D6.4 AGRICORE simulation module

agricore

Deliverable Number	D6.4
Lead Beneficiary	IDE
Authors	IDENER
Work package	WP6
Delivery Date	M58
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	25-ene-2024
0.2	Initial Content	IDE	15-feb-2024
0.3	Extended descriptions	IDE	25-mar-2024
0.4	Update to reflect software updates	IDE	18-jun-2024
1.0	Final version	IDE	10-sep-2024

Disclaimer

All the contributors to this deliverable declare that they:

- Are aware that plagiarism and/or literal utilisation (copy) of materials and texts from other Projects, works and deliverables must be avoided and may be subject to disciplinary actions against the related partners and/or the Project consortium by the EU.
- Confirm that all their individual contributions to this deliverable are genuine and their own work or the work of their teams working in the Project, except where is explicitly indicated otherwise.
- Have followed the required conventions in referencing the thoughts, ideas and texts made outside the Project.

Executive Summary

D6.4 consists of the software component "Simulation Module" developed within the AGRICORE project, and that is part of the overall AGRICORE simulation infrastructure. Accordingly, the actual deliverable is the software produced. However, the focus of this document is to describe how the Simulation Module was built, as well as how it works, in order to support the actual software which 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
EU	European Union
DWH	Data WareHouse
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 T6.4 Agent-based Simulation module, which objective covered the implementation of an object-oriented simulation module that allows the simulation of the evolution of the ABM population according to the dynamics of the agents implemented in WP3. Moreover, it covers the interconnection with external solvers and modules to properly simulate the evolution of the population.

The structure of this deliverable starts with a brief description of the different elements that compose the AGRICORE Simulation Module. 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

The simulation module is the central piece of the AGRICORE solution. As depicted in next Figure 1 in interconnects with different modules to extract data and generate the evolution of the populations. Specifically, the ABM Simulation Module (represented in the figure with the key D.6) extracts data from the Data Ware House, particularly the data regarding the Synthetic population. With this information, as well with the definition of the simulation scenario, the ABM simulation module is in charge of simulate the evolution of such population, interacting with required elements as the solvers and the algorithms used for the simulations. The results of such simulation is then saved back into the DWH to later consult through the AGRICORE interface.

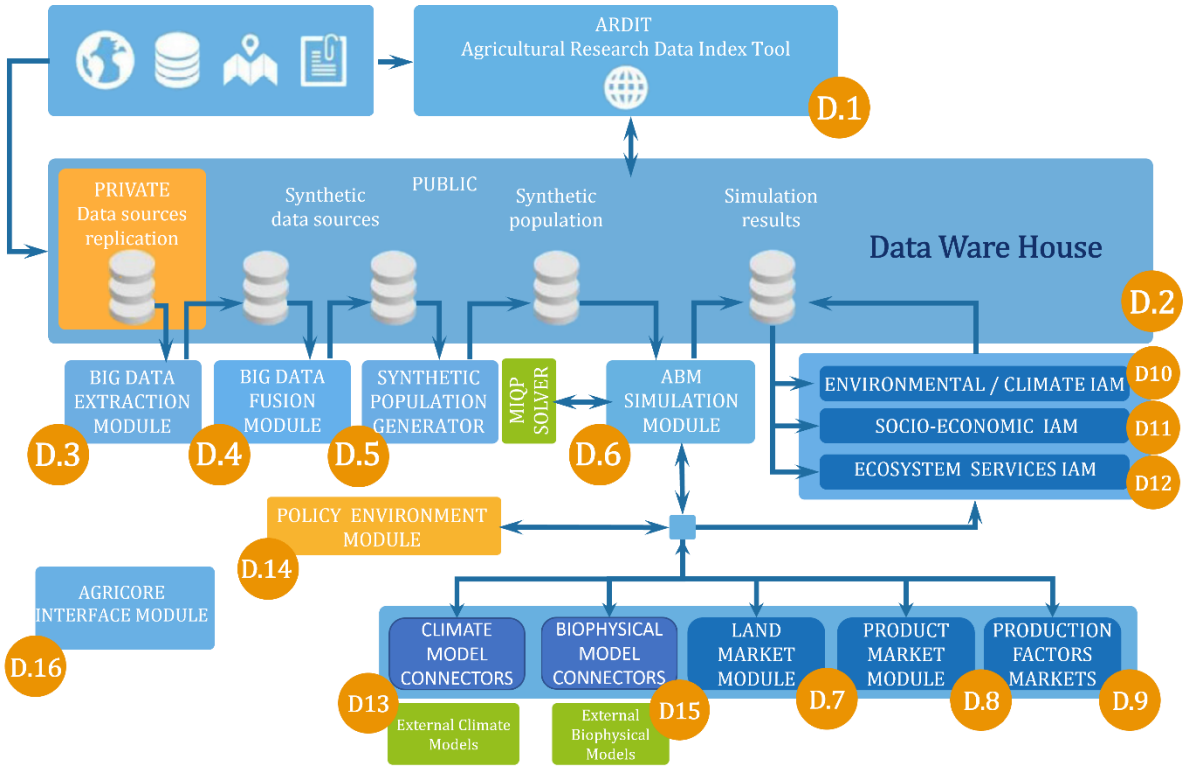

Figure 1: AGRICORE IT architecture

The ABM Simulation Module, despite being named "module" is a complex solution integrating several elements, as depicted in Figure 2. 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.

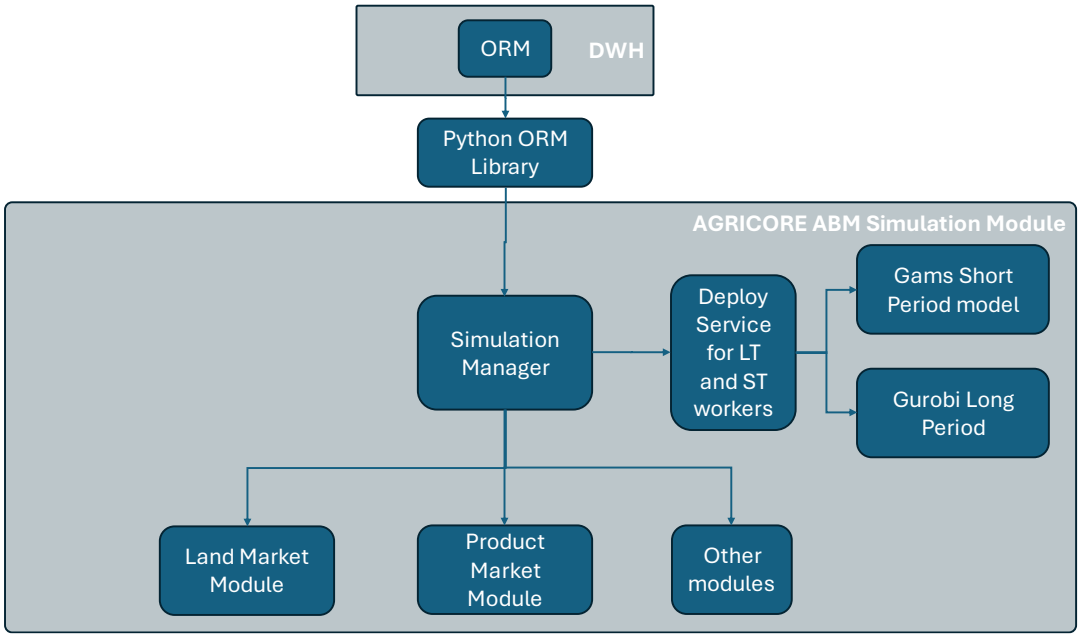

Figure 2: ABM Simulation module breakdown

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 ABM Simulation Module.

2.1 Working Framework

Before starting the explanation of each one of the ABM Simulation Module elements, it is important to describe the interaction with other elements that, while not strictly a part of the ABM Simulation Module, play a key role in its operation.

The first element in this category is the Agricore 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.

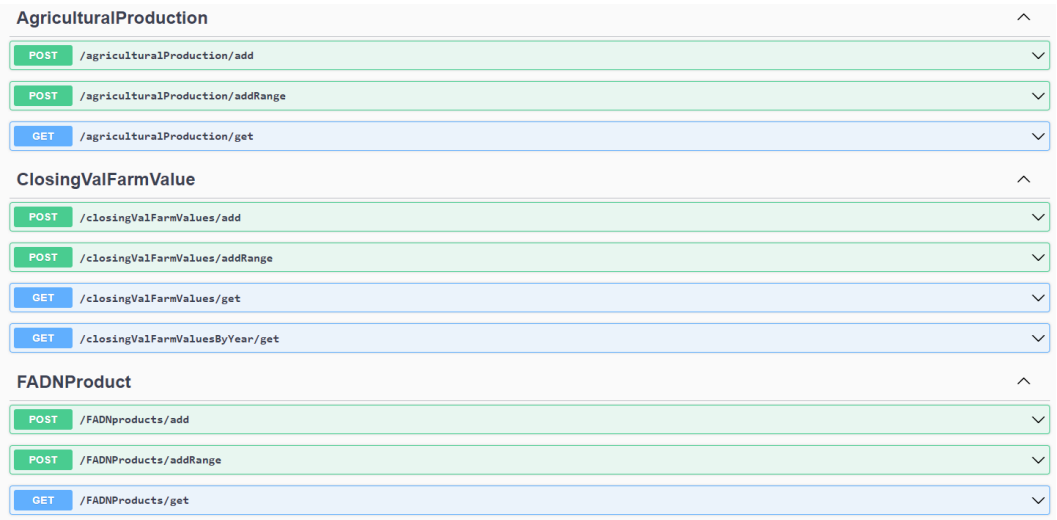

Figure 3: Extract of part of the API exposed by the AGRICORE ORM

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

The second main external element used by the Agricore ABM Simulation Module is the AGRICORE synthetic-population-model Python Library. This library allows an easy access to the information available through the ORM when using Python. As mentioned in the introduction, the AGRICORE ABM Simulation Module is developed in python and therefore, makes extensive use of this library to interact with the ORM (and therefore, the project DWH).

Finally, and although not a software module, the third element which is significantly used by the Simulation Module (but yet, external to it), is the actual data regarding the populations, which is accessible through the synthetic-population-model python library.

In order to facilitate the understanding of the next sections, the simplified workflow followed by the ABM Simulation Manager is represented in Figure 4.

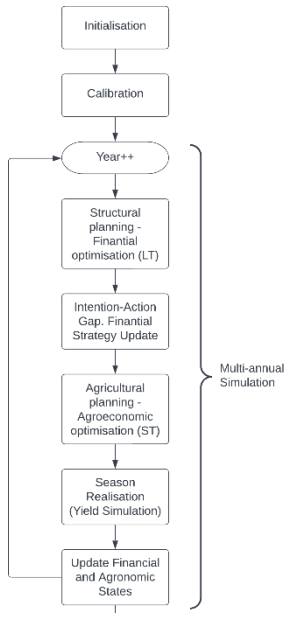

Figure 4: Simplified workflow of the ABM Simulation Module

The process starts with the Calibration of the Short period module, which is done once for each complete simulation. Once the model is calibrated, the behaviour of the farmers composing the population is calculated, iterating through the Long-Period simulation module (which provides long-term financial decisions), the Land Market Module (which deals with the exchange of production factors (land) between agents), the Short-Period module (that mimics the land allocation for each farmer) and the update of financial and agronomic states (which happens in the ORM when saving simulation results). For this last stage, connection to a Product market module and/or an Environmental module can be established to get realised production and economic data resulting of the decisions of the farmer. When not provided, prices and yield factors are considered un-changed across years.

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.

2.2 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 5: 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 6: Output data of the long-period algorithm

2.3 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).

2.4 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.

Figure 7: 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.

2.5 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.

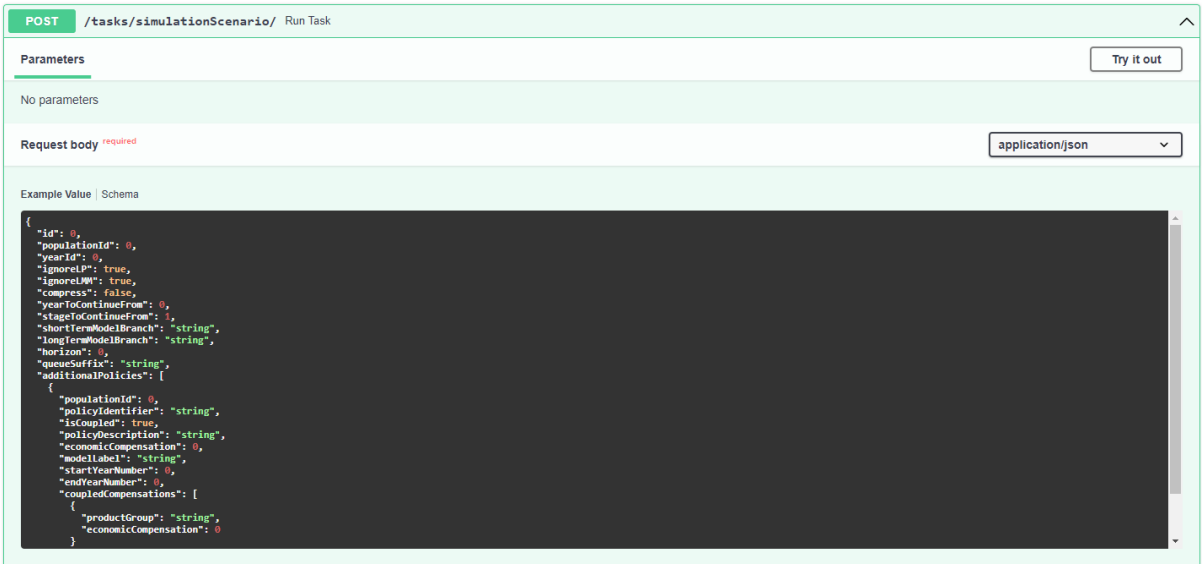

Figure 8: API exposed method for the simulation manager

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 Figure 4). 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 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 process of the simulation module and challenges

One of the greatest challenges for the AGRICORE project has been 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 simulation module, have been more or less straightforward, but as any software integration, quite exhaustive in terms of the required effort.

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 simulation module, this entails several repositories, including the simulation-manager, financial-optimization-module-long-term, model-short-period and the solver-workers-deploy, 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 ABM Simulation Module was completed after intense work and collaboration between the AGRICORE consortium. As a result, a system that allows simulating the evolution of synthetic populations (available in the project DWH through the ORM) in different conditions 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