IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 20, 2016, accepted June 14, 2016, date of publication July 18, 2016, date of current version August 26, 2016.

Digital Object Identifier 10.1109/ACCESS.2016.2592418

Practical Points for the Software Development of an Agent-Based Model of a Coupled Human-Natural System

SANTIAGO L. ROVERE[1], MICHAEL J. NORTH[2,3], (Senior Member, IEEE), GUILLERMO P. PODESTÁ[4], AND FEDERICO E. BERT[5,6]
[1]Grupo de Aplicación de Modelos de Agentes, Universidad de Buenos Aires, Buenos Aires C1127AAR, Argentina
[2]Argonne National Laboratory, Argonne, IL 60439-4854, USA
[3]Computation Institute, The University of Chicago and Argonne, Chicago, IL 60637-1403, USA
[4]Rosenstiel School of Marine and Atmospheric Science, University of Miami, Miami, FL 33149, USA
[5]Asociación Argentina de Consorcios Regionales de Experimentación Agrícola, Buenos Aires C1041AAZ, Argentina
[6]Universidad de Buenos Aires, Universidad de Buenos Aires, Buenos Aires C1127AAR, Argentina

Corresponding author: S. L. Rovere (srovere@gmail.com)

This work was supported in part by the Inter-American Institute for Global Change Research (IAI) under Grant CRN-2031 (Addendum), in part by the U.S. National Science Foundation (NSF) under Grant 0709681, Grant 1211613, and Grant 1049109, in part by NSF through IAI under Grant GEO-0452325. The work of F. E. Bert is supported in part by the Consejo Nacional de Investigaciones Científicas y Técnicas of Argentina and in part by the Argonne National Laboratory, a U.S. Department of Energy Office of Science Laboratory through The University of Chicago under Contract W-31-109-Eng-38.

ABSTRACT Modeling complex natural and human systems to support policy or management decision making is becoming increasingly common. The resulting models are often designed and implemented by researchers or domain experts with limited software engineering expertise. To help this important audience, we present our experience and share lessons learned from the design and implementation of an agent-based model of agricultural production systems in the Argentine Pampas, emphasizing the software engineering perspective. We discuss the model's design including the model classes; the activity diagram, and data flow; the package and folder layout; the use of design patterns; performance optimization; initialization approaches; the analysis of results; and model measurement, validation, and verification.

INDEX TERMS Complex adaptive systems, agent-based modeling, coupled human and natural systems, model design and implementation, software engineering.

I. INTRODUCTION

The Argentine Pampas is one of the leading cereal and oilseed producing areas in the world [1]. Production systems in the Pampas have evolved rapidly during recent decades [2]. This evolution has been shaped by multiple climatic [3], [4], technological [5]–[7] and global and local economic and political [2], [8] drivers. The most significant changes have been: (a) an increase in the area operated by individual farmers, accompanied by a decrease in the number of active farmers, (b) an increase in the amount of land operated by tenants, and (c) changes in land use patterns, in particular, the increasing dominance of soybean [9].

Agricultural systems are coupled human and natural systems (CHANS) in which people interact with natural factors. CHANS involve ecological variables (e.g., landscape patterns, crop growth, and groundwater depth), human variables (e.g., socioeconomic processes, land use patterns, land tenure regimes, and governance structures) as well as variables linked both to ecological and human processes (e.g., use of ecosystem services such as food and fiber provision) [10]. In CHANS, people and nature interact reciprocally and form complex feedback loops [10], [11]. Furthermore, changes in the environmental and socioeconomic contexts often induce individuals to adapt.

Systems that show both complexity and adaptability are known as complex adaptive systems (CASs). CASs are composed of autonomous agents (individuals, groups, institutions or other entities) that interact with, and influence each other, learn from their experiences, and adapt their behaviors [12]. Complexity in CASs involving natural and human systems (i.e., CHANS) is characterized by (a) nonlinearity (i.e., system behaviors are not directly proportional to inputs) (b) heterogeneity (i.e., attributes and behaviors may vary among individuals), (c) thresholds (i.e., abrupt transition points between alternate states), (d) time lags (i.e., delays between human-nature interactions and the perception or detection of their ecological or socioeconomic effects), (e) resilience (i.e., the capability to retain structures and functioning after disturbances), (f) legacy effects (i.e., impacts of prior human-nature couplings on later conditions) and (g) emergent patterns (i.e., macroscopic properties which emerge out of local small-scale interactions) [10], [13]. Not all these characteristics must be simultaneously present in a system for it to be considered complex. Another characteristic of CASs is adaptability – variations over time of the individuals' rules, behaviors and structures that are induced by changes in the environment or other contexts. Adaptation emerges as a result of (a) interactions among groups of individuals or (b) interactions between individuals and their environment and context [13].

Agricultural production systems in the Argentine Pampas can be considered as both a CAS and a CHANS as they combine complexity and adaptability, as well as interactions between human and natural components. Many characteristics of complexity are present in these systems: (a) crop yields show a non-linear response to rainfall, increasing first but decreasing after excessive rain (i.e., non-linearity); (b) farmers have individual characteristics such as wealth, land ownership, risk or loss aversion, and crop rotation preferences (i.e., heterogeneity); (c) shallow groundwater can provide a positive subsidy to crops if rainfall is insufficient, but if groundwater rises above certain depth it may drown roots and kill crops (i.e., thresholds); (d) loss of soil structure associated with lack of crop rotation may take years to cause noticeable yield decreases (i.e., time lags); (e) crop rotation has a positive effect on subsequent crop yields (i.e., legacy effects); and (f) regional-level patterns such as land use, farm structure and land tenure emerge from many individual decisions by farmers (i.e., emergent patterns). Similarly, adaptability is present in the behavior of farmers. For example, (a) changes in farmers' working capital may trigger an increase or reduction of the area they farm; (b) dissatisfaction with experienced outcomes (i.e., profits) may induce changes in subsequent land use choices; and (c) perception of better outcomes achieved by peers may induce imitation of their land use or management practices [9].

Agent-based modeling is well suited to the study of CHANS [14], [15]. This approach involves a set of autonomous decision-making entities (i.e., agents), an environment through which they interact, behaviors that define the agent-environment relationship, and rules that define the sequence of processes in the model [16]. Agent-based models (ABMs) often represent human decision-making, including processes such as social interactions, learning, and adaptation [17], [18]. ABMs also can serve as computational laboratories, as they may exhibit complex behavior patterns that provide valuable insights about the dynamics of the modeled system. There is a vast literature on ABMs of land use change (see [16], [18]) as well as on ABMs of agricultural applications (see [19]–[24]).

The Pampas Model (PM) is an ABM of agricultural production in the Argentine Pampas. Its development was motivated by the need to gain insight into the processes underlying recent structural and land use changes in the Pampas. The PM has many similarities with other agricultural land use models such as FEARLUS [25], AgriPoliS [26], MP-MAS [27], [28], and a model of the Canadian Prairies [24]. FEARLUS and AgriPoliS are the models most similar to the PM. Indeed, they guided the design of many of the processes in the PM [9]. The components of the PM have been described in [9], and validation protocols were discussed in [29].

In this article, we focus on analyzing and describing the PM from a software engineering point of view. As models of complex systems often are designed and implemented by researchers or domain experts with limited software engineering expertise, we seek to present helpful guidelines and good practices for the design and implementation of ABMs. Throughout the paper we provide examples drawn from our experience with the PM.

The paper is organized as follows. First we present a brief overview of the PM, followed by the main aspects of its design (Section II), namely: (a) model classes; (b) activity diagram and data flow; (b) package and folder layout; (d) use of design patterns; and (e) performance optimizations. Second, we discuss software engineering aspects related to the initialization of simulations (Section III); the analysis of results (Section IV); and code validation and verification (Section V). In Section VI we present code metrics estimated for the PM and comparable models. Finally, in Section VII we discuss the main lessons learned from our attempts to develop an extensible, maintainable, and optimized implementation of an ABM designed to study a CAS and CHANS, in the hope that our experience may help others undertaking similar tasks.

II. MODEL DESIGN AND ARCHITECTURE

This section describes the PM from a software engineering point of view. The description is intended to be helpful to software developers as well as researchers with some programming knowledge. For this reason, we aim to document our experiences in a form that is simple enough for domain experts or researchers, while also specific enough for software developers. For the description of the model, we apply the widely used Unified Modeling Language (UML) knowledge modeling approach.

UML is used by software developers to graphically represent a software program or a conceptual model. This allows developers to communicate ideas to each other succinctly and in a standard way [30]. It also helps to carry out the entire modeling process, from initial conception to final coding. Furthermore, UML is independent of particular programming languages and can be used to produce written descriptions that can be progressively updated and refined as increasing levels of detail become available [31].

In the following subsections we discuss topics related to the design of the PM using UML. First, we explain the framework selected to implement our model (Section II-A). Then, we describe the most important entities of the PM, their implementation as classes, and their interrelation using a class diagram (Section II-B). In Section II-C, we describe the flow of information along various model components and how this flow can be represented by an activity diagram. In Section II-D we illustrate the package layout as a means of keeping the code tidy. In Section II-E we provide multiple examples of design patterns we used to create efficient code. Finally, in Section II-F we explain performance optimizations we applied to reduce execution time.

A. MODELING FRAMEWORK

An ABM can be implemented from scratch using any programming language, but doing so often puts an unnecessary burden on researchers and modelers [32]. Instead, a collection of tools exist that can aid researchers in model implementation, allow module reuse in different models, and provide support for commonly needed services [33]. These tools are referred to as modeling toolkits, modeling frameworks, or modeling environments. ABM toolkits provide basic functions for modeling, but the user must integrate them into models. ABM frameworks provide integrated software that the user selectively specializes for their application. ABM environments provide fully integrated editors and debuggers that modelers can use to generate and test their applications. It is beyond the scope of this paper to review the available ABM software. Such a review can be found in [34]. Nevertheless, we explain here our selection of Repast Simphony as our implementation option.

The Recursive Porous Agent Simulation Toolkit (Repast, http://repast.sourceforge.net) is a leading free and open-source toolkit for implementing large-scale ABMs [35]. It has been widely used and offers great flexibility for researchers to develop ABMs [32] with an emphasis on social interactions. We choose Repast Simphony – one of the members of the Repast portfolio – as it supports Java, an object-oriented programming language widely used for the development of models and large enterprise applications.

B. MODEL CLASSES

The PM was implemented in Java, an object-oriented programming language [36]. Object-oriented programming is a paradigm based on the representation of every component of the model using classes. A class is a portion of code that represents an abstract or real entity and contains both attributes (i.e., variables) and methods (i.e., code). Classes define the structure and behavior of an entity, while objects are particular instances of that class.

In the following subsections we give a brief overview of the main components of the PM. These components are modeled as classes. As shown in Figure 1, class diagrams depict the main model classes and interrelations among them.

1) BRIEF MODEL OVERVIEW

This section describes briefly how real world entities in agricultural production systems map onto the implemented classes. A more thorough description of the PM can be found in Bert *et al.* [9] and Bert *et al.* [29].

The PM includes three main types of entities: the environment, farms and farmers. The environment represents the northern part of Buenos Aires Province, the most productive sub-region of the Argentine Pampas. This area encompasses about 1,000,000 hectares (100 km^2) and has a long agricultural history [1].

The environment includes the farms of different sizes defined during initialization. All farms are assumed to have the same soil, namely a *Typic Argiudol*, the most representative soil of the modeled region. All farms experience the same climate, represented by daily weather records from *Pergamino*, a location in the center of the region. Although the environment does not represent real farm boundaries, the model is spatially explicit because there is a topological relation among simulated farms that provides structure to interactions such as social comparisons or imitation.

Finally, the model includes farmers who grow soybeans, maize, or a wheat and short-cycle soybeans double crop on owned or leased land. Each farmer may have different land use allocation strategies and financial (e.g., working capital) characteristics. The Pampas Model's source code and documentation is available for free through the OpenABM model library (http://www.openabm.org) [37].

2) CLASS TYPES IN ABMs: AGENTS, PROTO-AGENTS AND NON-AGENT CLASSES

By definition, any ABM must have agents that are the decision-making components. Agents have sets of rules or behavior patterns that allow them to sense or gather information, process inputs, and then effect changes in the outside environment. Any class that is categorized as an agent should have the following characteristics: (a) be adaptive; (b) have the capability to learn and modify its behavior; (c) be autonomous; and (d) show heterogeneity in at least some attributes, resulting in a population of agents with diverse characteristics [31], [38].

Many ABMs include agents that do not have all the characteristics described in the previous paragraph. This fact, however, does not disqualify a model from being agent-based. Instead, these agents are called 'proto-agents.' Proto-agents are agents that lack one or more of the defining characteristics of agency. Depending on the model, these features can be added if there is a reason.

Finally, there are many other classes that do not qualify either as agents or as proto-agents but are still key components of the system being modeled. In the following subsection we introduce the most important classes in our model and illustrate the classification previously mentioned.

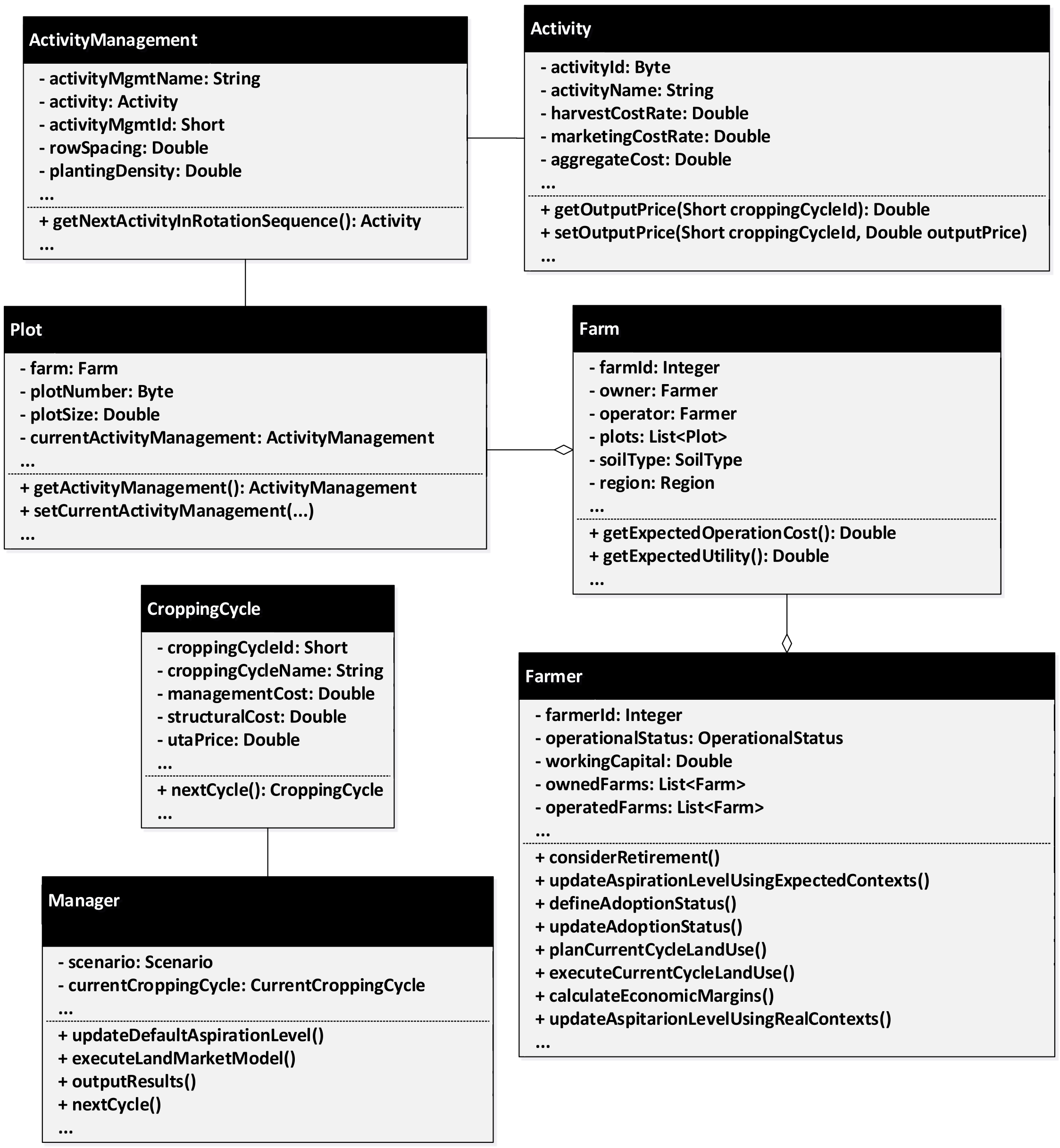
ActivityManagement
- activityMgmtName: String
- activity: Activity
- activityMgmtId: Short
- rowSpacing: Double
- plantingDensity: Double
...
+ getNextActivityInRotationSequence(): Activity
...
Activity
- activityId: Byte
- activityName: String
- harvestCostRate: Double
- marketingCostRate: Double
- aggregateCost: Double
...
+ getOutputPrice(Short croppingCycleId): Double
+ setOutputPrice(Short croppingCycleId, Double outputPrice)
...
Plot
- farm: Farm
- plotNumber: Byte
- plotSize: Double
- currentActivityManagement: ActivityManagement
...
+ getActivityManagement(): ActivityManagement
+ setCurrentActivityManagement(...)
...
Farm
- farmId: Integer
- owner: Farmer
- operator: Farmer
- plots: List<Plot>
- soilType: SoilType
- region: Region
...
+ getExpectedOperationCost(): Double
+ getExpectedUtility(): Double
...
CroppingCycle
- croppingCycleId: Short
- croppingCycleName: String
- managementCost: Double
- structuralCost: Double
- utaPrice: Double
...
+ nextCycle(): CroppingCycle
...
Farmer
- farmerId: Integer
- operationalStatus: OperationalStatus
- workingCapital: Double
- ownedFarms: List<Farm>
- operatedFarms: List<Farm>
...
+ considerRetirement()
+ updateAspirationLevelUsingExpectedContexts()
+ defineAdoptionStatus()
+ updateAdoptionStatus()
+ planCurrentCycleLandUse()
+ executeCurrentCycleLandUse()
+ calculateEconomicMargins()
+ updateAspitarionLevelUsingRealContexts()
...
Manager
- scenario: Scenario
- currentCroppingCycle: CurrentCroppingCycle
...
+ updateDefaultAspirationLevel()
+ executeLandMarketModel()
+ outputResults()
+ nextCycle()
...

FIGURE 1. Class diagrams specify the properties and behaviors of objects. Classes are represented by boxes, each of which contains the name of the class (at the top), and properties and behaviors at the bottom. Lines represent associations between classes.

3) MAIN CLASSES IN THE PM

The PM has many classes. For the sake of space, only the most important ones can be discussed here. A full listing of the model classes is available in Bert *et al.* [37]. We discuss the most relevant classes and identify them as agents, proto-agents and non-agents. In Fig. 1 we provide a standard UML class diagram to describe the interrelationships among the main classes. Class diagrams contain boxes that list the main attributes and behaviors of each class. A relationship between two classes is depicted by an arrow.

The **Farmer** class represents a farmer who may (i.e., active farmer) or may not (i.e., landlord) operate one or more farms. Farmers make several decisions during each cropping cycle. As discussed later, a cropping cycle is the fundamental time unit of our model. The farmers' decisions can include reducing, maintaining, or expanding area farmed; determining the portfolio of crops to grow; or exiting production (e.g., due to lack of capital) or return to an active status after having exited. All these behaviors are implemented as methods of the Farmer class. Moreover, these methods are annotated as *ScheduledMethods*,[1] so they are executed automatically

[1] *ScheduledMethods* are automatically put on Repast Simphony's simulation event calendar.

by Repast given a specific event schedule. As farmers are the decision-makers of our model, this class is an agent.

The **Farm** class represents a production unit operated by a farmer. The farm may be owned or rented by the operator. The identifiers of both the farm's owner and its operator on a given cycle are attributes of this class and are instances of the **Farmer** class. Every farm instance has six equally-sized plots, a reasonable approximation for this part of the Pampas [9]. Each plot is an instance of the **Plot** class and represents an area assigned to one and only one land-use in a given cycle. Also, all farms are part of a geographic space. This space is used to generate a network of spatial neighbors. This network is the basis of agent interactions (e.g., imitation). None of these classes make decisions, so they are classified as non-agents.

In the previous paragraph we stated that each plot of a farm is the area where one and only one land use takes place in a given cycle. We define the **Activity** class that represents a single land use or farming activity. Land uses are divided into two groups: agricultural activities (e.g., field crops such as soybeans, maize, and wheat) and cattle activities (e.g., cow-calf or pasture fattening). Each activity may be carried out using different agronomic managements (e.g., using different genotypes or input amounts). Each unique combination of an activity and a specific agronomic management is represented by an **ActivityManagement** object. Therefore, each plot of a farm is linked to a particular instance of ActivityManagement during a cycle. Neither of these two classes make decisions, so they are classified as non-agents.

CroppingCycle is a main class of the PM that represents the time unit of the model. Each cropping cycle represents a farming cycle beginning on May 1st and ending on April 30th of the following calendar year. All model processes are scheduled to take place within this time unit. When execution steps to the following cropping cycle, many environment variables may be updated as a result of exogenous changes, and thus be read as new inputs. All these updates are scheduled at specific moments and are carried out by a singleton class (i.e., a unique-instance class) called **Manager**. This class is responsible not only for carrying out updates, but also for performing common calculations needed by all agents in order to reduce model execution time. The Manager is a proto-agent [31].

C. REPRESENTING DATA FLOW: ACTIVITY DIAGRAMS

An important aspect of model design which is usually documented in agent-based models is the flow of information between software components. A specific UML diagram, the *activity diagram* [39], is used to document information flows between software components and other systems [31]. The activity diagram allowed us to show the model's data flow to various domain experts in order to ensure conceptual validity [29]. Furthermore, activity diagrams show how a system transitions from state-to-state over time, allowing documentation of parallel actions with the explicit inclusion

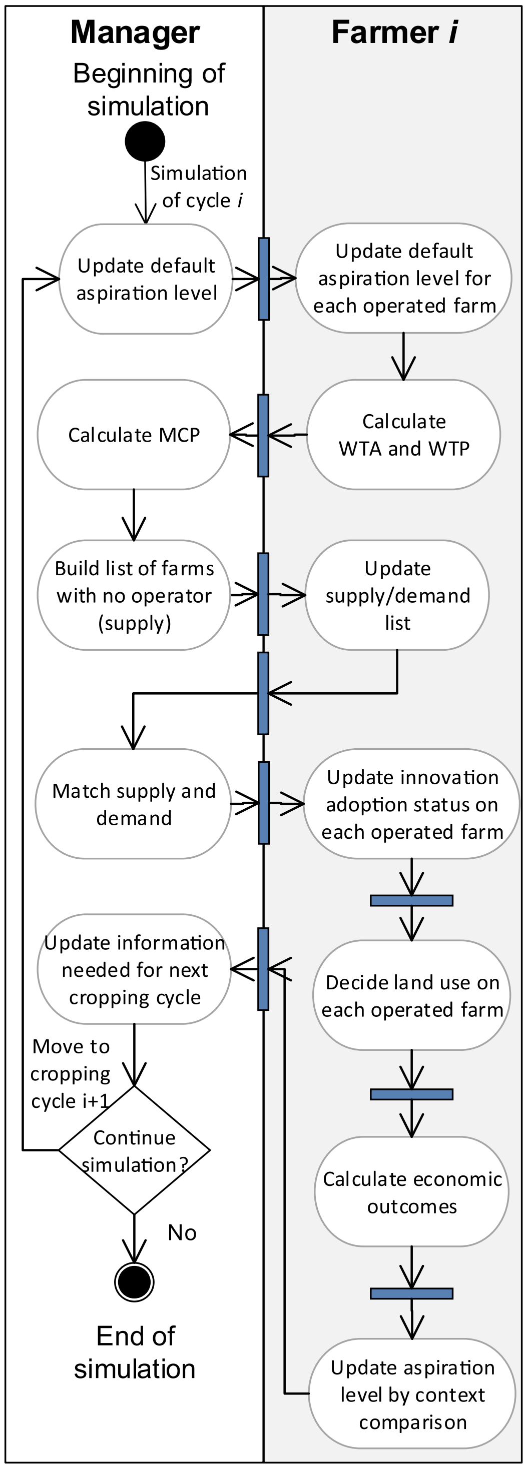
Manager
Farmer i
Beginning of simulation
Simulation of cycle i
Update default aspiration level
Update default aspiration level for each operated farm
Calculate MCP
Calculate WTA and WTP
Build list of farms with no operator (supply)
Update supply/demand list
Match supply and demand
Update innovation adoption status on each operated farm
Update information needed for next cropping cycle
Decide land use on each operated farm
Move to cropping cycle i+1
Continue simulation?
No
Calculate economic outcomes
End of simulation
Update aspiration level by context comparison

FIGURE 2. Activity diagrams are read from top to bottom. The initial state is represented by a filled black circle and the final state is represented by a hollow circle containing a smaller filled black circle. Intermediate states are represented by oval containers that indicate the meaning of the state. Transitions are represented by arrows. 'Swim lanes' (columns) represent the activities within a specific software component. Finally, short bars represent synchronization points for simultaneous activities.

of the flow of time. For this reason, activity diagrams help describe event scheduling within Repast.

The main processes and sub-processes in the PM are shown in an activity diagram provided in Fig. 2. At the beginning of each cropping cycle a farmer adjusts his economic aspirations based on the expected status of context factors (i.e., climate conditions, output prices, and input costs).

In the next step, the ''Update of Cropped Area'' sub-model, farmers decide whether they (a) can maintain the same area as in the previous cycle; (b) can farm additional land; or (c) must release some or all of the previously farmed area. In the current model, the only way to expand cropped area is by renting additional land. This is a reasonable assumption as land sales in the Pampas are rare.

Subsequently, the farmer allocates his land among a realistic set of activities: maize, full-cycle soybeans, and wheat-soybeans double cropping. After land is allocated, the yield of each activity is retrieved from lookup tables pre-calculated using biophysical crop models and climate conditions experienced during that cycle.

Economic returns are either (a) calculated from simulated yields and historical crop prices and input costs (specified as model inputs); or (b) provided as input data. The end result is an updated value of the farmer's Working Capital (WC) at the end of the production cycle.

Economic returns are then assessed in relation to both the farmer's initial aspiration and his peers' performance. This assessment drives an adjustment to the farmer's Aspiration Level (AL). AL is a special value that separates outcomes perceived as successes or failures [40]. The AL is used as an input to decisions in the following cropping cycle.

D. PACKAGE LAYOUT

In every software project, it is good practice to keep the source code and all other related information tidily organized. Any Java application, including the PM, includes many classes. As Java requires each public class to be defined in a separate file, the application ends up with as many files as classes. Maintaining a large set of files without any structure can quickly become unmanageable. To solve this problem, most object-oriented languages, including Java, provide a mechanism for organizing classes into packages. Packages are a way of grouping classes according to their functionality. The packaging mechanism organizes class files into a directory structure based on the package names used [41].

The PM includes a number of sub-models (see Section II-C), each of which is implemented as a collection of classes. The functionality involved in each sub-model defines the first-order layout of packages. Using the same rationale, the grouping of classes can be recursively applied to each package in order to create smaller sub-packages. In theory, there could be as many packages as classes, but this design would be as undesirable as having no packages at all. There is no strict rule about the number of classes that a package may contain, but a number not greater than 30 has been suggested [42].

From the beginning of model implementation we organized classes into separate packages. The package structure evolved throughout the development process, as model functionality was expanded. At the same time, we repeatedly refactored the package structure (i.e., reorganized its content) for the sake of maintainability.

Fig. 3 shows the current package layout of the PM. As our intention is to explain the thinking behind our package arrangement, only the top-level packages are displayed. The first row (orange) contains the root package: *fiuba.lamm.agromodel*. This package is named after the main institution where the PM was developed (*Facultad de Ingeniería de la Universidad de Buenos Aires – FIUBA*), the laboratory where the initial design took place (*Laboratorio de Modelos Matemáticos – LAMM*) and the original code name for the project (*Agromodel*). The root package includes common classes that represent the main reference entities of the model such as Activity, Region, SoilType, etc. In the second row we present the first-level packages (and one important second-level package) which include most of the code for all the sub-models of the PM, namely: *agents* (Farmer and Manager classes), *areaupdate* (area update sub-model; *mcp* also contains the code which implements the market clearing price component used in the area update); *decision* (land-use decision-making sub-model); *diffusion* (innovation diffusion sub-model); and *finance* (economic results calculation sub-model). Finally, in the third row we present other first level packages needed for the model, but indirectly related to the sub-models (e.g., math contains classes to perform mathematic operations such as function evaluations).

E. DESIGN PATTERNS

Design patterns are time-tested solutions to common computer programming problems that have been tried and shown to work on many different situations. The use of design patterns is a good practice in the design and implementation of any software project. Agent-based models are not an exception. Patterns are not programs, libraries, or ready-made templates. They instead offer conceptual strategies for solving problems and a language for communicating solutions [43].

The main advantage of using software design patterns is that they have been shown to improve the reliability of software as well as reduce development time and costs. For these reasons, it is strongly recommended to invest time before implementation to recognize those design patterns that can be used in planned code. Nevertheless, in our case, and also in general, this was not always possible. In some cases, this was because of time pressures. In the majority of cases, though, it was because the conceptual model was still not completely defined. For example, on many occasions we had to test different plausible model mechanisms before selecting the most appropriate.

We classify the design patterns used in our model into two groups: modeling design patterns and general design patterns. In the following subsections we provide examples of design patterns from both groups.

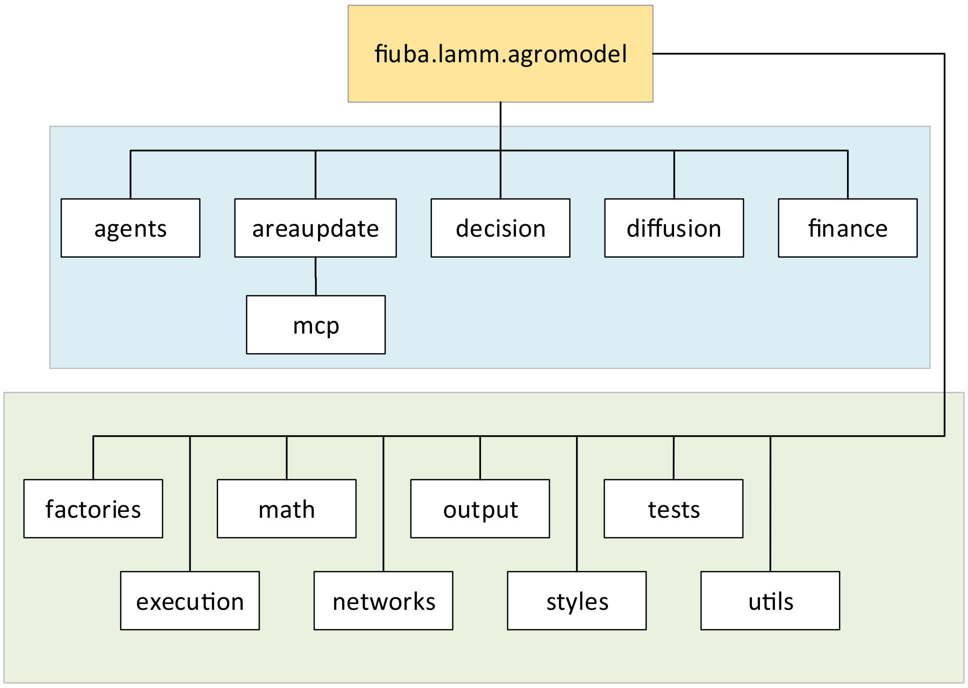

FIGURE 3. **Package layout of the PM. The first row shows the root package. In the second row, we show the first-level (and one important second-level) packages, each of which contains code for all the sub-models in the PM. The third row contains other first-level packages with additional needed code.**

1) MODELING DESIGN PATTERNS

a: AGENT-BASED MODEL PATTERN [43]

As in any complex adaptive system, we need to model the behavior of many interacting individuals. Agent-based modeling addresses this challenge by modeling each decision-maker as an independent component called an agent. Interdependence arises directly from behaviors that act on other individuals and indirectly from changes to shared items in the environment. Each agent has an individual set of attributes and behaviors. These attributes and behaviors may form themes, categories, or classes that can be shared among the agents. Methods such as object-oriented modeling can be used to represent agent commonalities and variations.

b: SCHEDULER PATTERN [43]

Agents have behaviors that need to be executed in a particular sequence. To address this need, Repast provides a Scheduler class. Agents provide information on when their rules must activate to a shared object (which is an instance of this class). The Scheduler merges all of the agents' activity sequences and then invokes rules in the resulting implied order. New activities are merged into the ordered sequence as they are passed to the Scheduler. Times are represented as either integers or real numbers. Priorities can be defined to break ties between events.

c: SCHEDULER SCRAMBLE PATTERN [43]

Besides needing a scheduler, we also need agents to act during the same clock tick without biasing the results. As two or more agents from the PM may attempt to execute behaviors simultaneously during the same clock tick, those competing behaviors are scheduled in a random order at each tick. This scheduling policy is actually implemented in Repast. The scheduling order ultimately depends on a random seed which can be fixed to produce a specific execution sequence.

d: DOUBLE BUFFERING PATTERN

Different agents often simultaneously view and change shared values during the same clock tick. The result of the updates and changes, however, should be independent of the order of event execution. In order to avoid execution order dependencies, we had to use two variables to store each value to be accessed. The first location is always used for reading or viewing, whereas the second one is used for temporary storage of updates during a clock tick. The temporary value is copied to the reading location at the end of each clock tick [31]. In our model, we implemented a diffusion of technological innovations sub-model where adoption is partly determined by the number of neighbors who already have adopted the innovation. Each agent keeps a state to indicate whether he/she is adopting or not an innovation at a given clock tick. When the *decideAdoptionStatus* scheduled method runs, the model needs to inspect the adoption status of the agent's neighbors in order to decide his own adoption status.

2) GENERAL DESIGN PATTERNS

a: STEP-BY-STEP PATTERN

The development of complex models often requires a substantial time. To ensure that the model remains aligned with changing stakeholders' requirements, it is a good practice to develop it incrementally, i.e., in small iterative steps. Each step is either co-developed with stakeholders or is presented to them immediately upon completion. The next development step is then adjusted based on the stakeholders' feedback [43]. To develop the PM, we collaborated closely with Asociación Argentina de Consorcios Regionales de Experimentación Agrícola (AACREA), a non-profit farmer organization in Argentina (http://www.crea.org.ar). During development, we structured the model into separate sub-models, each of which was either accepted by AACREA experts or validated against history data. This methodology allowed us to develop the model incrementally, where each sub-model was verified and corrected. Finding and correction of errors is easier as every component is developed and tested one at a time.

b: SINGLETON PATTERN

Singleton is a creational design pattern [44]. Essentially, it is a class which only allows a single instance of itself to be created, and usually provides simple access to that instance. There are different ways of implementing the singleton pattern. We implemented it by using: (a) a static member that contains the instance of the singleton class; (b) a private constructor that prevents anybody else to instantiate the class; (c) a static public method that provides a global point of access to the singleton object and returns the instance to the calling class. The Manager class (explained in Section II-B3) is an example of this pattern in our model. Use of the Singleton pattern ensures that, at any given time, if two objects create an instance of Manager they will hold the same object. This guarantees the existence of only one Manager instance which is self-updated by means of scheduled methods (i.e., no other model object is allowed to update the manager, it only provides read-only methods).

c: STRATEGY PATTERN

Strategy is a behavioral design pattern [44]. Its purpose is to define a set of algorithms from which the required mechanism or strategy is chosen at run-time. In an ABM, agents often can have multiple ways of carrying out a specific task, but each agent typically has a preferred mechanism according to his/her idiosyncrasy or needs. A good example for this pattern is our class LandUseSelectionMechanism. This class defines a couple of methods related to land use selection, one of which is abstract (search method). This method is implemented in all the subclasses, as each of them defines an algorithm to select the best land use according to the farmer's criteria. The use of strategies allowed us to encapsulate complex mechanisms within a class without needing to define many if-else branches that would have made the code unreadable and harder to maintain.

d: FACTORY PATTERN

This is a creational pattern [44] used to create objects without exposing the instantiation logic to the client. As we will describe in Section III, all the data for a specific scenario run is currently stored into a relational database. For instance, all the farmers' attributes are read from a database. In order to separate the behavior from its creation logic, we need an instance of FarmerFactory which creates all the necessary instances and returns these objects by means of specific finder calls (e.g., *findByOperationalStatus*). The use of this pattern, therefore, allowed us to separate the database querying from the class logic, making the code cleaner. In fact, if we ever change the underlying data support, we would only have to change the factory code, leaving everything else untouched.

F. OPTIMIZATION OF MODEL PERFORMANCE

One common issue faced by model developers is the balance between extensibility and performance. Models that can be extended tend to have clear code that avoid special cases and uses little caching, whereas fast models tend to have many special cases to reduce redundant data access [31]. North and Macal [31] suggest that optimization should be based on performance results from actual model runs, rather than suppositions. They also suggest the use of performance-profiling software to determine the actual causes of identified problems.

We identified several code bottlenecks by measuring the execution time of the various sub-models in the PM. For instance, the PM has a Land-Use sub-model in which, on each cropping cycle, a farmer selects the activity-managements he/she will use in each farm operated by that agent. There are many possible combinations of activity-managements for six-plot farms. Computing farm-wide gross margins for every combination of activity-managements may require a considerable time, especially if the calculation is performed by every farmer on each operated farm.

For this reason we created a proto-agent, namely the Manager. In general, the manager executes tasks for which results need to be available to all agents. One such task, for example, is the calculation of gross-margins per hectare for all possible combinations of activity-managements in a six-plot farm. On each cycle, this information is computed once by the manager and is subsequently accessed by all farmers when the Land-Use sub-model executes.

As a result of all optimizations, we were able to reduce the execution of a 20-year simulation from 40 to five minutes – almost one order of magnitude. However, we stress that this reduction required careful measurement of earlier run-times to identify bottlenecks in the code. In other words, we based our optimization decisions on run results rather than assumptions.

III. MODEL INPUT: INITIALIZATION

As any model, the PM requires a set of initial conditions for execution. These conditions include: (a) a population of farmers and their heterogeneous characteristics; (b) a set of external conditions that drive the simulation on each cropping cycle (e.g., expected and actual climate conditions, expected and actual input costs and output prices); and (c) the spatial layout and other attributes of simulated farms.

A particular combination of initial conditions – which we will subsequently refer to as a 'scenario' – can be provided to the model in various ways. Independently of the chosen approach, the ABM initializer class needs to access one or more data sources supported by Repast, such as (a) local or remote files with a specific format or (b) a relational database. As discussed below, initialization of the PM relies on both approaches.

Most of the initialization information (items *a* and *b* above) is stored in, and retrieved from a relational database. Relational databases provide an easy, efficient and transparent way to store and retrieve information. Furthermore, almost every programming language provides libraries to connect to a relational database.

In any relational database, information is stored into tables, each of which represents an entity of the model. By definition, an entity is an abstract or real object that is part of a given system. Entities have (i) attributes that describe the properties of the objects we are representing, and (ii) relations that define bindings with other entities [45]. Fig. 4 shows an Entity-Relationship (ER) diagram including the main entities in our model.

To initialize the PM with a spatial arrangement of simulated farms, the model initializer class reads a series of geographical information systems shapefiles. Shapefiles are widely used to represent geographic entities.[2] The input shapefile contains (1) polygons with outlines and (2) tables with the attributes of a synthetic set of farms with a size distribution consistent with real-world data.

In Section III-A we describe how both approaches (i.e., a relational database or files with a particular format) can be used to create initialization scenarios. Finally, in Section III-B we explain how initialization scenarios are used to launch a simulation.

A. GENERATION OF INITIALIZATION SCENARIOS

We described above a relational database used to store most initial conditions for the simulation (e.g., farmers, farms, plots' land use, etc.) and time series of exogenous variables (e.g., output prices, production costs, crop yields, etc.) that are updated on each cropping cycle. In this section we focus on the process that populates the relational database with initialization information available to the model.

We chose the MySQL relational database engine (http://www.mysql.com) because it is free, open-source, robust, easy-to-use, and widely used. MySQL provides a very simple way to read files with tab-separated values (TSV) into database tables. In turn, TSV files are very easy to generate from any scripting language. We used TSV files to populate the MySQL database, defining one-to-one correspondence between TSV files and database tables.

[2]Please see www.esri.com/library/whitepapers/pdfs/shapefile.pdf for more information on shapefiles.

Database tables in our model are classified into two groups: common tables and scenario tables. The former include tables that remain unchanged in every simulation. These tables were generated once and used in many simulations. The latter group includes tables that are specific to a simulation scenario. A simulation scenario is characterized by a unique combination of (i) model mechanisms and parameter values; (ii) initial conditions; and (iii) trajectories of exogenous input variables.

All the information associated with a particular scenario was generated by a script that gathers data from different sources and generates the necessary input TSV files. This script was developed using R [46]. The reasons for using R to generate scenario tables are multiple. First, it is free and open-source. Second, it has a very large user base throughout the world and multiple repositories and mailing lists where language-related questions usually are answered quickly and thoroughly. Third, in addition to multiple built-in functions for data analysis, R is a simple and effective programming language that includes conditionals, loops, user-defined functions and input/output capabilities [47]. Many research teams, including our own, involve scientists without SQL skills but with some experience in R programming. For this reason, many data handling functions can be implemented directly in R by researchers, thus reducing the burden on project programmers. Finally, at present R has about 8,200 user-contributed packages that considerably extend the language's core functionality.

In creating the script to generate scenario tables, we have taken full advantage of many existing R packages to save coding and testing time. For example, we used several packages that handle spatial data and give GIS-like capabilities to R. R was used to generate 'synthetic farms' for the model's environment because actual cadasters are not publically available. First, we generated distributions of farm numbers and sizes consistent with census data. Second, we used various spatial packages (see spatial task view in the Comprehensive R Archive Network, CRAN, cran.r-project.org/web/views/Spatial.html) to generate shapefiles with synthetic farm boundaries, and to build tables of spatial neighbors.

The R script to produce scenario tables is organized into a set of separate code blocks, each of which generates a single database table. The separation of code into blocks facilitates documentation and maintenance. Ultimately, users can organize various pieces of R code into separate functions and create their own packages (even if they are not distributed publically). Each block of code ends by calling a function that writes out each table to a TSV file. Although we used only TSVs, R also can write more structured output, such as YAML, JSON or XML files.

B. ACCESSING INITIALIZATION SCENARIOS FROM THE MODEL

After having populated the relational database and created the necessary shapefiles, the next step involves retrieving

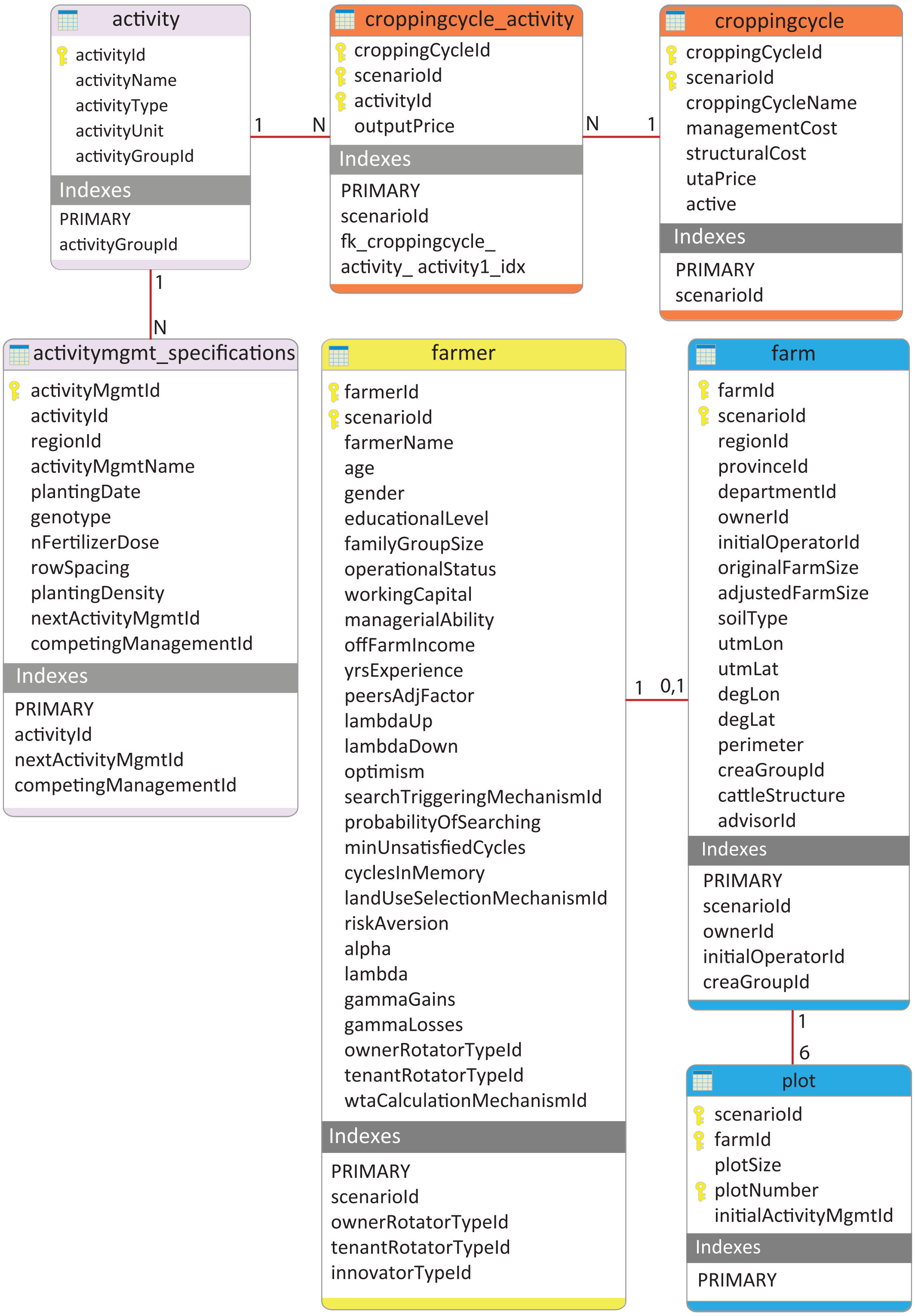

FIGURE 4. **Simplified ER diagram containing the main entities of the PM relational database. Entities are represented by boxes, while relations are represented by lines. A complete ER diagram is available from the authors. The key icons on the left of a table indicate fields that make up that table's primary key.**

that information during the model initialization stage. The model initialization takes place at the very beginning of a simulation. At that moment, all agents and other classes are instantiated. This instantiation is carried out by means of factory classes, explained in Section II.E. All the factories in our model implement a set of specific 'finders' which, in turn, return a collection of objects according to a search criterion. Every factory retrieves the appropriate information using

SQL (Structured Query Language), the standard language for interacting with relational databases [45].

After a table is read by the corresponding factory, the entity represented by the table is mapped to a Java class, with the exception of a couple of configuration tables. This mapping is straightforward, as the definition of an entity is similar to the definition of the corresponding class. For instance, each row of the farmer table in the relational database is mapped onto a different instance of the Farmer class.

In many cases, factories use an internal cache to store the content of a particular table. This is the case, for example, for reference tables listing data such as soil types, activity-managements and crop inputs required (e.g., fertilizer, biocides). These tables need to be accessed many times during the simulation. As the reference tables generally involve a small number of rows, it is more efficient to cache the whole table in memory than accessing the database whenever the information is needed.

IV. OUTPUT OF MODEL RESULTS

After a cropping cycle, a set of files is updated (or created, if on the first cycle) with a snapshot of the most important state variables. These files contain information that reflect the state of each farm and farmer, and other relevant information (e.g., the land rental price simulated endogenously) at the end of each cycle. These files are stored in TSV format for the sake of simplicity and convenience.

Repast provides a very simple way to create these output files, using what are known as *gatherers* and *outputters*. More information about these classes can be found in the Repast documentation (http://repast.sourceforge.net/docs.php). These classes provide mechanisms for (a) gathering needed attributes for all agents (gatherers) and, (b) outputting them into TSV files (outputters). In this case, however, we also need to output information about non-agent classes. For this reason, we had to implement our own method for information logging.

After the end of a simulation, output files are processed by an R script that carries out two main types of tasks: (a) consistency checks explained in Section V; and (b) generation of charts and tables to be analyzed later by domain experts. More detailed information on the results produced by our model can be found in [9].

The output process is the counterpart of the initialization stage. During initialization, an R script generates the model input as a set of TSV and shapefiles. This information – specified by domain experts – defines a unique simulation scenario. Similarly, at the end of a simulation, an output script provides domain experts with tables and charts for subsequent analysis and interpretation. The output script also was implemented in R for the same reasons listed in Section III.

V. VERIFICATION AND VALIDATION

A. CONSISTENCY CHECKS

Consistency is the lack of contradiction in data. Consistency of data sets can be measured by cross-checking fully or partly redundant fields [31]. The existence of inconsistent values in intermediate or final model results may reveal programming errors. For this reason, consistency checks are useful tools to verify agent-based models.

Before final results are analyzed by domain experts, an R script performs two main groups of checks to ensure that no obvious inconsistencies among results are present. The first group of checks involves numerical of logical comparison of interrelated variables within a cropping cycle. Examples of this group of checks include:

1) The area cropped by all farmers should be equal to the total simulated area. This comes from a model constraint that farms are valuable so every farm must be cropped on every cycle;
2) The area cropped by a farmer should be consistent with the farmer's operational status (i.e., whether a farmer is actively cropping or whether a farmer owns and/or rents land). Specific checks are as follows:
    a. Active farmers should operate an area greater than zero hectares;
    b. Non-active farmers (e.g., landlords) should operate an area equal to zero hectares;
    c. Farmers who crop only their owned land should have an operational status of 'Owner-Only;' and
    d. Farmers who crop both owned and rented land should show an operational status of 'Owner-Tenant.'

The second group of checks involves examination of the temporal trajectories of specific variables throughout the simulation. These checks enforce various model constraints, for example:

1) The total area farmed should be constant over time (model constraint: no farms are created or removed).
2) The number of active plus inactive farmers should remain constant over time (model constraint: no farmers are created; bankrupt farmers are replaced by inactive farmers created during initialization and held in reserve).

Consistency checks are very useful when making changes to the model, as they ensure that the assumptions and constraints on which these tests are based remain valid.

B. CODE WALKTHROUGHS

Code walkthroughs involve presenting software code to independent programmers and domain experts, and then manually stepping through example executions of the code. Code walkthroughs allow domain experts to assess whether the software follows the conceptual model design. At the same time, walkthroughs involving independent programmers allows an evaluation of the implemented code from the software engineering point of view. The audience is expected to criticize the code and the execution sequences for correctness. At no time during the presentation should the code be modified or executed. Furthermore, no suggestions should be made by auditing developers as this would make them contributing authors of the code, thus creating a conflict of interest [48].

We carried out code walkthroughs immediately after implementation of each model component. We stepped through the execution of all sub-models within the PM with two different domain experts. This practice allowed us to detect and correct inconsistencies between the implemented code and the conceptual model design. Furthermore, during the walkthroughs domain experts found flaws in the original conceptual model that was being implemented. Even though code walkthroughs are time-consuming, the effort invested in this approach is more than compensated by the time saved in debugging errors at runtime.

C. CONCEPTUAL MODEL VALIDATION

A thorough validation of an agent-based model can involve a range of approaches [49]–[51]. The PM was thoroughly validated as discussed in [30]. Here, we discuss only the conceptual validation process, as it took place in parallel with the design and implementation of the model. Conceptual validation – also referred to as ''micro-face validation'' by Rand and Rust [52] – is the process of ensuring that the mechanisms and properties of a model correspond to real-world mechanisms and properties.

Face validity checks rely on the knowledge and experience of experts familiar with the source problem. A conceptual design is presented to experts or stakeholders who are asked to assess if the design is reasonably compatible with their knowledge and experience. Domain experts also can examine preliminary results to compare initial output patterns with their own perceptions of how modeled events should have developed and progressed [53].

The conceptual validation stage involved a set of strategies aimed at ensuring the inclusion and realistic characterization of relevant processes and components in the model. Interactions with stakeholders and domain experts supported several strategies in this stage. The design of each model component started with a review of the relevant literature and a simple initial design. This initial design was subsequently discussed in small workshops with three to five AACREA domain experts.

One lesson we learned is that the validation of components and processes in cooperation with experts and other stakeholders must start from the very beginning of model development. Moreover, our experience clearly showed the valuable benefits of careful conceptual validation. The best evidence of such benefits was the reasonableness of early PM results. By including needed structural and behavioral components as well as realistic characterizations of important initial conditions and parameters, even initial and highly simplified simulations produced very sensible outcomes [9].

D. PARAMETER SPACE EXPLORATION

Most simulations of CHANS include random elements in both initial conditions and mechanisms. In addition, such models may include many uncertain parameters [54]. Consequently, a single run of the simulation model with a unique set of parameters will not be able to represent the real underlying uncertainty, thus model results can be misleading [54]. One step towards deriving robust results and quantifying uncertainty is to explore the model's output space in response to input parameter variations [55].

Parameter space exploration (PSE) provides additional insight about the behavior of a model by examining the output space in response to a range of parameter settings [56]. PSE can serve a range of purposes, including (a) parameter calibration and optimization; (b) uncertainty analysis; (c) sensitivity analysis; and (d) a search for specific model features such as regime shifts and tipping points [56].

There are many techniques for carrying out PSE. Human-guided parameter exploration conducted by the modeler, a domain-expert or a stakeholder allows testing of relevant hypotheses while minimizing the number of regions searched. Unfortunately, this approach is vulnerable to human bias and fatigue. In standardized sampling approaches (e.g., random, quasi-random, gridded/factorial, Latin hypercube, and sphere-packing), parameters are chosen systematically to ensure their having certain statistical or structural properties. Finally, PSE can be carried out through more sophisticated techniques such as meta-heuristic searching, optimization algorithms, genetic algorithms, machine learning, and query-based model exploration [56].

Repast provides a built-in solution to perform PSE, known as parameter sweeps (PSWs). PSWs are a quick way to conduct a series of simulations by running many instances of a model, each one with a given combination of input parameters [31]. PSWs can be implemented sequentially on one computing node or in parallel using multiple nodes. In order to carry out PSWs, the analyst defines an XML file describing the input parameter space and then uses the Repast batch run capability to iterate through all unique parameter combinations. In the following paragraphs, we focus on one specific type of PSE – sensitivity analysis (SA) – that we carried out to explore the PM solution space and identify key model parameters.

As part of SA, input parameters are systematically varied and the model's response to parameter uncertainty is assessed. SA allows exploration of relationships and mechanisms not yet well understood and highlights the most important processes [54]. SA also promotes the understanding of models, allowing their use both for theory development and applications [57]. Most researchers use SA to identify those parameters for which small variations induce high impacts in the model's output (i.e., to answer the question 'which inputs are most responsible for the variability of outputs?') [56], [58]. Many methodologies have been developed to perform sensitivity analysis. Here, two major types of SA will be discussed, namely local sensitivity analysis (LSA) and global sensitivity analysis (GSA) [59].

LSA involves the examination of model output by varying input parameters one at a time (OAT) while holding all others at nominal values [59], [60].

Ligmann-Zielinska and Jankowski [61] discuss this approach and its advantages. However, this method ignores the potential interactions between input parameters, preventing analysts from capturing their impacts on model output [54], [60]. In contrast, GSA involves varying all input parameters at the same time, thus allowing identification of non-linear interactions among parameters [54], [60]. As a drawback, GSA is computationally expensive and becomes particularly limiting in models with tens or hundreds of parameters such as the PM [62]. Lee *et al.* [56] describe many techniques to carry out GSA. In the following paragraphs we describe our use of both LSA and GSA to perform SA of the PM.

We used LSA to identify the key parameters in an important sub-model of the PM: the land rental market (LARMA) component; see details in [63]. Here, we illustrate the use of LSA to explore the impacts of three specific LARMA parameters. In each case, we varied one parameter at a time and assessed the impacts on time series of simulated land rental prices (LRPs). The parameters in this discussion were chosen to illustrate a range of influences on simulation outcomes (i.e., no impact, small, and large impacts).

The first parameter explored was the number of previous cropping cycles (NCYCLES) for which landlords assess economic outcomes to decide whether to rent out their land or, instead, farm it themselves. NCYCLES is used to calculate the rental fee that landlords are willing to accept (WTA) for their land. Following stakeholders' advice, we used a nominal value of three years for NCYCLES. We also explored the lower bound for NCYCLES namely one year. Although there is no theoretical upper bound, we considered a maximum value of six years to reflect the farmers' limited memories. The three PM outcomes, one for each NCYCLES value, showed that this parameter modified WTA values, which are an intermediate result, but it had no influence on LRPs, the main variable of interest [63]. We concluded that this parameter needed no further exploration and used the nominal value of three years in subsequent simulations.

The second parameter explored was the farmers' annual desired profitability rate (DPR). DPR represents the annual return on working capital that a farmer desires to achieve if he is to continue farming his land (e.g., inputs, land rental if applicable). DPR is used to calculate the amount that a potential tenant is willing to pay to rent in a new farm (this amount is referred to as 'WTP'). Following AACREA experts, we assumed a nominal DPR of 10%. Given the macroeconomic context at the time of these simulations, we explored alternative DPR values of 5% and 15%. Simulation outcomes showed a moderate impact of DPR values on LRPs. Higher DPRs lead to lower WTPs and consequently, lower LRPs. If tenant farmers desire higher profitability, they need a lower LRP, all else being equal. The lower DPR value (5%) slightly increased LRPs and brought simulated results closer to the real-world outcomes [63, Fig. 9(a)]. Nevertheless, AACREA experts considered a DPR of 5% too low for a risky activity such as agriculture. Because of its limited impact on the main quantity of interest (LRP), we used the nominal value of 10% in subsequent simulations.

Finally, the third parameter subjected to SA was LAMBDA (λ), a parameter associated with the dynamic adaptation of farmers' desired income on a cropping cycle. In making risky choices, decision makers often focus on reaching a special outcome – an aspiration level or AL. Outcomes above and below AL are respectively coded as successes and failures [40]. In its simplest form, AL for the following decision cycle (AL_{t+1}) is calculated as a weighted average of current AL (AL_t) and achieved economic performance on cycle t (EP_t). AL is adjusted upward when achievements equal or surpass aspirations (i.e., $EP_t \geq AL_t$), and downward otherwise [64]. This adjustment is formalized as $AL_{t+1} = \lambda\, AL_t + (1-\lambda)\, EP_t$, where $\lambda \in [0, 1]$ describes an individual's "resistance" or "inertia" to adjusting AL [65]. The baseline scenario assumed nominal LAMBDA values of 0.55 and 0.45 for upward and downward adjustments respectively [63] to reflect the fact that people "get used" to higher payoffs more rapidly than to lower ones, thus showing greater resistance to downward changes [66]. Alternative lower and upper values considered were zero and one for both upward and downward adjustments. LSA results show that extreme lambda values have a significant impact on the dynamics of LRPs [63, Fig. 9(b)]. The most significant effects appeared when $\lambda = 1$. LAMBDA has a large impact on LRP calculation, therefore stressing the importance of dynamic AL adjustment for realistic LRP calculation. Thus, a more thorough exploration of this parameter seems necessary in future simulations.

The PM is a complex model with about 100 parameters and mechanisms. With this kind of model, it would have been extremely difficult to perform GSA to assess the behavior of all parameters. Such an approach would have required very large computing power, and most importantly, even larger human resources to analyze the outcomes of numerous parameter combinations. Consequently, SA of most parameters was carried out using the LSA approach (see previous examples). Nevertheless, we used GSA in two experiments designed to explore the behavior of a set of parameters related to land-use and land-tenure processes. These experiments are described in Bert *et al.* [29] and rely on the experimental design approach shown in Happe [55]. This approach involves selection of three to seven key parameters related to the model behavior we want to explore, leaving the others constant at their nominal values. Each selected parameter is tested with two or three different values. Consequently, the resulting number of combinations is small enough to be analyzed in detail by domain experts.

VI. CODE METRICS

Analysis of implemented software plays a critical role in improving the quality of the code one is developing. There are multiple definitions of software quality. Please see [67]–[70] for details. Despite this ambiguity, we can generally define software quality as *"the degree to which a system,* *component, or process meets specified requirements"* (IEEE Standard Glossary of Software Engineering Terminology, https://standards.ieee.org/findstds/standard/610.12-1990.html). Quality requirements can be assessed from the users' point of view (i.e., externally) or from the software architecture perspective (i.e., internally) [70]. We will focus on the latter point of view, as we are mainly interested in evaluating the design of our model.

Improvement of software quality requires developers to understand exactly what is happening in the code and why [71]. To this effect, many standard metrics exist that allow measurement, evaluation, control, and improvement of software products and processes [72]. There is a vast literature on software metrics (see [71]–[78]). Nevertheless, there is very little guidance in the literature on software metric thresholds [72] to identify appropriate implementations. For this reason, we will focus on a few important metrics for which thresholds of acceptability have been defined.

TABLE 1. Software quality metrics evaluated for the PM and three other comparable models.

Metric	PM	LUXE	GUF	SAT
AC	12.44	7.55	16.75	16.40
EC	4.37	3.64	13.00	12.20
CC	1.95	2.19	2.74	2.67
LCOM	0.20	0.31	0.33	0.34
LOC	9,800	7,431	5,431	6,471
CLOC	1,079	1,818	1,230	1,481
DC	0.11	0.24	0.24	0.23

In the next paragraphs we calculate seven software metrics for the PM and three other similar ABM as shown in Table 1. These models have been selected for (a) having been implemented using Repast, (b) having a conceptual similarity to the PM, and (c) their source code being openly available. Our use of other models aims to put metrics for the PM in the context of similar ABMs. We do not intend to directly compare the various implementations or claim our implementation necessarily has higher quality. The models we selected are: (1) "Land Use in an eXurban Environment" (LUXE, www.openabm.org/model/3942/version/3/view), (2) "A model of groundwater usage by farmers in the Upper Guadiana, Spain" (GUF, www.openabm.org/model/2549/version/1/view), and (3) "Implementation of 'satisficing' as a model for farmers' decision-making in an agent-based model of groundwater over-exploitation" (SAT, www.openabm.org/model/3796/version/1/view).

A. AFFERENT COUPLING (AC)

This metric shows how many classes are associated with a class [78]. Classes with high AC values play a critical role, as errors in them or modifications to them may have impacts on other classes [72]. AC values in the range [1-20] are labeled as "regular" by [72] (see their Table 4). The value of 12.44 for the PM suggests that the software design could be improved, although such value does not necessarily represent a problem [72]. We note that [72] does not explicitly list AC thresholds for ABMs, thus we picked the category representing the highest standard. The other three models have AC values similar to the PM's.

B. EFFERENT COUPLING (EC)

This is a measure of the number of classes on which a given class depends [78]. Just as with AC, errors or modifications on classes with high EC values may have large impacts on the system. The acceptance threshold for this metric is usually the same as for AC. The PM remains in the "regular" range for the EC, although the value is considerably lower than for the AC. Also, as for the AC, the other models have EC values comparable to the PM.

C. CYCLOMATIC COMPLEXITY (CC)

This metric of complexity assigns a numerical score to each method based on the number of different branches through the method's source code [79]. CC quantifies the potential difficulties in testing, maintaining, or troubleshooting a piece of code. Code with high CC values are often challenging to understand and test. A CC acceptability threshold, however, remains controversial. The original threshold of 10 proposed by [79] has considerable supporting evidence, but limits as high as 15 have been used successfully as well. In any case, the CC value for the PM is well below both limits and is the smallest of all models.

D. LACK OF COHESION OF METHODS (LCOM)

This metric is a measure of the correlation between methods and local instance variables of a class [77]. High cohesion suggests solid class organization. Low cohesion suggests the need for refactoring. LCOM goes from zero to one, with zero being the best value. A high LCOM value is generally tied to a poorly cohesive class. As shown in Table I, the LCOM value for the PM is much closer to zero than to one, suggesting good cohesion. Values for the other models are comparable.

E. LINES OF CODE (LOC)

LOC is the number of code lines. LOC may be correlated with development and maintenance costs.

F. COMMENT LINES OF CODE (CLOC)

CLOC is the number of lines containing either comments or commented-out code. CLOC may be correlated with documentation levels and therefore code readability.

G. DENSITY OF COMMENTS (DC)

DC is CLOC divided by LOC. Higher DC values suggest that the code has more copious comments. A DC value of at least 0.2 is generally acceptable. The PM has a DC value of 0.11, suggesting that we should have invested more effort in documenting the code to meet the accepted standard. Nevertheless, we made a point to use clear and descriptive names for methods and attributes. This is another good practice in terms of maintainability, although there is no metric for it.

The previously described metrics were calculated using two different open-source tools, as no single tool computes all of them. We used Metrics 1.3.6 (metrics.sourceforge.net) to calculate AC, EC, LCOM, and CC. We used Sonar Qube 5.3 (www.sonarqube.org) to calculate LOC, CLOC, and DC.

VII. LESSONS LEARNED

The purpose of this paper was to present our experiences with the design and implementation of ABMs using the PM as a concrete example. In the following paragraphs, we share the main lessons learned, in the hope that they will be helpful for any researcher modeling a CAS.

A. SEARCH FOR AN APPROPRIATE MODELING FRAMEWORK

When it comes to implementing a CAS, we have found agent-based modeling to be a useful approach. Although an ABM can be implemented from scratch in any programming language, this choice involves unnecessary burden for researchers and modelers. Selecting an appropriate ABM toolkit is much more sensible.

B. DESIGN AND DOCUMENT THE SOFTWARE COMPONENTS USING SOFTWARE ENGINEERING TOOLS

Take time to identify the entities of the system (e.g., classes), their interrelations and behaviors, which in turn, can be documented using class diagrams. Model data flow should also be designed beforehand and documented using activity diagrams. Some model entities will need to be initialized at the beginning of simulations. Initialized entities can be persisted into a relational database that can be described with an entity-relationship diagram. All these documents will be of great help to carry out conceptual validation of the model with domain experts and stakeholders.

C. USE GOOD CODING PRACTICES

Follow the old adage, "try not to reinvent the wheel." Design patterns are time-tested solutions to common computer programming problems that have been tried and shown to work. For this reason, we strongly recommend that time be invested to recognize those design patterns that can be used in the code. The code also must be kept tidy in order to guarantee its maintainability. Also, having a large set of files without any structure can quickly become unmanageable. To avoid this problem, classes should be grouped into packages according to their functionality.

D. WRITE SCRIPTS TO GENERATE SCENARIOS AND ANALYZE RESULTS

Running the model is a task that might be executed hundreds or thousands of times. By using and saving scripts to generate each scenario or experiment, we reduce the probability of errors and enhance subsequent reproducibility [80]. The analysis of results is a task that should be carried out by domain experts or stakeholders. The development of scripts to generate exploratory tables and charts will aid researchers, allowing them to focus on rapid examination of model results, and not on development of basic visualizations.

E. CARRY OUT VERIFICATION AND VALIDATION (V&V) THROUGHOUT THE ENTIRE IMPLEMENTATION PROCESS

V&V are essential steps in the implementation of any model. Carrying out consistency checks, code walkthroughs, and conceptual validation from the early stages of implementation will help to identify sooner conceptual mistakes and programming bugs. Such practices will, in turn, reduce the time needed for subsequent debugging. During later stages of the implementation, parameter space exploration also will be needed to perform sensitivity analysis and calibration of the model, and to understand robustness of results to changes in model parameters or input data.

REFERENCES

[1] P. A. Calviño and J. P. Monzón, "Farming systems of Argentina: Yield constraints and risk management," in *Crop Physiology: Applications for Genetic Improvement and Agronomy*, V. O. Sadras and D. Calderini, Eds., 1st ed. San Diego, CA, USA: Elsevier, 2009, pp. 55–70.

[2] R. D. Schnepf, E. Dohlman, and C. Bolling, "Agriculture in Brazil and Argentina: Developments and prospects for major field crops," USDA, Washington, DC, USA, Agricult. Trade Rep. WRS-01-3, 2001.

[3] E. H. Berbery, M. E. Doyle, and V. Barros, "Tendencias regionales en la precipitación," in *El Cambio Climático en la Cuenca del Plata*, V. Barros, R. Clarke, P. Silva Días, Eds., 1st ed. Buenos Aires, Argentina: CONICET, 2006, pp. 67–79.

[4] G. O. Magrin, M. I. Travasso, and G. R. Rodríguez, "Changes in climate and crop production during the 20th century in Argentina," *Climatic Change*, vol. 72, pp. 229–249, Sep. 2005.

[5] P. Delvenne, F. Vasen, and A. M. Vara, "The 'soy-ization' of Argentina: The dynamics of the 'globalized' privatization regime in a peripheral context," *Technol. Soc.*, vol. 35, pp. 153–162, May 2013.

[6] D. Manuel-Navarrete and G. C. Gallopín, "Feeding the world sustainably: Knowledge governance and sustainable agriculture in the Argentine Pampas," *Environ. Develop. Sustain.*, vol. 14, no. 3, pp. 321–333, 2011.

[7] M. Qaim and G. Traxler, "Roundup ready soybeans in Argentina: Farm level and aggregate welfare effects," *Agricult. Econ.*, vol. 32, pp. 73–86, Jan. 2005.

[8] P. Lamers, K. McCormick, and J. A. Hilbert, "The emerging liquid biofuel market in Argentina: Implications for domestic demand and international trade," *Energy Policy*, vol. 36, pp. 1479–1490, Apr. 2008.

[9] F. E. Bert *et al.*, "An agent based model to simulate structural and land use changes in agricultural systems of the Argentine Pampas," *Ecol. Model.*, vol. 222, pp. 3486–3499, Oct. 2011.

[10] J. Liu *et al.*, "Complexity of coupled human and natural systems," *Science*, vol. 317, no. 5844, pp. 1513–1516, 2007.

[11] J. Liu *et al.*, "Coupled human and natural systems," *AMBIO, J. Human Environ.*, vol. 36, no. 8, pp. 639–649, 2007.

[12] M. J. North *et al.*, "Complex adaptive systems modeling with Repast Simphony," *Complex Adapt. Syst. Model.*, vol. 1, p. 3, Mar. 2013.

[13] C. Rammel, S. Stagl, and H. Wilfing, "Managing complex adaptive systems—A co-evolutionary perspective on natural resource management," *Ecol. Econ.*, vol. 63, pp. 9–21, Jun. 2007.

[14] M. Hare and P. Deadman, "Further towards a taxonomy of agent-based simulation models in environmental management," *Math. Comput. Simulat.*, vol. 64, pp. 25–40, Jan. 2004.

[15] M. D. A. Rounsevell *et al.*, "Challenges for land system science," *Land Use Policy*, vol. 29, pp. 899–910, Oct. 2012.

[16] D. C. Parker, S. M. Manson, M. A. Janssen, M. J. Hoffmann, and P. Deadman, "Multi-agent systems for the simulation of land-use and land-cover change: A review," *Ann. Assoc. Amer. Geographers*, vol. 94, pp. 314–337, Jun. 2003.

[17] P. Schreinemachers and T. Berger, "Land-use decisions in developing countries and their representation in multi-agent systems," *J. Land Use Sci.*, vol. 1, pp. 29–44, Jun. 2006.

[18] R. B. Matthews, N. G. Gilbert, A. Roach, J. G. Polhill, and N. Gotts, ''Agent-based land-use models: A review of applications,'' *Landscape Ecol.*, vol. 22, no. 10, pp. 1447–1459, 2007.

[19] T. Berger, ''Agent-based spatial models applied to agriculture: A simulation tool for technology diffusion, resource use changes and policy analysis,'' *Agricult. Econ.*, vol. 25, pp. 245–260, Sep. 2001.

[20] T. Berger, P. Schreinemachers, and J. Woelcke, ''Multi-agent simulation for the targeting of development policies in less-favored areas,'' *Agricult. Syst.*, vol. 88, pp. 28–43, Apr. 2006.

[21] K. Happe, A. Balmann, K. Kellermann, and C. Sahrbacher, ''Does structure matter? The impact of switching the agricultural policy regime on farm structures,'' *J. Econ. Behavior Org.*, vol. 67, pp. 431–444, Aug. 2008.

[22] K. Happe, H. Schnicke, C. Sahrbacher, and K. Kellermann, ''Will they stay or will they go? Simulating the dynamics of single-holder farms in a dualistic farm structure in slovakia,'' *Can. J. Agricult. Econ.*, vol. 57, pp. 497–511, Dec. 2009.

[23] J. Nolan, D. Parker, G. C. van Kooten, and T. Berger, ''An overview of computational modeling in agricultural and resource economics,'' *Can. J. Agricult. Econ.*, vol. 57, pp. 417–429, Dec. 2009.

[24] T. Freeman, J. Nolan, and R. Schoney, ''An agent-based simulation model of structural change in Canadian prairie agriculture, 1960–2000,'' *Can. J. Agricult. Econ.*, vol. 57, pp. 537–554, Dec. 2009.

[25] J. G. Polhill, L.-A. Sutherland, and N. M. Gotts, ''Using qualitative evidence to enhance an agent-based modelling system for studying land use change,'' *J. Artif. Soc. Soc. Simul.*, vol. 13, no. 2, p. 10, 2010.

[26] K. Happe, A. Balmann, and K. Kellermann, ''The agricultural policy simulator (AgriPoliS)—An agent-based model to study structural change in agriculture (version 1.0),'' in *Proc. Leibniz Inst. Agricult. Develop. Central Eastern Eur.*, Halle, Germany, 2004.

[27] S. Juhola and L. Westerhoff, ''Challenges of adaptation to climate change across multiple scales: A case study of network governance in two European countries,'' *Environ. Sci. Policy*, vol. 14, pp. 239–247, May 2011.

[28] P. Schreinemachers and T. Berger, ''An agent-based simulation model of human–environment interactions in agricultural systems,'' *Environ. Model. Softw.*, vol. 26, pp. 845–859, Jul. 2011.

[29] F. E. Bert, S. L. Rovere, C. M. Macal, M. J. North, and G. P. Podestá, ''Lessons from a comprehensive validation of an agent based-model: The experience of the Pampas model of Argentinean agricultural systems,'' *Ecol. Model.*, vol. 273, pp. 284–298, Feb. 2014.

[30] A. Collins, M. Petty, D. Vernon-Bido, and S. Sherfey, ''A call to arms: Standards for agent-based modeling and simulation,'' *J. Artif. Soc. Soc. Simul.*, vol. 18, no. 3, p. 12, 2015.

[31] M. J. North and C. M. Macal, *Managing Business Complexity: Discovering Strategic Solutions With Agent-Based Modeling and Simulation*, 1st ed. New York, NY, USA: Oxford Univ. Press, 2007.

[32] D. Murray-Rust, D. T. Robinson, E. Guillem, E. Karali, and M. Rounsevell, ''An open framework for agent based modelling of agricultural land use change,'' *Environ. Model. Softw.*, vol. 61, pp. 19–38, Nov. 2014.

[33] F. K. van Evert, D. Holzworth, R. M. Muetzelfeldt, A. E. Rizzoli, and F. Villa, ''Convergence in integrated modeling frameworks,'' in *Proc. MODSIM Int. Congr. Model. Simul.*, Melbourne, Australia, 2005, pp. 745–750.

[34] B. Heath, R. Hill, and F. Ciarallo, ''A survey of agent-based modeling practices (January 1998 to July 2008),'' *J. Artif. Soc. Soc. Simul.*, vol. 12, no. 4, p. 9, 2009.

[35] M. J. North, N. T. Collier, and J. R. Vos, ''Experiences creating three implementations of the repast agent modeling toolkit,'' *ACM Trans. Model. Comput. Simul.*, vol. 16, no. 1, pp. 1–25, 2006.

[36] B. Stroustrup, ''What is object-oriented programming?'' *IEEE Softw.*, vol. 5, no. 3, pp. 10–20, May 1988.

[37] F. E. Bert, G. P. Podestá, S. Rovere, M. North, and C. Macal. (2015). The Pampas model: An agent-based model of agricultural systems in the argentinean pampas. OpenABM. [Online]. Available: https://www.openabm.org/model/3872/version/1/view

[38] N. Gilbert, *Agent-Based Models*. Los Angeles, CA, USA: Sage, 2008.

[39] H. Bersini, ''UML for ABM,'' *J. Artif. Soc. Soc. Simul.*, vol. 15, no. 1, p. 9, 2012.

[40] E. Diecidue and J. van de Ven, ''Aspiration level, probability of success and failure, and expected utility,'' *Int. Econ. Rev.*, vol. 49, pp. 683–700, May 2008.

[41] T. R. Fisher, *Java Phrasebook*, 1st ed. New York, NY, USA: Pearson Education, 2006.

[42] M. Lippert and S. Roock, *Refactoring in Large Software Projects: Performing Complex Restructurings Successfully*, 1st ed. Chichester, U.K.: Wiley, 2006.

[43] M. J. North and C. M. Macal, ''Product and process patterns for agent-based modelling and simulation,'' *J. Simul.*, vol. 8, no. 1, pp. 25–36, 2014.

[44] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. New York, NY, USA: Pearson Education, 1994.

[45] T. M. Connolly and C. E. Begg, *Sistemas de Bases de Datos*, 4th ed. Madrid, Spain: Pearson Educación S.A., 2005.

[46] R Development Core Team. (2013). *R: A Language and Environment for Statistical Computing*. [Online]. Available: http://www.r-project.org

[47] H. Wickham, *Advanced R*, 1st ed. Boca Raton, FL, USA: CRC Press, 2014.

[48] M. J. North *et al.*, ''Multiscale agent-based consumer market modeling,'' *Complexity*, vol. 15, no. 5, pp. 37–47, 2010.

[49] Ö. Gürcan, O. Dikenelli, and C. Bernon, ''Towards a generic testing framework for agent-based simulation models,'' in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, Szczecin, Poland, Sep. 2011, pp. 635–642.

[50] C. Macal and M. North, ''Validation of an agent-based model of deregulated electric power markets,'' in *Proc. NAACSOS*, Notre Dame, IN, USA, 2005.

[51] G. K. Bharathy and B. Silverman, ''Holistically evaluating agent-based social systems models: A case study,'' *J. Simul.*, vol. 89, no. 1, pp. 102–135, 2013.

[52] W. Rand and R. T. Rust, ''Agent-based modeling in marketing: Guidelines for rigor,'' *Int. J. Res. Marketing*, vol. 28, pp. 181–193, Sep. 2011.

[53] P. Cooley and E. Solano, ''Agent-based model (ABM) validation considerations,'' in *Proc. SIMUL*, Barcelona, Spain, 2011, pp. 126–131.

[54] Y. Hu, O. Garcia-Cabrejo, X. Cai, A. J. Valocchi, and B. DuPont, ''Global sensitivity analysis for large-scale socio-hydrological models using Hadoop,'' *Environ. Model. Softw.*, vol. 73, pp. 231–243, Nov. 2015.

[55] K. Happe, ''Agent-based modelling and sensitivity analysis by experimental design and metamodelling: An application to modelling regional structural change,'' presented at the 11th Int. Congr. Eur. Assoc. Agricult. Econ., Future Rural Eur. Global Agri-Food Syst., Copenhagen, Denmark, Aug. 2005.

[56] J.-S. Lee *et al.*, ''The complexities of agent-based modeling output analysis,'' *J. Artif. Soc. Soc. Simul.*, vol. 18, no. 4, p. 4, 2015.

[57] J. C. Thiele, W. Kurth, and V. Grimm, ''Facilitating parameter estimation and sensitivity analysis of agent-based models: A cookbook using netlogo and R,'' *J. Artif. Soc. Soc. Simul.*, vol. 17, no. 3, p. 11, 2014.

[58] G. ten Broeke, G. van Voorn, and A. Ligtenberg, ''Which sensitivity analysis method should i use for my agent-based model?'' *J. Artif. Soc. Soc. Simul.*, vol. 19, no. 1, p. 5, 2016.

[59] R. Confalonieri, G. Bellocchi, S. Bregaglio, M. Donatelli, and M. Acutis, ''Comparison of sensitivity analysis techniques: A case study with the Rice model WARM,'' *Ecol. Model.*, vol. 221, pp. 1897–1906, Aug. 2010.

[60] J. Cariboni, D. Gatelli, R. Liska, and A. Saltelli, ''The role of sensitivity analysis in ecological modelling,'' *Ecol. Model.*, vol. 203, pp. 167–182, Apr. 2007.

[61] A. Ligmann-Zielinska and P. Jankowski, ''Spatially-explicit integrated uncertainty and sensitivity analysis of criteria weights in multicriteria land suitability evaluation,'' *Environ. Model. Softw.*, vol. 57, pp. 235–247, Jul. 2014.

[62] V. Makler-Pick, G. Gal, M. Gorfine, M. R. Hipsey, and Y. Carmel, ''Sensitivity analysis for complex ecological models—A new approach,'' *Environ. Model. Softw.*, vol. 26, pp. 124–134, Feb. 2011.

[63] F. Bert, M. North, S. Rovere, E. Tatara, C. Macal, and G. Podestá, ''Simulating agricultural land rental markets by combining agent-based models with traditional economics concepts: The case of the Argentine Pampas,'' *Environ. Modell. Softw.*, vol. 71, pp. 97–110, Sep. 2015.

[64] S. J. Mezias, Y.-R. Chen, and P. R. Murphy, ''Aspiration-level adaptation in an American financial services organization: A field study,'' *Manage. Sci.*, vol. 48, no. 10, pp. 1285–1300, 2002.

[65] R. Karandikar, D. Mookherjee, D. Ray, and F. Vega-Redondo, ''Evolving aspirations and cooperation,'' *J. Econ. Theory*, vol. 80, pp. 292–331, Jun. 1998.

[66] I. Gilboa and D. Schmeidler, ''Reaction to price changes and aspiration level adjustments,'' *Rev. Econ. Design*, vol. 6, no. 2, pp. 215–223, 2001.

[67] B. Kitchenham and S. L. Pfleeger, ''Software quality: The elusive target [special issues section],'' *IEEE Softw.*, vol. 13, no. 1, pp. 12–21, Jan. 1996.

[68] A. V. Feigenbaum, *Total Quality Control*, 3rd ed. New York, NY, USA: McGraw-Hill, 1991.

[69] J. M. Juran and F. M. Gryna, *Juran's Quality Control Handbook*, 4th ed. New York, NY, USA: McGraw-Hill, 1988.
[70] S. McConnell, *Code Complete: A Practical Handbook of Software Construction*, 2nd ed. Redmond, WA, USA: Microsoft Press, 2004.
[71] T. Remencius, A. Sillitti, and G. Succi, "Assessment of software developed by a third-party: A case study and comparison," *Inf. Sci.*, vol. 328, pp. 237–249, Jan. 2016.
[72] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," *J. Syst. Softw.*, vol. 85, pp. 244–257, Feb. 2012.
[73] N. E. Fenton and M. Neil, "Software metrics: Roadmap," in *Proc. Conf. Future Softw. Eng.*, Limerick, Ireland, 2000, pp. 357–370.
[74] G. Subramanian and W. Corbin, "An empirical study of certain object-oriented software metrics," *J. Syst. Softw.*, vol. 59, pp. 57–63, Oct. 2001.
[75] G. Baxter *et al.*, "Understanding the shape of Java software," in *Proc. 21st ACM SIGPLAN Conf. Object-Oriented Program. Syst., Lang., Appl.*, Portland, OR, USA, 2006, pp. 397–412.
[76] B. Kitchenham, "What's up with software metrics?—A preliminary mapping study," *J. Syst. Softw.*, vol. 83, pp. 37–51, Jan. 2010.
[77] A. D. Bakar, A. Sultan, H. Zulzalil, and J. Din, "Predicting maintainability of object-oriented software using metric threshold," *Inf. Technol. J.*, vol. 13, no. 8, pp. 1540–1547, 2014.
[78] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, 1st ed. New York, NY, USA: Pearson Education, 2002.
[79] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. 2, no. 2, pp. 308–320, Dec. 1976.
[80] G. K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig, "Ten simple rules for reproducible computational research," *PLoS Comput. Biol.*, vol. 9, no. 10, p. e1003285, 2013.

SANTIAGO L. ROVERE received the B.S. degree in software engineering from the Universidad de Buenos Aires (UBA), Argentina, in 2013. He is also a member of the Grupo de Aplicaciones de Modelos de Agentes with UBA's School of Engineering. Since 2007, he has been developing agent-based models for different research projects.

He is also an Independent Software Consultant for both private and public institutions, such as Columbia University (NY), the Argentine Met Service, and Argentina's Water Research Institute.

His research interests include the use of agent-based models of complex social-ecological systems to support decision-making.

MICHAEL J. NORTH (M'02–SM'08) received ten college degrees, including the Ph.D. in computer science from the Illinois Institute of Technology. He is a Senior Member of the Association for Computing Machinery and the American Society for Quality. He also holds the Professional Software Engineering Master Certification from the IEEE Computer Society, the Project Management Professional credential from the Project Management Institute, and the Certified Software Quality Engineer credential from the American Society for Quality.

He has authored over 25 years of experience developing advanced analytics applications for industry, government, and academia. He is the Lead Developer for the widely used free and open source Repast agent-based modeling suite. He has co-authored two books, five conference proceedings, one journal special issue, eight book chapters, four invited encyclopedia entries, 25 journal articles, and over 75 conference papers.

Dr. North is the Group Leader of the Integrated Analytics Group and a Principal Computer Scientist with the Global Security Sciences Division, Argonne National Laboratory. He is also a Senior Fellow in the joint Computation Institute of the University of Chicago and Argonne.

GUILLERMO P. PODESTÁ received the Ph.D. degree in biological oceanography from the University of Miami, FL, USA, in 1987. He was a U.S. National Research Council Post-Doctoral Associate with NASA's Goddard Space Flight Center, where he was involved in remote sensing of ocean color.

He is currently a Research Professor with the Rosenstiel School of Marine and Atmospheric Sciences, University of Miami. His research interests include the use of climate information and forecasts to support decision-making in climate-sensitive sectors such as agricultural production. He is also interested in the modeling of complex social-ecological systems.

Dr. Podestá is a member of the American Geophysical Union and the American Meteorological Society.

FEDERICO E. BERT was born in Rawson, Buenos Aires, Argentina, in 1980. He received the B.S. degree in agronomy from the Universidad de Buenos Aires in 2002, and the Ph.D. degree in agricultural sciences from the Universidad de Buenos Aires in 2007.

From 2002 to 2009, he was a Doctoral and Post-Doctoral Fellow of CONICET (Consejo Nacional de Investigaciones Científicas y Técnicas de Argentina) with the School of Agronomy of the Universidad de Buenos Aires. From 2009 to 2015, he was an Associate Researcher with CONICET. He is directing the Research and Development Department of AACREA, the main farmers' organization in Argentina. He has authored over 14 articles and over 50 conference abstracts and papers. His research interest is on modeling agricultural systems.

Dr. Bert co-led the development of public and private tools aimed to support agricultural decisions.