International Environmental Modelling and Software Society (iEMSs)
8th International Congress on Environmental Modelling and Software
Toulouse, France, Sabine Sauvage, José-Miguel Sánchez-Pérez, Andrea Rizzoli (Eds.)
http://www.iemss.org/society/index.php/iemss-2016-proceedings

AgroDB – Integration of Database Management Systems with Crop Models

Alexandre Tagliari Lazzaretti[a], José Maurício C. Fernandes[b], Willingthon Pavan[c], Josué Toebe[a], Roberto Wiest[a]
[a]*Instituto Federal Sul-Rio-Grandense (alexandre.lazzaretti, josue.toebe, roberto.wiest@passofundo.ifsul.edu.br*
[b]*Empresa Brasileira de Pesquisa Agropecuária (mauricio.fernandes@embrapa.br)*
[c]*Universidade de Passo Fundo (pavan@upf.br)*

Abstract: The usefulness of crop simulation models for understanding and transferring agricultural technology is widely acknowledged. The decision support system for agrotechnology transfer (DSSAT) has been in use for the last 3 decades by researchers worldwide. The DSSAT-Cropping System Model (CSM) is a general model for simulating crop growth and development, as well as the soil and plant water, nitrogen phosphorus, and carbon dynamics. The DSSAT-CSM comprises a suite of crop simulation models, including the CSM-CROPSIM wheat. However, depending on the type and number of simulations, it is usually necessary to handle large volumes of input and output data. A database management system (DBMS) provides technological resources and accessible methods for handling big data. In this study, we integrate simulation models with a DBMS and we demonstrate the advantages of this approach. We present a database called AgroDB that acts as a generic tool for crop simulation model users, which facilitates input data quality control, tracks the users selections during simulation model parameterization, and enables the visual analysis of simulation model outcomes. In addition, the system serves as communication route for model coupling. We describe a proof of concept application of the database management system approach and evaluate how well our approach meets its goals.

Keywords: Integrative approach; Procedural Languages; Database design; Model coupling.

1 INTRODUCTION

Crop simulation models can estimate the final yield, as well as simulating the growth and developmental dynamics of crops via numerical integration (Graves et al., 2002). Thus, the use of crop simulation models to predict production trends and to estimate risk has become an important tool for decision making (Donatelli et al., 2002; Sinclair and Seligman, 1996).

Moreover, these models support the scientific community in the organization of knowledge and hypothesis testing. Depending on the type and number of simulations, it is usually necessary to handle large volumes of input and output data. Thus, without adequate mechanisms for handling and storage, this task is difficult or even impossible to perform. Databases can support data management, but the database design should provide an adequate representation of all aspects of the real world.

A database has real value when it matches the needs of both the applications and users (Heuser, 2009). A database management system (DBMS) can store data, as well as providing technological resources and accessible methods for defining, manipulating, and controlling data (Date, 2003; Elmasri and Navathe, 2005; Silberchatz et al., 2006).

In this study, we demonstrate the integration of an object-relational DBMS and crop simulation models. The agronomy database (AgroDB) model acts as a data repository that facilitates integration with crop simulation models. The main features of AgroDB are as follows.

- The flexibility of the DBMS means that it can be used with different crop simulation models, where only a few changes to the physical structures.

- The object-relational model and normalization concept minimize the number of tables and optional fields. Thus, they can characterize the application semantics in a realistic manner.
- The DBMS can be integrated with the simulation model via a set of functions to simplify simulations in different scenarios.
- The proposed database is a tool that facilitates the definition, manipulation, control, and visualization of data.

2 MODEL OVERVIEW

2.1 Architecture

AgroDB comprises a support tool that can be integrated with crop simulation models. AgroDB is useful for storing data in a database, organizing experimental simulations, integrating a database with a crop simulation model via a set of functions, and data visualization. AgroDB has an interconnected modular structure, where each module is separated into parts according to its functionality.

In addition to integrating with crop simulation models, AgroDB comprises a data repository, which can import and export data files in various formats. Data handling, verification, and visualization are facilitated by database technology, such as SQL and procedural languages. For this work, the AgroDB was implemented in PostgreSQL Database Management System (PostgreSQL, 2016).

Crop simulation models require input and output data in a specific format. After it has been integrated with the models, AgroDB can export data in the requisite format for model execution and import the data generated by the simulation. Finally, all the data stored in AgroDB are available for access and manipulation by applications and decision makers (stakeholders).

2.2 Conceptual Model

During the development of AgroBD, a model with an extended relational database and object-oriented features was employed (Elmasri and Navathe, 2005). Two design methodologies were used and adapted according to the object-oriented model characteristics: entity Model transformation and reversing engineering. The entity model transformation considering the applications needs. The reversing engineering is based in the existing data (Heuser, 2009).

Following the object-relational paradigm, the data were stored in structures called tables. The data in a single table represent a relation using object identifier values, or by reference values with primary and foreign keys.

AgroDB is accessed using Structured Query Language (SQL) 3, which is a standard language for working with object-relational databases that was defined in 1999 (Silberchatz et al., 2006). In the present study, table creation was based on a vertical schema rather than a horizontal schema (Dehainsala, 2007). The use of abstract data types was another choice during the design stage, which allows the same data to be stored in different field formats (PostgreSQL, 2016).

In modeling AgroDB, we followed the DSSAT cropping system model design (Jones et. al., 2003). The database model comprised 63 tables, which were separated into the following modules: basics, crops, experiments, integration, simulation models, soil, and weather. The basics and experiments modules had different internal features, so they were subdivided. (Figure 1). Also is available, by request to the first author, the conceptual descriptions based on the Unified Modeling Language (UML) class diagram. The following descriptions explain some of the characteristics of each module.

Basics Module: the main features of this module are required for the maintenance of the basic tables to determine the functioning of the database system. It is subdivided into four sub-modules, as follows:

- basics-control: information required for control by the users and programs that access the data. In this module, classes of people are described with attributes, which characterize the network of people who will use the tools provided by AgroDB. The contacts among these people, their occupations, and the modules that use the system are also described.

- basics-data: information that characterizes and identifies the stored data, thereby identifying the types and sources of data. This module registers the values of the variables used in the database system, which can be categorized and associated with simulation models by setting exact values for the variables used by each simulation model.
- basics-localization: descriptions of geographical locations and data storage. This module registers data about countries, states, cities, macrozones, and regions.
- basics-station: storage of information related to the stations, i.e., the locations where the data were collected. Stations can be classified (e.g., soil and weather) according to the data types provided. Data can also be associated in the format provided by the station.

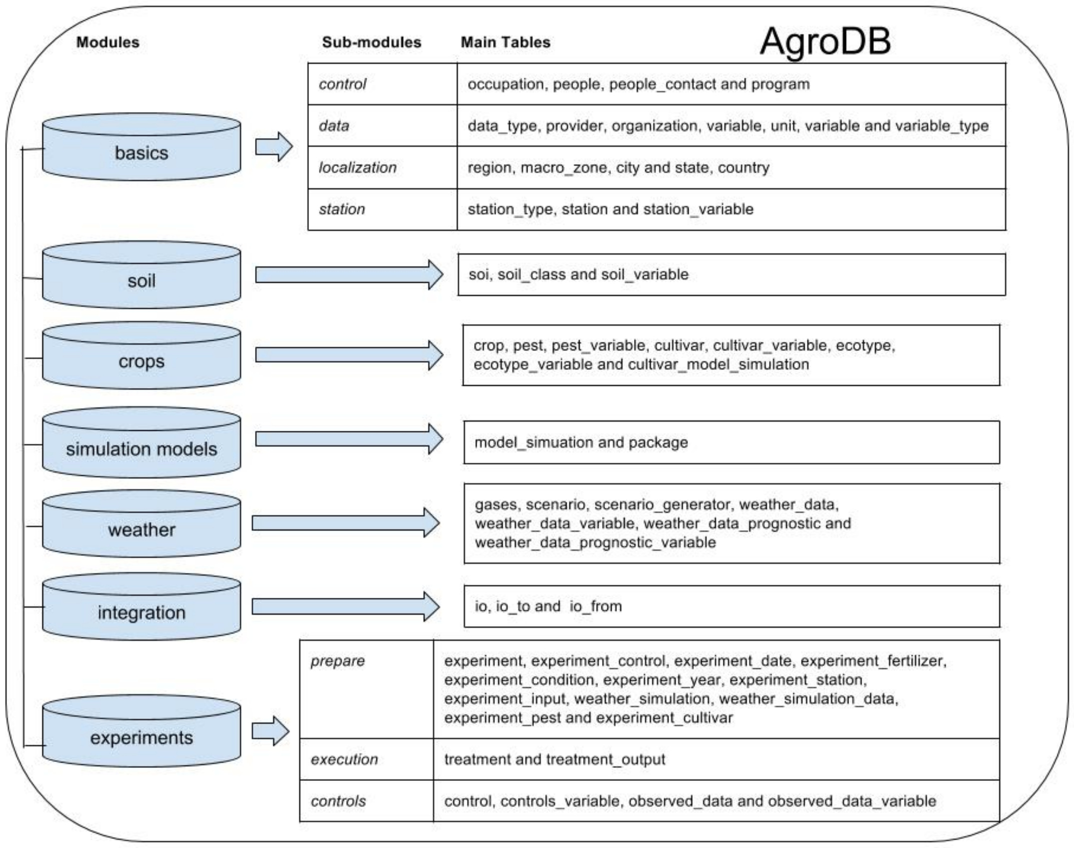
Modules
Sub-modules
Main Tables
AgroDB
basics
control
occupation, people, people_contact and program
data
data_type, provider, organization, variable, unit, variable and variable_type
localization
region, macro_zone, city and state, country
station
station_type, station and station_variable
soil
soi, soil_class and soil_variable
crops
crop, pest, pest_variable, cultivar, cultivar_variable, ecotype, ecotype_variable and cultivar_model_simulation
simulation models
model_simuation and package
weather
gases, scenario, scenario_generator, weather_data, weather_data_variable, weather_data_prognostic and weather_data_prognostic_variable
integration
io, io_to and io_from
experiments
prepare
experiment, experiment_control, experiment_date, experiment_fertilizer, experiment_condition, experiment_year, experiment_station, experiment_input, weather_simulation, weather_simulation_data, experiment_pest and experiment_cultivar
execution
treatment and treatment_output
controls
control, controls_variable, observed_data and observed_data_variable

Figure 1. AgroDB modules, sub-modules and main tables.

Integration Module: the main feature of this module is to maintain the data associated with integration using the database, where integration involves the input and output data managed by the database. Three tables are described in this module: the *io* table represents general information about the data, the *io_to* table represents how the data should be exported, and the *io_from* table represents the imported data. In general, experiments that involve large volumes of data may become too complex to handle using conventional methods such as spreadsheets and text files. Thus, AgroDB could be an important tool for crop simulation model users, especially if large volumes of data and different formats are involved. There are several key features when preparing data for a simulation model run. During module integration, the data types stored in AgroDB can be distinguished easily. For example, we found that it was possible to recognize weather data, soil types, or other data types. These data were stored in the *io* table. It was also possible to store information about the variables and their respective units, which were stored in the *io_from* table. Another important feature was information about the data format required by the simulator. For example, the cropsim-wheat model uses the solar radiation 6 (MJm^2), maximum air temperature (^{o}C), minimum air temperature (^{o}C), and rainfall (mm). Based on the information in the *io_to* table, it was possible to indicate the order, format, and units passed to the simulator. Therefore, knowing the data type and how data should be handled facilitates integration with the crop simulation model.

Simulation Models Module: contains the information required by the crop simulation models and the system to which they belong (Figure 1). For example, CSM-CROPSIM belongs to the DSSAT Cropping System Model.

Soil Module: manages the data associated with the soil characteristics (Figure 1), i.e., information about the soil variables, geographical location, and data sources.

Crops Module: the main feature is the maintenance of the information associated with the crop, including data for the cultivar genetic coefficients, diseases, and information about how a particular cultivar is characterized relatively to a particular crop simulation model (Figure 1).

Weather Module: corresponds to weather data (Figure 1), where it stores information related to observed and predicted weather data. In the latter case, it is possible to associate scenarios that indicate different representations of the climate. This module also stores the data and predictions associated with the accumulation of carbon dioxide in the atmosphere.

Experiment Module: mainly organizes the input data required to design and run virtual experiments. It is subdivided into three sub-modules, which differ according to the stages of the experiment. These modules are described as follows. Figure 2 shows UML class diagram of this module.

- experiments-prepare: contains information that corresponds to stored data required for design of an experiment to be executed later. At this stage, the corresponding data are associated with the experiment, including the weather, sowing dates, years of the experiment, soil data, initial conditions, and details of the control of the simulation model, such as the control of diseases, nitrogen, meteorological stations, and cultivars. The data and files are also specified that will be imported as the results of the simulation.
- experiments-execution: the information required for the execution of the experiment in terms of the simulator and the insertion of the simulation results into AgroDB. The treatment class stores the data obtained as the Cartesian product of the factors of the experiment. The data from each treatment comprise: cultivar, start simulation, experiment, station, soil (data identifier soil, soil class), and weather simulation (data identifier weather, weather simulation class).
- experiments-controls: information related to the organization of the results obtained from the simulations.

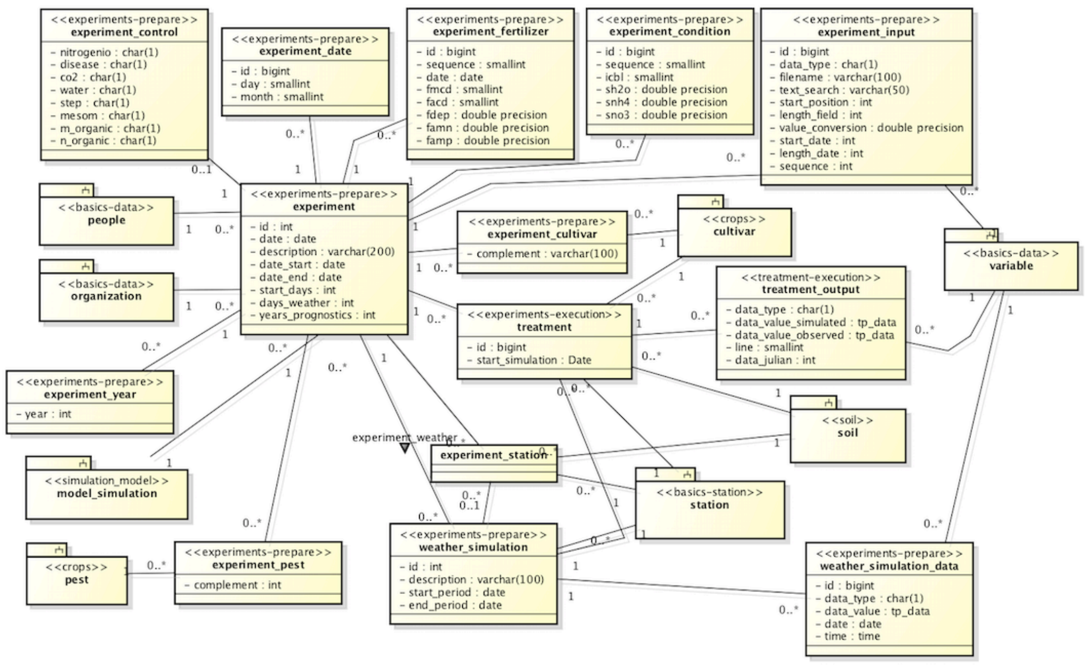
<<experiments-prepare>> experiment_control
- nitrogenio : char(1)
- disease : char(1)
- co2 : char(1)
- water : char(1)
- step : char(1)
- mesom : char(1)
- m_organic : char(1)
- n_organic : char(1)
<<experiments-prepare>> experiment_date
- id : bigint
- day : smallint
- month : smallint
<<experiments-prepare>> experiment_fertilizer
- id : bigint
- sequence : smallint
- date : date
- fmcd : smallint
- facd : smallint
- fdep : double precision
- famn : double precision
- famp : double precision
<<experiments-prepare>> experiment_condition
- id : bigint
- sequence : smallint
- icbl : smallint
- sh2o : double precision
- snh4 : double precision
- sno3 : double precision
<<experiments-prepare>> experiment_input
- id : bigint
- data_type : char(1)
- filename : varchar(100)
- text_search : varchar(50)
- start_position : int
- length_field : int
- value_conversion : double precision
- start_date : int
- length_date : int
- sequence : int
<<basics-data>> people
<<basics-data>> organization
<<experiments-prepare>> experiment
- id : int
- date : date
- description : varchar(200)
- date_start : date
- date_end : date
- start_days : int
- days_weather : int
- years_prognostics : int
<<experiments-prepare>> experiment_cultivar
- complement : varchar(100)
<<crops>> cultivar
<<basics-data>> variable
<<treatment-execution>> treatment_output
- data_type : char(1)
- data_value_simulated : tp_data
- data_value_observed : tp_data
- line : smallint
- data_julian : int
<<experiments-execution>> treatment
- id : bigint
- start_simulation : Date
<<experiments-prepare>> experiment_year
- year : int
<<soil>> soil
experiment_weather
experiment_station
<<simulation_model>> model_simulation
<<basics-station>> station
<<crops>> pest
<<experiments-prepare>> experiment_pest
- complement : int
<<experiments-prepare>> weather_simulation
- id : int
- description : varchar(100)
- start_period : date
- end_period : date
<<experiments-prepare>> weather_simulation_data
- id : bigint
- data_type : char(1)
- data_value : tp_data
- date : date
- time : time

Figure 2. UML Class diagram of the Experiment module.

2.3 Integration

Four integration steps were defined, and their steps and main features are described as follows. The first step is: Data preparation - this involves the preparation of the data needed for the experimental run. We highlight the importance of AgroDB as a repository of data, where data are stored and then extracted by integration with the simulation model. The main data used in the simulations included: climate (predictions and observed), soil, cultivar, ecotypes, and data locations.

The second step is: Preparation of the experiment - in this step, the factors listed in the experimental design were registered, such as the initial conditions related to the location and dates. Next, a set of functions written in PL/PgSQL and PL/R, as well as SQL operations, were used to generate the treatments, data structure, and files needed to run the simulation model. During this stage, it was necessary to specify the experiment that would be performed and whether it was related to future climate data or observed data scenarios. The initial soil conditions were then specified as well as the controls on the model, e.g., irrigation, disease control, and the method used to calculate organic matter. These controls were specified as the simulation model. It was also necessary to register the cultivars to be tested, the dates of planting and sowing, the years when the experiments occurred, and the localities. In this stage, the values of the fertilizer and cover were also specified. Finally, it was registered that the variables would be imported as simulation model results.

The third step is: *Execution* - the integration functions were first implemented in the crop simulation model and the results obtained were entered into AgroDB. All of the functions were parameterized, some of which might call other functions. This explains why more than one language was used. For example, the create directories() function was written in PL/pgSQL and it passed the treatments to the PL/R function, which generated the experimental directories and the respective treatments. The functions with the same order number had the same level of priority, regardless of which was called first. Among the integration functions, generate treatment functions() and create directories() were generic for all the simulation models. Depending on the simulation model, some functions were specific. In the case of Cropsim Wheat, functions were used to generate the files needed to run the model (create files model()), to create files with the soil data (create experiment soil()), to create files for data management and to execute processing (create experiment X()), and to create files for the cultivars data (create experiment cultivar()) and their ecotypes using the function create experiment ecotype(). Files were also created for the weather data (create experiment weather()). Finally, after all the files required to execute the simulator had been created, the simulation was executed for each treatment using the execute cropsim treatment() function. The import results() function imported the simulation results for AgroDB, as configured in the experiment, which were stored in the experiment input table.

The last step is: *Data visualization* - the data were visualized using SQL queries via PL/R functions, R code, and database geographical features. The data stored in the database tables could be manipulated using code.

2.4. Model Coupling

Presently, we are using two types of techniques to coupling multiple models, a scheduled and a communication based approach. In the scheduled approach, models are executed as independent programs and do not communicate with each other while they are executing. Instead, each model produces output data sets at the end of its execution. These output data sets are used as the initial data sets in other models. This approach is not suitable when models need to exchange data while they are executing (Solano et al., 2013; Pavan and et al., 2014).

In the communication approach, models are executed as independent programs and communicate with each other during execution using some form of message passing. In this approach, no significant reprogramming is required to add or replace a model, and models need not be implemented in the same programming language (Solano et al., 2013). Our communication based approach include independent applications (couplers) to mediate the execution and communication among the models.

The core of the model coupling approach is a relational database management system, in this case, AgroDB (Figure 3). The concomitant execution provides infinite possibilities for exchanging data between models, without changing the business logic of individual models. In this way, models legacy can be preserved.

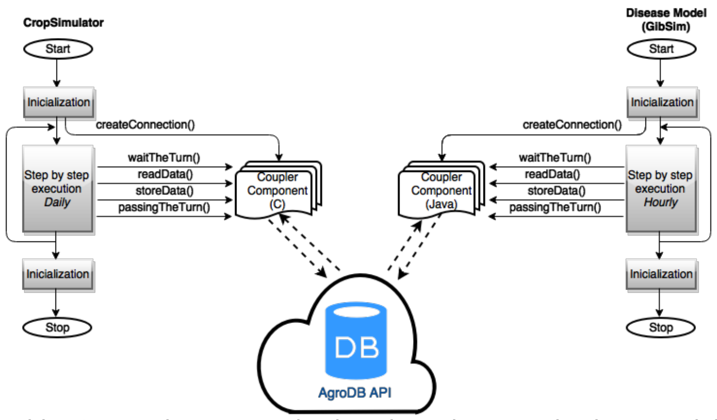

Figure 3. Global architecture to the communications-based approach where models run concurrently while exchanging data.

The AgroDB Architecture source code is available by request to the first author (alexandre.lazzaretti@passofundo.ifsul.edu.br).

3 PROOF OF CONCEPT

A proof of concept application is presented to show the suitability of the database (AgroDB) to (*i*) make data ready, (*ii*) build the experiment , (*iii*) execute a simulation and (*iv*) produce output data visualization. The application aims to simulate an experiment to evaluate the growth and development of the wheat crop in the presence/ absence of a foliar fungal disease. The database (AgroDB) is used to route the communication between a generic plant disease model and a simulation model for the growth and development of wheat.

Briefly, the plant disease model implements a deterministic compartmental model to simulate the response of a generic fungal pathogen to hourly meteorological variables. At the day of disease onset the model is initialized with a proportion of host tissue that is infected. Disease progress depends on weather and availability of susceptible host tissue. The plant disease model implemented was parameterized to reproduce the spring wheat rust (causal agent: Puccinia triticina Eriks) pathosystem. Model description and parameters values to run simulations were reported by Pavan and Fernandes (2009). The leaf rust model used UML and object-oriented programming languages (Java), to simulate the dynamics of a leaf rust epidemic and its impact on wheat production. A standalone version of the CSM CROPSIM Wheat model (Hunt and Pararajasingham, 1995), coded in Fortran 77, was chosen to simulate the growth and development of wheat.

Couplers components were developed for each model family, one in C to be compiled with Fortran 77 code and another in Java to be part of the model architecture. The couplers components mediate data traveler and the turn for each model. While one model goes through the execution the other one waits for its turn. The time step used for each model (daily or hourly) does not affect the unattached execution, allowing the coupling of more than two models, if necessary.

Because the coupled system we developed is a proof of concept only, we did not run it under model comparison scenarios with the ambition of drawing substantive conclusions. However, we did run it to verify that the coupled models interacted with the coupling components correctly and that data moved between the coupled models appropriately.

Sample simulations were carried out using an experimental site located at Londrina, PR, Brazil (lat. -23.30, long. -51.16) during the years 2002-2009. The weather, soil and crop species data needed to run the simulation are present in the AgroDB.

Figure 4 displays the results of a sample scenario. Plots A and B display the daily leaf area index

(m^2/m^2) with and without the disease effect. Plot B shows the disease severity as a percentage of the models total leaf area. Plot D shows a histogram for the final grain yield (Kg/ha) with and without the disease effect. From these plots, we observe that wheat leaf rust impact leaf area and final grain yield. Despite disease presence leaf area and yield varied according to years.

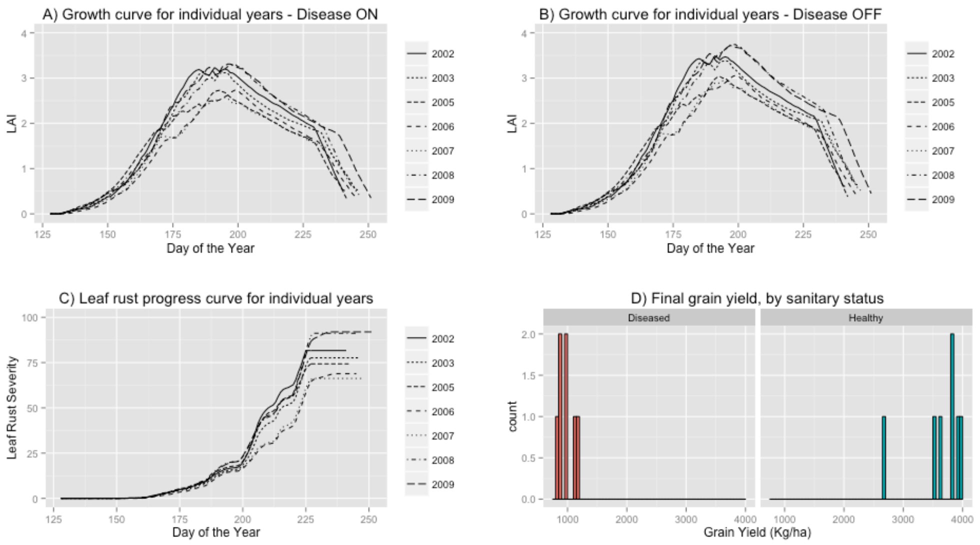

Figure 4. Results from the sample simulation scenarios of the proof of concept coupled system: Spring wheat Puccinia triticina Eriks. pathosystem.

4 DISCUSSION

The AgroDB system, largely automates the labor intensive processes of creating and running data ingest and transformation pipelines and allows researchers to run crop simulators that extend over locations and growing seasons, or evaluate management practices or other input configurations.

The current state of development of AgroDB certainly has not reached its final stage and needs further improvement while tests and applications continue. However, as it stands, provides tools and methods for integrating disciplinary knowledge into comprehensive models of cropping systems. Through model coupling, for example, crop modelers have an opportunity to improve greatly the efficacy with which knowledge from different disciplines combines.

The increasing demand for predicting crop yield, which can neither deny the effect of plant diseases, nor set it as a constant for an environment, requires process based models in which would be possible to extend the results to unknown situations (Bregaglio and Donatelli, 2015). The tools and methods here presented reshape the status of crop model linkages with pest and disease models for impact, adaptation analyses.

Model coupling component grant that models written different programming languages such as Java, R, and Fortran. This provide plasticity to run simulation models concurrently while exchanging data through the AgroDB, regardless of the language used.

Integrating different aspects of a crop simulation model that mimics the growth and development of crops and also accounts for the plant diseases impacts on yield should be useful for analyzing and implementing disease management practices. Moreover, in the case of emergent invasive species, such integrated models can identify areas where the climatic conditions may be suitable for a pathogen to persist from one season to another and to cause economic loss.

Finally, a modeler who decided to add a new model to our AgroDB system can take advantage of the ACE (AgMIP Crop Experiment) REST API to translate data into the input format requirements (Porter et al., 2014). Alternatively, functions have to be changed/added to our AgroDB database to generate particular model reads or accommodating facets of the new model. While we did not attempt to anticipate every possible change, we did design the database having one overriding purpose: Scaling to take in as many features as possible.

REFERENCES

Ali, I., Whisler, F.D., Iqbal, J., Jenkins, J.N., Mckinion, J.M., 2004. Soil Physical Properties Web Database for Gossym and Glycim Crop Simulation Models. Agronomy Journal. 96, 1706–1710.

Bechini, L., Stocle, C., 2007. Integration of a Cropping Systems Simulation Model and a Relational Database for Simple Farmscale Analyses. Agronomy Journal 99, 1226–1237.

Bregaglio, S., Donatelli, M., 2015. A Set of Software Components for the Simulation of Plant Airborne Diseases. Environmental Modelling and Software 0, 119. Http://dx.doi.org/10.1016/j.envsoft.2015.05.011.

Caldeira, C., Pinto, P.A., 1998. Linking DSSAT v3 to a Relational Aatabase: The Agrosys DSSAT Interface. Comput. Eletron. Agric. 21, 69–77.

Date, C. J., 2003. Introdução a Sistemas de Bancos de Dados. 8.ed. Rio de Janeiro, RJ: Elsevier, 2003. 865 p. ISBN 8535212736.

Dehainsala, H.; Pierra G.; Bellatreche, L., 2007. Ontodb: An Ontology Based Database for Data Intensive Applications. in: 12th Int. Conf. on Database Systems for Advanced Applications, DASFAA'07.

Donatelli, M., Ittersum, V., Bindi, M., Porter, J.R., 2002. Modelling Cropping Systems: Highlights of the Symposium and Preface to the Special Issues. Agronomy Journal. 18, 1–11.

Elmasri, R., Navathe, S., 2005. Sistemas de Banco de Dados. 4. ed. São Paulo, SP: Pearson addison wesley, 724 p. ISBN 8588639173.

Graves, A.R., Mathhews, H.T., Stephens, R.B., Middleton, E.T., 2002. Crop Simulation Models as Tools in Computer Laboratory and Classroom Based Education. J. Nat. Resour. Life Sci. Educ. 31, 48–54.

Haley, S.D., May, R.D., Seabourn, B.W., Chung, O.K., 1999. Relational Database System for Summarization and Interpretation of Hard Winter Wheat Regional Quality Data. Crop Science. 39, 309–315.

Heuser, C.A., 2009. Projeto de Banco de Dados. 6. ed. Porto Alegre, RS: Bookman, 282 p. (Série livros didáticos informática ufrgs ; 4). ISBN 9788577803828.

Hunt, L.A., Pararajasingham, S., 1995. Cropsim-wheat: A Model Describing the Growth and Development of Wheat. Canadianjournal Plant Science 75, 612632.

Jones, J. W. et al., 2003. The DSSAT cropping system model European Journal of Agronomy, Vol. 18, No. 3-4. (January 2003), pp. 235-265, doi:10.1016/s1161-0301(02)00107-7.

Pavan, W., et al., 2014. An Integrative Modeling Framework to Evaluate Wheat Production Systems: Fusarium Head Blight, in: 12th Int. Conf. on Database Systems for Advanced Applications, http://www.iemss.org/society/index.php/iemss-2014-proceedings.

Pavan, W., Fernandes, J.M., 2009. Uso de Orientacao a Objetos no Desenvolvimento de Modelos de Simulacao de Doencas de Plantas Genericos. Revista Brasileira de Agroinformatica 9, 12–27.

Porter, C.H., Villalobos, C., Holzworth, D., Nelson, R., White, J.W., Athanasiadis, I.N., Janssen, S., Ripoche, D., Cufi, J., Raes, D., Zhang, M., Knapen, R., Sahajpal, R., Boote, K., Jones, J.W., 2014. Harmonization and Translation of Crop Modeling Data to Ensure Interoperability. Environmental Modelling and Software 62, 495 – 508. URL: http://www.sciencedirect. com/science/article/pii/S1364815214002576, doi:http://dx.doi.org/ 10.1016/j.envsoft.2014.09.004.

PostgreSQL, 2016. Postgresql Database. Avaiable in: http://www.postgresql.org (last acessed 20/02/2016).

Scott, J.M., Lord, C.J., 2003. Gs: Database: Use a Relational Databases to Enhance Data Management for Multisite Experiments. Exp. Agric. 43, 729– 743.

Silberchatz, A., Korth, H., Sudarschan, S., 2006. Sistemas de Banco de Dados. 5. ed. Rio de Janeiro, RJ: Elsevier. xxiii, 781 p. ISBN 9788535211078.

Sinclair, T.R., Seligman, N., 1996. Crop Modeling: From Infancy to Maturity. Agronomy Journal. 88, 698–704.

Solano, E., Morris, R.J., Bobashev, G.V., 2013. Coupling Models by Routing Communication Through a Database. Research Triangle Park, NC: RTI Press. 0. Http://www.rti.org/rtipress.

Stigliani, L., Santospirito, G., Cardinale, N., Resina, C.A., 1996. A Relational Database as Decision Suport System in Chemical Weed Control. Weed Technol 10, 781–794.

Van, E., Spaans, A., E.J., Krieger, S.D., Carlis, J.V., Backer, J.M., 1999. A Database for Agroecological Research Data: I. Data Model. Agronomy Journal 91, 54–62.