Scope

The goals of this whitepaper are: 1) to present key concepts of multiscale biological modeling. 2) To present a language hierarchy and tool framework for (1) that clarifies the work-flow for model and simulation development. 3) To describe how (2) supports model sharing, integration and model transport among modeling and simulation methodologies. 4) To identify key enabling components missing from current model representations, especially with reference to multiscale, multi-cell models and simulations. 4) To enable the treatment of repositories of models and results as mineable data.

Design Goals

1) The ability to specify models (at any spatial level) in a way that insulates the model description of the underlying biology from any metadata related to the specific **methodology** used to solve the corresponding simulation. *E.g.* a model describing a set of genetic, regulatory or metabolic pathways should be solvable using ODEs, Gillespie or other methodologies; a model of tissue development should be solvable using GGH, Finite Element, Center-Model or Vertex Model methodologies.

2) The ability to combine multiple models at the same or different spatial scales to create larger models without excessive demands for user input. This ability allows the creation of repositories of reusable components which can be modified and extended. *E.g.* the assembly of a signaling pathways model with a cell-cycle regulation pathway model; the assembly of a cell-cycle pathway model with a model of tumor growth at the multi-cell scale and a tissue-scale model of nutrient supply.

3) Mash-up tools to assist with model integration.

4) The ability to package any combination of models into an encapsulated model, *e.g.* the definition of a set of pathways models and cell behavior models into a model of a cell type; or the assembly of multiple cell types into a model of a tissue or organ.

5) The ability to replace model or submodel parameters without changing the structure of a model, *e.g.* replacing the parameters in a human hepatocyte cell model with those appropriate for a chicken to create a chicken hepatocyte model.

6) The ability to format both simulation output and experimental data so that it can provide model initial conditions and parameters.

7) The ability to annotate both simulation output and experimental data so they can be compared quantitatively and qualitatively.

8) The ability to identify and specify missing model parameters.

9) The ability to identify and specify missing methodology-specific metadata.

10) Annotation to allow model and data archiving, searching, mining and versioning.

Aside: To effectively use model repositories as databases may require that both successful and unsuccessful models are included in the database. An unsuccessful model may contain information (or insight), though probably less information than a successful model. Nonetheless, failed models may provide useful insight into what works and what does not. Scientists rarely publish "failed experiments"; perhaps due to the cost and effort of publication or because of the difficulty of getting negative results published. In the case of a simulation environment "publication" might simply be electronic archiving of the model, scripts and results.

Additional Definitions

We begin with a few definitions:

People often use the terms "model" and "simulation" interchangeably. We will refer to specific code executable by a **simulation environment** and its internal representation during execution as **simulations** and all other descriptions of biological phenomena as **models**.

Ontologies are lists of terms, their definitions and relationships. They include both **non-instantiable** categorical ontologies (like FMA and GO) and **instantiable** ontologies that we can use to describe a model or part of a model (like FMA or a markup language).

Languages include both mark-up type descriptions (which have a lot in common with ontologies as noted above) and the "native" Languages of particular simulation/computational environments. In this document, a Language does not refer to a particular programing language, such as C++ or Python, but instead refers to a suite of computational objects that represent the concepts to be modeled or the capabilities of a particular simulation environment.

Tools are programs or subprograms that allow manipulation of models and language components and/or instantiate their concepts. Tools should align, at some level, with appropriate Ontologies.

Use Cases are "proof of concept" models. They describe the domain to which a set of Languages and Tools apply. Use Cases specify the types of questions the models or simulations can answer.

A **Toolkit** is the ensemble of ontologies, languages and tools relevant to a modeling and simulation domain.

Multicell models and simulations span the size domain from single cells to tissues. A Multicell model can have varying levels of sub-cellular detail; the exact level of detail depends on the behaviors that the individual cells (or set of cells) must exhibit.

Cell clumps refer to fragments of tissues, like a liver lobule, that can be modeled as a biologically logical unit.

General Structure of Language Standards, Tools and Toolkits

To be useful, a language standard requires a Toolkit with at least three components, the language standard itself, a set of tools for writing, editing, viewing and translating the language, and a set of use-cases to illustrate the language application (Figure 1). If the language is to be logically consistent, and be searchable for model-as-data applications, it also requires a reference ontology. The reference ontology must cover the important biological concepts needed in the use-cases.

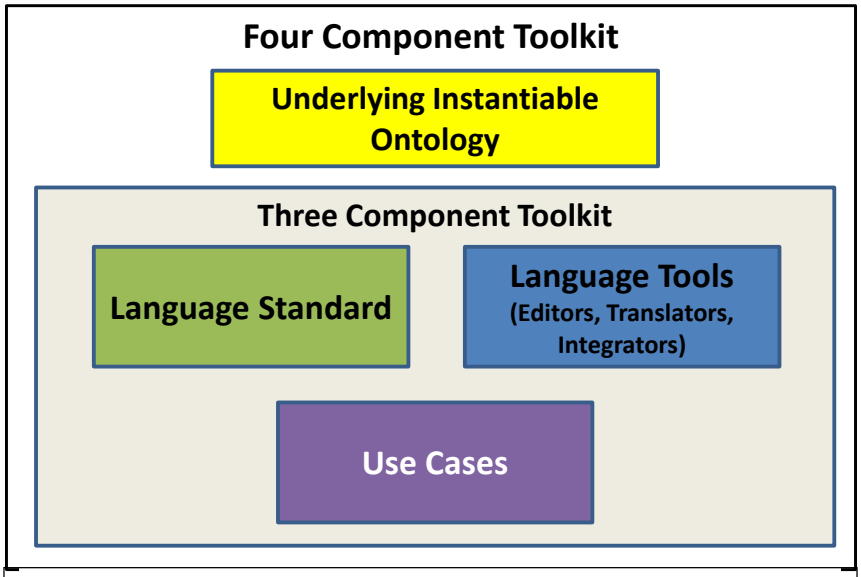

Figure 1: The elements of a three and four-component language standard toolkit.

Needs

Figure 2 adapts the Physiome framework to indicate scales missing from the initial Physiome concept and their corresponding modeling languages. Because of its history, Physiome lacks a concept of a *cell* as a spatially extended motile (active) agent. Despite its name, CellML actually describes molecular-level chemical and electrical objects and their behaviors. ModelML is a multiscale integration language and FML is a language for the representation of spatial configurations and partial differential equations (**PDE**s). Both were developed at UNSW. CBML and CBO are respectively, a proposed interchange language and ontology to represent the missing “cells as agents” scale.

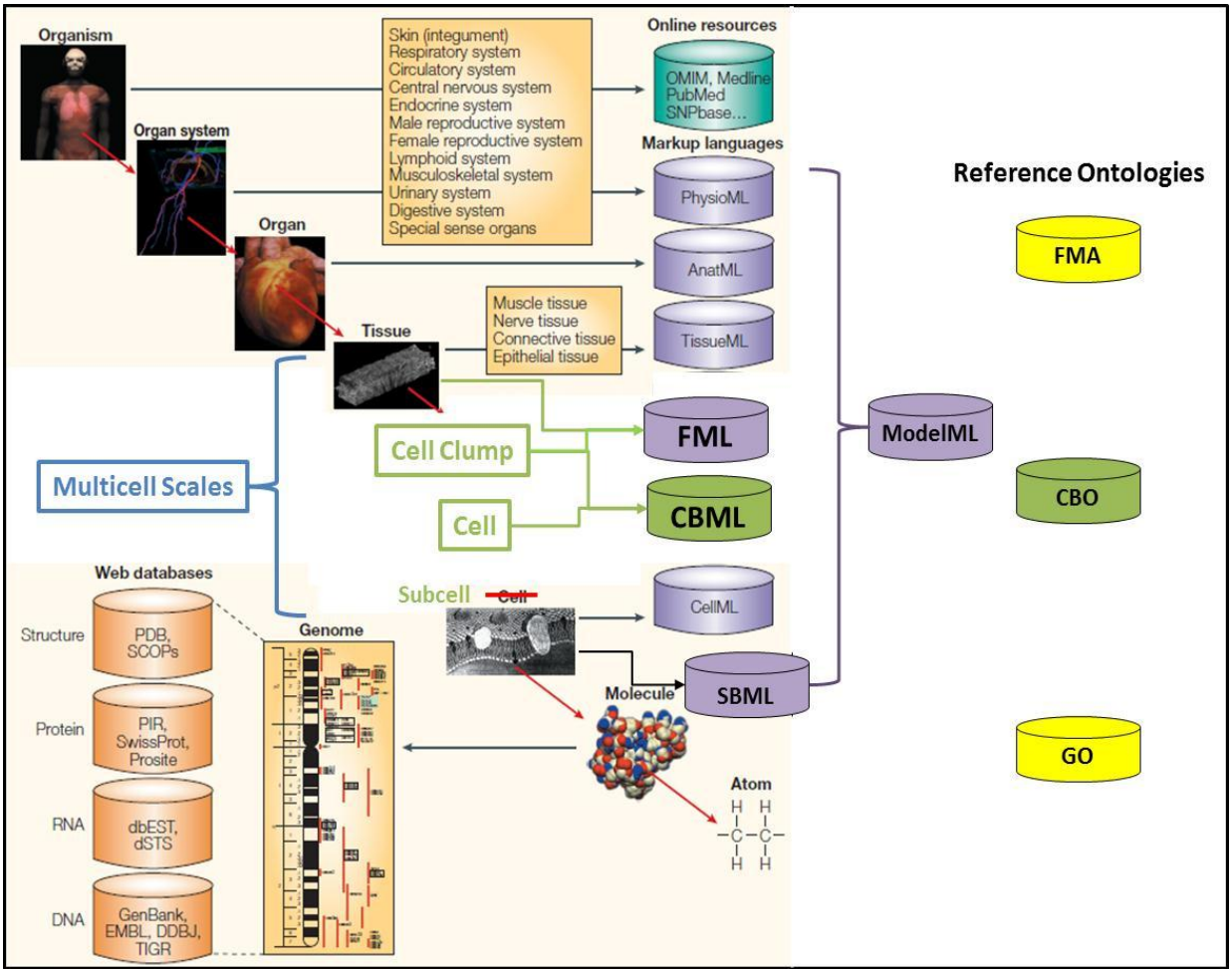

Figure 2. After P. Hunter & T. Borg, "Innovation: Integration from proteins to organs: the Physiome Project", *Nature Reviews Molecular Cell Biology* **4**, 237-243 (2003). Elements added to show the missing scales are shown in **green and blue**.

The Process of Modeling, Levels of Model Description and Model Description Languages

Models and simulations consist of at least four elements; **objects**, **object properties / interactions**, **initial conditions** and **dynamics / processes** (Figure 3). The execution of a simulation produces **data**, which may or may not have the same form as elements of the model or simulation. The results of a simulation might be a spatial structure which could also serve as an initial condition for another simulation, a kinetic constant which might be a parameter in an

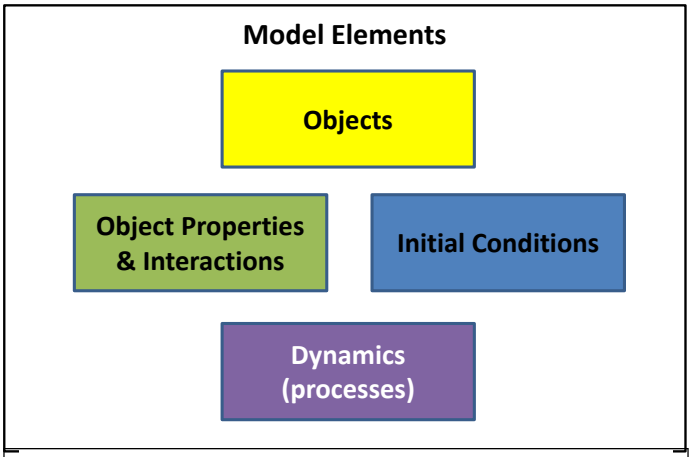

Figure 3: Model Components.

object property, or a new object, all of which might be used as input to a new model.

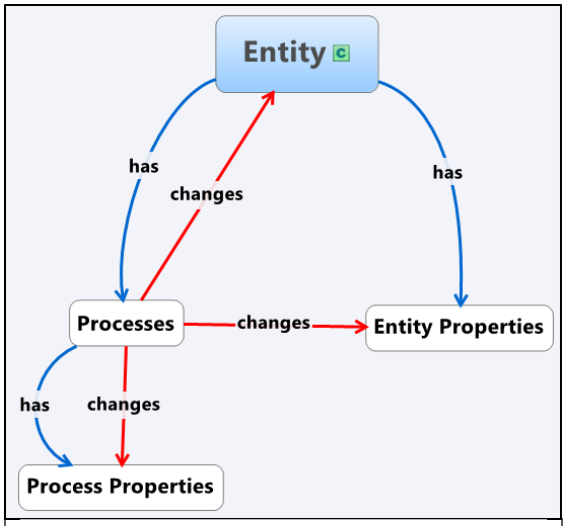

Figure 3b: Ontological Relationships for Model Description.

Ultimately, we require languages to describe all of these elements, as well as meta-languages to describe their interactions. However, the number and nature of these languages are not immediately obvious. We can determine these by analyzing the workflow of developing a simulation and the simulation output.

To develop a simulation of a biological phenomenon, we begin with a qualitative verbal description (a "Biological Model") described using the reference ontologies, and eventually transform it into a script which a simulation environment can execute (Figure 4). We begin with concepts and languages as close as possible to those in common use and at each stage, disambiguate them and add the necessary detail until we arrive at concepts that correspond to those of the simulation environment. Each model level corresponds to one or two language levels and a language links each adjacent pair of modeling levels. Moving downwards in the workflow of Figure 4 requires disambiguation and added detail. Moving upward requires generalization and abstraction. This organization shows clearly the value of languages in mediating model-development workflow.

If we want the workflow to allow the various models, submodels and simulations to be searchable, interoperable, programmable and editable, we must employ at least four levels of language abstraction and three levels of model (Figure 4). At the top, we start with a biological model, as it might be described by a wet-lab biologist, or biology textbook. This level might include genes, proteins, behaviors, cells, tissues and linkages (interactions). Existing ontologies (such as FMA and GO) act as "naming authorities" that link objects and concepts to other resources. The ontologies guarantee the logical integrity of the other layers and map to important biological concepts, or at a minimum, to the concepts important for the particular process being modeled.

To produce a well-defined instantiation of the biological model, we must generate a mathematic model which provides quantitative meaning to the biological concepts. Many simulation environments jump directly from the Biological Model to the Simulation (Computational Model), but doing so mixes specific simulation methodologies with the biological and mathematical concepts of the model, destroying portability and obscuring important details and design choices. To maintain portability, we **must** separate the process of mathematical disambiguation from the translation of the mathematical description into a script for a specific computational methodology. To allow the Mathematical Models to be parsed and searched, we represent them as human-readable interchange languages like CellML and SBML. However, to manipulate, integrate and translate models requires that we have an intermediate representation friendly to analysis tools. These intermediate languages consists of the APIs to describe the concepts in the Interchange Languages, metadata and Executable Scripts, the first two of which

must be standardized and the third of which is methodology specific, and the internal representation of the models used in the translation and integration tools, usually a tree. While it can be helpful for all tools to use the same tree definitions, it is not essential as long as they conform to the APIs.

The third level converts the mathematical model into an executable simulation script. This conversion sets the model's granularity (in space and time), how continuous functions in the mathematical model will be converted into computable functions, how boundaries and systemic quantities will be handled, and the various computational methodologies and solvers that will be employed.

Simulation Output

If the output data of the simulation are to be sharable and interpretable, they must be represented (or at least representable) using one or more interchange languages (which may not be pure Markup languages, which are verbose for very large data sets). The use of interchange languages to represent data allows simulation results to be compared to experiments, other simulations and to feed back to the higher levels of the modeling allowing model validation and refinement.

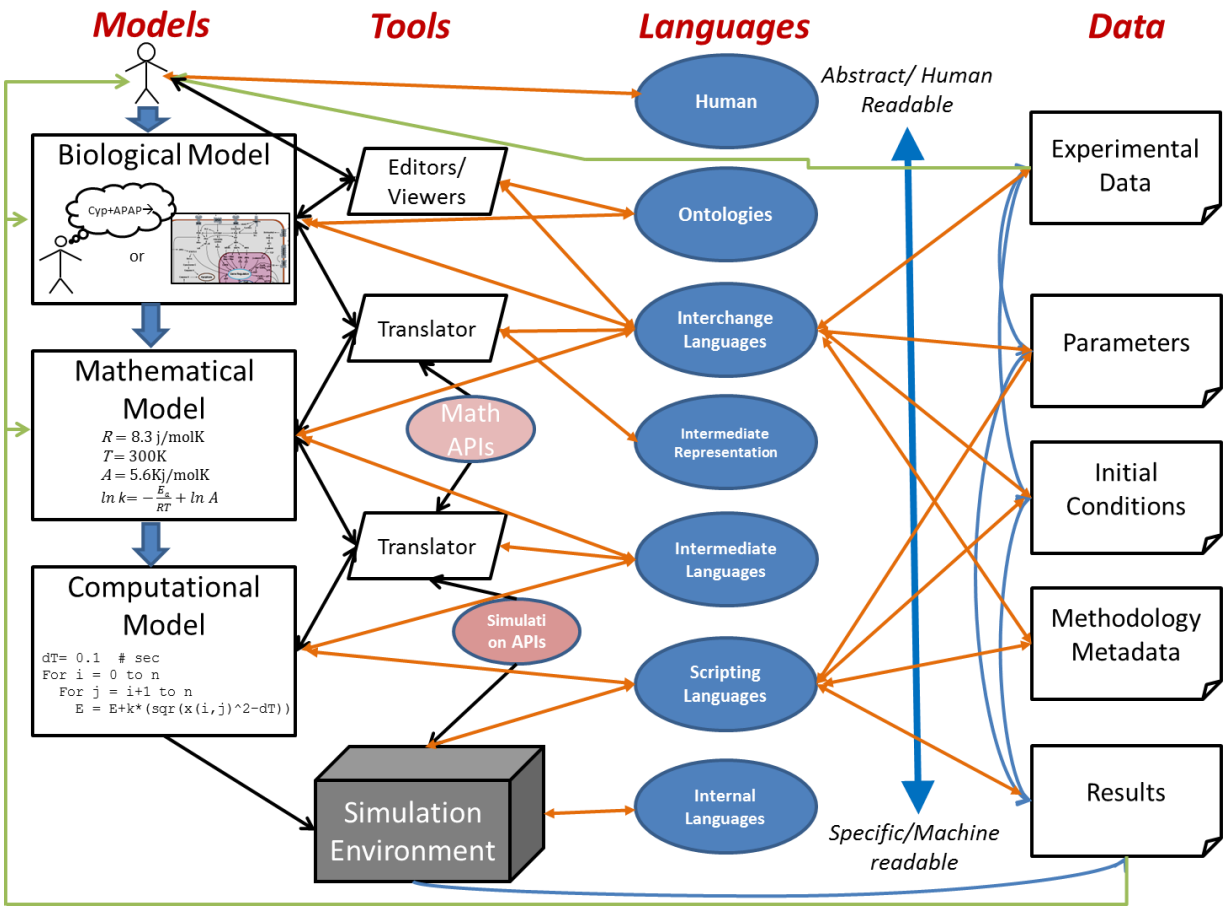

Figure 4: Model to Simulation Workflow. Languages, Modeling Levels and Selected Interrelationships.

Conclusions

References