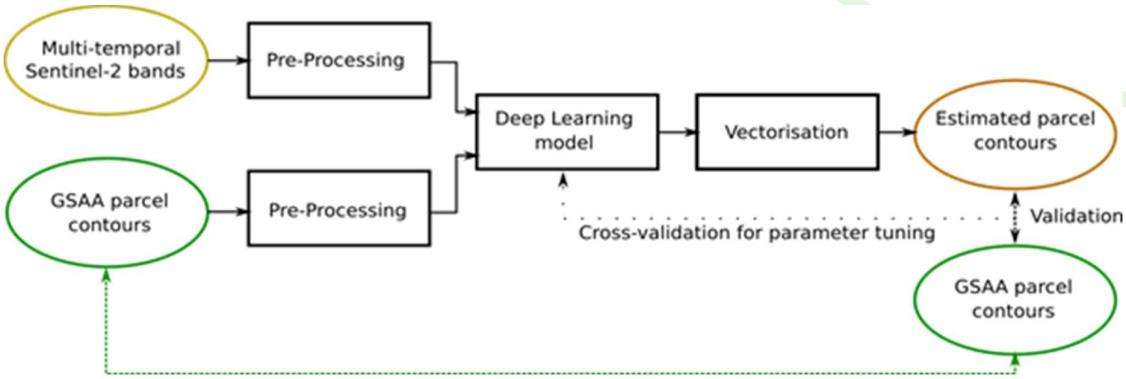
- Anej Visković (anej.viskovic@sinergise.com): contact point and project manager in charge of the component
- Devis Peressutti (devis.peressutti@sinergise.com): data scientist and technical reference for the development of the component

6.4.2.2 Features description

The component allows to detect and delineate agricultural parcel boundaries from Sentinel-2 imagery. During a training phase, the component takes as input Sentinel-2 images and target reference parcels (i.e. in suitable vector format, such as ESRI shapefile or gpkg) and produces a model that estimates parcel boundaries that match the reference parcels. At evaluation phase, the component uses the trained model and Sentinel-2 images to output polygons of agricultural parcels in an Esri shapefile or gpkg format.

6.4.2.3 Architecture description and technology stack

The following block diagram shows the training phase of the component, where multi-temporal Sentinel-2 images and reference GSAA are processed and inputted to the deep learning supervised model. Following the training phase, post-processing is applied to vectorise the estimated parcel boundary probabilities. During the evaluation phase, only the trained model and the Sentinel-2 images are required to delineate parcel boundaries.



The component has been implemented in Python 3.5+, with the creation of a Python package that includes all required Python dependencies. In addition, docker containers, PostgreSQL and the gdal geospatial processing library are used for data pre- and post-processing.

The SentinelHub service was employed to access the Sentinel-2 imagery. Amazon Web Services (AWS) were used to store the datasets downloaded through the large-scale batch processing.

Copyright © NIVA Project Consortium

6.4.2.4 System requirements

The provided code was tested on Linux Ubuntu 18.04 machines, although it is not dependent on the operating system, since it relies on technologies supported by Windows, MacOS and Linux.

6.4.2.4.1 Hardware

Hardware requirements depend on the area-of-interest and, consequentially, on the size of data to be processed. The training of the deep learning model is significantly sped up with use of Graphic Processing Units (GPUs), while the majority of the pipelines constituting the component support parallel processing of the Sentinel-2 imagery.

As a reference for disk storage requirements, the data storage required for processing the 2020 season for the entire Lithuania amounted to ~1.3TB, while for Castilla y Leon for a month it amounted to ~500GB. The datasets were stored on AWS S3 buckets.

As a reference for the computational resources required, the processing pipelines were ran on machines with 32 CPU cores and 72GB of RAM, while the training of the deep learning model was executed on a machine with 8 CPU cores, 61GB of RAM and NVIDIA Tesla V100 GPU.

6.4.2.4.2 Software

A Python package *fd* has been created to ease the running of the component in an end-to-end fashion. Jupyter notebooks showcase each sub-component, documenting the entire process. In addition to the Python libraries required by the *fd* package, the following open-source software was used:

- Docker containers for the data pre- and post-processing;
- PostgreSQL to quickly access and process the reference vector parcels;
- Gdal to post-process the parcel boundary probabilities and obtained the final vector parcels.

The SentinelHub payable service is used to seamlessly access and process the Sentinel-2 image archive.

6.4.2.5 Repository access (link and other information)

This component is available with a user guide on NIVA Gitlab:

https://gitlab.com/nivaeu/uc2_fielddelineation

The repository contains the Python package implementing the component, as well as Jupyter notebooks that showcase the end-to-end component, including both training and boundary parcel estimation.

6.4.2.6 Dependencies

The Sentinel-Hub service was used to download Sentinel-2 imagery, while AWS S3 buckets were employed to store the processed data. The component can be easily adapted to work with alternative non-payable services to access and store the data.

Copyright © NIVA Project Consortium

6.4.2.7 Libraries

A list of the Python libraries required by the *fd* field delineation Python package can be found in the *requirements.txt* file in the https://gitlab.com/nivaeu/uc2_fielddelineation repository.

6.4.3 Component Sen4CAP system results as preliminary data

6.4.3.1 Contact information (author, technical reference)

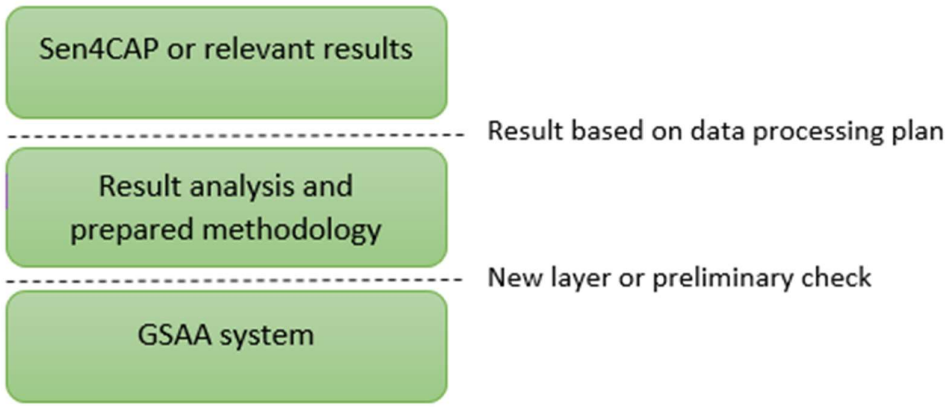
- Martynas Rimgaila (martynas.rimgaila@nma.lt): contact point in charge of the component

6.4.3.2 Features description

Sen4CAP system is an external component (not being developed by NIVA partner) which will facilitate the GSAA application submission process by providing land cover classification, crop classification and activity monitoring thus reducing time spent filling applications and limiting number of farmer errors. By using results of land cover classification, crop classification and activity monitoring it will lead to availability to check if any results could be used before declaration and provide benefits in the beginning and during declaration period as preliminary crop type (summer, winter, land cover, land use).

6.4.3.3 Architecture description and technology stack

The component describes the process of analysis and recommendations. It will let to check if farmers can be provided with additional data for the application submission which will reduce time spent filling applications and limiting number of farmer errors. Paying agency will get less errors in the farmers' applications.



6.4.3.4 System requirements

Sen4CAP system result analysis and methodology is is an operating system and application independent.

Copyright © NIVA Project Consortium

6.4.3.4.1 Hardware

The methodology has no hardware interfaces and has no constraints on hardware architecture.

6.4.3.4.2 Software

ArcGIS software is used in order to perform analysis tasks.

6.4.3.5 Repository access (link and other information)

More information can be found in NIVA sharepoint UC2 folder.

6.4.3.6 Dependencies

This system has no external dependencies.

6.4.3.7 Other modules

GSAA system to provide preliminary data from Sen4CAP system.

Copyright © NIVA Project Consortium

6.5UC3 - Farm Registry

6.5.1 Component Farm Registry

6.5.1.1 Contact information (author, technical reference)

This component has been developed by a TRAGSATEC team leaded by Mercedes Forteza (mfr@tragsa.es).

6.5.1.2 Features description

This Use Case are a Data Model for a Farm Registry and the linked services for publication and publishing of that farms database. This Registry for agricultural areas provides the following information:

- Basic information about the personal data of the owners
- Farm and agricultural plots that make part of it
- Geospatial delimitation of the Farm
- Rainfed or watered crop
- Seedtime
- Use of fertilizers and phytosanitary products: product, dose and application dates

The approach to the UC3 Data Model could be as shown below:

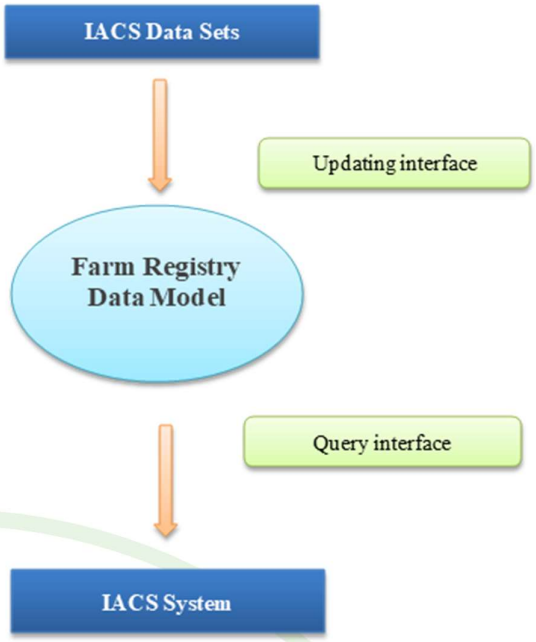
6 Focus on the territorial dimension of farms: Farm Registry
7 A unique farm identification code for the farms across Europe. *The proposal has been included in the WP3 questionnaire to be filled this month.*
8 The agricultural plot would be the minimum work unit: defined as farmer + crop + rainfed/watered
9 The information will be organized in three levels :
10 Where is the farm?: farm, geospatial delimitation and basic use (Agricultural Land, Permanent Pastures, and Permanent Crops).
11 What does the farm produce?: Annual geospatial declaration of rainfed or watered crop
- **How do you produce in the farm?** seedtime, agricultural activity, fertilizers, plant-protection products.

And all this information is made available through JSON interfaces.

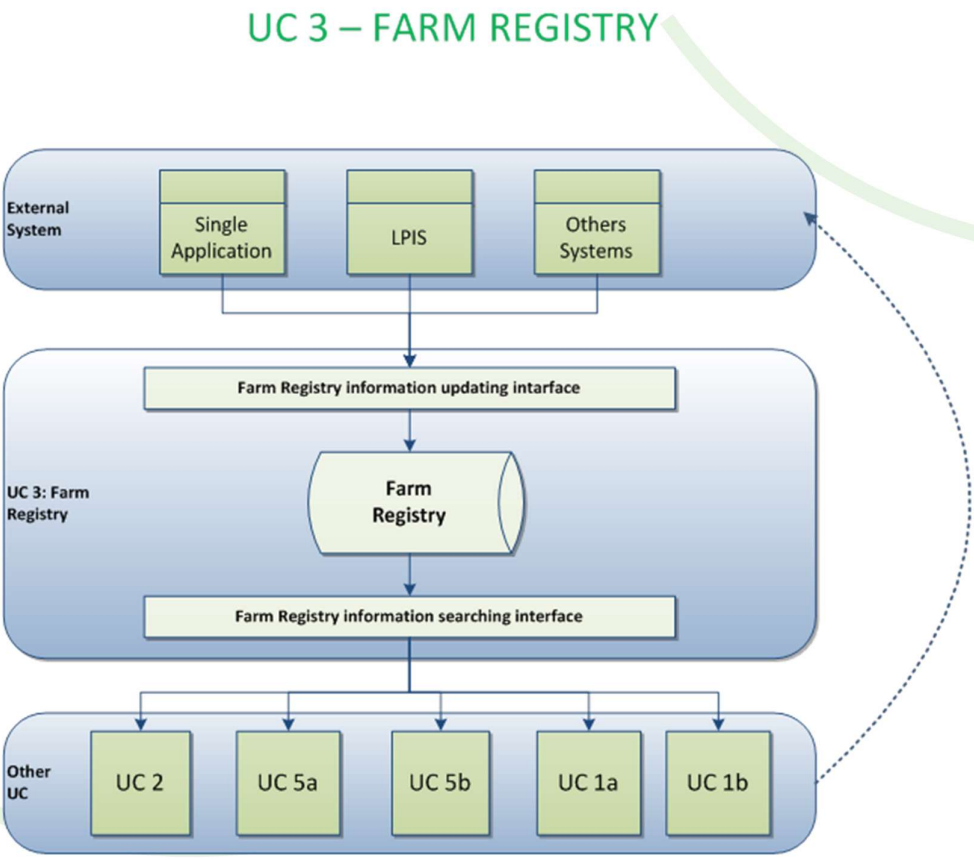
6.5.1.3 Architecture description and technology stack

Basically, this service provides access to the data processed under the Farm Registry data model through two different interfaces: a passive-query interface and an active-writing-updating-deleting interface.

Copyright © NIVA Project Consortium



A more precise architecture diagram, showing the links with external UCs can be seen below:



Copyright © NIVA Project Consortium

6.5.1.4 System requirements

Farm Registry is an independent system and application.

6.5.1.4.1 Hardware

The prototype has no hardware interfaces and has no constraints on hardware architecture. It would be required just a standard publication service.

6.5.1.4.2 Software

In order to perform demo tasks, these software configurations were used:

- First of all, we need that a PostgreSQL database exists, with a schema.
- A Tomcat publication service

6.5.1.5 Communication protocol

JSON interfaces. A communication example could be:


```
{ "CropReportDocument": { "ReportCount": 1, "Description": "eCrop report message", "Type": 1, "Copy": false, "ControlRequirement": false, "Purpose": 1, "LineCount": 440, "Information": "Some random information text", "Status": 1, "Sequence": 1 }, "AgriculturalProducerParty": [{ "Country": "ES", "Name": "Luis Miguel Sánchez", "Classification": 1, "PersonalID": "2260069R", "StructuredAddress": { "Country": "ES", "CityName": "Getafe", "StreetName": "Julián Camarillo", "BuildingNumber": "6B", "Postcode": "28080", "Additional": "nothing", "Email1": "lmsanchez@tragsa.es", "Email2": "email2@tragsa.es", "Phone1": "+34 611 742 101", "Phone2": "0034 612 742 102" }, "AgriculturalProductionUnit": [{ "Key": "APU1", "Country": "ES", "DateFrom": "2015-01-26", "DateTo": "2015-12-31", "StructuredAddress": { "Country": "ES", "CityName": "Madrid", "StreetName": "Alcalá", "BuildingNumber": "100", "Postcode": "28082", "Additional": "parcel", "Email1": "parcel1@tragsa.es", "Email2": "parcel2@tragsa.es", "Phone1": "+34687791122", "Phone2": "+34666777888" }, "FarmingType": [{ "Type": "152", "DateFrom": "2020-01-01", "DateTo": "2020-01-31" }], "Workunit": [{ "Type": 1, "AWU": 8.5, "DateFrom": "2020-02-01", "DateTo": "2020-02-10" }], "CropPlot": [{ "Key": "CP_01", "Start": "2019-09-01", "End": "2019-12-31", "SoilType": [{ "Type": 1, "DateFrom": "2020-07-01", "DateTo": "2020-07-07" }], "Geometry": [{ "Type": "Polygon", "Coordinates": [ [[5.41412794564089, 52.148451591648602], [5.41784357231444, 52.154784683175897], [5.4232748279405, 52.157515523436402], [5.41412794564089, 52.148451591648602]] ], "Surface": 9873.25, "DateFrom": "2021-01-01", "DateTo": "2021-12-31" }], "ReferencePlot": [{ "Key": "RP_01", "DateFrom": "2020-02-01", "DateTo": "2020-02-10", "ReferenceParcel": "28074A017003930000UA00000001", "Geometry": [{ "Type": "Polygon", "Coordinates": [ [[5.49469492804353, 52.124697887324601], [5.49691768162249, 52.119557486966002], [5.5053539453497, 52.119750446627997], [5.49469492804353, 52.124697887324601]] ], "Surface": 5688.45, "DateFrom": "2021-10-01", "DateTo": "2021-10-31" }], "FieldCrop": [{ "Key": "FC_01", "Irrigation": true, "CropProduct": "B18",
```


Copyright © NIVA Project Consortium

```
"LandscapeFeatures": true, "LandTenure": 3, "DateFrom": "2020-01-01",
"DateTo": "2021-12-31", "Geometry": [{ "Type": "Polygon", "Coordinates": [
[[5.47582548628898, 52.112071103593401], [5.47603713431143,
52.105012153653398], [5.47370706432387, 52.104759628267502],
[5.47582548628898, 52.112071103593401]] ], "Surface": 754.12, "DateFrom":
"2021-01-01", "DateTo": "2021-12-31" }], "Activity": {
"AgriculturalActivity": true, "CultivationSystem": 2, "CultivationDetail":
5, "SeedType": 3, "OrganicFarming": 1 }, "ActivityDetail": [{ "LaborType":
1, "StartDate": "2021-01-01", "EndDate": "2021-01-01" }], "CropIrrigation":
[{ "Type": 1, "WaterOrigin": 1, "Volume": 49.99, "Unit": 1, "Date": "2021-
12-31" }], "Phytosanitary": [{ "ID": "0", "Country": "ES", "Dose": 55.99,
"Unit": 1, "Equipment": 1, "StartDate": "2020-02-01", "EndDate": "2020-02-
10" }], "Fertilizer": [{ "ID": 5, "Type": 1, "FertilizationType": 2,
"FertilizationMethod": 1, "Dose": 25.52, "Unit": 1, "StartDate": "2021-01-
01", "EndDate": "2021-01-01" }] }] }] }] }] }] }
```


6.5.1.6 Repository access (link and other information)

This component is located at:

https://gitlab.com/nivaeu/uc3

6.5.1.7 Dependencies

This system has no external dependencies

Copyright © NIVA Project Consortium

6.6UC4a - Geotagged photos

6.6.1 Contact information (author, technical reference)

The GeoTag Photo App (AgriSnap) has been developed by TSSG (WIT) and DAFM (Department of Agriculture Food and the Marine) The contact person for technical enquiries is Dave Hearne (dhearne@tssg.org) and all other questions to Eoin Dooley (Eoin.Dooley@agriculture.gov.ie)

6.6.1.1 Features description

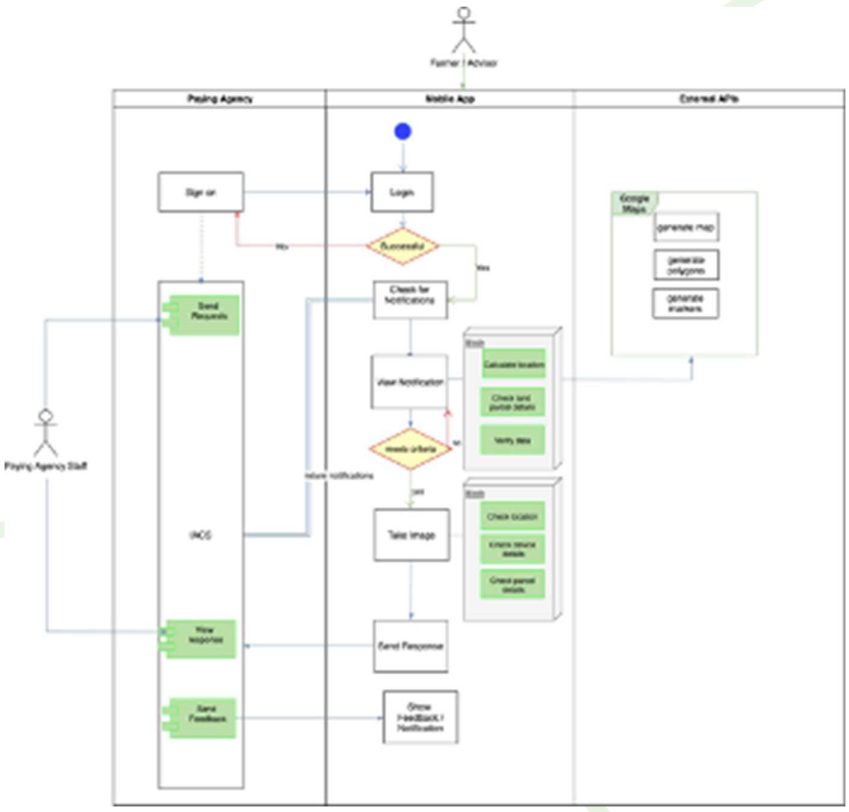
AgriSnap is a hybrid mobile application developed to run on both iOS and android devices.

- It allows the Paying Agency to send requests to an end user (e.g. Farmer, Advisor)
- The user can return images to the paying agency
- Each image is tagged with essential information
  - GeoLocation
  - Device Information
  - User Details
  - Cell Tower Information (Android)
  - GNSS Data (Android)
- Device location is tracked through high accuracy GPS data
- Land Parcel and Request information is available through a map for better user experience
- AR component to assist in image taking (in progress)
- Help functionality with audio clips (in progress)

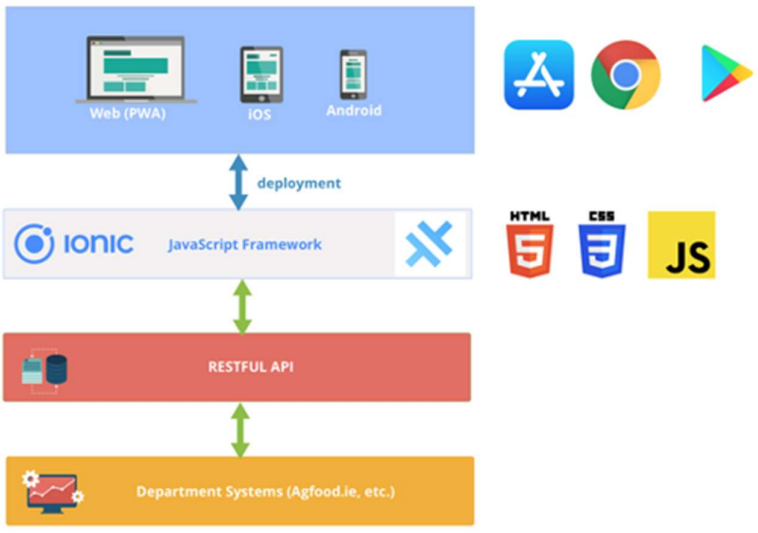
6.6.1.2 Architecture description and technology stack

The main architecture can be seen in the diagrams below:

Copyright © NIVA Project Consortium



Technology Stack



Copyright © NIVA Project Consortium

The technology being used for building the hybrid application is Ionic, an open source multi-platform environment for development.

With Ionic (MIT license) we are using the following stack:

- Capacitor – MIT license
- Angular – MIT license
- SASS – MIT license
- Typescript – Apache 2.0
- HTML

Capacitor is the native runtime that the application runs on , allowing us to interact with native operations on the device.

6.6.1.3 System requirements

6.6.1.3.1 Hardware

Any mobile device will run AgriSnap, for best results we recommend a modern phone as the GPS chip will have a higher accuracy.

6.6.1.3.2 Software

There is no software needed for this, the application is hosted in the Play Store (android) and the App Store (iOS)

6.6.1.4 Communication protocol

For all communications a REST API is used to exchange data, over a secure encrypted HTTPS connections

Communication with the Paying Agency

GET Requests:

A request is made to the Paying Agencies communication layer through an API call, this call returns a JWT (JSON Web Token) which is used to authenticate the user and the response. The data returned is GEOJSON (example below):


```
{
  "id": 218,
  "type": {
    "code": 1,
    "description": "Overclaim MEA land"
  },
  "context": [
    {
```


Copyright © NIVA Project Consortium

```
        "type": "land",
        "label": "J00000011",
        "comment": "DAFM Geotag Photo Request: Please capture a photograph within the indicated parcel.",
        "hash": "786548CA1575BF503BBCB2A5EE890063",
        "type": "land",
        "geoCoordinates": {
          "type": "Polygon",
          "typeName": "Polygon",
          "coordinates": [
            [
              [
                -7.1142588845,
                52.2744269261
              ],
              [
                -7.1136639424,
                52.2742532674
              ],
              [
                -7.1136703742,
                52.2742050288
              ],
              [
                -7.1138343853,
                52.2741600062
              ],
            ]
          }
        },
        {
          "type": "land",
          "label": "J00000011_PT",
          "comment": "DAFM Geotag Photo Request: Please capture a photograph within the parcel, close to the point illustrated on the map. The app will indicate when to capture the photograph.",
```


Copyright © NIVA Project Consortium

```
        "hash": "160206F9A850E42D5DF1AB13F0254D45",
        "type": "land",
        "geoCoordinates": {
          "type": "Point",
          "typeName": "Point",
          "coordinates": [
            -7.1134645565,
            52.2741278471
          ]
        }
      }
  ],
  "scheme": "BPS",
  "geotag": "Y",
  "herdNumber": "J0000001",
  "year": 2021,
  "correspondenceId": 3352497060,
  "correspondenceDocNo": 3386371088,
  "hash": "CAD6E"
}
```


POST Requests:

The data that is returned from the app to the paying agency is also sent over encrypted HTTPS, this data include a BLOB, a binary representation of the image, and JSON data related to the request including geolocation, user details and land parcel details.

Communication with the 3rd Party Dependencies

We communicate with 2 3rd party providers, unwired labs for cell tower information and leaflet for map tiles for Open Street Maps satellite view.

Both of these are GET requests through an API over a secure HTTPS connection.

6.6.1.5 Repository access (link and other information)

All source code is available here:
https://gitlab.com/nivaeu/uc4a_geotagapp

Copyright © NIVA Project Consortium

6.6.1.6 Dependencies

Open Street Maps: This is used to provide a map interface for the end user, it is open source.

Unwired Labs: Used from GNSS4Cap for receiving cell tower information, limited to 100 calls a day for the free tier.

6.6.1.7 Libraries

A list of all installed dependencies can be found below (as of version 0.30)

agrisnap@0.3.0

- @angular-devkit/build-angular@0.1002.0
- @angular-devkit/core@10.2.0
- @angular-devkit/schematics@10.2.0
- @angular/cli@10.2.0
- @angular/common@10.2.1
- @angular/compiler@10.2.1
- @angular/compiler-cli@10.2.1
- @angular/core@10.2.1
- @angular/forms@10.2.1
- @angular/language-service@10.2.1
- @angular/platform-browser@10.2.1
- @angular/platform-browser-dynamic@10.2.1
- @angular/router@10.2.1
- @byteowls/capacitor-oauth2@2.1.0
- @capacitor/android@2.4.2
- @capacitor/cli@2.4.2
- @capacitor/core@2.4.2
- @capacitor/ios@2.4.2
- @ionic-native/android-permissions@5.29.0
- @ionic-native/core@5.29.0
- @ionic-native/device@5.29.0
- @ionic-native/device-orientation@5.29.0
- @ionic-native/email-composer@5.29.0
- @ionic-native/file@5.30.0
- @ionic-native/geolocation@5.30.0
- @ionic-native/gyroscope@5.29.0
- @ionic-native/http@5.29.0
- @ionic-native/network@5.29.0
- @ionic-native/splash-screen@5.29.0
- @ionic-native/sqlite@5.29.0
- @ionic-native/status-bar@5.29.0
- @ionic/angular@5.4.1

Copyright © NIVA Project Consortium

- @ionic/angular-toolkit@2.3.3
- @ionic/pwa-elements@3.0.1
- @ionic/storage@2.3.1
- @ngx-translate/core@13.0.0
- @ngx-translate/http-loader@6.0.0
- @schematics/angular@10.2.0
- @types/axios@0.14.0
- @types/hammerjs@2.0.36
- @types/jasmine@3.6.0
- @types/jasminewd2@2.0.8
- @types/node@14.14.6
- axios@0.21.0
- codelyzer@6.0.1
- cordova-plugin-advanced-http@3.1.0
- cordova-plugin-android-permissions@1.1.0
- cordova-plugin-badge@0.8.8
- cordova-plugin-device@2.0.3
- cordova-plugin-device-orientation@2.0.1
- cordova-plugin-email-composer@0.9.2
- cordova-plugin-file@6.0.2
- cordova-plugin-geolocation@4.1.0
- cordova-plugin-gyroscope@0.1.4
- cordova-plugin-network-information@2.0.2
- cordova-sqlite-storage@5.1.0
- core-js@3.6.5
- ErasmicoinGnssStatus@0.0.1
- ErasmicoinNetworkInfo@0.0.1
- hammerjs@2.0.8
- ion-bottom-drawer@2.0.0
- UNMET PEER DEPENDENCY jasmine-core@3.4.0
- jasmine-spec-reporter@6.0.0
- jetifier@1.6.6
- jwt-decode@3.0.0
- karma@5.2.3
- karma-chrome-launcher@3.1.0
- karma-coverage-istanbul-reporter@3.0.3
- karma-jasmine@4.0.1
- karma-jasmine-html-reporter@1.5.4
- leaflet@1.7.1
- leaflet.offline@2.0.0-beta.4
- protractor@7.0.0

Copyright © NIVA Project Consortium

- rxjs@6.6.3
- ts-node@8.0.3
- tslib@2.0.3
- tslint@6.1.3
- typescript@4.0.5

Copyright © NIVA Project Consortium

6.7UC4b - Machine data in Geo-spatial 'on-line' aid application GSAA as added value data

6.7.1 Component machine data

6.7.1.1 Contact information (author, technical reference)

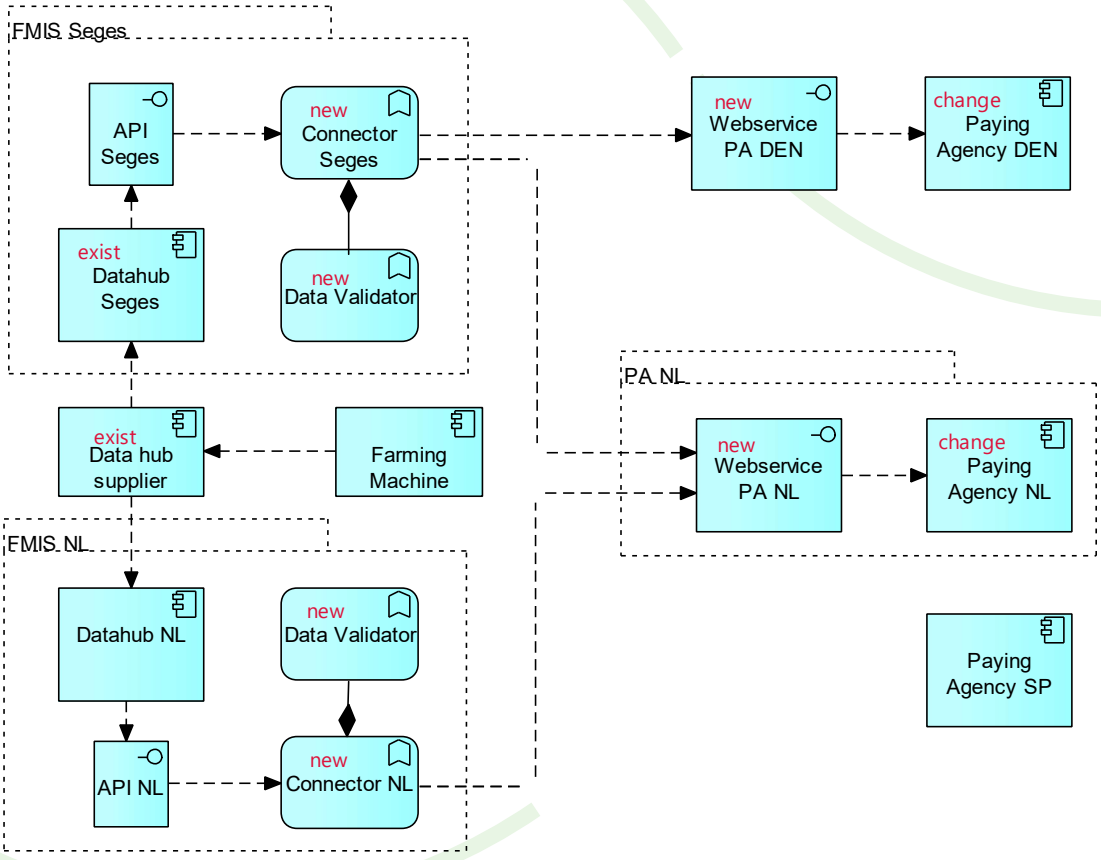
The machinedata app has been developed by WUR and Seges. The contact person for technical enquiries is Yke van Randen (yke.vanranden@wur.nl) for the PA webservice and Finn Møller Andreasen (fima@seges.dk) for the connector.

6.7.1.2 Features description

The aim of the NIVA UC4b project is to gain experience with data from precision agriculture. NIVA wants to investigate how this data can be used in agricultural subsidy processes.

6.7.1.3 Architecture description and technology stack

The application structure (blueprint) focuses on how the application is logically structured.



Functional description of the components

Copyright © NIVA Project Consortium

Component	Description
Farming machine	The farmer works his field with the agricultural machine.
Datahub Machine Supplier	The data hub of the machine supplier, for example from John Deere or Bogballe.
Datahub FMIS	The data hub of the FMIS supplier, for example from Seges or Dacom.
API Seges	The API is the interface of the data hub. The connector reads the data through the API.
Connector	The connector is the application for the farmer, which gives the farmer insight into the activities of precision agriculture. Functions: - Display the performed agricultural activities - Giving an agreement for sharing the agricultural activity with the paying agency. - Show the agricultural activities shared with the paying agency.
Data validator	The data validator is the component that assesses the data from the agricultural machine that actually concerns an activity carried out by the farmer.
Webservice PA	The web service of the paying agency that receives data from precision agriculture.
Paying agency NL	The database that stores precision farming data.

Technology stack of the components

Component	Technology stack
Farming machine	No NIVA component
Datahub Machine Supplier	No NIVA component

Copyright © NIVA Project Consortium

Datahub FMIS	No NIVA component
API Seges	See 6.7.1.4.1.2 below
Connector	See 6.7.1.4.1.2 below
Data validator	See 6.7.1.4.1.2 below
Webservice PA	Java on a JBoss application server
Paying agency NL	Postgres SQL in Postgres database

6.7.1.4 System requirements

6.7.1.4.1.1 Hardware

Component	Technology stack
API Seges	Azure App Service Plan or IIS in windows server environment
Connector frontend	Azure App Service Plan or IIS in windows server environment
Connector backend	Azure App Service Plan or IIS in windows server environment
Data validator	Azure App Service Plan or IIS in windows server environment
Webservice PA	No specific requirements for hardware exist.
Paying agency NL	No specific requirements for hardware exist.

6.7.1.4.1.2 Software

Component	Technology stack
API Seges	Programming language: C# Middleware: .NET
Connector frontend	Programming language: TypeScript Middleware: Angular
Connector backend	Programming language: C# Middleware: .NET Core 3.1
Data validator	Programming language: C# Middleware: .NET Core 3.1
Webservice PA	The Pa Webservice was developed as an JEE WEB Application, JAVA 8, providing several webservices. These services are published by deploying the application in an application server. The webservices use the UTF-8 character encoding. Currently JBOSS Wildfly application server 22.0.0 is used to publish the services for the webapplication. The web application uses basic authentication to authenticate and authorize access. Tis is

Copyright © NIVA Project Consortium

	provided by the application server. The following roles must be defined in the application server nivauc4bR (readonly) ,nivauc4bW (readwrite).
Paying agency NL	The Pa Database is implemented using postgres (PostgreSQL 9.4.26 on x86_64-unknown-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-39), 64-bit) at WENR.
References	Visual Studio 2019: https://visualstudio.microsoft.com/downloads/ .NET Core 3.1: https://dotnet.microsoft.com/download/dotnet-core/3.1 git: https://git-scm.com/downloads

6.7.1.5 Communication protocol

UC4b uses the following communication protocols.

- Oauth for authentication, between the connector and the FMIS.
- eCrop-JSON message between the connector and the PA webservice. The message book is specified in “eCrop used by NIVA Final.docx”

6.7.1.6 Repository access (link and other information)

This component is located at:

https://gitlab.com/nivaeu/uc4b

Name	Description	Supplier model	Number of units	Deployment Site(s)
NIVA front end farmer	Application for the farmer, for sending machine data to the payin gagency	Open source	1	Gitlab
NIVA Connector	Sends data to validate and put through to PA	Open source	1	Gitlab
NIVA Validator	Elementary data check	Open source	1	Gitlab
Web service PA	Webservices to pull in the data.	Open source	1	Gitlab
Database PA	Database to store machine data	Open Source	1	Gitlab

6.7.1.7 Dependencies

The following dependencies exists:

Copyright © NIVA Project Consortium

Component	Description
Proprietary ISOXML message form machine	There are three of four diffenent type of ISOXML messages. The type of ISOXML message depend on the machine builder. Each type of ISOXML message has to be supported. UC4b does not support all types.
FMIS	80% of the information from the eCrop message is read from the FMIS. The software for reading this information is FMIS prpprietary and not open source.

6.7.1.8 Other modules

Other modules are described in the section on dependencies.

Copyright © NIVA Project Consortium

6.8UC5a - Land Parcel Identification System LPIS update ,& change detection

6.8.1 Contact information (author, technical reference)

- Jesper Stendal Sørensen (jesste@lbst.dk): UC5a co-lead and project manager in charge of the technical coordination for Danish development.

- Mette Holst Nielsen (meho@lbst.dk): UC5a co-lead and project manager in charge of the technical coordination for Danish development.

- Guillaume Marchand (guillaume.marchand@ign.fr): UC5a co-lead and project manager in charge of the technical coordination for French development.

6.8.2 Features description

The developments under use case 5a are a set of algorithms developed in Denmark, and two components developed in France. The end goal of the both parts are to be used for updating LPIS.

The Danish set of algorithms consists of five different algorithms for five different types of elements of the updating of LPIS:

- Ponds
- Groups of trees
- Trees in line
- Unfortified roads and paths
- Artificial covered surfaces.

The output of the algorithms are a set of polygons for each type of element, on the area where the algorithms have been run.

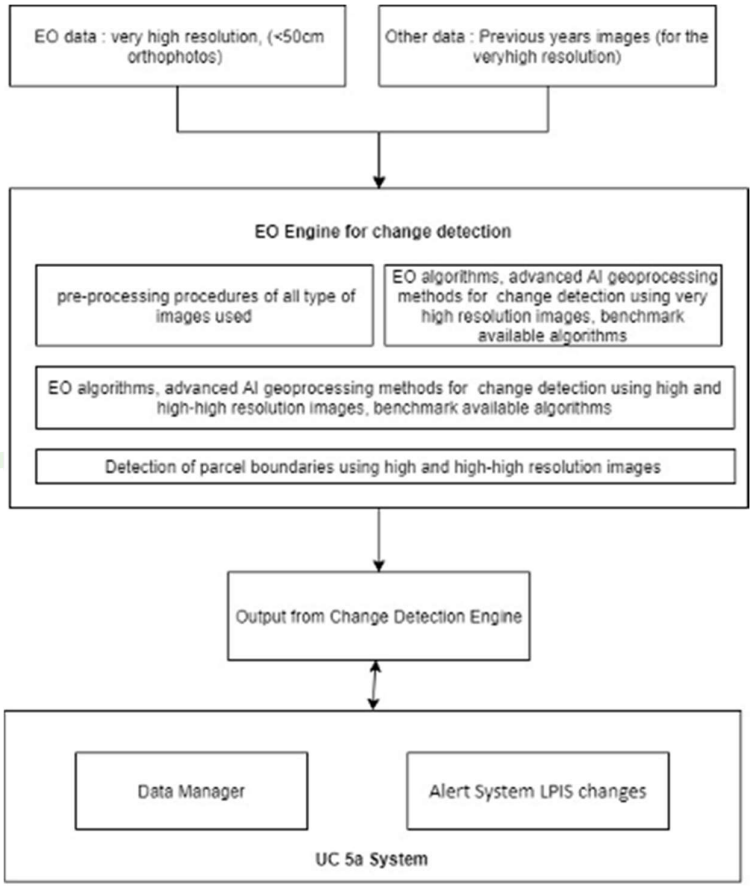
The French components are:

- A segmentation tool to generate vegetation and buildings masks by deep-learning with orthophotos, 3D models and LPIS (main component tested with partners)

A post-processing component to generate geolocated alerts to target changes (second component in addition to the first)

Copyright © NIVA Project Consortium

6.8.3 Architecture description and technology stack



The input data type for the algorithms are orthophotos (raster), RGB, NIR, LPIS (vector) for the Danish development and the same plus DTM and DSM for the French development. The format are reference parcels from LPIS in Shape or GDAL-compatible.

6.8.4 System requirements

Land Parcel Identification System LPIS update, & change detection is a set independent algorithms.

6.8.4.1 Hardware

The Danish algorithms has no hardware interfaces and has no constraints on hardware architecture. French algorithms are working on Linux OS only (Ubuntu 18.04) with 32GB of memory, a single 8GB or 11GB GPU and SSD for code. Computations are not distributed across several GPUs. CPU or GPU can be used for the detection computations.Software

Copyright © NIVA Project Consortium

6.8.4.2 Software

French algorithms called ODEON (Object Delineation on Earth Observations with Neural network) is a set of command-line tools performing with **Conda** which facilitates the creation of an IT environment with all dependencies and open libraries (e.g. Python 3.6 + several geo libs or Pytorch 1.x).

6.8.5 Repository access (link and other information)

The component are located here:
https://gitlab.com/nivaeu/uc5a_daa_segmentation
https://gitlab.com/nivaeu/uc5a_ign_alertcreation
https://gitlab.com/nivaeu/uc5a_ign_maskdetection

6.8.6 Dependencies

Dependencies and open librairies for ODEON French detection algorithms:

appdirs-1.4.4	13 KB
libiconv-1.16	680 KB
lerc-2.2.1	133 KB
blas-2.107	13 KB
boost-cpp-1.74.0	16.1 MB
glib-2.66.4	212 KB
m2w64-libwinpthread-	31 KB
idna-2.10	52 KB
cairo-1.16.0	2.2 MB
chardet-4.0.0	224 KB
win_inet_pton-1.1.0	8 KB
pyqt-impl-5.12.3	4.3 MB
attrs-20.3.0	41 KB
torchvision-0.8.1	7.2 MB
cycler-0.10.0	9 KB
pixman-0.40.0	482 KB
ninja-1.10.2	273 KB
zfp-0.5.5	150 KB
openssl-1.1.1i	5.8 MB
cloudpickle-1.6.0	22 KB

Copyright © NIVA Project Consortium

libdeflate-1.7	61 KB
libzopfli-1.0.3	203 KB
kiwisolver-1.3.1	58 KB
wincertstore-0.2	15 KB
libwebp-base-1.1.0	366 KB
curl-7.71.1	122 KB
proj-7.1.1	2.4 MB
liblapack-3.9.0	3.9 MB
affine-2.3.0	16 KB
intel-openmp-2020.3	2.0 MB
qt-5.12.9	106.3 MB
blas-devel-3.9.0	12 KB
kealib-1.4.14	158 KB
click-7.1.2	64 KB
pywavelets-1.1.1	4.3 MB
jsonschema-3.2.0	45 KB
fiona-1.8.18	991 KB
mkl-devel-2020.4	5.6 MB
toolz-0.11.1	46 KB
libcblas-3.9.0	3.9 MB
m2w64-gcc-libs-core-	214 KB
vs2015_runtime-14.28	1.4 MB
xz-5.2.5	211 KB
tiledb-2.1.6	1.6 MB
threadpoolctl-2.1.0	15 KB
cudatoolkit-11.0.3	942.6 MB
mkl-include-2020.4	696 KB
pyproj-2.6.1.post1	377 KB
pooch-1.3.0	40 KB
pyparsing-2.4.7	60 KB

Copyright © NIVA Project Consortium

decorator-4.4.2	11 KB
tifffile-2020.12.8	124 KB
urllib3-1.26.2	98 KB
tornado-6.1	649 KB
joblib-1.0.0	206 KB
libxml2-2.9.10	3.4 MB
tqdm-4.56.0	76 KB
ca-certificates-2020	173 KB
libkml-1.3.0	8.9 MB
m2w64-gmp-6.1.0	726 KB
zipp-3.4.0	11 KB
imageio-2.9.0	3.1 MB
munch-2.5.0	12 KB
pip-20.3.3	1.1 MB
libtiff-4.2.0	1.1 MB
cligj-0.7.1	10 KB
pytorch-1.7.0	1003.4 MB
libglib-2.66.4	3.3 MB
dask-core-2020.12.0	669 KB
lcms2-2.11	783 KB
scipy-1.6.0	23.3 MB
pyqt-5.12.3	22 KB
m2w64-gcc-libgfortra	342 KB
libnetcdf-4.7.4	602 KB
rtree-0.9.7	45 KB
setuptools-49.6.0	962 KB
poppler-data-0.4.10	3.8 MB
pytz-2020.5	244 KB
requests-2.25.1	51 KB
sqlite-3.34.0	1.2 MB

Copyright © NIVA Project Consortium

hdf4-4.2.13	1.1 MB
gdal-3.1.4	1.3 MB
matplotlib-3.3.3	7 KB
mkl-2020.4	172.4 MB
brotli-1.0.9	882 KB
libuv-1.40.0	330 KB
scikit-learn-0.24.0	6.5 MB
matplotlib-base-3.3.	6.8 MB
pillow-8.1.0	751 KB
postgresql-12.3	21.0 MB
vc-14.2	12 KB
freetype-2.10.4	493 KB
libspatialindex-1.9.	380 KB
charls-2.1.0	103 KB
icu-67.1	16.1 MB
gettext-0.19.8.1	5.0 MB
cytoolz-0.11.0	326 KB
python-dateutil-2.8.	220 KB
pyqtwebengine-5.12.1	143 KB
importlib_metadata-3	3 KB
libcurl-7.71.1	278 KB
poppler-0.89.0	2.2 MB
snappy-1.1.8	50 KB
giflib-5.2.1	85 KB
six-1.15.0	14 KB
click-plugins-1.1.1	9 KB
pysocks-1.7.1	28 KB
packaging-20.8	34 KB
olefile-0.46	32 KB
python-3.8.6	19.1 MB

Copyright © NIVA Project Consortium

jxrlib-1.1	620 KB
liblapacke-3.9.0	3.9 MB
cryptography-3.3.1	593 KB
zstd-1.4.8	958 KB
tk-8.6.10	3.2 MB
libaec-1.0.4	31 KB
cffi-1.14.4	228 KB
yaml-0.2.5	61 KB
libblas-3.9.0	3.9 MB
certifi-2020.12.5	144 KB
numpy-1.19.5	4.9 MB
brotlipy-0.7.0	368 KB
networkx-2.5	1.2 MB
libclang-10.0.1	22.0 MB
libffi-3.3	37 KB
wheel-0.36.2	31 KB
pyyaml-5.3.1	162 KB
libpng-1.6.37	724 KB
lz4-c-1.9.2	279 KB
krb5-1.17.2	856 KB
pandas-1.2.0	10.2 MB
geotiff-1.6.0	136 KB
freexl-1.0.5	56 KB
bzip2-1.0.8	149 KB
hdf5-1.10.6	19.6 MB
pyqt5-sip-4.19.18	298 KB
scikit-image-0.18.1	10.7 MB
libssh2-1.9.0	218 KB
pycparser-2.20	94 KB
xerces-c-3.2.3	3.8 MB

Copyright © NIVA Project Consortium

snuggs-1.4.7	8 KB
expat-2.2.9	261 KB
shapely-1.7.1	426 KB
imagecodecs-2020.12.	6.3 MB
m2w64-gcc-libs-5.3.0	520 KB
libgdal-3.1.4	8.8 MB
pyopenssl-20.0.1	48 KB
blosc-1.21.0	688 KB
libpq-12.3	3.7 MB
typing_extensions-3.	25 KB
python_abi-3.8	4 KB
rasterio-1.1.8	8.0 MB
importlib-metadata-3	21 KB
pcre-8.44	498 KB
msys2-conda-epoch-20	3 KB
geos-3.8.1	1.0 MB
openjpeg-2.3.1	225 KB
cfitsio-3.470	596 KB
pyqtchart-5.12	212 KB
zlib-1.2.11	126 KB
geopandas-0.8.1	925 KB
jpeg-9d	366 KB
pyrsistent-0.17.3	92 KB
libspatialite-5.0.0	10.1 MB

Copyright © NIVA Project Consortium

6.9 UC5b - Scheme Eligibility and Payment Eligibility: Click-and-Pay

6.9.1 Component 1 "(Simulate Payment Entitlements (EU-wide)"

6.9.1.1 Contact information (author, technical reference)

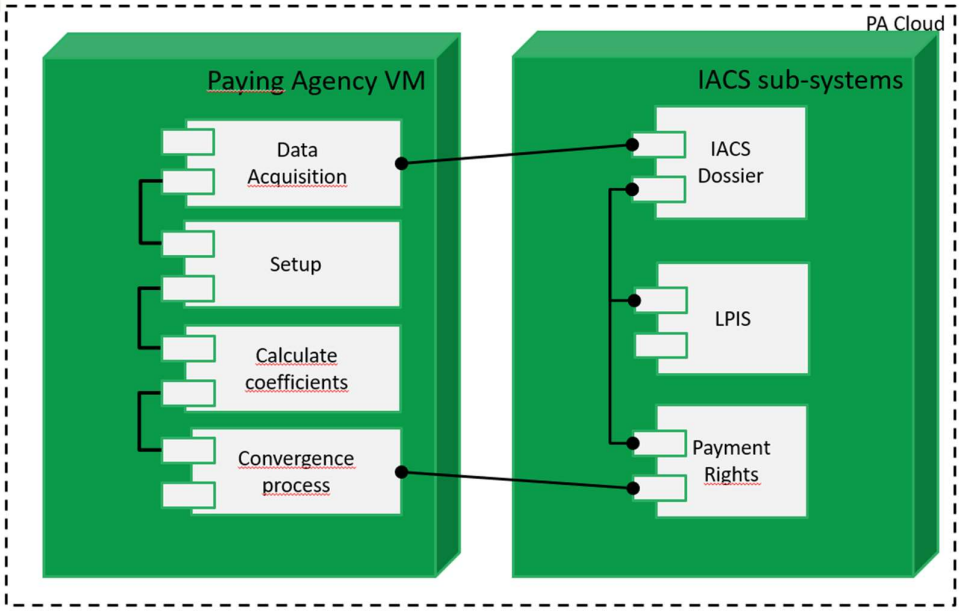
Fabio Pierangeli (fabio.pierangeli@crea.gov.it), Flavio Lupia (flavio.lupia@crea.gov.it)

6.9.1.2 Features description

Quantify the value of payment entitlements under the BISS, during the post-2020 programming period.

6.9.1.3 Architecture description and technology stack

In progress



6.9.1.4 Communication protocol

IN PROGRESS

6.9.1.5 Repository access (link and other information)

IN PROGRESS ON GIT-LAB

Copyright © NIVA Project Consortium

6.9.2 Component "(UC5b.l – Attribute Farm typology in Farm Dossier)"

6.9.2.1 Contact information (author, technical reference)

Antonio Giampaolo (antonio.giampaolo@crea.gov.it), Simonetta De Leo (simonetta.deleo@creag.gov.it), Carlo Di Laura (carlo.dilaora@crea.gov.it) , Stefano Cafarelli (stefano.cafarelli@crea.gov.it)

6.9.2.2 Features description

The Community typology uses the economic criteria to classify farms and to represent the framework of the productive structure of agriculture in a specific territory. The typological classification of farms is a fundamental element to optimize the information framework of the Farm Register, both to allow for a correct identification of the types of production present in a territory according to economic criteria and to favour benchmarking activities to support the PA and all potential users interested in this information.

The implementation of this tool within the Farm Register will allow national PAs and public authorities to initiate the comparability of farms within the MS and between MS.

6.9.2.3 Architecture description and technology stack

The component will include 2 modules:

- a file library (dll dynamic link library) that carries out the processing logic of the classification farm;

- a seamless API (REST JSON based WebService) module for interfacing and exchanging data with the outside integrated services.

The library is in the middle (communication), between the database containing base data and the API module.

It reads some base data in RDBMS MS SQL SERVER (for example the values for specific activities and for a specific UE Region code) and then identify properly, with standard EU algorithms, the agricultural holding in terms of activities for dimension and therefore calculates Type Farming and Economic Size classification.

6.9.2.4 Communication protocol

Some technical details of API (Communication protocol).

REST JSON API is intended for system-to-system integration and for External Systems.

All requests of the current REST JSON API will be received in POST method and with Content-Type: application/json.

All responses returned will be in JSON format (successful responses, business errors), so, will be return with Content-Type: application/json.

In case of successful execution, the API System will return the status code 200 OK and with a valid JSON response.

Copyright © NIVA Project Consortium

In case of error, the API will return, to the external system, the proper status code different from 200 OK and a response with detail error message.

For example, we could back a return status code +400 (bad request) if there are activities unknown in the payload, because will be impossible to understand the proper classification of farm.

6.9.2.5 Repository access (link and other information)

IN PROGRESS ON Git-Lab

Copyright © NIVA Project Consortium

6.9.3 Component 3 "(Click&Pay)"

6.9.3.1 Contact information (author, technical reference)

Fabio Slaviero (f.slaviero@abacogroup.eu); Tommaso Nolli (t.nolli@abacogroup.eu)

6.9.3.2 Features description

The Click&Pay is a seamless claim engine and management that allows to manage EoI (=Expression of Interest) through the user friendly front-end portal.

The front -end portal shows two main lists:

- the "My Expression of Interest" contains the already signed agreements,
- the "New Expression of Interest" contains the list of possible interventions

For each possible intervation, the user can see the "Terms and Conditions" and the requirements to access the fund, and their current status.

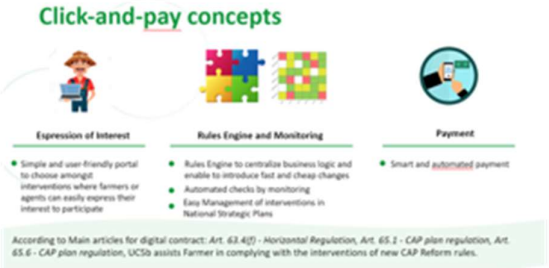
These requirements are checked through "indicators" calculated from several sources:

- Farmer Dossier
- Geospatial application
- LPIS
- Entitlements registry

The subscribed EoI are automatically monitored and, if all requirements are met, the payment can be done.

For each possible intervention, PA users can define the eligibility rules based on indicators (Decision Tables).

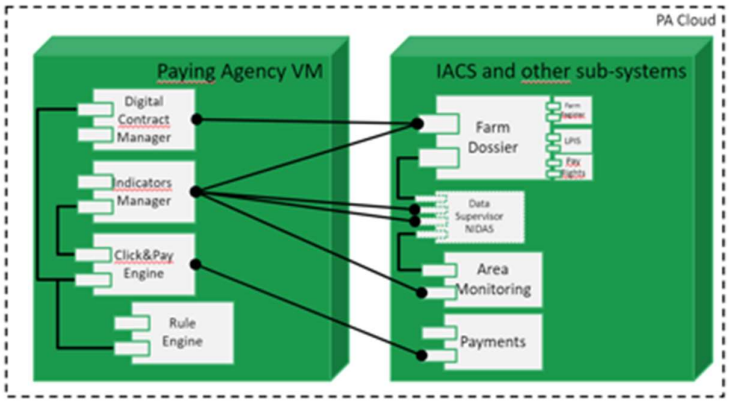
The interface is organized to allow "non-programmers" to do changes to rules. Changes can be even done in MS Excel. The system takes care of versioning of rules.



Copyright © NIVA Project Consortium

6.9.3.3 Architecture description and technology stack

Deployment view (Click-and-Pay)



6.9.3.4 System requirements

Postgresql 12.3, PostGIS 3.0 e OpenJDK 11

6.9.3.4.1 Hardware

Virtual machine with 4 vCPU, 8gb RAM and 50gb Hardisk

6.9.3.4.2 Software

Linux 108

6.9.3.5 Communication protocol

The softare contains a maven module named seamless-claim-sdk; this module contains all the interfaces and classes needed to integrate with external systems

(http; database Postgresql specific protocol, for the publication)

GET /api/contract-types

GET /api/contract-types/parties/{partyId}/is-eligible

POST /api/contracts

GET /api/contracts/farmers/{party_id}

PUT /api/contracts/{contractId}

GET /api/contracts/{contractId}/contract.pdf

GET /api/contracts/{contractId}/land-uses

GET /api/contracts/{partyId}/{contractId}/monitoring-status

GET /api/registry/parties

Copyright © NIVA Project Consortium

GET /api/test-login

6.9.3.6 Libraries

The server side software is written in java and requires java 8 as a minimum version; the dependencies are tracked using standard maven pom files.

The main external libraries are:

- io.dropwizard/dropwizard-core
- io.dropwizard/dropwizard-auth
- com.fasterxml.jackson.core/jackson-annotations
- jakarta.validation/jakarta.validation-api
- org.hibernate.validator/hibernate-validator
- io.dropwizard/dropwizard-jdbi3
- io.dropwizard/dropwizard-client
- org.postgresql/postgresql

Each libraries can have additional dependencies that are automatically imported by maven

The client side software is written in typescript using vue.js; the dependencies are tracked usign standard npm package.json file

The main dependencies are:

- @fortawesome/fontawesome-free: ^5.13.0
- axios: ^0.19.2
- core-js: ^3.6.4
- vue: ^2.6.11
- vue-class-component: ^7.2.3
- vue-debounce-decorator: ^1.0.1
- vue-i18n: ^8.17.6
- vue-property-decorator: ^8.4.1
- vue-router: ^3.1.6
- vue-toasted: ^1.1.28
- vuex: ^3.4.0

6.9.3.7 Other modules

- DROOLS

Copyright © NIVA Project Consortium

6.9.4 Component 4 "(NIDAS)"

6.9.4.1 Contact information (author, technical reference)

Rosario Savarese →rosario.savarese@gmail.com

Manuela Ferri

6.9.4.2 Architecture description and technology stack

The software, of which the primary language is Python, is built upon a Model-View-Controller (MVC) architecture. One of the primary goals of this architecture is to separate the visual display of the information from the business rules and management of the underlying data. For example, if you need to change the way data is organized in the model, it is desirable not to have to make dramatic changes to how you view data. This is true for maintaining flexibility in viewing data. Today, it is common to have many different client applications sharing the same underlying data.

Designing models

The model is essentially the data that makes up your ERP software installation, which is stored in the PostgreSQL database. The ERP software is unique in that database structures are typically defined by the modules at the time they are installed. The software takes the model definitions and automatically creates the necessary table structures inside of the PostgreSQL database. Furthermore, a web interface allows administrators to easily extend the data model in a variety of ways without having to modify the source code.

Rendering views

Each view in the ERP is defined in XML documents. The software framework is responsible for rendering these view files in a web browser. Alternative views can be built to render the ERP functionality upon other platforms such as mobile devices.

Authoring controllers

The controller component of the architecture is where the business logic and workflow rules of the application are applied. The controller components are written in Python code and stored as objects in the modules.

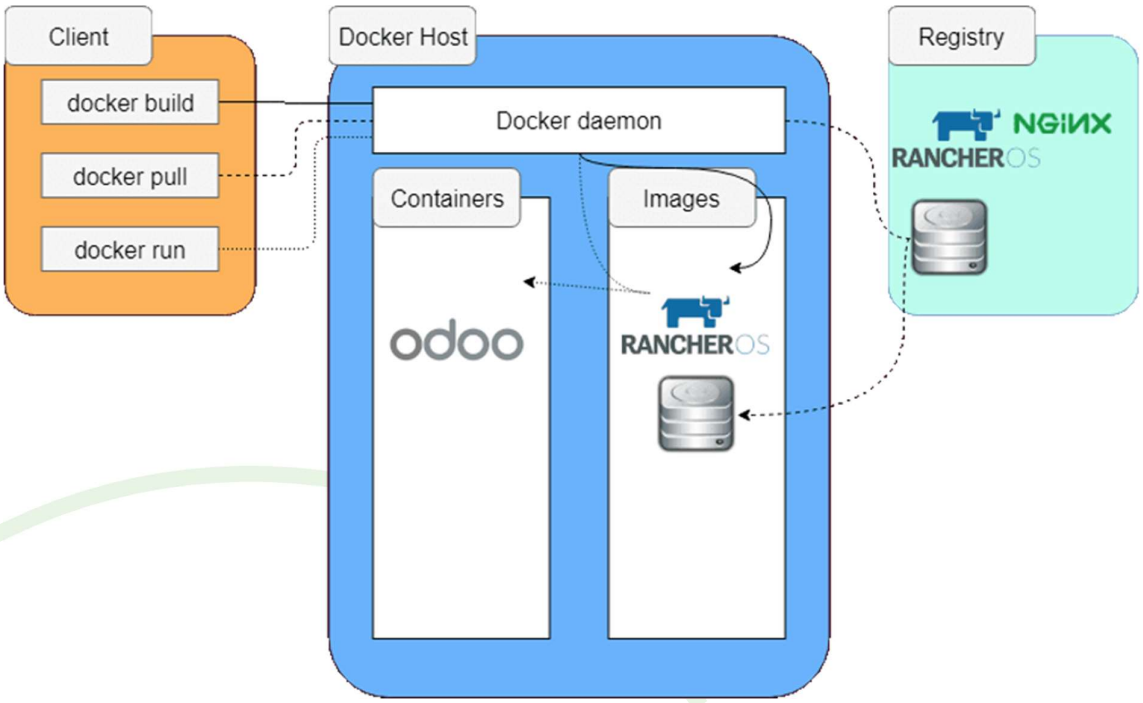
The docker environment

The software runs in a docker environment that comes to give the same solution as Virtual Machines without impacting the performance and efficiency of the real machine. It's kind of a Lightweight Virtual Machine, thanks to its new architecture model.

Dockerize the application means to have an isolated environment "container" that contains the packages up code and all its dependencies so the application runs quickly and reliably. This container (Docker container) that holds your app can be saved as an image (Docker image) and used multiple times in different machines. The Docker engine turns your image to a container in your machine so that you can use your app from this container.

For more information about docker it is suggested to read the official documentation at this link: https://www.docker.com/resources/what-container

Copyright © NIVA Project Consortium



6.9.4.3 System requirements

6.9.4.3.1 Hardware

- cpu: 2
- ram: 4GB
- hdd: 40GB

6.9.4.3.2 Software

- Operating system: Rancher OS
- Docker containers
- Backend: Postgresql, Python 3, XML
- Frontend: HTML, Javascript, QWeb (proprietary), XML

6.9.4.4 Libraries

The Nidas app doesn't have external dependencies.

Copyright © NIVA Project Consortium

6.9.4.5 Other modules

The Nidas app makes use of the mail module, which is needed to add features such as: internal and external communication (via email or simple notifications); chatter used to store the document history; activity scheduling.

Copyright © NIVA Project Consortium

7. Conclusions

The document contains all the components developed for the individual UCs, and highlights the work carried out by the individual owners. As you can see, there are UCs for which more components have been developed using different technological stacks. Although this, they are designed for integration and reuse, fundamental characteristics for the NIVA project. All the projects have been uploaded to the gitlab repository, and some are still subject to daily review. The instructions for the installation and use of the components has not yet been uploaded by everyone, but for those in which it is present, it is instead exhaustive. Missing instructions will be integrated soon by UC Technical leaders.

Copyright © NIVA Project Consortium