

BioLabs: A Framework for Simulations of Biological Systems

Tomas Mikula

July 26, 2010

1 Introduction

Explaining complex phenomena in organisms requires understanding biological systems at various scales. We describe BioLabs, a framework for simulations of biological systems that supports more layers of abstraction with emphasis on interoperability of different models and composability of smaller components into complex systems.

Unlike many simulation environments, BioLabs are not built around a single model of biological processes. Instead, they provide a platform on which different models can be implemented. BioLabs' main focus is on models that are computational (meaning the evolution in time is described by an algorithm¹) and composable of simple interacting components (not unlike the way software components are assembled into complex software systems).

¹as opposed to, for example, differential equations. Arguments have been reported why differential equations are unsuitable for modeling complex biological systems.

2 Simulation Basics

This section introduces basic notions of BioLabs' simulation.

The main computational unit of the simulation is a *bio-object*. It represents a (biological) object from the real world, such as an organelle, cell, tissue... A bio-object is (typically) stateful — it changes its state in time. Each bio-object has a method `work(period)` that tells it to update its state as if *period* units of time passed, without any external interaction.

Bio-objects in an experiment are organized in a hierarchy, which corresponds with hierarchical organization of living organisms — organs consist of tissues, which themselves consist of cells... There is one top-level bio-object, called *root*. Every other bio-object in the experiment has exactly one *parent* bio-object (see Fig. 1). Interaction with a bio-object and invocations of its `work()` method are always controlled from its parent. Running the simulation for a *period* of time then comes down to calling the `work(period)` method on the root of the hierarchy. Since the computation within a subtree of a bio-object *x* (i.e. within `x.work()`) is completely indepen-

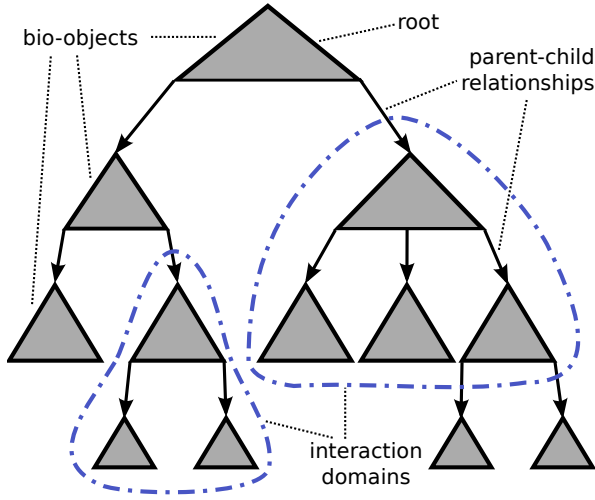


Figure 1: A hierarchy of bio-objects.

dent of anything outside this subtree, the simulation fits well into the divide & conquer paradigm and is therefore very well-suited for parallelization.

Interaction of a set of bio-objects is an action performed on these objects that typically causes a change of their state. An interaction is always initiated from the `work()` method of some bio-object and can directly involve only that bio-object and its children² (illustrated in Fig. 1 as *interaction domain*). Interaction takes no simulation time and children of the initiating bio-object must not be “working” when the interaction is performed. This design strictly separates interaction from bio-object’s internal computation.

Note that interaction with a bio-object cannot in general be performed simply by a method call on this object, because the algorithm that computes the interaction out-

come will typically work on data from all involved bio-objects. We introduce *interaction points* which hold the input and output data of an interaction. The general schema for interaction initiated from bio-object p is then: 1) p gathers interaction points from its children, 2) p applies the interaction algorithm on the data from interaction points, 3) p notifies its children about the result of the interaction.

Natural examples of interactions are signalling or substance exchange between cells, but even cell growth and movement must be modeled as interactions if they require some coordination with cell’s environment. Each of various interaction types has its unique identifier and a defined format of interaction points. Two bio-objects can then communicate over their common subset of supported interactions.

3 BioLabs Platform

In addition to specifying APIs for bio-objects that enable their interaction, BioLabs come with a simulation environment in which experiments are built from bio-objects, executed, visualized and analyzed. Much of this experiment control is done by scripts written in JavaScript.³

3.1 Visualization

There is no one visualization technique that suits all models. BioLabs keep this in mind

²Interaction stretching over more levels of the hierarchy is still possible indirectly, but this is transparent to the initiating bio-object.

³Support for other scripting languages can easily be added, but to keep some uniformity among BioLabs scripts we won’t do so unless the other language proves to be significantly more suitable for BioLabs’ purposes.

and were designed to support multiple visualization methods. For example, an experiment can be visualized spatially (in 2D or 3D), schematically, by a graph displaying evolution of some attributes... When the developer of a bio-object (e.g. a neural cell) wants to provide, say, a spatial 3D visualization for it, she writes a *visualizer* that produces visual representation of her bio-object using some 3D technology.⁴

3.2 Querying and Analysis

The developer of a bio-object decides what attributes it has.⁵ Queries can be asked about the experiment.⁶ The result of a query is a set of bio-objects (for example, all bio-objects of type *cell* with attribute *stress* set to *true*). Query results can be used as input for statistical analysis or graphs. Since attributes and queries are accessible to scripts, this analysis can be fully automated.

3.3 Intervention into the Simulation

During the simulation, the user can intervene in the experiment by changing bio-objects' attributes and performing *operations*. An operation is an action performed on a bio-object while it is not working. Operations model interventions in the organism, such as injections or surgical operations. Again, the bio-object's developer de-

cides what operations it supports.⁷ Operations can also be automated by scripts (for example, inject a drug dose whenever the number of inflammatory cells exceeds a given threshold).

4 Future Challenges

While using a scripting language to control the simulation is very flexible, it is not very accessible to a wide group of potential users—biologists. This could be improved by introduction of an expressive domain specific language for BioLabs' experiments. The language would be able to express the composition of an experiment from bio-objects and interactions and to control its simulation, analysis and intervention in it.

To promote BioLabs as a unifying platform adding interoperation of different models, various (third-party) models should be ported to the BioLabs platform and their interaction in an experiment should be demonstrated.

⁴BioLabs currently support Java3D.

⁵Fields and methods of the bio-object's class that are considered as attributes are marked by the @Attribute annotation.

⁶Querying is similar to how the XPath language is used to query XML documents.

⁷Operations are methods marked by the @Operation annotation.