D6.5 Integrated AGRICORE tool

agricore

Deliverable Number	D6.5
Lead Beneficiary	IDE
Authors	IDE
Work package	WP6
Delivery Date	M54
Dissemination Level	Public

www.agricore-project.eu

The Agricore project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 816078

Document Information

Project title	Agent-based support tool for the development of agriculture policies
Project acronym	AGRICORE
Project call	H2020-RUR-04-2018-2019
Grant number	816078
Project duration	1.09.2019-31.8.2023 (48 months)

Version History

Version	Description	Organisation	Date
0.1	Deliverable ToC	IDE	05-feb-2024
0.2	Initial Content	IDE	18-feb-2024
0.3	Extended descriptions	IDE	26-mar-2024
0.4	Update to reflect software updates	IDE	24-jun-2024
1.0	Final version	IDE	11-sep-2024

Disclaimer

Executive Summary

D6.5 consists of the software component "Agricore Suite" developed within the AGRICORE project, and that encompass all the software elements developed within it. Accordingly, the actual deliverable is the software produced. However, the focus of this document is to describe how the integration process was done, which elements compose the final AGRICORE Suite, as well as how they works and cooperate between them to support the final goal of the suite, the simulation and analysis of synthetic populations evolution in a given policy framework. The actual results (this is, the developed software) is published as open source (as agreed for most of the AGRICORE project results).

Abbreviations

Abbreviation	Full name
ABM	Agent Based Model
AGRICORE	Agent-based support tool for the development of agriculture policies
DWH	Data WareHouse
EU	European Union
gRPC	general-purpose Remote Procedure Call
I/O	Input/Output

List of Figures

List of Tables

No table of figures entries found.

Table of Contents

1 Introduction

This deliverable is part of the results of WP6 of the AGRICORE project. WP6's main objective is the actual implementation of the software solution of the AGRICORE project. As such, WP6 covered most of the aspects of software development, from architecture definition, interface definition, implementation, integration and monitoring. This WP is strongly related to all WPs in the project, as it covers the integration of previous results (WP1-WP5) which are later demonstrated in WP7.

Specifically, this deliverable is the result of the execution of Task T6.5 Integration of all AGRICORE modules, which objective covered the integration of all the elements composing the AGRICORE tool and the release of the final set of tools whose integration has been validated, together with the platform ready for its use for the envisaged demonstration (WP7) activities.

The structure of this deliverable starts with a brief description of the different elements that compose the AGRICORE Suite. It later delves into the integration process of such elements and describe the main challenges overcome during such integration. Finally, it provides a direct link to the repositories where the different modules composing the AGRICORE simulation module have been published

2 Components of AGRICORE simulation module

Next Figure 1 depicts the overall AGRICORE architecture, with all the elements that compose it. In this image, D.1 (ARDIT) servers as the entry point for the simulation work, providing an straightforward way to identify datasets that contain the features of interests for the simulation to be executed. D.2 (DWH) represents the IT infrastructure where the platform is deployed and includes the necessary tools (e.g. Agricore ORM) to store and access data for the simulations. D.3, D.4 and D.5 represent the modules used to generate synthetic populations from the available census and sample available data. D.6 represents the ABM simulation module, the core of the AGRICORE simulation suite, where the coordination of all the required steps for the simulation takes place. This module includes and/or provides connectivity to the different modules (D.7, D.8, D.9) that interact with the simulation engine to account for transfer of production factors and track market evolution. D.10, D11 and D12 represent the Impact Assessment Modules that calculate the required KPIs to facilitate the evolution of the population on environmental or socio-economic levels. D13 represents the interconnection with external climate modules, connection that finally was not done online (therefore integrated in the suite) but rather established manually to extract yield variances depending on the climate conditions. Finally, D16 (Agricore Interface) provides a GUI to easily define simulation scenarios and to alter the specific policy conditions (D14) to be simulated.

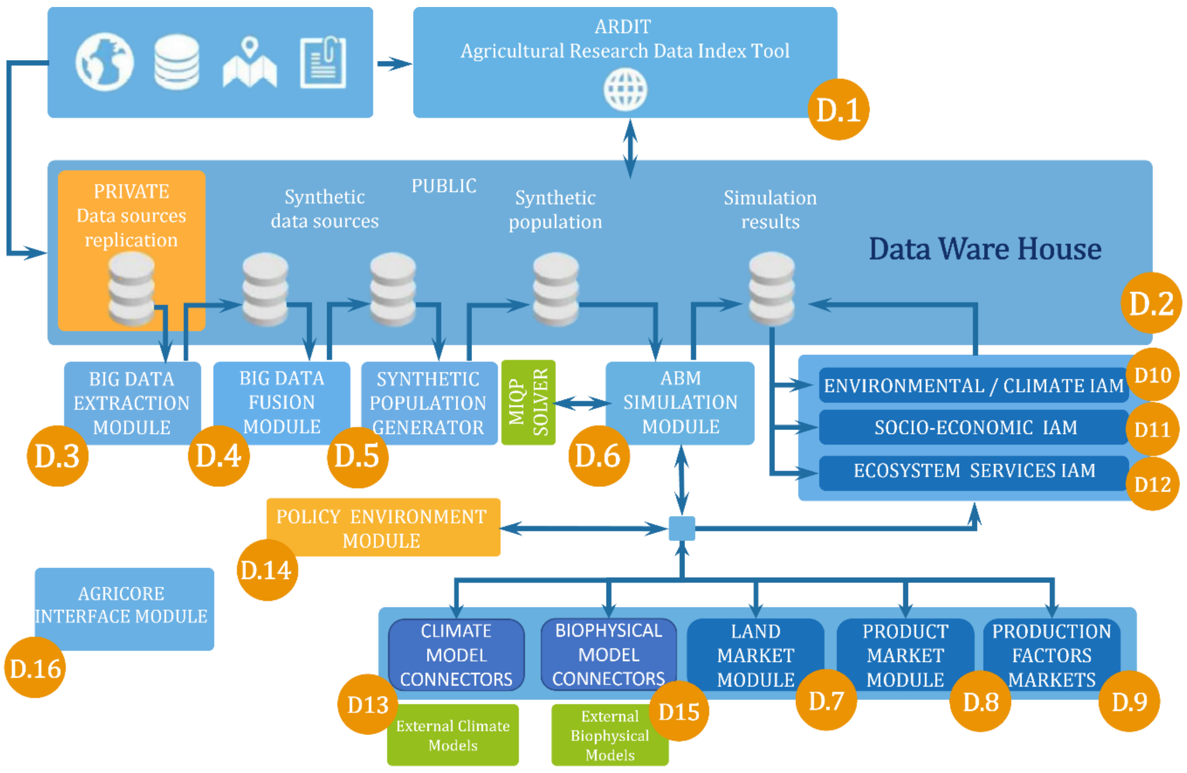


Figure 1: AGRICORE IT architecture

The data workflow is straightforward. A researcher should acquire the required datasets for the definition of the population. Once compiled, the Synthetic Population Generator should be used to generate a synthetic population, which is then uploaded to the DWH. Then, the user can go to the Agricore Interface to define the simulation parameters to be used (e.g. simulation horizon, included policies, etc.). The interface then will interact with the ABM simulation module to schedule all the steps of the population simulation, which status can be tracked from the

AGRICORE interface. Once completed, the user can analyse the data using the embedded panels in the interface, being able to navigate through the results at different levels of granularity.

At the pure software level, the AGRICORE suite is a complex infrastructure that coordinates several modules using a wide set of languages (GAMS, python, C#, R, ...) through proper interfaces. The next Figure 2 depicts the actual software elements that compose the final Agricore suite, and that are published under an open source license.

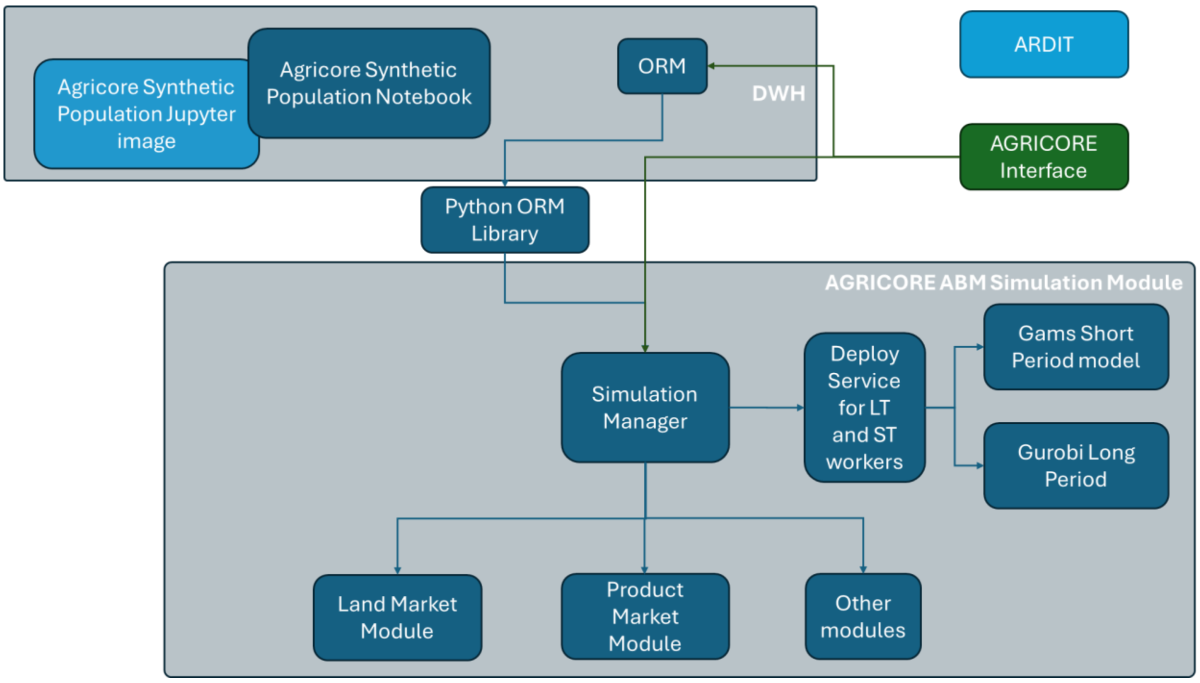


Figure 2: Software modules composing the final Agricore Suite

To further explain how these elements have been developed and how they interact among them, the next sub-sections delve into the elements of the AGRICORE Suite. In any case, for a complete description of the next modules, please check the corresponding modules repository. Please check also the repository and the corresponding documentation for the python library, which provides a clear description of all the typed objects used in the Python developments

2.1 ARDIT

The ARDIT system is loosely coupled with the rest of the Agricore Suite as there is no automatic and direct data exchange with other elements of the suite. However, it has a high relevance to support the identification of data sources that must be used across the synthetic population generation process. It has been implemented as an Angular JS interface and a backend in Java.

2.2 Agricore Interface

The AGRICORE interface is one of the most relevant parts of the AGRICORE Suite as it serves almost as single-entry point for the user interaction. The interface offers the user the possibility to navigate across the entire workflow of the population simulation, as depicted in next Figure 3.

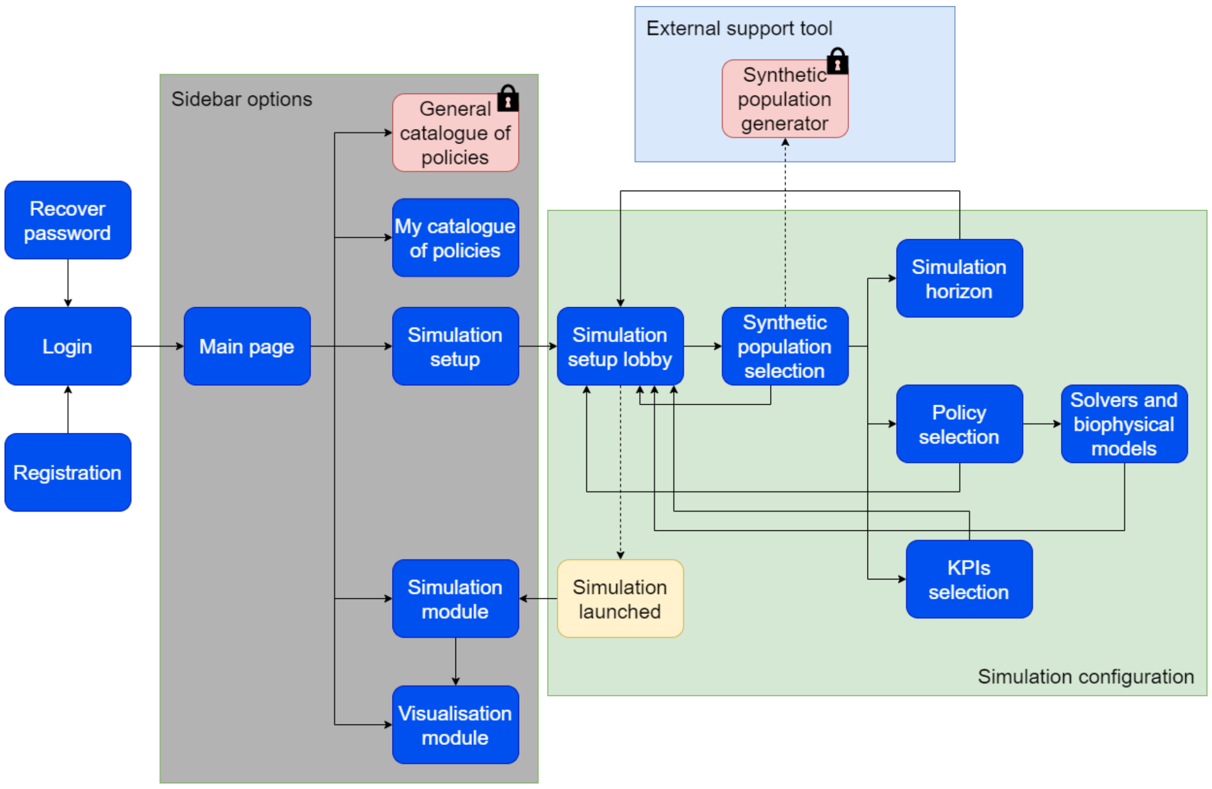


Figure 3: AGRICORE interface functionality

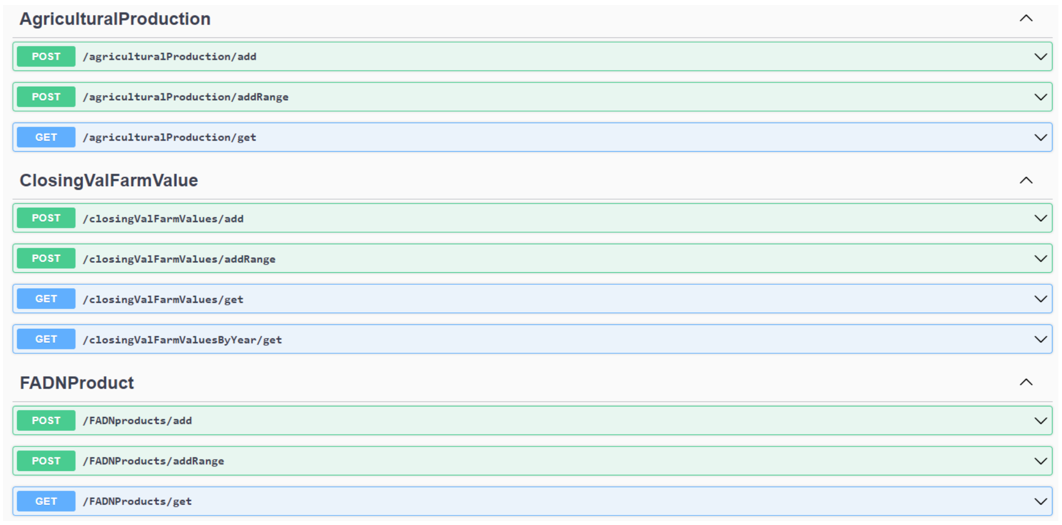
As in ARDIT, the interface is a combination of a Angular JS frontend with a Java backend, which is also coupled with LDAP to manage user permissions.

2.2.1 Impact Assessment Modules

Although the initial approach regarding the Impact Assessment Modules was to use a separate service for their calculation, the final definition of the linked KPI's allowed for a much straightforward implementation, which is the one finally used. Specifically, the KPI's required for giving the functionality expected from the IAM's have been finally implemented as specific equations and calculations within the dashboards, graphs and datasets that are used to display the results to the user, which make the calculations of such KPI's to take place online.

2.3 ORM

The Agricore ORM is a .NET standard solution which has been developed to work as a single-entry point to extract information from the DWH during the simulations. This ORM exposes an extensive API that allows Creating, Reading, Updating and Deleting all the entities involved in the overall scenario simulation, ranging from the synthetic populations to the individual financial registers for each farm and each year in a specific scenario.


AgriculturalProduction

POST /agriculturalProduction/add

POST /agriculturalProduction/addRange

GET /agriculturalProduction/get

ClosingValFarmValue

POST /closingValFarmValues/add

POST /closingValFarmValues/addRange

GET /closingValFarmValues/get

GET /closingValFarmValuesByYear/get

FADNProduct

POST /FADNproducts/add

POST /FADNProducts/addRange

GET /FADNProducts/get

Figure 4: Extract of part of the API exposed by the AGRICORE ORM

The ORM exposes an extensive API that is available for testing through Swagger, and that becomes online when the AGRICORE solution is deployed.

2.4 Agricore Synthetic Population items

For supporting the generation of synthetic populations, 2 main elements have been developed and are used in the process. First, the consortium build a Docker Image that exposes a Jupyter system to facilitate the accessing of the synthetic population generator functionality. This has been done because the actual Synthetic Population Generator uses a complex combination of Python and R code, which requires very specific versioning of the underlying libraries and that is complex to prepare. In order to facilitate this, all these steps a Docker Image is automatically compiled. This image also contain all the source code implemented as a library to support the synthetic population generation. The second element consists of a notebook which explains the steps required to create a synthetic population, with several utilities to check the suitability of the generated populations.

2.5 Python Library

The synthetic-population-model Python Library is a core element of the AGRICORE suite. This library allows an easy access to the information available through the ORM when using Python. Many modules, including the core AGRICORE ABM Simulation Module are developed in python and therefore, makes extensive use of this library to interact with the ORM (and therefore, the project DWH).

2.6 ABM Simulation Module

The ABM Simulation Module, despite being named “module” is a complex solution integrating several elements, as depicted in Figure 5. Specifically, the core of the ABM Simulation Module is the “Simulation Manager”, which is a software (developed in Python) that is in charge of coordinating all the steps of the simulation process, interacting with other elements. On a first stage, the ABM Simulation Module is invoked from the AGRICORE interface, with the specific definition of the to-be-simulated scenario. Then, the ABM Simulation Module coordinates the cyclic process of extracting information from the AGRICORE ORM – which is the element within the DWH that provides the interface to access the information on it -, and invoking the different elements that process this information, specifically the Long Period Model, the Short Period Model (both representing the main ABM dynamics), the Land Market Module, the Product Market Module and other minor actors in the simulation.

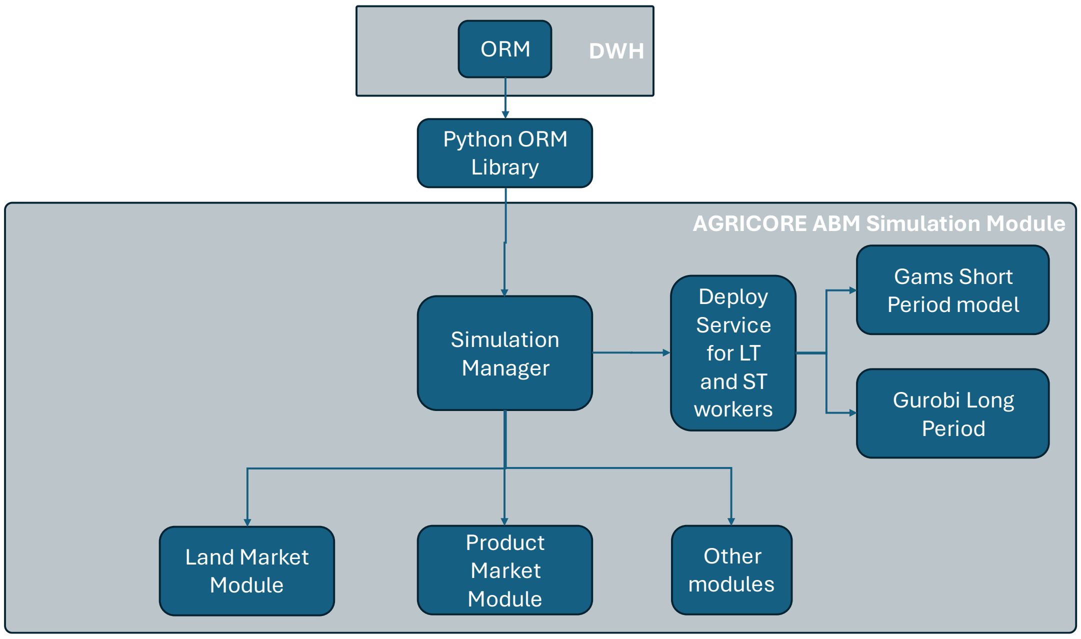
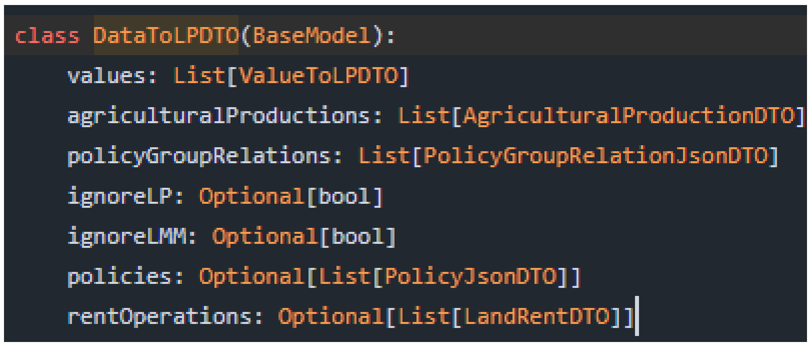


Figure 5: ABM Simulation module breakdown

2.6.1 Long-period simulation module

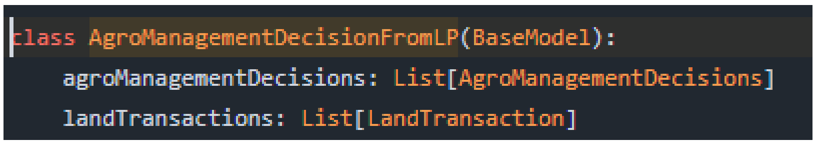
The long-period simulation module has been implemented in python and is also available as a Docker image. The solution exposes a single entry point as a function (process_inputs) which received different inputs and some simulation options. The inputs of the process_inputs mthod are properly typed (DataToLPDTO) in the simulation-models library, as shown next:



```
class DataToLPDTO(BaseModel):
    values: List[ValueToLPDTO]
    agriculturalProductions: List[AgriculturalProductionDTO]
    policyGroupRelations: List[PolicyGroupRelationJsonDTO]
    ignoreLP: Optional[bool]
    ignoreLMM: Optional[bool]
    policies: Optional[List[PolicyJsonDTO]]
    rentOperations: Optional[List[LandRentDTO]]
```

Figure 6: Input data for the Long-period algorithm

The algorithm also allows extra configuration parameters as the number of the year being simulated, the Id of the simulation run, an option to enable/disable parallel processing (to accelerate simulation) among others. The function produces an AgroManagementDecisionFromLP object which is described next:



```
class AgroManagementDecisionFromLP(BaseModel):
    agroManagementDecisions: List[AgroManagementDecisions]
    landTransactions: List[LandTransaction]
```

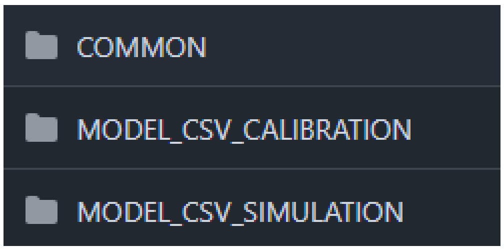
Figure 7: Output data of the long-period algorithm

2.6.2 Land market module

The implementation of the land market module has been done as a sub-module of the previously described long-period model. The reason for this is the adoption of a 2 stages approach where farmers establish their desired financial actions but are then confronted with the reality. This means that the farmers do not always achieve their targeted (un)investments as they participate in a bidding process. Acordingly, the long-period optimisation process is executed twice, one to decide their optimal decisions (and therefore, their land purchase/sell goals), and another to optimised their behaviour accounting for a new realised reality (which includes the actual land transfer operations executed in the market).

2.6.3 Short-period simulation module

The short-period model has been implemented in GAMS, advancing previous research efforts from UNIPR on PMP. The short-period receives from the long-period the long-term goals of the farm (e.g. available cash) and the available land (after land transfers) as well as the status of the farm at the end of the previous season. With this information, the short-period model establishes the optimal operation of the farm in terms of land allocation, considering all applicable constrains as product prices, applicable policy, etc.



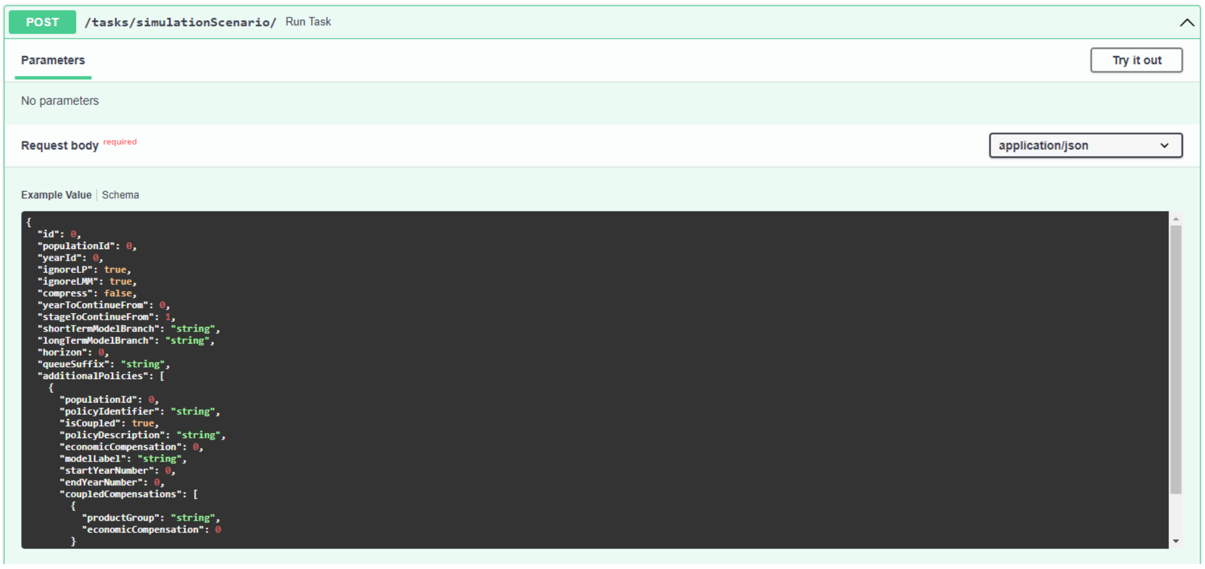
COMMON
MODEL_CSV_CALIBRATION
MODEL_CSV_SIMULATION

Figure 8: Code structure for the short-period model

The implementation is split into 3 main sections of the code. One folder includes all the code regarding the calibrating process, where individual production factors and costs are estimated using PMP. A second folder includes the code for the simulation process, which uses the previously established calibration data and the before described inputs to calculate the updated status of the farm after the year, especially regarding land allocation (as other parameters are overwritten later with realized information). Finally, the COMMON folder stores all the code and algorithms that are shared by the two stages of the simulation.

2.6.4 Simulation Manager

The simulation manager is the core of the ABM Simulation Module implementation. This module is implemented in Python and exposes an API with a single-entry point (/tasks/simulationScenario). This entry point is called by the AGRICORE interface when running a simulation, and its request body includes all the identification data regarding the scenario to be simulated. It also allows the user to disable different stages of the simulation (e.g. Long-Period, Market module) as well as to continue previously started simulations.



```
{
  "id": 0,
  "populationId": 0,
  "yearId": 0,
  "ignoreLP": true,
  "ignoreLMM": true,
  "compress": false,
  "yearToContinueFrom": 0,
  "stageToContinueFrom": 1,
  "shortTermModelBranch": "string",
  "longTermModelBranch": "string",
  "horizon": 0,
  "queueSuffix": "string",
  "additionalPolicies": [
    {
      "populationId": 0,
      "policyIdentifier": "string",
      "isCoupled": true,
      "policyDescription": "string",
      "economicCompensation": 0,
      "modelLabel": "string",
      "startYearNumber": 0,
      "endYearNumber": 0,
      "coupledCompensations": [
        {
          "productGroup": "string",
          "economicCompensation": 0
        }
```

Figure 9: API exposed method for the simulation manager

The module integrates an engine that allows for multiple simulations to be requested in parallel, each one been processed by different workers. The module makes use of celery task queues to split the different stages of the simulation (as depicted in previous **Error! Reference source not found.**). The deployment (further detailed in Deliverable 6.5) generates a number (10 by default) of workers that can process stages separately, even for different simulation scenarios.

The repository containing the simulation-manager code not only includes the API and the core code for the instantiation of the simulation manager service, but also the code and corresponding Dockerfiles that can be used to instantiate the required workers. This covers the actual simulation-manager worker, the long-term worker, the short-term worker and short-term scheduler, as well as a panel (flower celery) that allows the monitoring of the active and historic tasks.

2.6.5 Solver interconnection modules.

During the first tests executed on the completed modules, we realised that the current implementations of the short-period model and of the long-period model were not exploiting the hardware capabilities of the servers where the system was being tested (and would be later deployed). The reasons for that were:

- The short-term model was implemented in GAMS. The version that was being used of GAMS did not incorporate appropriate multi-threading or multi-processing capabilities. Although GAMS offers a product (GAMS Grid) that extended GAMS functionality to support multi-CPU alternatives, it was deemed not reasonable to require such a license to allow AGRICORE simulations.
- Similarly, the long-term model, implemented in python, did not directly incorporate efficient multi-threading or multi-processing capabilities either.

Due to this situation, several options were explored and tested to achieve the maximum performance for a given hardware capabilities. As a result, two types of optimisations were implemented:

- On the one hand, the long-term worker (and therefore, the corresponding long-term model algorithm) were modified to integrate multi-processing capabilities. In this case, the software was adapted to include Dask, a known library for multi-processing Pandas, which also have a task system that enables parallel execution of python code in the same machine.

- On the other hand, for the short-term model, a different approach was followed. In this case, given that the implementation of the short-model was suitable to circumscribe its optimisation only for small regions (where land renting was allowed), the decision was to split the overall simulation by regions (region level 3 – in AGRICORE refers to Agricultural region). Then, the simulation manager implements an scheduling system that split the simulation data, creating different simulation problems and generating the required folders and requests. Then, each solver where GAMS has a proper license implements a number of short-term workers that ultimately allow for parallel simulation of different agrarian regions at the same time, while properly maximising the usage of the underlying hardware infrastructure.

All these modifications were implemented in the repository of the simulation manager and embedded in the corresponding worker implementations.

3 Integration of the Agricore complete platform and challenges

One of the greatest challenges for the AGRICORE project has been during the development and integration of the ABMS Simulation Manager, specifically in the interconnection of the short-period model and the long-period models. First, the initial approach targeted and consistent implementation of a single model in python (which would later interact with the corresponding solvers). However, the availability of previously tested and community validated (only on single year scenarios) short-term model by UNIPR and the fact that the overall optimisation that AGRICORE propose was suited better for a 2-stage approach led the consortium to embrace the current implementation. This consisted of 2 different optimisation systems, one for long-term operations, implemented in Python, and one for short-term optimisation, implemented in GAMS.

Although this decision has brought several advantages (as use of previously validated models) it also entailed several complications, as the usage of two different languages for the implementation of algorithms, the need for different approaches to optimise time execution, the increased complexity for the definition and implementation of interfaces (especially in the case of GAMS), etc.

The rest of the integration activities within the AGRICORE Suite, have been more or less straightforward. However, the consortium members have different level of expertise in terms of software coding, and the project adhered to the principle of freedom of code (this is, allow each entity to use the programming language and frameworks of their choice). This has a high relevance in terms of showing the modularity of the system, as every actor on it communicates only through properly defined interfaces which are tested during the integration. However, this also poses some challenges regarding cross-partner collaborations and lack of consistence within the project. Overall, as in any software integration, great efforts were needed to complete the integration of the system, which, nonetheless, were completed succesfully.

4 Code availability and documentation

As stated in the original AGRICORE proposal, all the project results are published as opensource in the project repository, including clear documentation that supports the usage and further implementation and improvement of the project outputs.

For the Agricore suite, this entails several repositories, including the Agricore-deployment, the interface-module, the synthetic-population-models, the synthetic-population-jupyter-image, the synthetic-population-notebookm, the ABM-object-relational-mapping, the simulation-manager, the financial-optimization-module-long-term, the model-short-period and the solver-workers-deploy ones, which are all available at the project GitHub page: https://github.com/orgs/AGRICORE-project.

The content of this repositories is being populated after the completion of the corresponding documentation, the verification of the included comments and the removal of any testing and/or deployment data that could have been used during the project implementation.

5 Conclusions

The development process of the AGRICORE suite was completed after intense work and collaboration between the AGRICORE consortium members. As a result, a system that allows generating synthetic populations, simulating their evolution in different policy conditions and analysing the results was properly completed. All the code used for its implementation is being published under open-source conditions for further study, reuse and improvement.

For preparing this report, the following deliverables have been taken into consideration:

Deliverable Number	Deliverable Title	Lead beneficiary	Type	Dissemination Level	Due date
D6.4	AGRICORE Simulation Module	IDE	O	Public	M54