Computer Science Journal of Moldova, vol.**24**, no.1(70), 2016

Large-scale executable biology using rapid integration of computational models[*]

Vladimir Rogojin, Ion Petre

Abstract

We plan to develop a systematic framework for assembling large-scale computational biological models by reusing and combining already existing modelling efforts. Our goal is to build a software platform that will compile large-scale biomodels through successive integrations of smaller modules. The modules can be arbitrary executable programs accompanied by a set of (I/O) interface variables; they may also have an internal structure (such as a metabolic network, interaction network, etc.) that yields its executable part in a well defined way. Firstly, wherever possible, modules with the compatible internal structure will be joined by combining their structure and by producing new larger executable modules (like, combining two metabolic networks, etc.). Then, irrespective of the underlying internal structure and modelling formalisms, all the modules will be integrated through connecting their overlapping interface variables. The resulting composed model will be regarded as an executable program itself and it will be simulated by running its submodules in parallel and synchronizing them via their I/O variables. This composed model in its turn can also act as a sub-module for some other even large composite model. The major goal of this project is to deliver a powerful large-scale modeling methodology for the primary use in the fields of Computational Systems Biology and Bioinformatics.

Keywords: computational biomodelling, multiscale whole-cell modeling, model integration, model refinement, executable biology.

[*]The results published in this article were presented on November 13, 2015 at the seminar dedicated to the memory of Prof. Iu. Rogojin

1 Introduction

Predictive and comprehensive models of biological cells are highly significant for the understanding and engineering biological systems. Such large-scale whole-cell models have the potential to direct experiments in molecular biology, facilitate computer-aided design and simulation in synthetic biology, and enhance personalized therapeutic methods.

The ultimate goal of our research is to develop and implement a generic technique that will allow for automatic building of custom large-scale multi-level comprehensive models that are able to describe accurately a biological phenomenon of interest with a desired level of details. The main idea here stays in reusing and integrating of already existing disparate modeling efforts into more representative, and therefore, more accurate, simulations of the targeted phenomenon. The models will be organized into a hierarchical structure by their levels of abstraction, so that a user can easily navigate through the hierarchy and choose the appropriate levels of details for the resulting integrated model.

In Computational Systems Biology a large number of individual teams of researchers target isolated subsystems and processes from living cells for modeling and simulation. In many cases, different researchers model independently overlapping, highly related or even the same cellular structures and processes. Moreover, those computational models focus on some particular cellular sub-systems while ignoring cross-talks with the other sub-systems and the other factors that also influence the sub-system under studies. A step forward towards building representative cellular models will be to join the disparate efforts of different modelers and build complex large-scale models via integration of already existing ones.

The greatest challenge here comes from the fact that different research teams come with their own concepts, abstraction levels, formalisms and methodology preferences as well as with their own technological limitations for their models. They contribute to the literature a mass of knowledge in the form of models and simulations in different formats, level of details and data types. One has to overcome those challenges in order to bring numerous modeling and simulation efforts into a significantly more representative larger comprehensive model that will capture all the features of initial models and will agree still with all the respective initial experimental data.

Our method will involve searching for a common ground of integration between models, regardless of their formalisms and implementation. That common ground can be used to join models automatically into a single comprehensive simulation system. This will allow models to be integrated on several levels of details, depending on the required analysis. We will base our developments on a formal framework for integration of models of different formalisms, implementations and level of details that was presented in [1].

A software implementation of an integrated whole-cell model was presented in [2]. A model Mycoplasma genitalium was obtained through joining of 28 different independent computational submodels, where each submodel represents some specific cellular process [2]. The model described the dynamics of every molecule over the entire life cycle and accounts for the specific function of every annotated gene product. Each submodel was using formalism most appropriate to its functioning and existing knowledge. The submodels were assumed to run approximately independently on a short time scales. Simulations were performed by running through a loop in which the submodels were run independently at each time step but depended on the values of variables determined by the other submodels at the previous time step. The simulation software worked as a set of communicating running in parallel Matlab programs, where each program implements a submodel. The whole-cell software managed to predict the observed experimental data very accurately as well as it helped also to discover new previously unknown cellular behaviors. In the similar spirit, we will let for coordinated synchronized execution of different simulation instances within our simulation framework.

In our research, we propose a follow-up for the effort of developing large-scale biological model building techniques and model integration [1], [2]. We will represent a model as an executable simulating program that can be integrated with the other executable simulating programs standing for other models via a well defined API. We will allow for hierarchical type of model integration, that is, a number of complex models obtained due to previous integrations of simpler models can also be integrated together and form even more complex models.The practical outcome of our research will be a software modeling platform that will provide a systematic framework for integration and coordinated execution of different simulation instances, each simulating different parts of the biological phenomenon of interest. We are going to validate our methodology and computational platform through building a prototype for a whole-cell model of the life cycle of either yeast or E.coli, two of the most studied model organisms [3], [4]. Existing data in well-curated model database such as Biomodels [5] should be enough at least for a rough prototype of such a whole-cell model.

2 Background

A formal framework for integration of models of different formalisms, implementations and level of details was presented in [1]. The framework involves searching for a common computational ground between the models that will allow to integrate them, regardless of their formalism and implementation, across different methodologies into a single agent-based simulation system. Our abstract model descriptor will be based in particular on the concept of behavioral inclusion trees from [1].

Parameter fitting of a model to the experimental data is formulated as a global optimization problem. The goal is to tweak parameters of the model in such way that the predicted behavior of the model is as close as possible to the experimental data. The objective function here shows how far are the simulated data points from the experimental ones and takes as arguments the set of parameters to fit. The optimization goal is to find values of the parameters where the objective function reaches the global optimum (minimum). In particular, COPASI provides functionality for quantitative ODE-based model parameter estimation. It allows to fit both reaction kinetic rate parameters as well as initial concentrations of the metabolites.

Quantitative model refinement is an essential step in the model development cycle. Starting with a high level, abstract representation of a biological system, one often needs to add details to this representation to reflect changes in its constituent elements. Any such refinement step has two aspects: one structural and one quantitative. The structural aspect of the refinement defines an increase in the resolution of its representation, while the quantitative one specifies a numerical setup for the model that ensures its fit preservation at every refinement step. The refinement should be done so as to ensure the preservation of the numerical properties of the model, such as its numerical fit and validation. In [6]–[10] there were presented methods for quantitative model refinement in a number of modeling frameworks, such as ODE-based models, rule-based models, Petri net models, guarded command language. We plan to use quantitative refinement when bringing different models to the same level of details.

A number of efforts towards integrating a plethora of publicly available molecular-scale experimental measurements that render intracellular molecule functions and interactions has facilitated data-driven large-scale model development [11]. For instance, in [11] there was developed a toolkit called Moksiskaan that can integrate information about the connections between genes, proteins, pathways, drugs and other biological entities from a large number of databases. As the result, one can obtain a comprehensive network model encompassing signaling, metabolic, gene regulatory, etc. intracellular relations. We will employ Moksiskaan for collecting a vast range of biological data needed for model construction and fitting.

Anduril [12] is an open source component-based workflow framework for scientific data analysis developed at the Computational Systems Biology Laboratory, University of Helsinki. Anduril also provides API that allows integrating rapidly various existing software tools and algorithms into a single data analysis pipeline. An Anduril pipeline comprises a set of interconnected executable programs (called components) with well-defined I/O ports, where an output port of a component may be connected to the input ports of some number of other components. (for example, see Figure 1). During execution of a pipeline, a component will be executed as soon as all the input data are provided by the up-stream components. After execution of the component, its results become available for the downstream components that can be executed in their turn as soon as all the necessary input data are provided for them. Anduril is highly scalable computational platform, it runs well both on desktop personal computers as well as on powerful supercomputers and clusters. Anduril can run multiple processes in parallel and its pipelines can be scaled for a distributed execution across the network. The size of Anduril pipelines can range from several component instances that could run in few seconds up to thousands of component instances requiring several days of execution. We will base our model integration platform on Anduril.

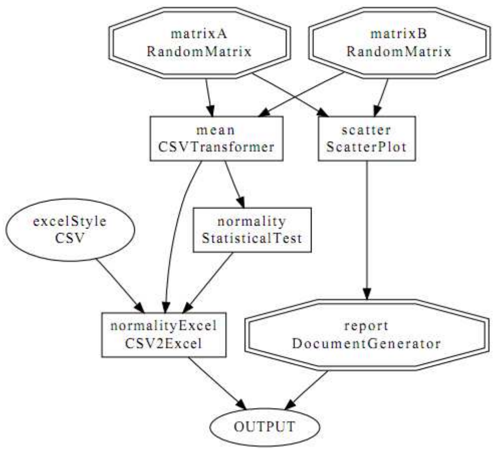

Figure 1. Example of an Anduril workflow (pipeline). Workflow is a set of tasks (component instances) organized in a directed network. Each component instance has some defined set of inputs and outputs, where outputs of an instance are connected to inputs of some other instances. Instances without any inputs import data into the workflow. Instances without any outputs normally output the workflow's computation results. For example, here *matrixA* and *matrixB* are instances of a component *RandomMatrix* that is a program generating a random matrix; instance *mean* of component *CSVTransformer* calculates mean for each row of matrices coming from *matrixA* and *matrixB* instances; instance *scatterPlot* generates a specific plot for the matrices from *matrixA* and *matrixB*, etc.

3 Integration of computational models

We are planning to obtain a software platform, that will allow a user to specify the input data (like set of initial models with their software implementations, model and environment parameters, abstraction levels, experimental data to fit to, etc.) for some target biological phenomenon and to query for prediction on some target features of the behavior of the modeled biological system (for instance, a time course for concentrations of some biochemical species, the time of cell growth in particular environmental conditions, etc.).

Here, we focus on developing model refinement and integration techniques for biological computational models. This project is a first step towards building a system that, based on a user query, will allow collecting and reuse existing biological computational models in order to generate automatically custom case-specific comprehensive models for particular biological phenomenon at the desired abstraction level. We will develop a methodology that will allow minimizing or avoiding refitting the initial models to their corresponding experimental data while performing model refinement and model integration activities.

Our research is a follow-up for the effort of developing large-scale biological model building techniques and model integration [1],[2]. Particularly, we will allow for an arbitrary nature of a modeling formalism and its implementation. A model will be represented by an executable simulating program and can be integrated with the other executable simulating programs standing for other models via a well defined API. We will allow for hierarchical type of model integration, that is, a number of complex models obtained due to previous integrations of simpler models can also be integrated together and form even more complex models. We will allow for a high level of scalability and flexibility in the sense that a model can be tuned to a particular use case to fit the expected/desired behavior and to incorporate the desired level of details. In particular, we will employ here the previous practices related to numerical model parameter fitting [13] and model refinement [6]–[10].

We will address the following challenges:

1. Develop and implement intra-formalism integration methods for a number of different formalisms across different abstraction levels. That is, we are aiming to develop a systematic approach for automatic abstraction level adjustment via refinement and building a single model through joining a number of initial models, while working within the same formalism;

2. Develop and implement inter-formalism integration methods;

3. Deploy a generic (plug&play) platform that allows to “plug” computational models into the composite model, refine them and obtain the respective large-scale models and simulations.

4 Methods

We discuss here the methods that we are planning to apply in our research.

4.1 Abstract model descriptor

We will develop a methodology to describe in an abstract way models so that it will be possible to decide automatically how to join the models and how to instantiate the resulting integrated model. This *abstract model descriptor* will abstract from the model type (discrete or continuous, stochastic or deterministic), the modeling frameworks and models implementations.

The model descriptor will include such concepts as an *entity* (formalizes a real world physical object), a *process* (represents an activity involving one or more entities), a *variable* (a measurable/observable and/or affectable property of an entity or process). An entity may consist of a number of other entities, a process can be split in subprocesses, a variable may characterize/affect the current state of an entity or a process. Also, an entity may be an abstraction of some other entity and a process may be an abstraction of some other process.

We regard the abstract model descriptor as a digraph with annotated nodes and edges, Figure 2. In order to enable automatic integration of models basing on their descriptors, the user has to map nodes and edges of the descriptors to the corresponding components of the models or simulating instances. The exact way, how this mapping should be performed will depend on the particular type, formalism and implementation of each of the models.

4.2 Model integration and simulation

Here, the user should provide a set of initial models to integrate along with their abstract descriptors and mappings between the models and their descriptors. We remind, that we regard a model as an executable program or as a formal construct (a set of chemical reactions, signaling pathways, gene regulatory nets, etc.) together with its well-defined mapping onto an executable simulation instance.

We consider the following steps for integrating multiple user-provided models into one:

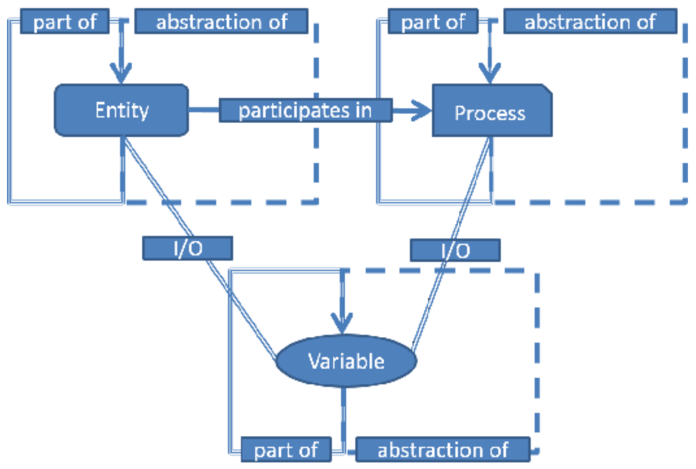

Figure 2. Schematic representation of abstract model descriptor. An entity can be a part of/abstraction of some other entity, a process can be a part of/abstraction of some other process and a variable can be a part of/abstraction of some other variable. The *part-of* relation is represented by a triple-line edge on the plot, and *abstraction-of* is represented as a dotted line. Entity can participate in a process. The relation *participates-in* is represented by solid line. A variable can be regarded as an interface to the state of an entity or process. The relation *input/output (I/O)* is represented by a double-line edge.

1. For all the models for which we have formally defined constructs, decide what constructs can be combined and how (for instance, two metabolic networks can be joined into one). We need to develop methods for joining models with the "compatible" internal structures into one model with its internal structure being a union of the original structures. See for example Figure 3. The internal structures will be joined through their overlapping components. Those overlapping components from two different structures that are at different levels of details will be brought to the same abstraction level via model refinement techniques [6]–[10]. The internal structure integration methodology and model refinement process strictly depend on the formalism being used, and we will develop the integration and refinement techniques for a

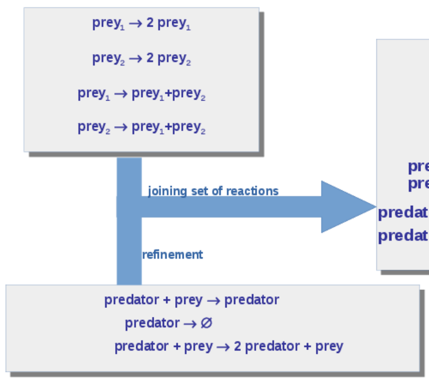
$prey_1 \to 2\ prey_1$
$prey_2 \to 2\ prey_2$
$prey_1 \to prey_1 + prey_2$
$prey_2 \to prey_1 + prey_2$
joining set of reactions
refinement
$predator + prey \to predator$
$predator \to \varnothing$
$predator + prey \to 2\ predator + prey$
$prey_1 \to 2\ prey_1$
$prey_2 \to 2\ prey_2$
$prey_1 \to prey_1 + prey_2$
$prey_2 \to prey_1 + prey_2$
$predator + prey_1 \to predator$
$predator + prey_2 \to predator$
$predator + prey_1 \to 2\ predator + prey_1$
$predator + prey_2 \to 2\ predator + prey_2$
$predator \to \varnothing$

Figure 3. Example of integration of two models within the same formalism into one. The result: modified classic Lotka-Volterra Pray-Predator model [14] with two types of prey. The model is regarded in terms of chemical reactions (reaction rates and initial species concentrations are omitted here). One of the initial models describes dynamics (basically proliferation) of two types of prey. The other initial model describes predator's dynamics and its relation to prey specie (note that this model does not consider different types of prey species). During the integration process, the predators' model gets refined in order to include relations to the two types of prey species, then sets of reactions from the prey and the refined predator models get united.

number of classical formalisms (such as mass-action based ODEs, Petri nets, Boolean logic networks, etc.) separately. As the result of such integration of models with the compatible internal structures we will get one simulation instance that captures the behavior of the original models;

2. The user may provide a set of experimental data to fit the initial models. Fit the models to the user-provided time series data where necessary. Translate the resulting formal models into executable simulation instances;

3. Basing on the abstract model descriptors, decide connections between the executable simulation instances. The connections can be uni- or bidirectional. Here, a connection will also mean translation of an output

from a source simulation instance into an appropriate input format for the destination simulation instance. The type of translation will be decided automatically basing on the classes of the source and destination instances.

As the result, the integrated model will be the set of executable simulation instances with the scheme of their interconnections. The integrated model can be considered as a simulation instance itself and can be further integrated with the other models.

During the simulation the simulation instances will send to each other synchronizing messages that include the current states of some of their components as well as their contexts. The connection scheme defines which instance sends messages to which instances and what components states are included in each of the messages. In particular, the states of entities, processes and variables will be communicated. The abstract model descriptor of each of the initial models will be used to identify the overlapping or related components between different models. This information will be used to decide what components of the respective models should be refined, and what connections to form between instances. If in one of the models one identifies a set of components that is a refinement of a component from some other model, then one has to perform the respective refinement in the other model in order to obtain a set of the components that coincides exactly with the components from the former model. The refinement procedure depends on what type, formalism and model implementation is used in the respective model. When having same component (at the same level of abstraction) in two different models, one can form a connection between those two models that specifies in particular what components in these models are connected (also, probably, in which direction). One also associates to the connection a data transformer/adapter in order to transform the representation of states of the model component between the respective formalisms and data formats. See Figures 4, 5, 6 and 7 for an example of combining two simulation instances basing on their abstract descriptors.

The set of connections between a pair of instances defines what components communicate their states to the partner model within a message. Also, the context of one model that is being sent to the other model along with the states includes the simulated time point in order to synchronize two models.

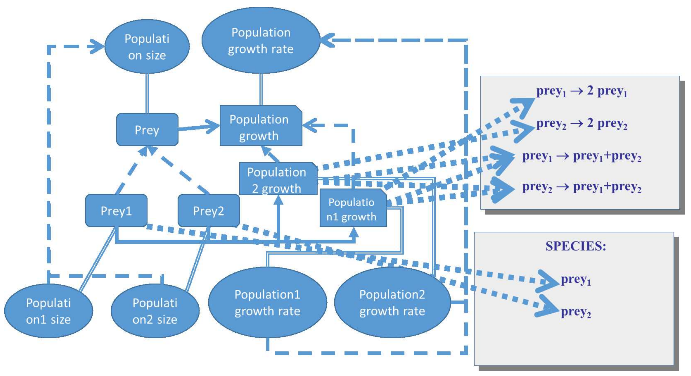

Figure 4. An abstract model descriptor for the prey population dynamics and its mapping to the internal model structure (set of reactions and species)

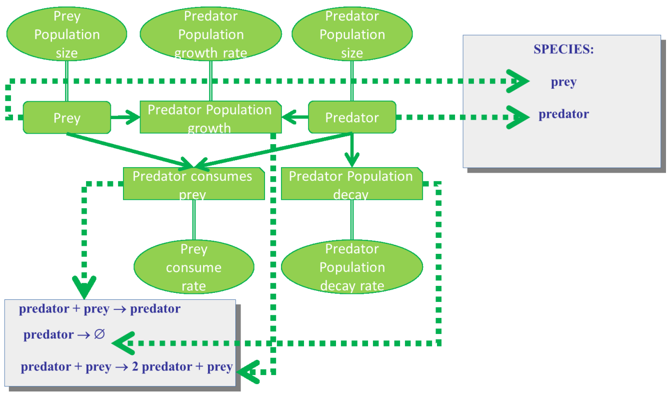
Prey Population size
Predator Population growth rate
Predator Population size
SPECIES:
prey
predator
Prey
Predator Population growth
Predator
Predator consumes prey
Predator Population decay
Prey consume rate
Predator Population decay rate
$predator + prey \to predator$
$predator \to \varnothing$
$predator + prey \to 2\ predator + prey$

Figure 5. An abstract model descriptor for the predator population dynamics (including population growth/decay and prey consumption) and its mapping to the internal model structure

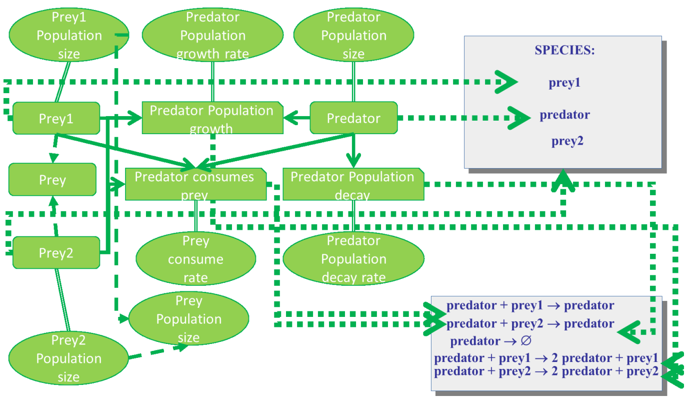
Prey1 Population size
Predator Population growth rate
Predator Population size
SPECIES:
prey1
predator
prey2
Prey1
Predator Population growth
Predator
Prey
Predator consumes prey
Predator Population decay
Prey2
Prey consume rate
Predator Population decay rate
Prey Population size
Prey2 Population size
predator + prey1 → predator
predator + prey2 → predator
predator → ∅
predator + prey1 → 2 predator + prey1
predator + prey2 → 2 predator + prey2

Figure 6. An abstract model descriptor for the predator population dynamics that was refined with two types of prey in order to have overlapping components between submodels from (a) and (b) at the same level of details

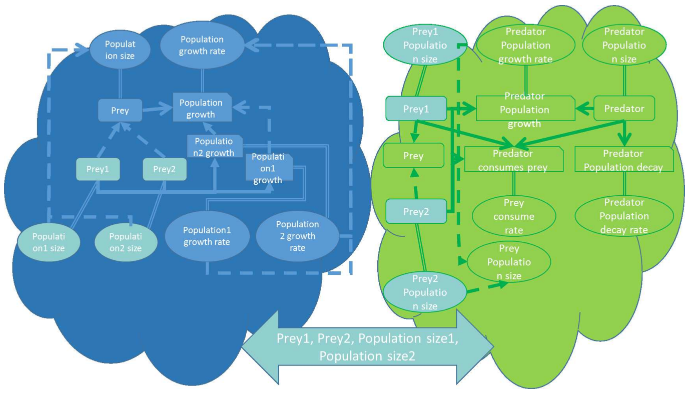

Figure 7. Basing on the abstract model descriptors for submodels from (a) and (b) it was decided to refine submodel for (b) into (c), and the overlapping components were detected between (a) and (c), that are “Prey1” and “Prey2” entities, “Population1 size” and “Population2 size” variables. Hereby, we establish connection between simulation instances corresponding to (a) and (c) that update each other with the states of “Prey1”, “Prey2”, “Population1 size” and “Population2 size” components.

We will develop an API which the simulation instances should handle in order to exchange messages with each other during the simulation. When a simulation instance receives a message, it should decide how to update its internal states of the respective components, especially in the case when the received message brings set of states contradicting the instance's own internal states.

4.3 Software implementation

We will produce a software platform that will construct a large-scale model through integration and refinement of the existing models.

Our platform will be built on top of the Anduril workflow framework due to the fact that Anduril provides systematic computational environment for rapid integration of existing computational tools and algorithms into an organized pipeline. An integrated model will be implemented as an Anduril pipeline, where components will represent the initial simulation instances, and connections between the components will represent connections between the respective models/instances. We also note, that an Anduril pipeline can serve as a component to be included in the other pipeline. In other words, Anduril will allow integrating a complex model within the other more complex model.

A simulation instance will be a software program wrapping the respective existing simulation software for the respective model, while handling the API to communicate to other instances and updating the local states of the instance according to the states received from the other instances. Also, the wrapper has to handle translation of states between different abstraction levels. In particular, we can incorporate COPASI as a dynamically linked library and access all of its simulation and parameter estimation functionality from our simulation instance program, that can "talk" to other instances and input/output data into/from the COPASI-simulated model from/to the other simulation instances. Petri nets can be simulated with S4 Snoopy modeling software that we are going to access from the respective simulation instance via well-define API of S4. Simulation instances for the Rule-Based modeling will use BioNetGen software as a command-line tool, or incorporate directly BioNetGen program code. We will use CellNetAnalyzer Matlab library in the simulation instance for Boolean networks. Generally speaking, any third-party simulation software can be incorporated into our integrative simulation framework via a wrapping program that handles our API for the communication with the other simulation instances. Schematically, this idea is represented in Figure 8.

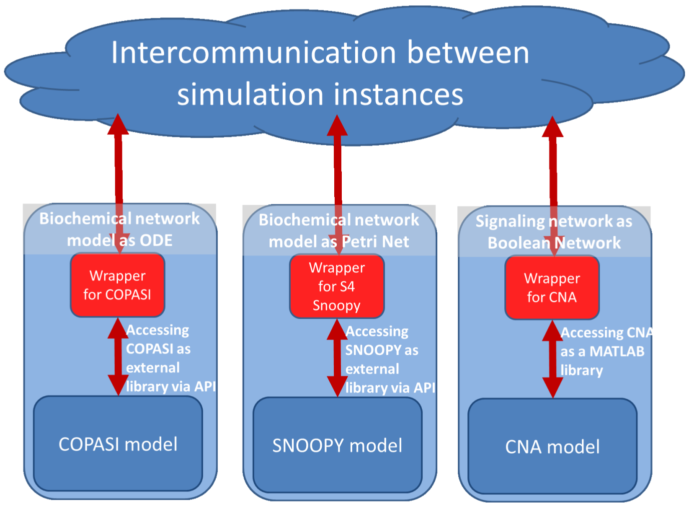
Intercommunication between simulation instances
Biochemical network model as ODE
Wrapper for COPASI
Accessing COPASI as external library via API
COPASI model
Biochemical network model as Petri Net
Wrapper for S4 Snoopy
Accessing SNOOPY as external library via API
SNOOPY model
Signaling network as Boolean Network
Wrapper for CNA
Accessing CNA as a MATLAB library
CNA model

Figure 8. Scheme of the general organization of simulation instances. Each simulation instance simulates a model of some biological phenomena (for instance, biochemical networks, signaling networks, DNA transcription, RNA translation, etc.) probably in some well-defined formalism (for instance, mass-action based ODE model, Petri nets, Boolean networks, etc.) that is being implemented/simulated by some software (for instance, COPASI for ODE, Snoopy for Petri nets, CellNetAnalyzer for Boolean networks, etc.) A simulation instance consists of the original simulating software and the wrapper that translates between the simulation software and the messages for the other simulation instances.

5 Discussion

As the result, this project should deliver a powerful methodology for construction of large-scale models via integration of existing models and data.

The methodology will combine model refinement and integration techniques and assumes both data- and hypotheses-driven model construction approaches.

We believe that our research will contribute to the scientific modeling community through the development of a methodology for connecting different computational biological models irrespective of their formalisms, concepts and implementations. The platform that we are planning to implement will allow to build large-scale models and run comprehensive simulations by joining independent modeling and simulation efforts from a great number of research biomodeling projects. As the result, one will be able to build rapidly and easily large-scale custom models from a set of already existing models that will satisfy user-specific particular needs. In particular, the experimental biologist researchers should be the first to benefit from our platform due to the fact that comprehensive large-scale modeling can help in discovering new features of cellular processes without the need to conduct multiple expensive laboratory experiments. We may list among the other stakeholders bioinformaticians, medical-related researchers, synthetic biologists, bio-engineers, pharmacologists and any one else who needs to run expensive bio-experiments in-vivo or in-vitro.

References

[1] M. Patel and S. Nagl, “The role of model integration in complex systems modelling,” *Understanding Complex Systems*, 2010. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15603-8

[2] J. R. Karr, J. C. Sanghvi, D. N. Macklin, M. V. Gutschow, J. M. Jacobs, B. Bolival Jr., N. Assad-Garcia, J. I. Glass, and M. W. Covert, “A Whole-Cell Computational Model Predicts Phenotype from Genotype,” *Cell*, vol. 150, no. 2, pp. 389–401, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0092867412007763

[3] L. Pray, “L. h. hartwell’s yeast: A model organism for studying somatic mutations and cancer,” *Nature Education*, vol. 1, no. 1, p. 183, 2008.

[4] P. S. Lee and K. H. Lee, “Escherichia colia model system that benefits from and contributes to the evolution of proteomics,” *Biotechnology and Bioengineering*, vol. 84, no. 7, pp. 801–814, 2003. [Online]. Available: http://dx.doi.org/10.1002/bit.10848

[5] C. Li, M. Donizelli, N. Rodriguez, H. Dharuri, L. Endler, V. Chelliah, L. Li, E. He, A. Henry, M. I. Stefan, J. L. Snoep, M. Hucka, N. Le Novère, and C. Laibe, “Biomodels database: An enhanced, curated and annotated resource for published quantitative kinetic models,” *BMC Systems Biology*, vol. 4, no. 1, p. 92, 2010. [Online]. Available: http://www.biomedcentral.com/1752-0509/4/92

[6] D.-E. Gratie, B. Iancu, S. Azimi, and I. Petre, “Quantitative model refinement in four different frameworks,” in *From Action Systems to Distributed Systems*, L. Petre and E. Sekerinski, Eds. Taylor & Francis Group, 2016.

[7] D.-E. Gratie and I. Petre, “Hiding the combinatorial state space explosion of biomodels through colored petri nets,” *Annals of University of Bucharest*, vol. LXI, pp. 23–41, 2014.

[8] C. Gratie and I. Petre, “Fit-preserving data refinement of mass-action reaction networks,” in *Language, Life, Limits*, ser. Lecture Notes in Computer Science, A. Beckmann, E. Csuhaj-Varju, and K. Meer, Eds., vol. 8493. Springer, 2014, pp. 204 – 213.

[9] B. Iancu, D.-E. Gratie, S. Azimi, and I. Petre, “On the implementation of quantitative model refinement,” in *Algorithms for Computational Biology*, ser. Lecture Notes in Computer Science, A.-H. Dediu, C. Martin-Vide, and B. Truthe, Eds., vol. 8542. Springer International Publishing, 2014, pp. 95–106.

[10] B. Iancu, E. Czeizler, E. Czeizler, and I. Petre, “Quantitative refinement of reaction models,” *International Journal of Unconventional Computing*, vol. 8, no. 5-6, pp. 529–550, 2012.

[11] M. Laakso and S. Hautaniemi, “Integrative platform to translate gene sets to networks,” *Bioinformatics*, vol. 26, pp. 1802–1803, 7 2010. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btq277

[12] K. Ovaska, M. Laakso, S. Haapa-Paananen, R. Louhimo, P. Chen, V. Aittomki, E. Valo, J. Nunez-Fontarnau, V. Rantanen, S. Karinen, K. Nousiainen, A.-M. Lahesmaa-Korpinen, M. Miettinen, L. Saarinen, P. Kohonen, J. Wu, J. Westermarck, and S. Hautaniemi, “Large-scale data integration framework provides a comprehensive view on glioblastoma multiforme.” *Genome medicine*, vol. 2, no. 9, pp. 65+, Sep. 2010. [Online]. Available: http://dx.doi.org/10.1186/gm186

[13] E. Czeizler, V. Rogojin, and I. Petre, “The phosphorylation of the heat shock factor as a modulator for the heat shock response,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 5, pp. 1326–1337, 2012.

[14] F. Hoppensteadt, “Predator-prey model,” *Scholarpedia*, vol. 1, no. 10, p. 1563, 2006, revision 91666.

Vladimir Rogojin, Ion Petre

Received February 9, 2016

Computational Biomodelling Laboratory
Turku Centre for Computer Science, and
Department of Computer Science
Åbo Akademi University
Agora, Domkyrkotorget 3
20500 Åbo
Phone: +358 2 215 3361
E–mail: `vrogojin@abo.fi, ipetre@abo.fi`