ADVANCES IN MODELING
AGRICULTURAL SYSTEMS
Aims and Scope
Optimization has been expanding in all directions at an astonishing rate during the past few decades. New algorithmic and theoretical techniques have been developed, the diffusion into other disciplines has proceeded at a rapid pace, and our knowledge of all aspects of the field has grown even more profound. At the same time, one of the most striking trends in optimization is the constantly increasing emphasis on the interdisciplinary nature of the field. Optimization has been a basic tool in all areas of applied mathematics, engineering, medicine, economics, and other sciences.

The *Springer Optimization and Its Applications* publishes undergraduate and graduate textbooks, monographs, and state-of-the-art expository works that focus on algorithms for solving optimization problems and also study applications involving such problems. Some of the topics covered include nonlinear optimization (convex and nonconvex), network flow problems, stochastic optimization, optimal control, discrete optimization, multiobjective programming, description of software packages, approximation techniques, and heuristic approaches.
To our children:
Dea Petraq Papajorgji
and
Miltiades Panos Pardalos
Agriculture has experienced a dramatic change during the past decades. The change has been structural and technological. Structural changes can be seen in the size of current farms; not long ago, agricultural production was organized around small farms, whereas nowadays the agricultural landscape is dominated by large farms. Large farms have better means of applying new technologies, and therefore technological advances have been a driving force in changing the farming structure.

New technologies continue to emerge, and their mastery and use require that farmers gather more information and make more complex technological choices. In particular, the advent of the Internet has opened vast opportunities for communication and business opportunities within the agricultural community. But at the same time, it has created another class of complex issues that need to be addressed sooner rather than later. Farmers and agricultural researchers are faced with an overwhelming amount of information they need to analyze and synthesize to successfully manage all the facets of agricultural production.

This daunting challenge requires new and complex approaches to farm management. A new type of agricultural management system requires active cooperation among multidisciplinary and multi-institutional teams and refining of existing and creation of new analytical theories with potential use in agriculture. Therefore, new management agricultural systems must combine the newest achievements in many scientific domains such as agronomy, economics, mathematics, and computer science, to name a few.

This volume came to light as the result of combined efforts by many researchers in different areas with the goal of providing the readers with a wide spectrum of advanced applications in agriculture. Readers will find new software modeling approaches such as Unified Modeling Language (UML), Object Constraint Language (OCL), model driven architecture (MDA), and ontologies. Readers will also find a large arsenal of advanced mathematical tools used to study the multiple aspects of agricultural production such as calculation of leaf area index (LAI) from Earth observation (EO) data, accurate estimation of chlorophyll level by remote sensing methods, data mining techniques applied in machine vision, analysis of remotely sensed forest images,
fruit and wine classification, and finally packaging of agricultural products. The main message authors would like to transmit through the chapters of this book is that modeling is a very serious activity, and many types of models must be developed to cope with the complexity of agricultural systems.

Two years ago, we published the book *Software Engineering Techniques Applied to Agricultural Systems: An Object-Oriented Approach and UML*, whose main goal was to provide researchers and students in agricultural and environmental areas with new developments in the field of software engineering. At the time of working on the current volume, the first book has been used as the basic text for the course “Biological Simulation” at the Agricultural and Biological Engineering Department, University of Florida, thanks to the kind decision of Dr. Gregory Kiker with whom we co-teach this course.

While teaching this course, students provided many useful comments and suggestions. They asked questions regarding issues we never thought about before. After 2 years of teaching, the issue that came to us for discussion over and over again was that students wished to see more applications developed using this modeling paradigm. They wished to see more examples of modeling complex agricultural systems. We hope this additional volume will satisfy some of their wishes.

We would like to take this opportunity to thank all contributors and referees for their valuable contributions. Without them, this volume would not have been possible. We would like to thank our students for their valuable feedback and comments; we taught them the science, and they taught us how to improve our work. Last but certainly not least, we would like to thank Springer for giving us another opportunity to work with them.

*Gainesville, Florida*  
Petraq J. Papajorgji  
*Gainesville, Florida*  
Panos M. Pardalos
Contents

1 The Model Driven Architecture Approach: A Framework for Developing Complex Agricultural Systems .................. 1
   Petraq Papajorgji, Ryan Clark, and Eric Jallas

2 A New Methodology to Automate the Transformation of GIS Models in an Iterative Development Process ........... 19
   André Miralles and Thérèse Libourel

3 Application of a Model Transformation Paradigm in Agriculture: A Simple Environmental System Case Study .......... 37
   André Miralles and Thérèse Libourel

4 Constraints Modeling in Agricultural Databases .................. 55
   François Pinet, Magali Duboisset, Birgit Demuth, Michel Schneider, Vincent Soulignac, and François Barnabé

5 Design of a Model-Driven Web Decision Support System in Agriculture: From Scientific Models to the Final Software ...... 67
   Loudovic Tambour, Vianney Houlès, Laurence Cohen-Jonathan, Valerie Auffray, Pierre Escande, and Eric Jallas

6 How2QnD: Design and Construction of a Game-Style, Environmental Simulation Engine and Interface Using UML, XML, and Java ................................................................. 103
   Gregory A. Kiker and Rohit Thummalapalli

7 The Use of UML as a Tool for the Formalisation of Standards and the Design of Ontologies in Agriculture ............ 131
   François Pinet, Catherine Roussey, Thomas Brun, and Frédéric Vigier
8 Modeling External Information Needs of Food Business Networks
Melanie Fritz .................................................................................. 149

9 Enterprise Business Modelling Languages Applied to Farm Enterprise: A Case Study for IDEF0, GRAI Grid, and AMS Languages
Vincent Abt, Frédéric Vigier, and Michel Schneider ................................................................ 167

10 A UML-Based Plug&Play Version of RothC
Petraq Papajorgji, Osvaldo Gargiulo, James W. Jones, and Sibiri Traore .................................................. 193

11 Ontology-Based Simulation Applied to Soil, Water, and Nutrient Management
Howard Beck, Kelly Morgan, Yunchul Jung, Jin Wu, Sabine Grunwald, and Ho-young Kwon .............. 209

12 Precision Farming, Myth or Reality: Selected Case Studies from Mississippi Cotton Fields
Jeffrey L. Willers, Eric Jallas, James M. McKinion, Michael R. Seal, and Sam Turner ......................... 243

13 Rural Development Through Input–Output Modeling
Konstadinos Mattas, Efstratios Loizou, and Vangelis Tzouvelekas ...................................................... 273

14 Modeling in Nutrient Sensing for Agricultural and Environmental Applications
Won Suk Lee, Ismail Bogrekci, and Min Min .................................................................................. 297

15 Estimation of Land Surface Parameters Through Modeling Inversion of Earth Observation Optical Data
Guido D’Urso, Susana Gomez, Francesco Vuolo, and Luigi Dini .......................................................... 317

16 A Stochastic Dynamic Programming Model for Valuing a Eucalyptus Investment
M. Ricardo Cunha and Dalila B. M. M. Fontes .................................................................................. 339

17 Modelling Water Flow and Solute Transport in Heterogeneous Unsaturated Porous Media
Gerardo Severino, Alessandro Santini, and Valeria Marina Monetti .................................................. 361
Contributors

Vincent Abt
Cemagref, Clermont Ferrand, France, LAMSADE, Université Paris-Dauphine, Paris, France, Vincent.abt@cemagref.fr

Valérie Auffray
SAS ITK, Montpellier, France

François Barnabé
Cemagref, Clermont Ferrand, France, Francois.barnabe@cemagref.fr

Amalia Barone
Department of Soil, Plant, Environmental, and Animal Production Sciences, University of Naples Federico II, Naples, Italy, amalia.barone@unina.it

Howard Beck
Department of Agricultural and Biological Engineering, University of Florida, Gainesville, Florida, USA, hwb@ufl.edu

Jevin Bhorania
Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida, jevin@ufl.edu

Ismail Bogrekci
Department of Agricultural and Biological Engineering, University of Florida, Gainesville, Florida, USA, bogrekci@ufl.edu

Nikita Boyko
Department of Industrial and System Engineering, University of Florida, Gainesville, Florida, USA, nikita@ufl.edu

Thomas Brun
Cemagref, Clermont Ferrand, France, Thomas.brun@cemagref.fr

Domenico Carputo
Department of Soil, Plant, Environmental, and Animal Production Sciences, University of Naples Federico II, Naples, Italy, domenico.carputo@unina.it
Contributors

Silvana Cavella
Department of Food Science, University of Naples Frederico II, Naples, Italy, silvana.cavella@unina.it

Radnaabazar Chinchuluun
Department of Agriculture and Biological Engineering, University of Florida, Gainesville, Florida, USA, radnaach@ufl.edu

Maria Luisa Chiusano
Department of Soil, Plant, Environmental, and Animal Production Sciences, University of Naples Federico II, Naples, Italy, marialuisa.chiusano@unina.it

Claudio Cifarelli
Department of Industrial and System Engineering, University of Florida, Gainesville, Florida, USA, claudioc@ufl.edu

Ryan Clark
Sosy Inc., San Francisco, CA, USA, rclark@integranova.net

Laurence Cohen-Jonathan
SAS ITK, Montpellier, France, Laurence.cohen-jonathan@itkweb.com

M. Ricardo Cunha
College of Economics and Management, Catholic University of Portugal and Lancaster University, Porto, Portugal, mrcunha@porto.ucp.pt

Nunzio D’Agostino
Department of Soil, Plant, Environmental, and Animal Production Sciences, University of Naples Federico II, Naples, Italy

Guido D’Urso
Department of Agriculture, Engineering, and Agronomy, University of Naples Federico II, Naples, Italy, durso@unina.it

Birgit Demuth
Department of Computer Science, Dresden University of Technology, Dresden, Germany, Birgit.Demuth@inf.tu-dresden.de

Luigi Dini
Geodesy Centre, Earth Observation Unit, Italian Space Agency, Matera, Italy

Magali Duboisset
Cemagref, Clermont Ferrand, France, Magali.duboisset@cemagref.fr

Pierre Escande
SAS ITK, Montpellier, France

Dalila B. M. M. Fontes
Faculdade de Economia da Universidade de Porto, Rua Dr. Roberto Frias Porto, Portugal, Fontes@fep.up.pt
Contributors

Melanie Fritz
International Research Center on Food Chain and Network Research,
Department of Food and Resource Economics, University of Bonn, Bonn,
Germany, m.fritz@uni-bonn.de

Luigi Frusciante
Department of Soil, Plant, Environmental, and Animal Production Sciences,
University of Naples Federico II, Naples, Italy, luigi.frusciante@unina.it

Maria de Sousa Gallagher
Department of Process and Chemical Engineering, University College, Cork,
Ireland, m.desousagallagher@ucc.ie

Osvaldo Gargiulo
Agricultural Engineering and Biological Department, University of Florida,
Gainesville, Florida, USA, ogargiul@ufl.edu

Susana Gomez
Institute of Applied Mathematics, National University of Mexico, Mexico City,
Mexico, susanag@servidor.unam.mx

Sabine Grunwald
Soil and Water Science Department, University of Florida, Gainesville,
Florida, USA, sabgru@ufl.edu

Vianney Houlès
SAS ITK, Montpellier, France

Eric Jallas
SAS ITK, Montpellier, France, Eric.jallas@wtkweb.com

James W. Jones
Department of Agricultural and Biological Engineering, University of Florida,
Gainesville, Florida, USA, ifur@ufl.edu

Yunchul Jung
Department of Agricultural and Biological Engineering, University of Florida,
Gainesville, Florida, USA

Gregory A. Kiker
Department of Agriculture and Biological Engineering, University of Florida,
Gainesville, Florida, USA, gkiker@ufl.edu

Ho-young Kwon
Soil and Water Science Department, University of Florida, Gainesville,
Florida, USA

Won Suk Lee
Department of Agricultural and Biological Engineering, University of Florida,
Gainesville, Florida, USA, Wslee@ufl.edu
Thérèse Libourel
National Center for Scientific Research, Montpellier Laboratory of Computer Science, Robotics, and Microelectronics, University of Montpellier II, Montpellier, France, libourel@lirmm.fr

Efstratios Loizou
Department of Agriculture Products Marketing and Quality Control, Technological Educational Institution of Western Macedonia, Terma Kontopoulou, Florina, Greece, Lstratos@agro.auth.gr

Pramod V. Mahajan
Department of Process and Chemical Engineering, University College, Cork, Ireland, p.mahajan@ucc.ie

Paolo Masi
Department of Food Science, University of Naples Frederico II, Naples, Italy, paolo.masi@unina.it

Konstadinos Mattas
Department of Agriculture Economics, Aristotle University of Thessaloniki, Thessaloniki, Greece, mattas@auth.gr

James M. McKinion
USDA-ARS-Genetics and Precision Agriculture Research Unit, Mississippi State, Mississippi, USA, mckinion@ra.msstate.edu

Min Min
Department of Agricultural and Biological Engineering, University of Florida, Gainesville, Florida, USA

André Miralles
Centre for Agricultural and Environmental Engineering Research, Earth Observation and GeoInformation for Environmental and Land Development Unit, Montpellier, France, andremiralles@teledetection.fr

Valeria Marina Monetti
Department of Water Resource Management, University of Naples Federico II, Naples, Italy

Kelly Morgan
Soil and Water Science Department, University of Florida, Gainesville, Florida, USA, conserve@ufl.edu

Antonio Mucherino
Center for Applied Optimization, University of Florida, Gainesville, Florida, USA, Amucherino@ufl.edu

Fernanda A.R. Oliveira
Department of Process and Chemical Engineering, University College, Cork, Ireland, f.oliveira@ucc.ie
Contributors

Petraq Papajorgji
Center for Applied Optimization, University of Florida, Gainesville, Florida, USA, petraq@ufl.edu

Panos M. Pardalos
Department of Industrial and System Engineering, University of Florida, Gainesville, Florida, USA, pardalos@cao.isc.ufl.edu

François Pinet
Cemagref, Clermont Ferrand, France, Francois.pinet@cemagref.fr

Catherine Roussey
LIRIS CNRS, University of Lyon, Lyon, France, Catherine.roussey@liris.cnrs.fr

Alessandro Santini
Division of Water Resource Management, University of Naples Federico II, Naples, Italy, alessandro.santini@unina.it

Michel Schneider
Cemagref, Laboratory of Computer Science, Modeling, and System Optimization, Blaise Pascal University, Clermont Ferrand, France, Michel.schneider@cemagref.fr; Schneider@isima.fr

Michael R. Seal
ITD, Spectral Visions, Stennis Space Center, Mississippi, USA

Onur Seref
Center for Applied Optimization, University of Florida, Gainesville, Florida, USA, seref@ufl.edu

Gerardo Severino
Division of Water Resource Management, University of Naples Federico II, Naples, Italy, gerardo.severino@unina.it

Vincent Soulignac
Cemagref, Clermont Ferrand, France, Vincent.soulignac@cemagref.fr

Ludovic Tambour
SAS ITK, Montpellier, France, Ludovic.tambour@itkweb.com

Rohit Thummalapalli
Summer Science Training Program, University of Florida, Gainesville, Florida, USA

Elena Torrieri
University of Naples, Federico II, Italy, Elena.torrieri@unina.it
Sibiri Traore  
Agricultural Engineering and Biological Department, University of Florida, Gainesville, Florida, USA

Sam Turner  
USDA-ARS-Genetics and Precision Agriculture Research Unit, Retired, Starkville, Mississippi, USA

Vangelis Tzouvelekas  
Department of Economics, University of Crete, Rethymno, Crete, Greece, Vangelis@econ.soc.uoc.gr

Frédéric Vigier  
Cemagref, Clermont Ferrand, France, Frederic.vigier@cemagref.fr

Francesco Vuolo  
Ariespace s.r.l., University of Naples Federico II, Ercolano, Italy

Jeffrey L. Willers  
USDA-ARS-Genetics and Precision Agriculture Research Unit, Mississippi State, Mississippi, USA, jeffrey.willers@ars.usda.gov

Jin Wu  
Soil and Water Science Department, University of Florida, Gainesville, Florida, USA

Vitaliy Yatsenko  
Space Research Institute of National Academy of Sciences of Ukraine and National Space Agency of Ukraine, Kiev 03187, Ukraine, vyatsenko@gmail.com
The Model Driven Architecture Approach: 
A Framework for Developing Complex Agriculture Systems

Petraq Papajorgji, Ryan Clark, and Eric Jallas

Abstract  Development and application of crop models is increasingly constrained by the difficulty of implementing scientific information into an efficient simulation environment. Traditionally, researchers wrote their own models and tools, but as software has become much more complex, few researchers have the means to continue using this approach. New modeling paradigms provided by the software engineering industry can be successfully used to facilitate the process of software development for crop simulation systems.

This chapter outlines a model driven architecture (MDA)-based approach to construct a crop simulation model. This new modeling paradigm is a Unified Modeling Language (UML) -based approach. A conceptual model of the problem is first constructed to depict concepts from the domain of the crop simulation and their relationships. The conceptual model is then provided with details about the role each of the concepts plays in the simulation. The multiplicity of the associations between concepts is determined, and the behavior of each of the objects representing concepts of the domain is defined. Mostly, an object’s behavior in the crop simulation domain is expressed using equations. For this type of behavior, this new modeling paradigm offers a declarative way to write equations using attributes of objects participating in the conceptual diagram. For behavior that cannot be expressed through equations, a formal language is used to model behavior without the ambiguities that can be introduced by the use of natural language. Models can be validated and logical flows can be discovered before code generation.

An Extensible Markup Language (XML) representation of the conceptual model is used by an engine that generates automatically executable code in several programming environments such as Java, Enterprise Java Beans, Visual Basic, and .NET. Results obtained from this new approach are presented, and they coincide with results obtained with other approaches.

P. Papajorgji (*)
Center for Applied Optimization, University of Florida, Gainesville, FL, USA
e-mail: petraq@ufl.edu

1 Introduction

The model driven architecture (MDA) is a framework for software development defined by the Object Management Group (OMG) [14]. At the center of this approach are models; the software development process is driven by constructing models representing the software under development. The MDA approach is often referred to as a model-centric approach as it focuses on the business logic rather than on implementation technicalities of the system in a particular programming environment. This separation allows both business knowledge and technology to continue to develop without necessitating a complete rework of existing systems [20].

The MDA approach is making its advance in the software industry consistently. There are a considerable number of software companies providing MDA-based tools such as Kabira (http://kabira.com/), Accelerated Technology (http://www.acceleratedtechnology.com/), Kennedy Carter (http://www.kc.com/), and Sosy, Inc. (http://sosyinc.com/), and a more important number of companies are developing their applications using this technology. According to a recent survey organized by the well-respected Gartner, Inc. [8], given the potential savings and the linkage to requirements that MDA promises, many analysts say it is only a matter of time before MDA-like environments will be mandated by management. A study undertaken by The Middleware Company (http://www.middleware-company.com) encourages organizations wishing to improve their developer productivity to evaluate MDA-based development tools. The Middleware Company has relationships with several hundreds of thousands of developers through TheServerSide Communities and provides the most in-depth technical research on middleware technology available in the industry. Visionary Bill Gates in the Gartner symposium [7, 8] predicted that visual modeling tools will reduce software coding “by a factor of five” over the next 10 years.

The main goal of this study is to evaluate the application of this modeling paradigm in the domain of crop simulation systems. Crop simulation applications are different from business applications. Business applications have the tendency to be linear and usually do not involve a great number of iterations. Crop simulations are repetitive, and calculations are done for each time step of the simulation. There are many examples of applications using the MDA approach in the business area but none in the domain of crop simulation systems. Modeling the relationship between a client and purchases and orders is a relatively well-known process. Crop simulation systems tend to be more abstract than are business systems. Expressing the relationships between plant, soil, and weather and the processes occurring in each of these elements may not be as straightforward as modeling a client–supplier relationship. The level of calculations used in a business model is relatively simple whereas crop simulation models make heavy use of equations.

The MDA-based tool used in this study is the Oliva Nova Model Execution of Sosy, Inc. (http://sosyinc.com/). Because all MDA-based tools implement
the same principles, we believe that the type of the problems to be addressed during model construction would be similar.

2 MDA and Unified Modeling Language

The OMG characterizes the MDA approach as fully specified, platform-independent models that can enable intellectual property to move away from technology-specific code, helping to insulate business applications from technology and to enable further interoperability [21, 22, 30].

One of the key architects of the MDA approach [31] states that one of the goals of MDA is that models are testable and simulatable. Thus, models developed using this new paradigm are capable of execution. In order for a model to be executed, its behavior must be represented. Behavioral modeling is essential to reach the goals of the MDA [17].

MDA uses the Unified Modeling Language (UML) to construct visual representations of models. UML is an industry standard for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system [1], and it has a set of advantages that makes it fit to be the heart of the MDA approach. First, by its nature, UML allows for developing models that are platform-independent [23, 26]. These models depict concepts from the problem domain and the relationships between them and then represent the concepts as objects provided with the appropriate data and behavior. A model specified with UML can be translated into any implementation environment. The valuable business and systems knowledge captured in models can then be leveraged, reused, shared, and implemented in any programming language [3].

A second advantage is that UML has built-in extension mechanisms that allow the creation of specialized, UML-based languages referred to as UML profiles [6]. In the case that modeling agricultural systems would require special modeling artefacts, then an agricultural UML profile would be created and plugged into the UML core system.

Modeling artifacts in UML are divided into two categories: structural models and behavioral models. Structural models include class and implementation diagrams. Behavioral models include use-case diagrams, interaction (sequence and collaboration) diagrams, and state machine diagrams. An important amount of code can be generated using class diagrams (structural models). Even before the MDA approach, many tools vendors provided code generators that produce “code skeletons” using class diagrams. Although this type of code generation is useful, it is very different from the code generation that the MDA approach offers. The source code generated using class diagrams (structural models) has no behavioral semantics. The programmer has to add code to include the business logic into the code.

The MDA approach consists of three levels of models as shown in Fig. 1. As shown in this figure, a set of transformations is needed to transform a model from the current level to the next one.
The approach starts with constructing a conceptual diagram that represents our knowledge of the problem domain expressed through concepts, abstractions, and their relationships [9, 28, 29]. Conceptual diagrams are the result of an activity referred to as conceptual modeling. Conceptual modeling can be defined as the process of organizing our knowledge of an application domain into hierarchical rankings or ordering of abstractions to obtain a better understanding of the phenomena under consideration [2, 33]. Conceptual diagrams have the advantage of presenting concepts and relationships in an abstract way, independent of any computing platform or programming language that may be used for their implementation. During this phase, the focus is on depicting the concepts of the system and providing them with the right data and behavior. The fact that the implementation technology may be Java, a relational database, or .NET is irrelevant at this point. Therefore, the intellectual capital invested in the model is not affected by changes in the implementation technologies. A conceptual model thus is a platform independent model (PIM).

Model construction is done visually using UML, and the participation of domain specialists in the model construction process is greatly facilitated. The MDA approach frees domain specialists from the necessity of knowing a programming language in order to be an active participant. PIMs are developed in UML, which is visual and uses plain English that can be easily understood by programmers and nonprogrammers alike [24]. A PIM is the only model that developers will have to create “by hand.” Executable models will be obtained automatically by applying a set of transformations to the PIM.

After the business issues related to model construction are well-defined and presented in a PIM, then implementation matters such as the programming environment and the computing platform can be addressed. As implementation details in a specific computing environment are considered, a platform specific model (PSM) is constructed using a PIM as starting point. A PSM is a computational model that is specific to some information-formating technology, programming language, distributed component middleware, or messaging middleware [6].

A PSM is obtained by applying a set of transformations to a PIM. A PIM could be transformed into several PSMs, in the case that different implementation technologies are selected to implement the same original PIM. Figure 2 shows an example of a PIM that will be transformed into two different PSMs: one PSM is implemented in a relational database environment and the other is implemented in a Java environment.

**Fig. 1** Transformations are applied to a model level to obtain the next level.
The model presented in Fig. 2 does not refer to any particular computing platform or computing environment. It only shows that objects of type `Plant` and `Soil` are related to each other by the means of associations `growsIn` and `hosts`. The model says that zero or one plant can grow in a soil unit (a small area of soil used in the simulation) and that one unit of soil can host zero or one plant. Both objects have access to each other. `Plant` has a link referred to as `lnkSoil`, which allows access to data and behavior from object of type `Soil`. `Soil` has a link referred to as `lnkPlant`, which allows access to data and behavior from an object of type `Plant`. The model presented in Fig. 2 shows the type of the attributes for each of the objects. For example, `plantingDate` is of type `int`, and `fieldCapacity` is of type `float`. The presented model has the required level of detail to allow the transformations needed to obtain a precise model that includes implementation details.

If the PIM shown in Fig. 2 is to be implemented in a relational database environment, then the following transformations need to occur:

1. A table named `tbPlant` is created, and the names of its columns are `plantingDate`, `rowSpace`, `numberOfLeafs`, and `maxLeafNumber`.
2. A table named `tbSoil` is created, and the names of its columns are `soilDepth`, `soilWaterContent`, and `fieldCapacity`.
3. Both tables will have a column in common, referred to as `soil–plant`, which is a foreign key (a bidirectional link) that links these two tables.

To implement the PIM presented in Fig. 2 in a Java environment, the following transformations need to occur:

1. A class referred to as `Plant` is created having for attributes `plantingDate` of type `int`, `rowSpace`, `numberOfLeafs`, and `maxLeafNumber` of type `float`.
2. A class referred to as `Soil` is created having for attributes `soilDepth`, `soilWaterContent`, and `fieldCapacity` of type `float`.
3. Class `Plant` has an attribute of type `Soil` that allows access to data and behavior defined in class `Soil`. Class `Soil` has an attribute of type `Plant` that allows for accessing data and behavior defined in class `Plant`.

PSMs are provided with enough details so that code generators can automatically translate the model in code in several programming languages. Note
that the example in Fig. 2 is simple and does not contain any behavior defined in either of the objects in the diagram. Generating class skeletons (the class definition with attributes and names of the methods without the body of the method) is not new. Currently, most UML tools on the market produce class skeletons. What makes MDA different and superior from the existing UML tools is that MDA provides the means for representing the behavior of classes. To better understand how to present an object’s behavior, for example the behavior of the method `calculateRate` defined in class `Soil` or `Plant`, the concept of modeling behavior needs to be introduced.

3 Modeling Behavior

Efforts for generating behavior automatically are not new in the history of software engineering. Currently, all of the integrated development environments (IDEs) provide ample support for the drag-and-drop approach. An icon representing a process is dropped on the canvas and the corresponding code is automatically generated. In this case, the icon represents a well-known process that is precoded and ready for use. Although the drag-and-drop approach facilitates enormously the process of software development, it is not a general means for modeling behavior because it is limited to cases where the behavior is known prior to model construction. As it is difficult to predict the behavior of the potential objects used in a system, this approach has not solved the problem of modeling behavior.

The problem of finding ways to express behavior has been addressed in two different ways by the researcher community. The reason for this could be found in the gap that exists between the expressive power of structural models and the potential complexity of the behavioral requirements that need to be expressed [16].

3.1 The Object Constraint Language

One line of researchers uses a formal language to address the problem of modeling behavior. There is a branch of computer science that studies how to provide precise and unambiguous descriptions of statements using formal languages. These languages are highly mathematical [36] and difficult to use by mainstream developers [35]. Efforts were undertaken to create a language that is simple and yet rigorous enough to express the behavior of objects. Thus in 1995, IBM’s software engineers created the Object Constraint Language (OCL), a subset of UML that allows software developers to apply constraints to objects. These constraints are useful and provide developers with a well-defined set of rules that controls the behavior of an object. The behavior of a system can be expressed by preconditions and postconditions on operations [14]. This approach is largely inspired by ideas of the well-known modeling
philosophy of “Design by Contract” [19]. Advocates of this approach do not pretend that complete code generation is possible. They state that for relatively simple operations, the body of the corresponding operation might be generated from the preconditions and postconditions, but most of the time the body of the operation must be written in PSM [14]. Furthermore, they clearly state that the dynamics of the system still cannot be fully specified in the UML–OCL combination [11, 14].

The following example shows the use of OCL to describe how to define the phenological phase of object Plant. The evaluation context is Plant, meaning that self represents object Plant. phenologicalPhase is an attribute of class Plant.

```
Plant
phenologicalPhase = if self.numberOfLeaves < self.maximumLeafNumber
                 then vegetative
                 else reproductive
              endif
```

It is important to note the main characteristics of OCL. First, OCL is purely an expression language. An OCL expression does not have side effects; it does not change the status of the model. Whenever an OCL expression is evaluated, it simply delivers a value. Second, OCL is not a programming language, and therefore it is not possible to write program logic in OCL. A complete description of OCL can be found in Ref. 35. This approach has been used by a number of authors such as D’Souza and Wills [5], Cook and Daniels [4], and Walden and Nerson, [34].

### 3.2 The Action Language

Another group of researchers followed a different direction, the one of state machine–based models to describe behavior. They created the action language action language that allows describing actions an object performs when receiving a stimulus [18, 32]. Action languages abstract away details of the software platform so that the designer can write what is needed to be done without worrying about distribution strategies, list structure, remote procedure calls, and the like. The use of the state machines to specify behavior assumes no gap between expressive power and behavioral complexity [17]. Thus, it is possible to construct tools that express complex behavior and generate complete code from well-thought models. Because of the ability to capture arbitrarily complex behavior at a high level of abstraction, this approach seems capable of fully supporting the vision of MDA [17].

The following shows examples of statements in action language:

```
create object instance newSoil of Soil;
newSoil.name = "sandy soil";
```
soilName = newSoil.name;
delete object instance newSoil;

The first line creates an instance of class Soil and assigns it to newSoil. Therefore, newSoil refers to an instance of Soil class. The second line assigns to attribute soilName a value that is the name of the soil. The third line assigns to variable soilName the value of the attribute name. The last line deletes the newly created object. Selection expressions are used to select single objects or a set of objects. The following shows examples of the use of section expressions:

select many soils from instances of Soil;
select many soils from instances of Soil where selected.name = "sandy soil";

The first line selects in the set of instances created from class Soil some of these instances. The second line selects only the instances of class Soil that satisfy the condition soils should belong to the category “sandy soils.”

The action language approach is widely accepted by the embedded systems community, and only recently are there applications in the business information area. Among software companies that have adopted the action language approach are Accelerated Technology (http://www.acceleratedtechnology.com/), Kennedy Carter (http://www.kc.com/), and Sosy, Inc. (http://sosyinc.com/) to name a few.

4 Modeling a Crop Simulation

The crop simulation approach chosen for this study is the Kraalingen approach to modular development [13]. We choose to investigate this approach for two reasons. First, we had developed an object-oriented implementation using this approach, so this provided a basis for comparing results obtained with the new approach. Second, the Kraalingen approach is used by DSSAT-CSM crop model [12]. The experience obtained in this study may be useful in future studies when more complex crop simulation models could be considered for development using this new paradigm.

4.1 The Conceptual Model, or PIM

As previously mentioned, the MDA approach starts by constructing a conceptual model that depicts concepts from the problem domain and their relationships. The conceptual model for the Kraalingen approach is shown in Fig. 3. In this figure, object Simulator is the center of the model, and it plays a supervisory role. Simulator has relationships with Plant, Soil, and Weather objects. The nature of the relationship is a composition; Simulator plays the role of whole and Plant, Soil, and Weather play the role of parts. This means that object Simulator “owns”
the related objects and therefore can control the manifestation of their behavior. *Simulator* is provided with behavior to send the right message at the right time to related objects to carry out the simulation process [24].

The cardinality of the association between *Simulator* and *Soil*, referred to as controls, shows that one instance of *Simulator* controls zero or one instance of *Soil*, and one instance of *Soil* is controlled by one instance of *Simulator*. Furthermore, besides the cardinality, the association shows that the role of *Simulator* is static, and the role of *Soil* is dynamic. Let us provide some more information on the nature (static and dynamic) of the role that classes can play in an association as it is an important concept.

The fact that *Simulator* plays a static role means that a relationship between *Simulator* and *Soil* can be established when an instance of *Soil* is created. Furthermore, the relationship can be deleted only when the instance of *Soil* is deleted. As *Simulator* has the same type of relationship with *Plant* and *Weather*, then, when an instance of *Simulator* is created, instances of *Soil*, *Plant*, and *Weather* are created and the corresponding relationships are established. The 1:1 cardinality allows *Simulator* to navigate through all the related objects.
The relationship referred to as `plantGrowsInSoil` expresses the interaction between objects of type `Plant` and `Soil` in the simulation. The `Soil` class models the unit of soil used in the simulation process. Note that `Soil` plays the static role and `Plant` the dynamic one. We modeled this relationship in this way to express the fact that an instance of `Soil` must exist before an instance of `Plant` is related to it. This relationship represents truthfully the natural relationship between a plant and a soil in the real world.

The `Weather` component is considered as a container of `DailyWeather` data, and their relationship is referred to as `contains`. The multiplicity of this relationship shows that an instance of the container `Weather` may contain zero or many instances of `DailyWeather` and that instances of `DailyWeather` will be stored in one container. Note that the static role in this relationship is played by container `Weather`, meaning that the container must exist before instances of `DailyWeather` are stored into it. There is another relationship between container `Weather` and `DailyWeather`, referred to as `currentDay` to represent the weather data for the current day. The calculation occurring during the phases of rate calculation and integration [27] use data stored in the instance of `DailyWeather` pointed by `currentDay`.

### 4.2 Providing Objects with Behavior

Providing objects of the conceptual diagram with behavior is one of the most exciting features of the MDA approach. In the world of the simulation models, most of the behavior that objects should provide is expressed in the form of equations. Equations are constructed in a declarative way using attributes of objects participating in the conceptual diagram. Figure 4 shows the example of calculation of the soil attribute albedo using a declarative approach.

As shown in the figure, leaf area index data stored in `Plant` is needed to calculate albedo. Attributes of all objects participating in the conceptual diagram are available for use in formulas. Note that calculations may be associated with some conditions that must be satisfied before the calculations take place. To avoid errors occurring when leaf area index data is requested from a nonexistent object of type `Plant`, the condition “`EXIST(Plant) = true`” ensures that the calculations will take place only when an instance of `Plant` exists. In the case that the required instance does not exist, then an error will be displayed and the system halts.

In the case that the behavior of an object cannot be expressed by an equation, a formal language is provided to model behavior. The formal language used to model behavior is a type of action language. The behavior is referred to as services that an object provides. This language offers three types of services: events, transactions, and operations.

An event is an atomic unit of activity that occurs at specific instances of time. Events can create an object from a class, destroy an object, and modify
the life cycle of the corresponding object. Processes \textit{calculateDailyNetPhotosynthesis}, \textit{calculateDeltaLeafAreaIndex}, and \textit{calculateOrganDryWeight} for \textit{Plant} are modeled as events. These processes are part of the rate calculation for \textit{Plant} [27].

Processes that are modeled as events should be simple and deal with a well-defined activity. As an example, \textit{calculateDeltaLeafAreaIndex} is a simple process that has as a result the calculation of the value for attribute \textit{deltaLeafAreaIndex}. An event does not consider the order in which processes should occur. Therefore, the process \textit{calculateRate} cannot be modeled as an event. The order in which \textit{calculateDailyNetPhotosynthesis}, \textit{calculateDeltaLeafAreaIndex}, and \textit{calculateOrganDryWeight} are executed matters. The process \textit{calculateOrganDryWeight} uses \textit{deltaTotalDryWeight} and the latter uses \textit{dailyNetPhotosynthesis}. Therefore, \textit{calculateOrganDryWeight} cannot be executed before \textit{calculateDailyNetPhotosynthesis}. If processes need to be executed in a defined order, then another type of service, transactions, should be used.

A transaction is an atomic processing unit composed of one or more events or other transactions. Similar to events, transactions are used to create, destroy, and modify the life cycle of objects. The state of an object involved in a transaction cannot be queried until the transaction is finished. If an error occurs

\textbf{Fig. 4} Example of creating formulas in a declarative way
during the execution of a transaction, the state of objects involved in the transaction will be restored. The capability of restoring initial values is referred to as rollback. They will exit the transaction with the state objects had before the execution of the transaction. Processes initialize for Plant, Soil, and Weather and calculateRate for Plant [27] are modeled as transactions.

The following is the example of the transaction SIMULATE that initializes the Simulator and starts the simulation process:

```
INITIALIZESIMUL(pt_pt_pt_p_atrsoilDepth, pt_pt_p_atrwiltPointPercen,
pt_pt_in_plantingDate, pt_pt_in_FilePath).
FOR ALL Weather.DailyWeather DO SENDMESSAGESIF(THIS,
Weather.DailyWeather)
```

The first line shows the transaction INITIALIZESIMUL with a list of parameters provided by the user. The parameters are soil depth, wilting point percent, and planting date. This transaction creates a new instance of Simulator, Plant, Soil, and Weather and establishes the relationships between them. The second line represents a statement in the action language that has as a result looping over all instances of DailyWeather stored in the container Weather. For each instance of DailyWeather, the transaction SENDMESSAGEIF executes the two instances given in its parameter set: the Simulator itself and the current DailyWeather. The body of this transaction is as follows:

```
Soil.calculateRate(Soil).
{Plant.isPostPlanting = TRUE}
Plant.CALCULATERATE(Plant).
Soil.integrate(Soil).
{Plant.isPostPlanting = TRUE}
Plant.integrate(Plant)
```

Note that for each instance of DailyWeather, Plant and Soil will receive messages calculateRate and integrate as defined by [13].

The third type of service is referred to as operations. An operation is similar to a transaction but does not offer rollback capabilities. Because of the similarity with transactions, operations are not used in this project.

The simulation process is controlled by the behavior of object Plant. A state-transition diagram is used to model the behavior of Plant [1]. This diagram shows the valid execution order of the services of the class and the set of possible life cycles of Plant. Figure 5 shows the state-transition diagram of Plant. The diagram has two types of elements: states and transitions. States represent the different situations through which an object of type Plant can pass, depending on the value of its attributes. Transitions represent executed services, events, or transactions, which produce state changes and modify the value of object’s attributes.
According to the state-transition diagram, Plant will remain in the state vegetative and will continue to receive messages calculateRate and integrate as long as the guard condition number of leaves > maximum number of leaves is not satisfied. When the guard condition is satisfied, Plant will move to state reproductive. For this transition, the source state is vegetative and the target state is reproductive. Plant will remain in the state reproductive as long as the guard condition cumulative thermal time > reproductive thermal time is not satisfied. When the guard condition is satisfied, Plant will move to state mature and the simulation will terminate.

4.3 Data Requirements

The system uses weather data saved locally in a text file as described in Ref. 27. The process of reading weather data is not part of the model and cannot be modeled using the available artifacts. The Oliva Nova Model Execution provides access to external functionalities through user functions. A user function can be written in any of the languages that are generated by the MDA-based tool such as Java, .NET, and so forth. In this project, a user function referred to as importWeatherData is designed to provide access to weather data. The source of the weather data can be a text file, a database, or a weather station online [25].
The software system generated by the MDA tool is a Web-based application. Interested readers can test the system using the URL http://mda.ifas.ufl.edu. The user is prompted to enter some initial plant and soil data such as planting data, soil depth, and wilting point percent before running a simulation. Different simulation scenarios can be obtained by changing the values of the initial data. If other initial data are needed as input parameters, then they need to be added to the user interface. Note that the user interface is automatically generated based on the user’s specification using the presentation model. The presentation model is part of a set of models provided by the MDA tool, designed to generate the application’s user interface.

4.4 Code Generation

As mentioned previously, the heart of the MDA approach is generating code by applying a series of transformations to the PIM. The conceptual model expresses all the elements of the problem domain and their relationships. The model must be complete and rigorous so that code could be generated by applying transformations to the model.

According to the code-centric approach, only when code is obtained can errors be found and corrected. In the MDA approach, the process of checking the validity of the model can start while designing the model. MDA-based tools provide ample capabilities to check the correctness of the conceptual model, the behavior of each object in the model, and relationships between objects. An Extensible Markup Language (XML) file containing detailed specifications about the model is created and is used by code engines/compilers to generate code in several programming languages. Several scenarios can be considered as different parts of the system can be implemented in different languages. For example, a user can choose the C# environment for developing the user interface and a CORBA-EJB, Java-based environment for the implementation of the server. Because the conceptual model is detailed and precise, code generators can find all the needed information to translate the model into several programming environments. Besides the code representing objects of the conceptual model, code generators will provide all the wiring code that links the client and the server applications.

4.5 Results

As previously mentioned in this chapter, a Java version of the Kraalingen simulation model was developed by the authors. The validation of the Java version was done in collaboration with crop specialists and the results were compared with a FORTRAN version developed earlier [27]. Results obtained by the software developed using the MDA approach are presented in Fig. 6.