Florida Climate Institute
AT THE UNIVERSITY OF FLORIDA

Visiting Scholar Seminar
November 10, 2015

Enhancing model reuse: shifting the focus from frameworks to components in the development of biophysical simulation systems

Marcello Donatelli

crea

Research Center for Agriculture and Environment

Bologna, Italy

marcello.donatelli@entecra.it

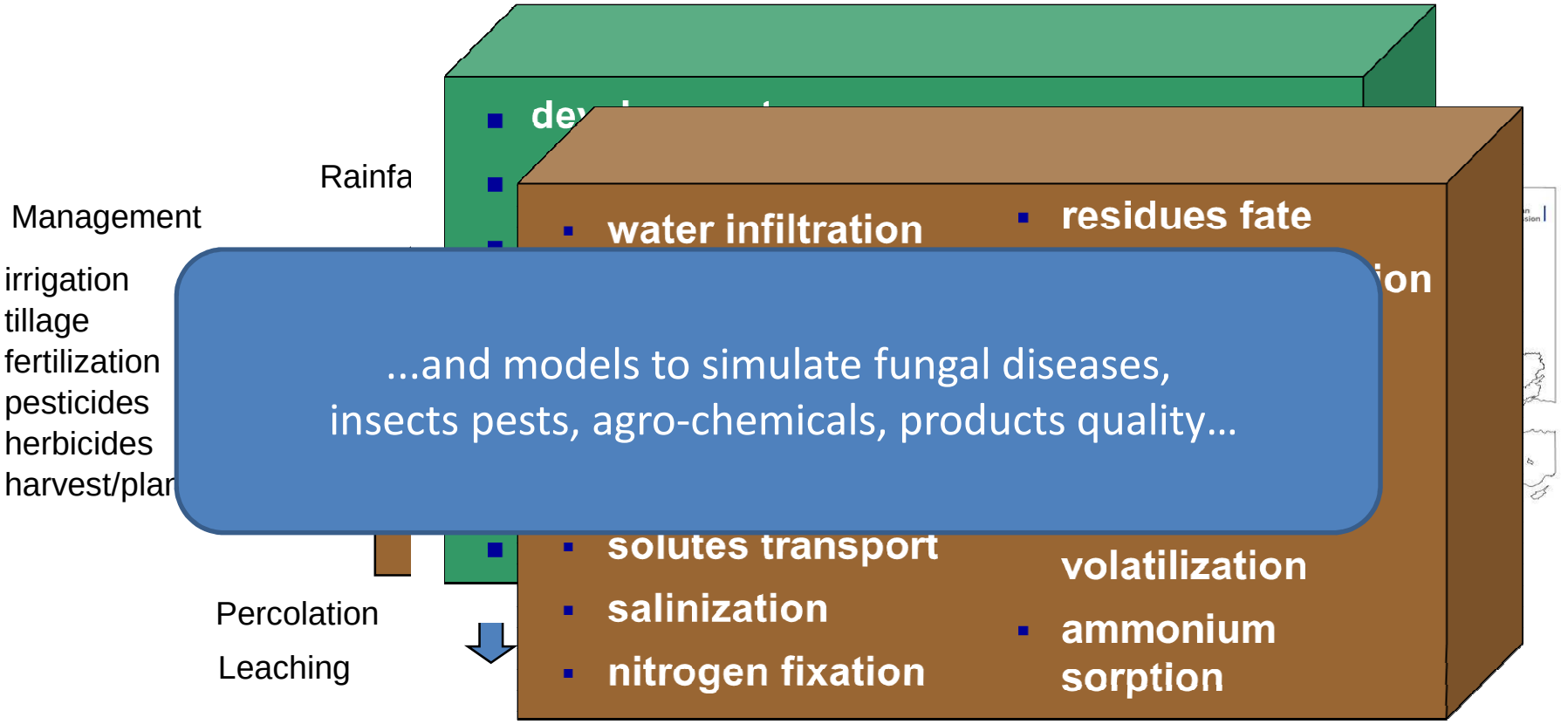
Outline

- Modelling frameworks
- Model reuse
- Imperative vs. declarative
- From models to viewvers
- BioMA applications
- Conclusions

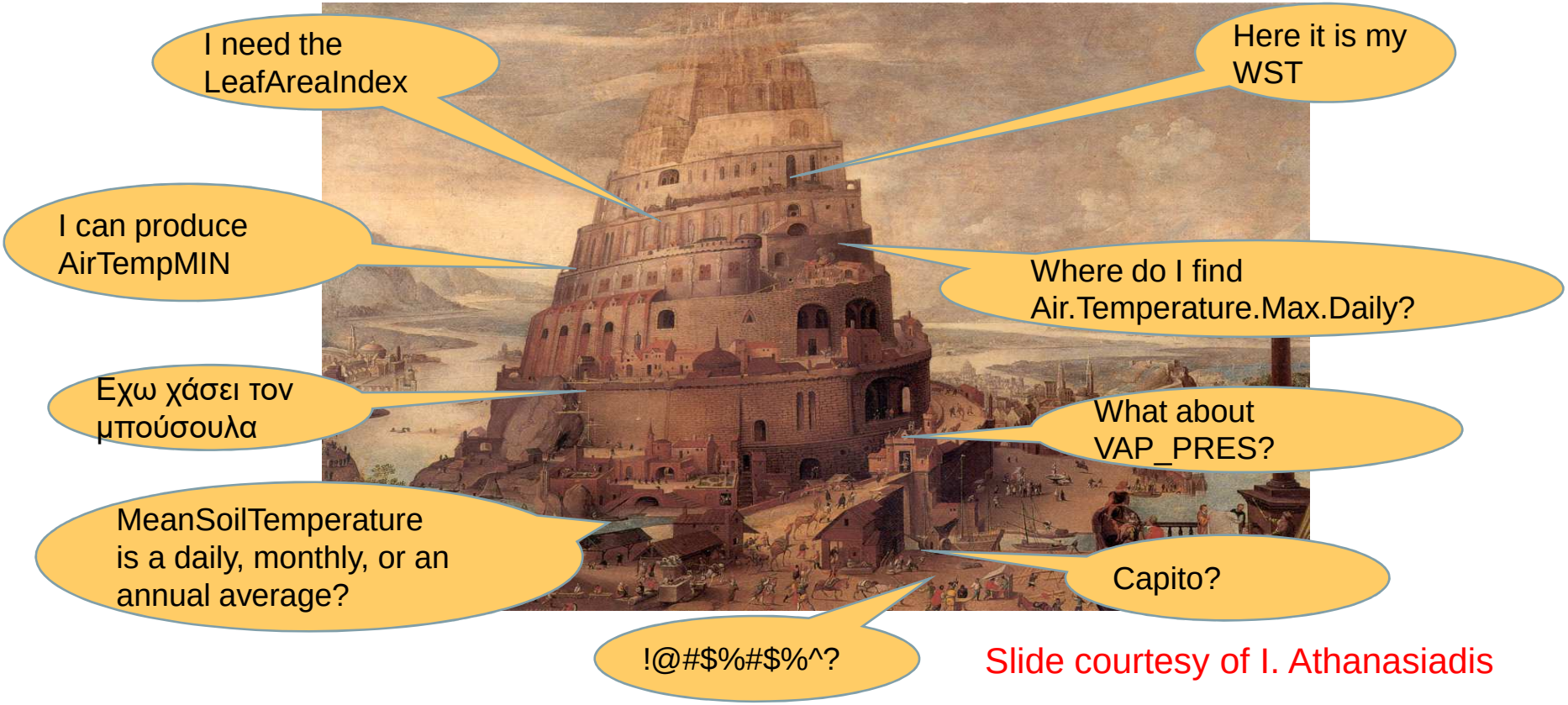
Model development and reuse

- The demand of model tools to perform integrated evaluation of agro-ecological systems has further increased in the last decade.
- New requirements for simulation capabilities have emerged, and the capability to timely transfer research results to modelling tools is key to meet the demand of various stakeholders.
- The major obstacle to develop such simulation systems has been the fragmented availability of modelling resources, partly due to technical bottlenecks.
- Extension of modelling resources by adding modules, and replacing or changing existing ones to accommodate new modules, has not been at reach except by duplication via full recoding.

The generic modelling problem



Agronomists, Soil scientists, Geographers, Meteorologists, ...



Slide courtesy of I. Athanasiadis

Model frameworks

- Since many years model frameworks have represented a substantial step forward with respect to monolithic implementations of biophysical models.
- The separation of algorithms from data, the reusability of services such as I/O procedures, have brought a solid advantage in the development of simulation systems.
- However, the diffusion of such frameworks beyond the groups developing them, as model development environment, can be considered modest.
- The reusability of models has also proved to be modest; a model unit for a given framework is not used in other frameworks.

Model frameworks (cont.)

- Current implementations of modeling solution vary from structured Fortran code to modular realizations using C++, or via memory-managed object-oriented languages (C#, Java).
- Even if the technology available would allow much more functionalities of what is being exploited in most cases, software framework have been constantly improved in terms of quality; still, the problem of model reuse has been largely unsolved.
- The focus on frameworks has made software architects at least partially overlooking on the requirements of reusability per se of model units.

Why reuse is important?

- Avoiding duplication per se is not the main issue; instead, time and quality of modelling solution development are limiting factors:
  - Reusing a model unit to explore and compare options during model development shortens the time of delivery, allowing also comparative testing.
  - It may also allow using expertise in different domains via dicrete units of high quality.
- Some duplication can be considered acceptable in the final version of new tools, but not to explore options during model development.

Reuse of what?

- The words “model reuse” can have different meanings; within biophysical modelling, a “model” can be:
  - A process-based representation of either a single process or of a compartment;
  - A whole modelling solution allowing the simulation of the system of interest.
- The point in reuse however is not the granularity of the model unit, as in model development;
- The key difference is if either the discrete unit is to be reused in workflows, or is composed to iterate with other model units over the execution of time-steps.

Asynchronous or synchronous?

- If the reuse is of modelling solutions, adapters are built following the requirements of a specific software framework either to enable links in workflows, or for ensemble runs.
  - The “model” units run asynchronously
  - Incompatibilities of binaries can be overcome
- In the case of iteration across model units over time the requirements for an effective reuse increase:
  - Key features in the architecture of model units are required
  - Binaries incompatibility across models units becomes an insurmountable obstacle in practical terms
- Model development requires being able to modify and link model units working synchronously at run time.

New requirements

- Also, some new high level requirements emerged for modelling frameworks:
  - To increase the transparency of the modelling solutions being built compared to legacy code available, for each of the modelling solutions being built;
  - To increase the traceability of performance of each modelling unit used in modelling solutions;
  - To involve teams without requiring them to commit to a whole infrastructure they would not own.
- To maximize both reusability and accessibility, an option is to develop a simulation system based on framework-independent components, both for model and for tool components.

Outline

- Modelling frameworks
- Model reuse
- Imperative vs. declarative
- From models to viewers
- BioMA applications
- Conclusions

Imperative or declarative implementation?

- Declarative implementation via a mark-up language is appealing because it does not bind to either a specific architecture or software platform, and sharing declarative units is potentially easier.
- A model represented declaratively can be used by a code layer to produce an imperative implementation, which runs either autonomously or via specific tools, potentially in multiple platforms.
- However, the advantage attributed to declarative implementation implies that the major obstacle in model reuse is due to the difficulty in reproducing the algorithms to link.

Imperative or declarative … (cont.)

- Instead, other features of the software system such as I/O flexibility, data adequacy control, iteration and integration, may be the limiting factors in bridging from prototypes to operational systems in biological systems simulation.
- So far, the vast majority of model developers has preferred imperative implementation, considering the declarative implementation as another demanding layer with no clear added value.
- Declarative implementation may still have a role, but it does not appear to be the unique solution to model reuse.

What next?

- The adoption of a software framework to implement models implies committing large resources, and impacts on specific production environments.
- The current situation from one side shows no model reuse other than, at most, whole modelling solutions in asynchronous implementations using wrappers.
- Shared libraries of model units are de facto not available.
- There are however options to build a common model base which accounts for the constraints summarized above, at the same time fostering model reuse.

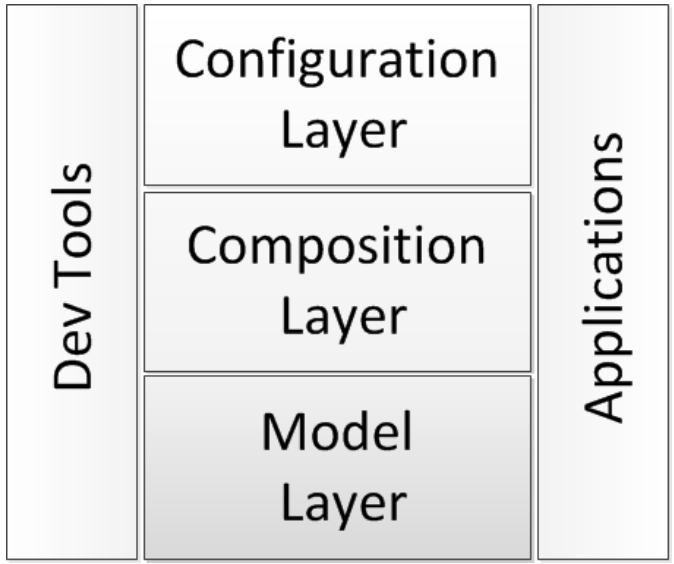
Outline

- Modelling frameworks
- Model reuse
- Imperative vs. declarative
- From models to viewers
- BioMA applications
- Conclusions

The high level architecture of a software modelling environment

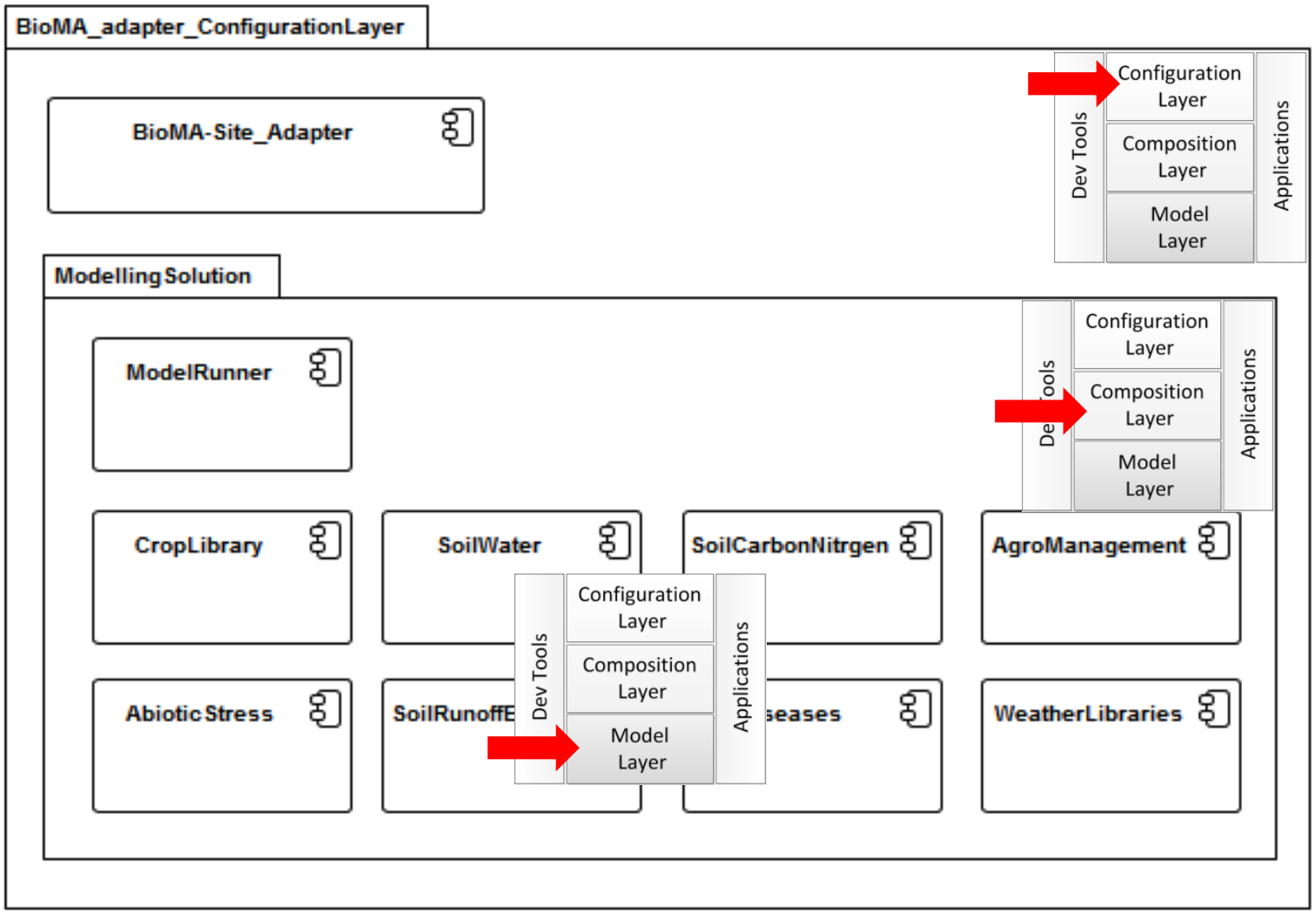
FROM MODELS TO VIEWERS

From models to viewers



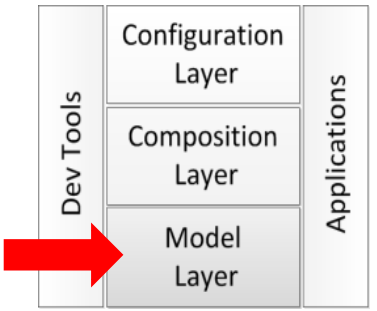
- **Model Layer**: fine grained/composite **models** implemented in components
- **Composition Layer**: **modeling solutions** from model components
- **Configuration Layer**: **adapters** for advanced functionalities in controllers
- **Applications:** from console to advanced MVC implementations
- **DevTools**: code generators, UI components and applications

From models to applications



fine grained/composite models implemented in components

THE MODEL LAYER



Defining the domain

- Domain classes contain the definition of the domain being modelled
- Variables are properties in domain classes which are homogeneous for content: states, rates, exogenous etc.
- Domain classes are input/output objects for model classes
- Each quantity is defined as name, definition, units and max/min/default values
- The same type of attributes is also used for parameters

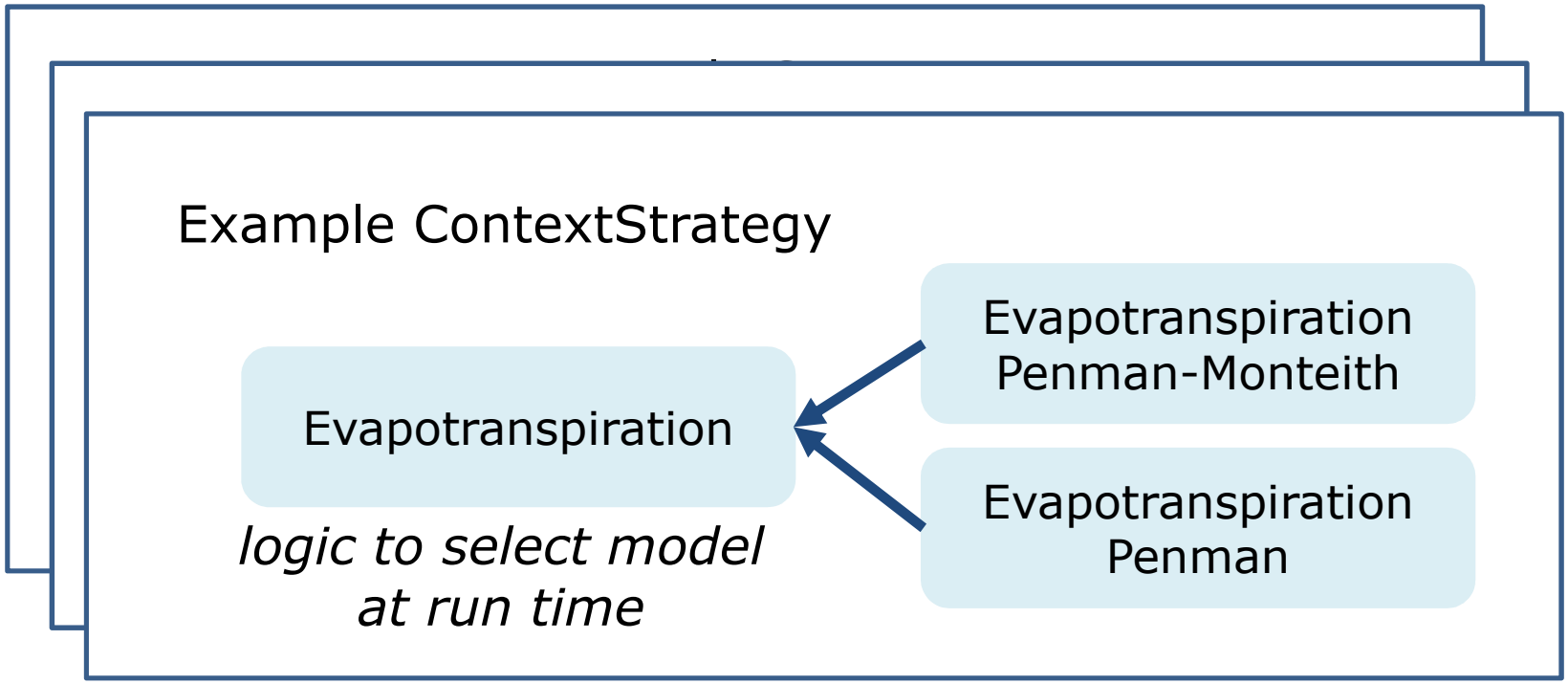
Model units

Each model is implemented as a stateless class which:

- Implements the definition of its own parameters;
- Implements the test of pre- and post-conditions;
- May use other classes sharing the same interface;
- Exposes the list of its inputs, outputs, simulation options, and parameters;
- Provides scalable logging;
- Implements default (Euler's) integration.

crea

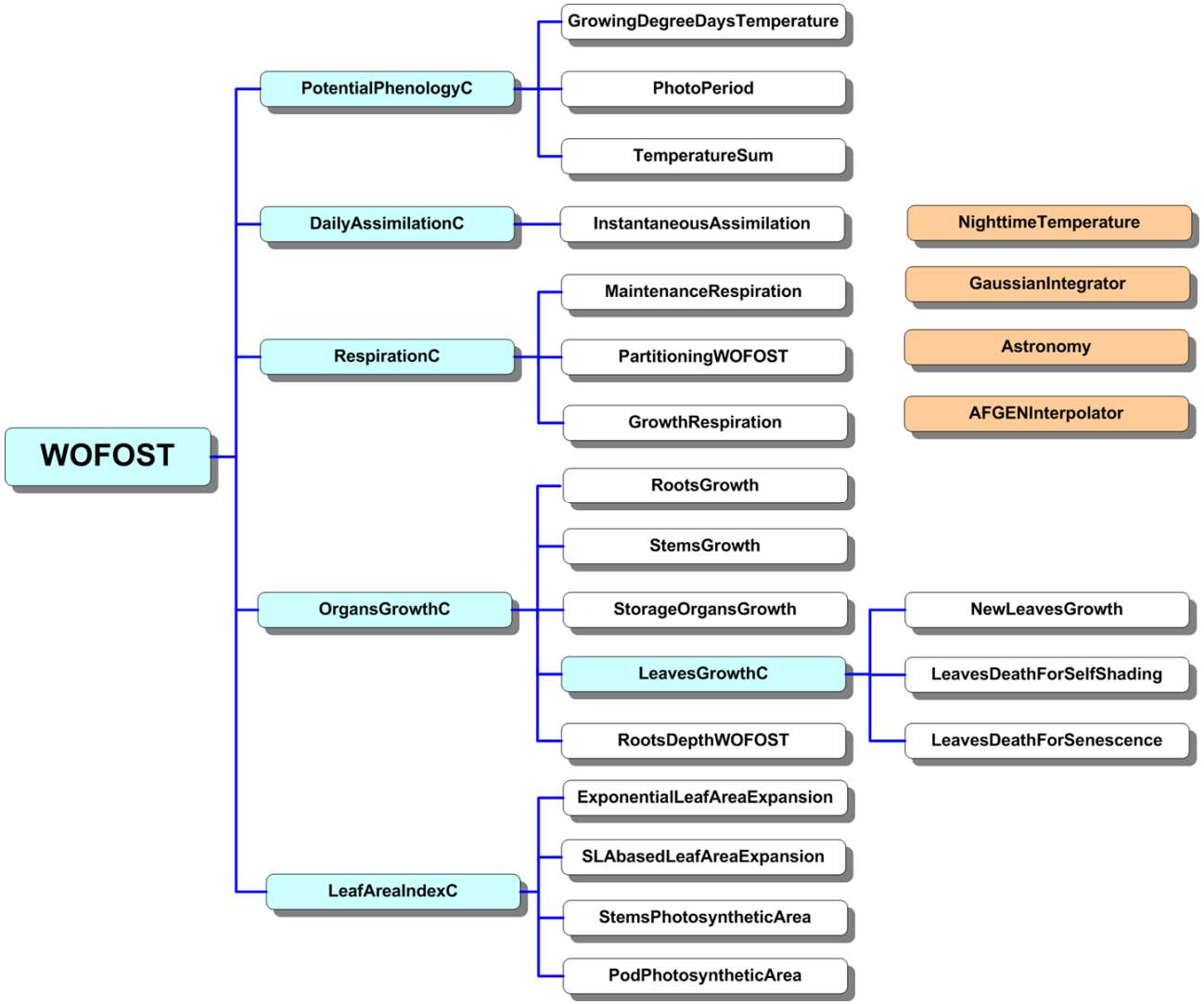
Types of model units implementation

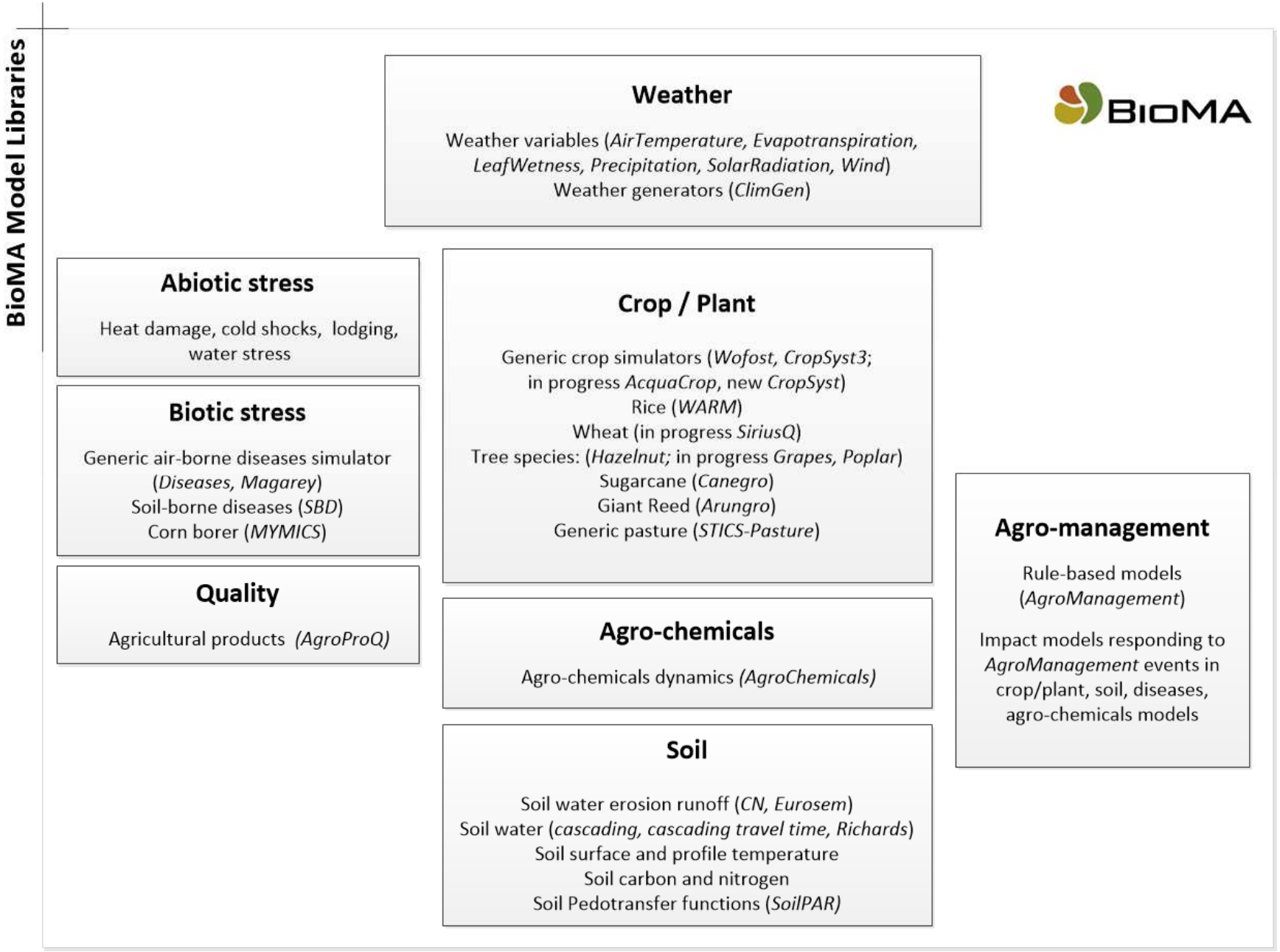


Model components

- A component is built including model units.
- Model components built with the BioMA architecture:
  - Are reusable across frameworks;
  - Have a semantically explicit interface;
  - Can be extended both for data and models;
  - Include the definition of their own parameters;
  - Allow running pre- and post-conditions;
  - Have a scalable logging.
- They are a way to share knowledge while providing operational software units, to be used alone or via composition.

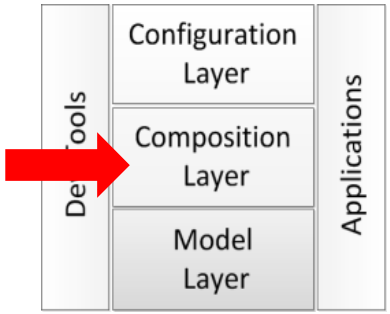
Crop model units: the example of Wofost




BioMA Model Libraries
Weather
Weather variables (AirTemperature, Evapotranspiration, LeafWetness, Precipitation, SolarRadiation, Wind)
Weather generators (ClimGen)
BioMA
Abiotic stress
Heat damage, cold shocks, lodging, water stress
Biotic stress
Generic air-borne diseases simulator (Diseases, Magarey)
Soil-borne diseases (SBD)
Corn borer (MYMICS)
Quality
Agricultural products (AgroProQ)
Crop / Plant
Generic crop simulators (Wofost, CropSyst3; in progress AcquaCrop, new CropSyst)
Rice (WARM)
Wheat (in progress SiriusQ)
Tree species: (Hazelnut; in progress Grapes, Poplar)
Sugarcane (Canegro)
Giant Reed (Arungro)
Generic pasture (STICS-Pasture)
Agro-chemicals
Agro-chemicals dynamics (AgroChemicals)
Soil
Soil water erosion runoff (CN, Eurosem)
Soil water (cascading, cascading travel time, Richards)
Soil surface and profile temperature
Soil carbon and nitrogen
Soil Pedotransfer functions (SoilPAR)
Agro-management
Rule-based models (AgroManagement)
Impact models responding to AgroManagement events in crop/plant, soil, diseases, agro-chemicals models

modelling solutions from model components

THE COMPOSITION LAYER



The Composition Layer

- The composition layer must include:
  - **Time handling**, hence allowing for calls to models at the time step chosen for communication across components in the modeling solution;
  - **Provide events handling,** in this case we refer to actions which are triggered not at all time steps.
- The composition layer may include:
  - Integration services;
  - Data services (in principle excluding persistence, which is part of the configuration, hence context specific);
  - Visual tools can be developed to assist creating code units to be compiled and used by applications.

crea

Requirements

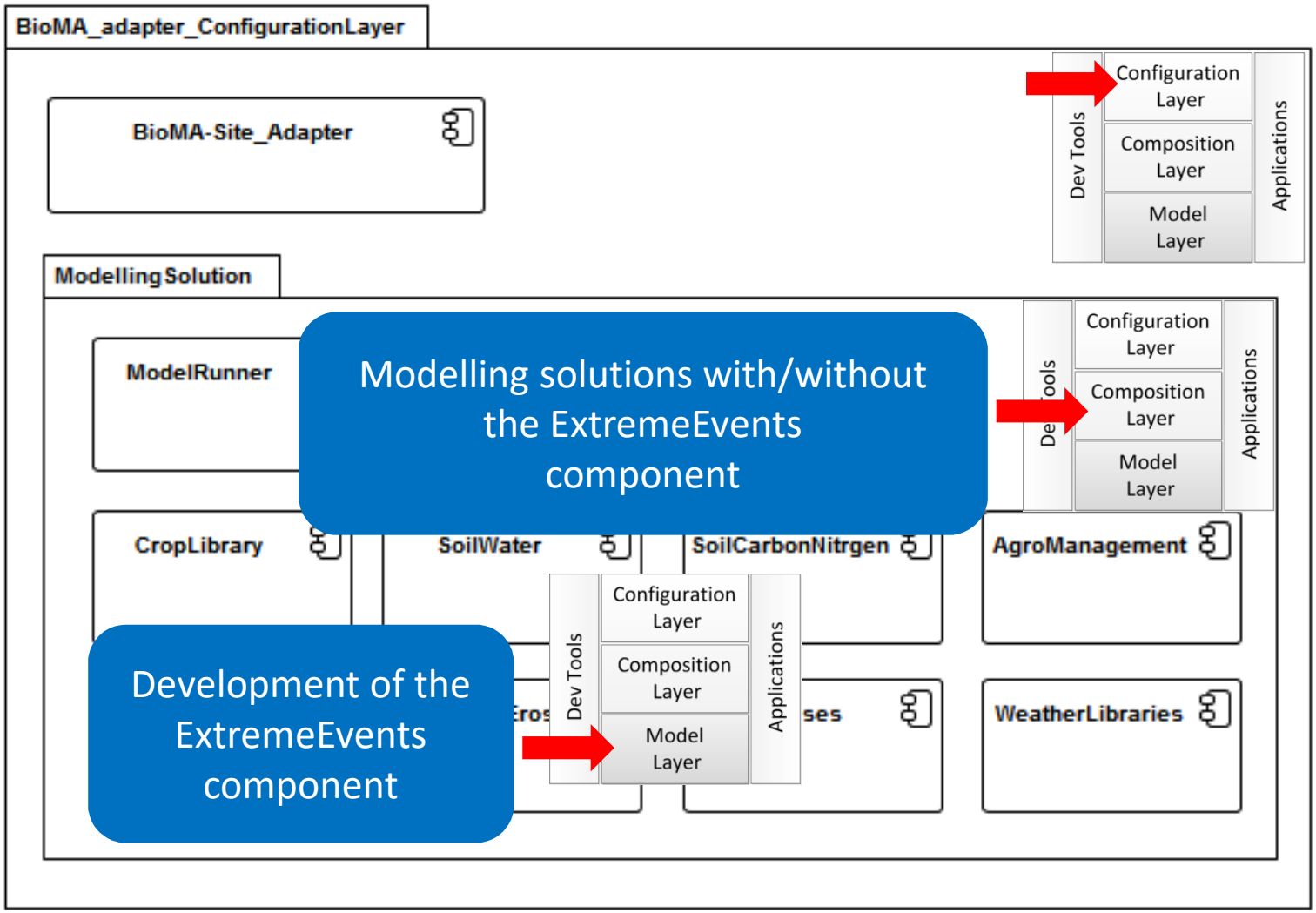
- Allow re-use of components data-types;
- Allow transfer of modeling/run options to/from the higher level (Configuration level, Application);
- Require simple implementation of adapters of components to an instance of the layer;
- Allow units conversion;
- Implement an initialization and finalization method;
- Have its own scalable logging;
- Allow discovering (via reflection)
  - links between components, and on the quantities involved;
  - the components used;
  - inputs, outputs, and parameters of modeling solutions;
  - modeling options made available as part of specific modeling solutions.

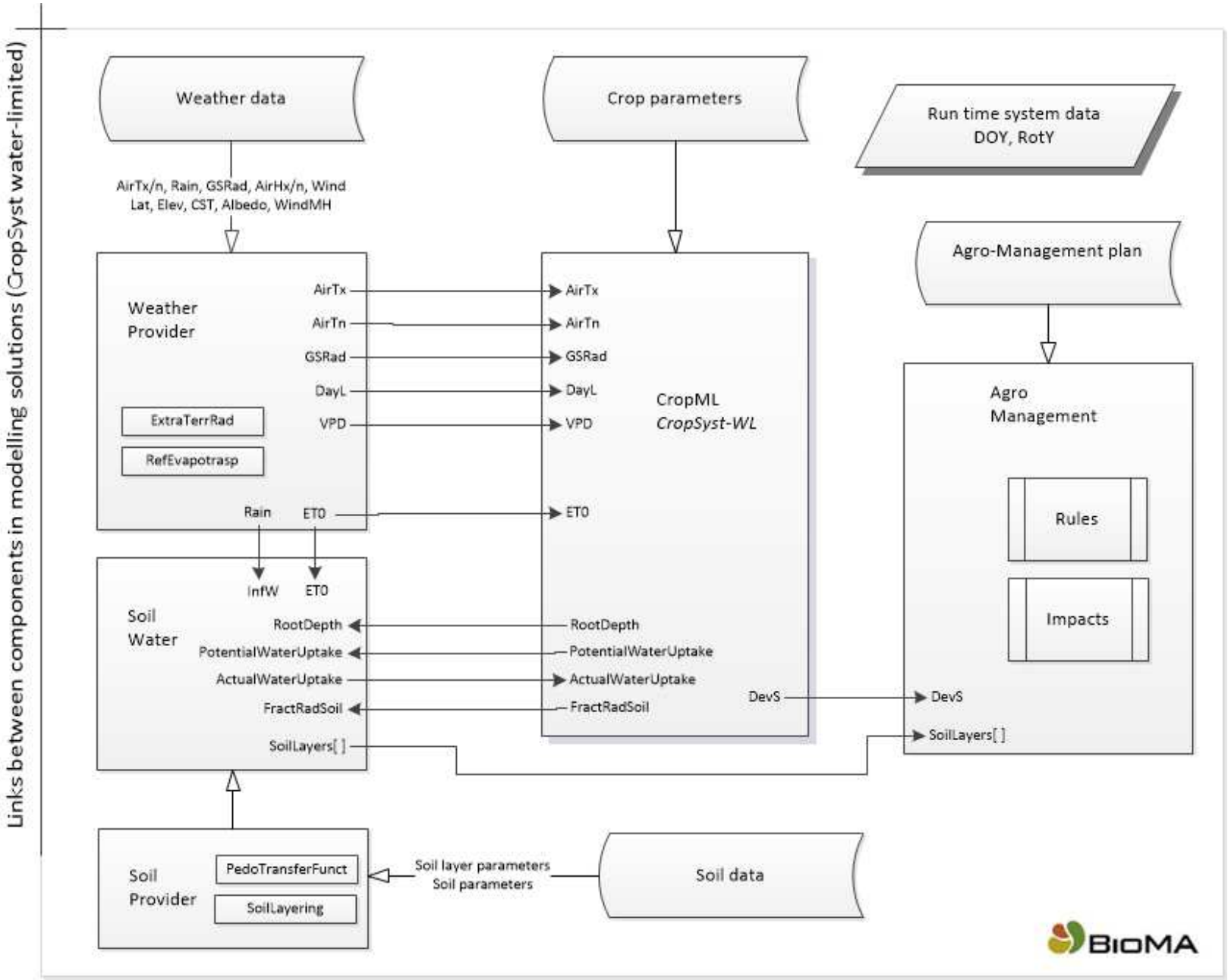
crea

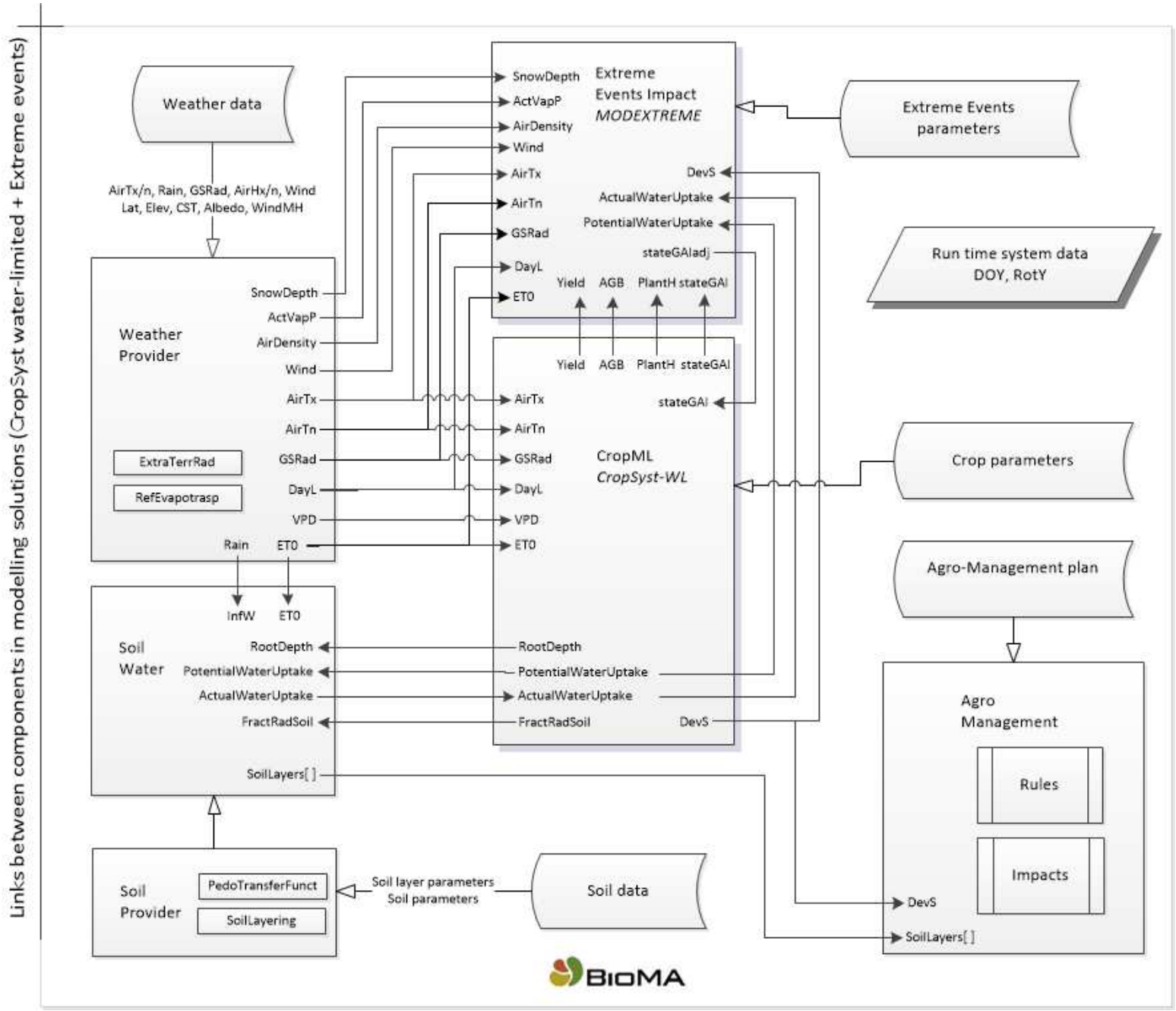
Modelling solutions

- Modelling solutions implemented according to the requirements of the Composition Layer are **framework-independent**;
- They expose information and functionalities that maximize the ease to create an adapter to different platforms, including (but not exclusively) BioMA (e.g. compatible with APSIM);
- They allow creating automatically a large part of documentation;
- Their code can be built almost entirely via code generators.

Adding a component for extreme events impact

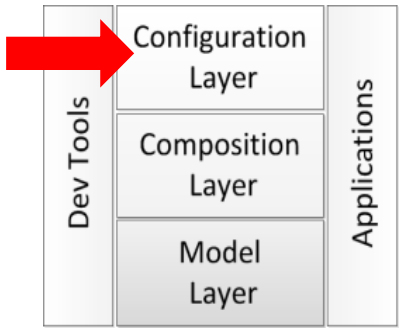



Links between components in modelling solutions (CropSyst water-limited)
Weather data
Crop parameters
Run time system data
DOY, RotY
AirTx/n, Rain, GSRad, AirHx/n, Wind
Lat, Elev, CST, Albedo, WindMH
Agro-Management plan
Weather
Provider
AirTx
AirTn
GSRad
DayL
VPD
ExtraTerrRad
RefEvapotrasp
Rain
ET0
CropML
CropSyst-WL
Agro
Management
Rules
Impacts
InfW
ET0
Soil
Water
RootDepth
PotentialWaterUptake
ActualWaterUptake
FractRadSoil
SoilLayers[ ]
DevS
Soil
Provider
PedoTransferFunct
SoilLayering
Soil layer parameters
Soil parameters
Soil data
BioMA


Links between components in modelling solutions (CropSyst water-limited + Extreme events)
Weather data
AirTx/n, Rain, GSRad, AirHx/n, Wind
Lat, Elev, CST, Albedo, WindMH
Weather Provider
ExtraTerrRad
RefEvapotrasp
SnowDepth
ActVapP
AirDensity
Wind
AirTx
AirTn
GSRad
DayL
VPD
Rain
ET0
InfW
ET0
Soil Water
RootDepth
PotentialWaterUptake
ActualWaterUptake
FractRadSoil
SoilLayers[ ]
Soil Provider
PedoTransferFunct
SoilLayering
Soil layer parameters
Soil parameters
Soil data
Extreme Events Impact
MODEXTREME
SnowDepth
ActVapP
AirDensity
Wind
AirTx
AirTn
GSRad
DayL
ET0
DevS
ActualWaterUptake
PotentialWaterUptake
stateGAIadj
Yield
AGB
PlantH
stateGAI
Extreme Events parameters
Run time system data
DOY, RotY
CropML
CropSyst-WL
Crop parameters
Agro-Management plan
Agro Management
Rules
Impacts
DevS
SoilLayers[ ]
BioMA

Interfacing modelling solutions to graphical user interfaces

THE CONFIGURATION LAYER



The Configuration Layer

- The data to run a modelling solution can originate from various deployment environments, for example using a database, xml files, or remote web services. Data persistence can also be diversified.
- A specific view on data given by an application requires specific information to allow user interaction according to the use cases needed.
- All these ways of providing a modeling solution and a GUI with needed data are abstracted in the concept of a configuration; this concept is addressed in the Configuration Layer.
- Also, the configuration layer must expose handles to run a modeling solution iteratively, as it is requested for instance in sensitivity analysis or when running an optimization.

Requirements

- Allow providing values for items constituting the configuration.
- Verify items validity with respect to the environment of execution.
- Save a configuration for later reloading.
- Create recursive configuration structures, in case one of the items constituting the configuration needs in turn to be configured. An implication of this requirement is that not only a modeling solution must own a configuration, but also the non-trivial components constituting the configuration itself.
- Support callback functions when the status of a configuration changes, to refresh views attached in a Model View Controller architecture.

View - controllers

- Different couples view-controller can be developed based on the Configuration Layer, hence allowing the development of different applications targeted to specific uses.
- The BioMA case, as an example, uses a controller to handle two user interfaces («Spatial» and «Site»).
- An adapter for a modelling solution to an instance of the Configuration may or may not origin from the Composition Layer; legacy code can be wrapped to represent a benchmark for testing the reimplementation of specific modelling solutions.

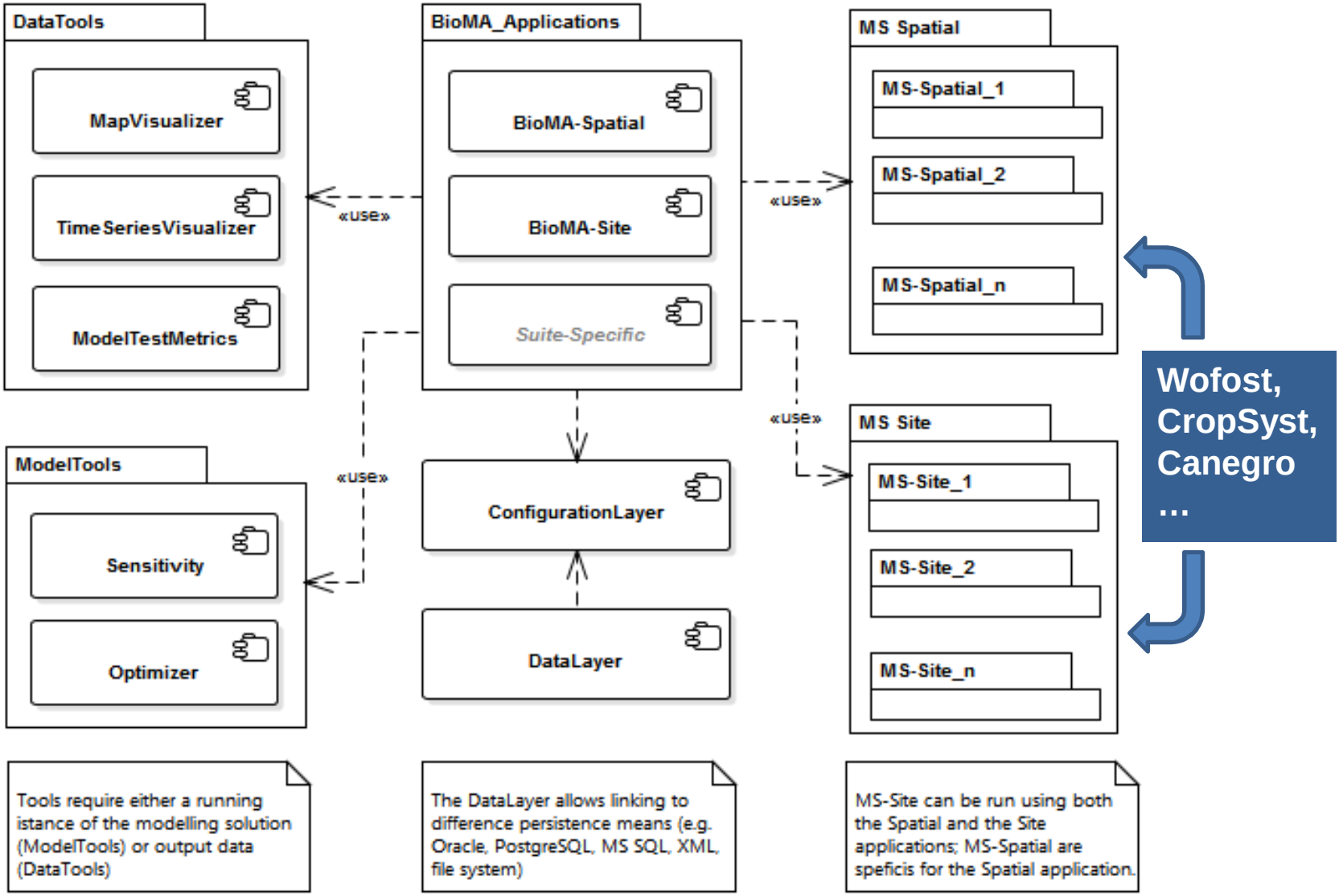
Outline

- Modelling framework
- Model reuse
- Imperative vs. declarative
- From models to viewvers
- BioMA applications
- Conclusions

BioMA applications

- BioMA applications have been used for different research projects, many under climate change scenarios (https://en.wikipedia.org/wiki/BioMA):
  - weather datasets for biophysical simulation
  - impact on crop production and adaptation in Europe
  - soil pathogens
  - plant pathogen impact on production
  - corn borers
  - modelling solutions comparison at sub-model level
  - impact on crop production in Latin America
  - fungal infections
  - functions to estimate soil hydraulic properties
  - quality of agricultural products

BioMA Deployments



The importance of the IPR model

- A modelling framework which explicitly targets model and software re-use implies:
  - Multi-actors work within a team/institution
  - Multi-team work across institutions
- When concepts as reuse and model sharing are considered, there is concern on IPR and also on “losing the identity” built over years and communities, on specific realizations of modelling systems
- A clear intellectual property rights model is consequently needed

crea

The IPR model

- Working with a model framework requires substantial resources, hence a medium-term perspective;
- No institution will do it on a code base of core components which are owned by someone else and which have code not accessible;
- BioMA is adopting a **MIT version with open source access to core components in GitHub**;
- A consortium is being developed to steer the branch of the official version of components, and also to share the code of development tools.

Conclusions

- A modelling system based on model implemented at fine granularity maximizes both the ease of testing alternate modelling approaches and the capability of extending to other processess or modelling domains.
- Targeting reuse requires matching specific requirements, also to provide functionalities which go beyond the mere output→input communication between modules.
- Working with a modular, transparent system is not only a technical mean to improve efficiency; it directly impacts on improving analytical capabilities on system performance.

crea

Conclusions (2)

- BioMA neither is a simulation model nor proposes a specific model; instead, it is an open platform to make available in operational software the results of research on biophysical modeling in agriculture.
- We make available BioMA as a platform, but also, and of no lesser importance, as a loose collection of model objects and software tools reusable in other modelling frameworks.

crea

Thank you

BioMA https://en.wikipedia.org/wiki/BioMA

BioMA Modelling
@BioMAFramework