Annual Workshop, University of Florida
February 27, 2015, Gainsville, USA

Enhancing Model Reuse via Component-Centered Modeling Frameworks: the BioMA Platform

Marcello Donatelli[1], Davide Fanchini[2], Davide Fumagalli[3],
Iacopo Cerrani[3], Simone Bregaglio[4], Antonio Zucchini[3], Luca Erba[1],
Ioannis Athanasiadis[5], Andrea Rizzoli[6]

1 CRA - Council for Agricultural Research and Economics, Bologna, Italy
2 Softeco Sismat IT, Genova, Italy
3 JRC - Joint Research Centre, European Commission, Ispra, Italy
4 UNIMI - University of Milan, Milano, Italy
5 DUTH - Democritus University of Trace, Xanthi, Greece
6 IDSIA - Dalle Molle Institute for Artificial Intelligence, Manno-Lugano, Switzerland

Introduction – The approach

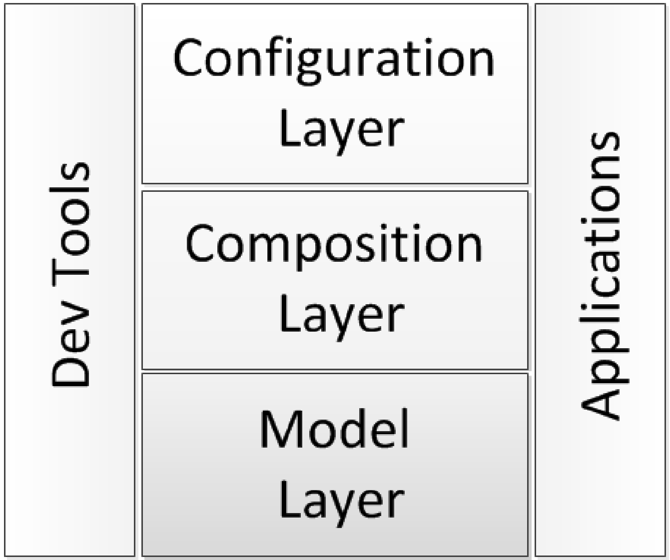
- In addition to the features known for the modelling frameworks available, some new high level requirements emerged:
  - To increase the transparency of the modelling solutions compared to legacy code available, for each of the modelling solutions being built;
  - To increase the traceability of performance of each modelling unit used in modelling solutions;
  - To involve teams without requiring them to commit to a whole infrastructure they would not own and possibly would not use.

To maximize both reusability and openness, we chose to develop a simulation system based on framework-independent components, both for model and for tool components.

The high level architecture of the software modelling environment

FROM MODELS TO VIEWERS

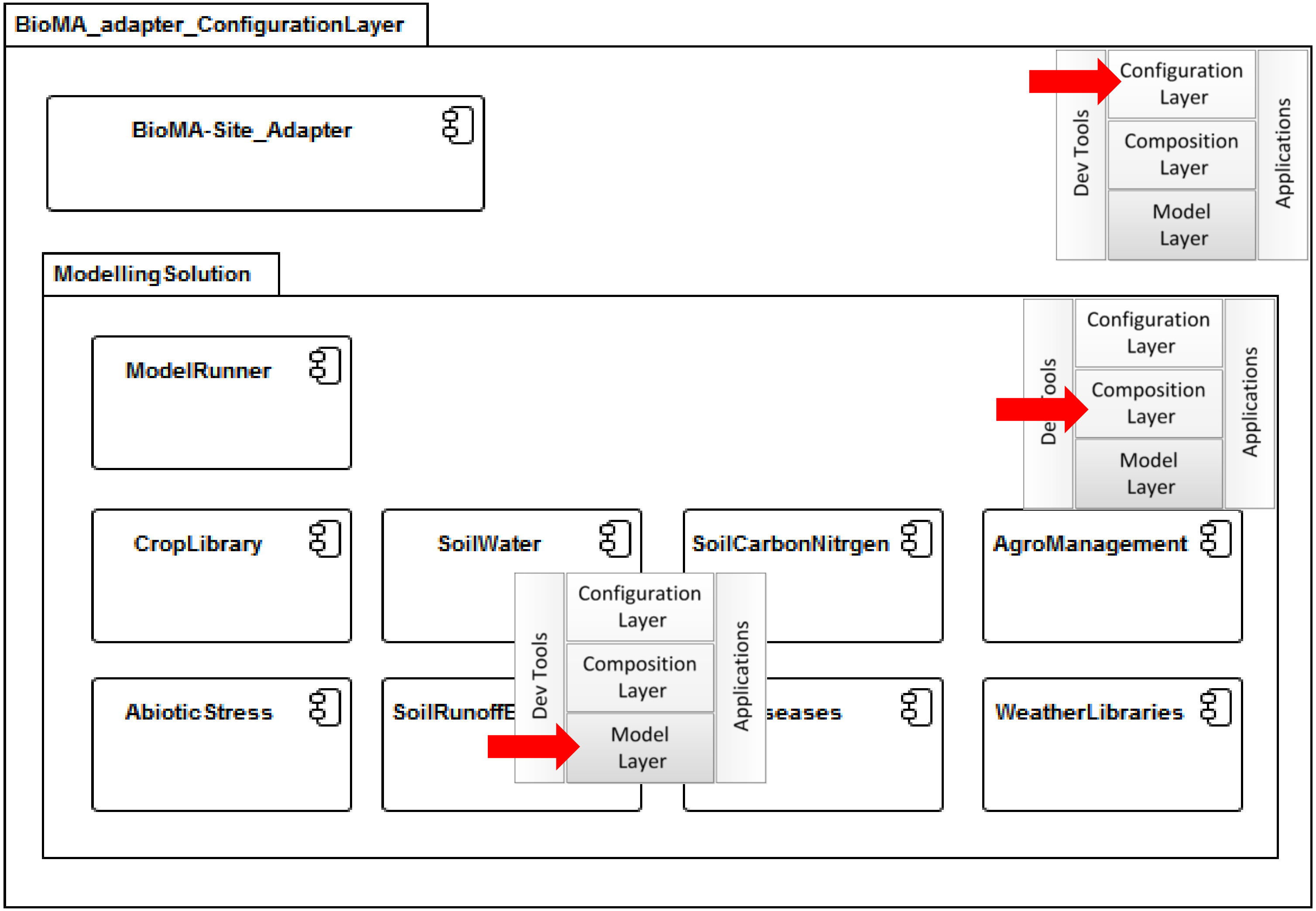
From models to viewers



- **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

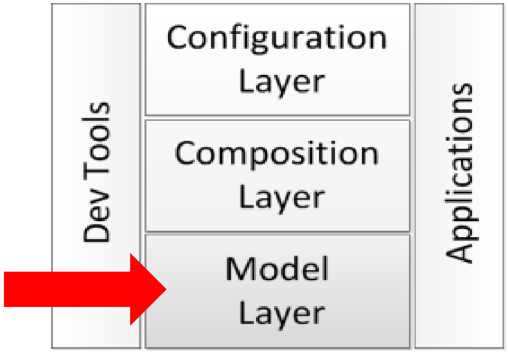
BioMA CRA

From models to applications

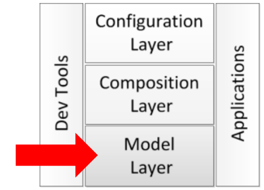


Implementing model units

THE MODEL LAYER



Working at the Model Layer level



- The first step is the implementation of «single» models – ideally including a single process, as stateless discrete units

Inputs

- **States**
- **Exogenous**
- **Auxiliary**
- **Management**
- …

(from domain description)

model algorithms

parameters (model specific)

Outputs

- **Rates**
- **States**

(from domain description)

Domain classes

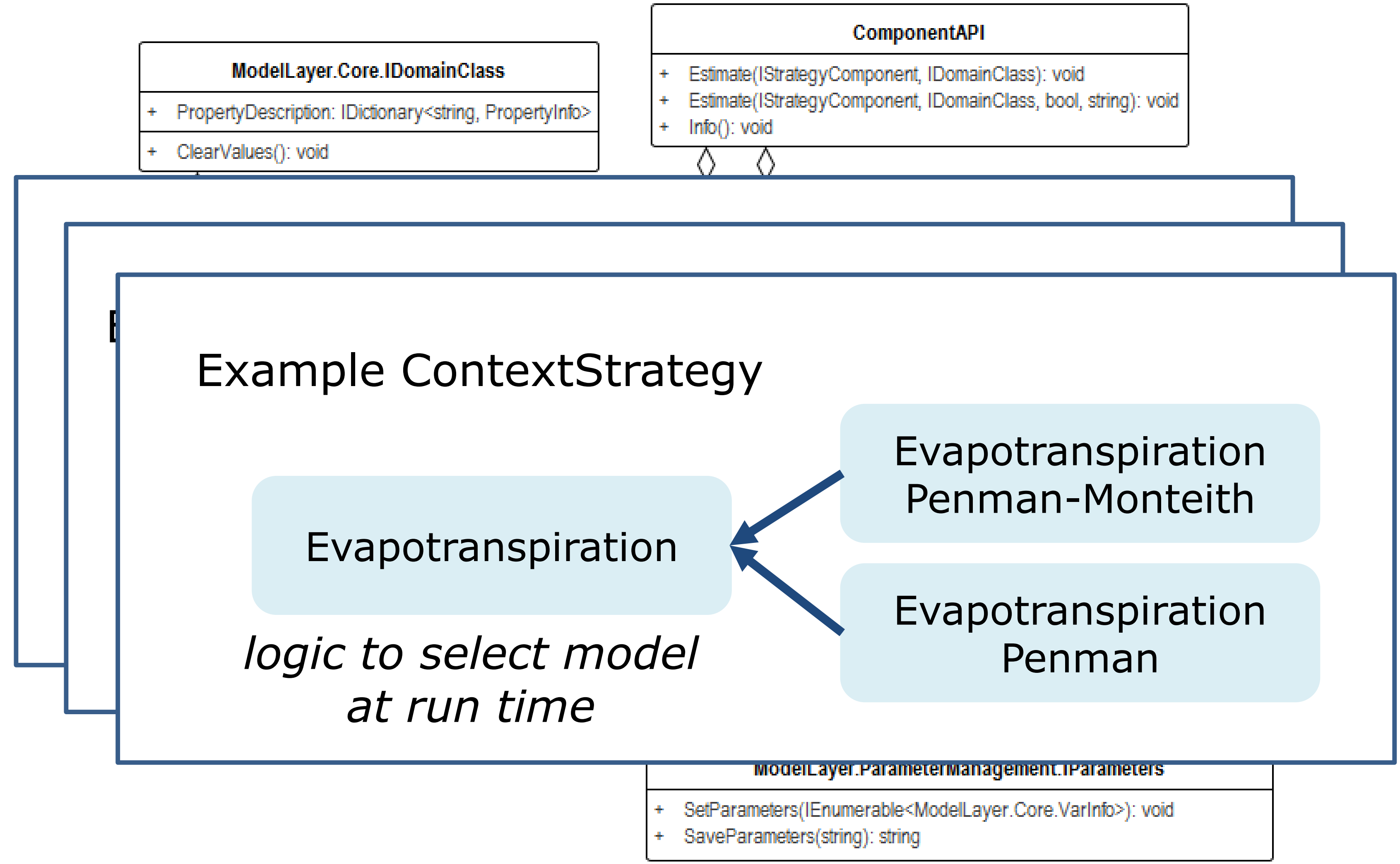
- Domain classes contain the definition of the domain being modelled
- The quantities are placed in domain classes which are homogeneous for content: states, rates, exogenous etc.
- The domain classes are input/output objects for model classes
- Each quantity is defined as name, definition, units and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

The Model layer

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) integration.

Model component architecture



Model components

- Model components built with this architecture:
  - Are framework-independent;
  - 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.

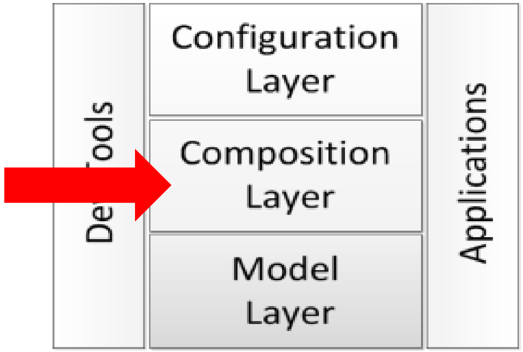
- They are a way to share knowledge while providing operational software units, to be used alone or via composition.

Development tools

- Applications (DevTools) are provided to minimize code writing by modellrs:
  - Explore model component interfaces and data-types
  - Generate the code of data types (domain classes)
  - Generate the code of model classes (strategies)
  - Generate the code of component API
  - Generate the code of modelling solution by composition of models from components

Composing models to build modelling solutions

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 (the time step chosen for communication is not necessarily the time step of the modeling approaches used);
  - 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.

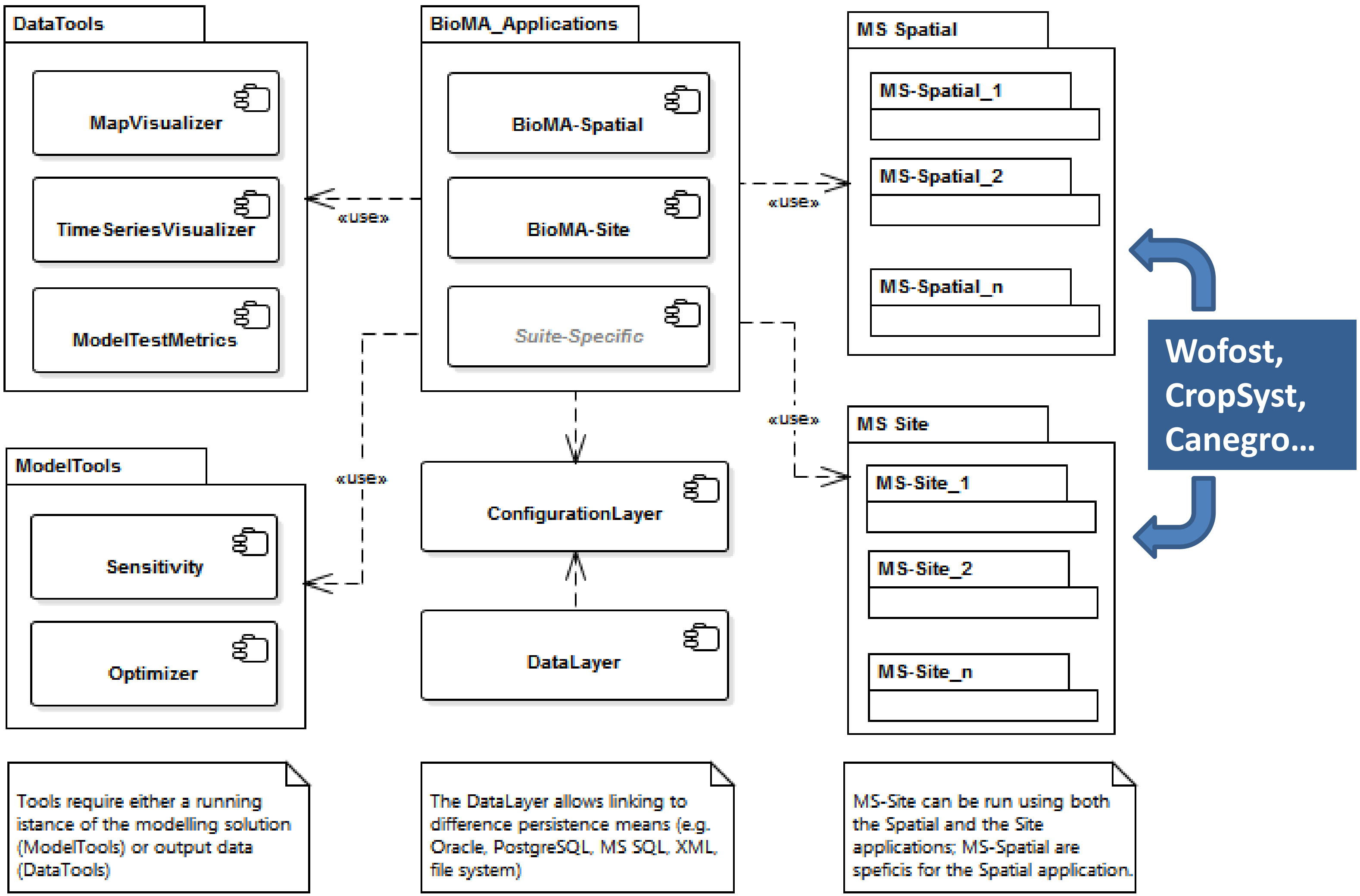
Modelling solutions

- These modelling solutions are **framework-independent**;
- They expose information and functionalities that maximizes the ease to create an adapter to different platforms, including (but not exclusively) BioMA;
- They allow creating automatically a large part of documentation;
- Their code can be built almost entirely via code generators.

An example realization of the four software layers

THE BioMA PLATFORM

Applications, Tools and Modelling Solutions



Model libraries available

Weather libraries

AirTemperature, EvapoTranspiration, LeafWetness,
SolarRadiation, Rainfall, Wind
Climatic indices
Weather Generators (ClimGen, CLIMAK)

Abiotic stresses

Heat damage, Rice cold shocks, Lodging

Biotic stresses

Generic air-borne diseases
simulator (Diseases, Magarey),
Generic soil-borne diseases growth (SBD),
CornBorer simulator (MYMICS)

Chemicals

Chemicals dynamics (AgroChemicals)

Plant libraries

Generic crop simulators (Wofost, CropSyst, STICS)
Generic tree simulator (Tree)
Rice (WARM)
Sugarcane (CaneGro)
Grain quality (AgPro-Q)

Soil libraries

Soil water runoff and erosion (CN, Eurosem),
Soil water redistribution (Cascading, FiniteDifferences)
Soil surface and profile temperature,
Soil nitrogen (SoilN)
Pedotransfer functions (SoilPAR)

Agricultural management

Rule-based modelling
(AgroManagement)

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 procesess or modelling domains.

- **Targeting reuse** requires matching specific requirements, also to provide functionalities which go beyond the mere output→input communication between modules.

- The architecture of BioMA is based on such requirements, facilitating development and maintenance.

- Applications developed on the base of the BioMA framework are increasingly capable to address many aspects related to the simulation of biophysics of agricultural production.

Conclusions (2)

- BioMA **neither is a simulation model nor proposes a model**; instead, it is an open platform to make available in operational software the results of research on biophysical modeling in agriculture;
- Adopting a component oriented development, extended both to models and tools, fosters reusability **without forcing third parties to invest on a specific framework they do not own** and possibly would not fully use;
- 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.**

modextreme
agriculture facing extreme climatic events

Credits

BioMA Scientific Leader
Marcello Donatelli

Biophysical Modellers
Marcello Donatelli, Roberto Confalonieri, Simone Bregaglio, Gianni Bellocchi, Giovanni Cappelli, Laura Carlini, Efthymia Chatzidaki, Gianni Fila, Caterina Francone, Andrea Maiorano, Tommaso Stella, Marco Acutis

Software Engineers
Davide Fumagalli, Davide Fanchini, Iacopo Cerrani, Antonio Zucchini, Luca Erba, Ioannis Athanasiadis, Andrea Rizzoli

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

BioMA Modelling
@BioMAFramework