

Lessons Learned at the Whole Cell Summer School

Chris J. Myers

Theory

A Whole-Cell Computational Model Predicts Phenotype from Genotype

Jonathan R. Karr,^{1,4} Jayodita C. Sanghvi,^{2,4} Derek N. Macklin,² Miriam V. Gutschow,² Jared M. Jacobs,² Benjamin Bolival, Jr.,² Nacyra Assad-Garcia,³ John I. Glass,³ and Markus W. Covert^{2,*}

¹Graduate Program in Biophysics

²Department of Bioengineering

Stanford University, Stanford, CA 94305, USA

³J. Craig Venter Institute, Rockville, MD 20850, USA

⁴These authors contributed equally to this work

*Correspondence: mcovert@stanford.edu

<http://dx.doi.org/10.1016/j.cell.2012.05.044>

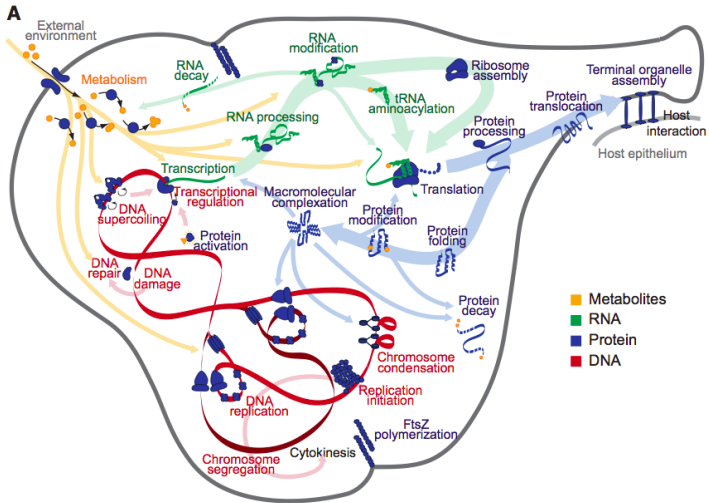
SUMMARY

Understanding how complex phenotypes arise from individual molecules and their interactions is a primary challenge in biology that computational approaches are poised to tackle. We report a whole-cell computational model of the life cycle of the human pathogen *Mycoplasma genitalium* that

First, until recently, not enough has been known about the individual molecules and their interactions to completely model any one organism. The advent of genomics and other high-throughput measurement techniques has accelerated the characterization of some organisms to the extent that comprehensive modeling is now possible. For example, the mycoplasmas, a genus of bacteria with relatively small genomes that includes several pathogens, have recently been the subject of an exhaustive experimental effort by a European consortium to determine

Cell (2012)

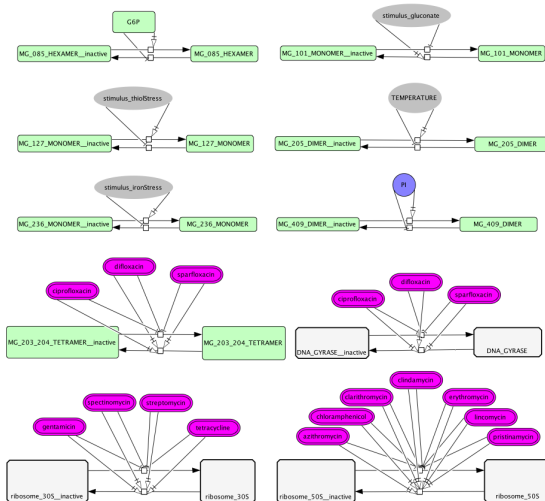
Whole Cell Model Processes



Goals of the Project

- Create an SBGN diagram for each process.
- Produce SBML for each process.
- Write SED-ML to test each process.
- Integrate and test the complete model.

SBGN for Protein Activation



Challenges Converting Matlab to SBML

- Matlab is inherently sequential while SBML is inherently parallel.
- Assignments, conditional behavior, and loops can be done with SBML events, but it can be a bit tricky.
- Random numbers and distributions not yet well supported by SBML (need distributions package).
- Matrices are efficient in Matlab and not well supported by SBML (need arrays package and a way to handle large, sparse matrices).
- Parallel access to mutually exclusive shared variables is difficult (chromosome).

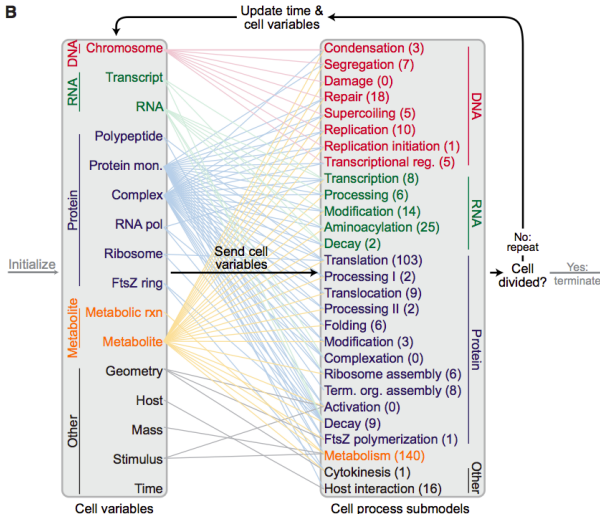
Key Challenge for Each Group

- Replication initiation - need to access shared variable, chromosome.
- Replication - must represent state of 500kb chromosome.
- Transcription - need to track positions of transcription factors and move RNAP along the chromosome, large number of variables.
- Translation - need to track positions of ribosomes on many types of mRNAs, again large number of variables.
- Protein - many small processes, requires random distributions.
- Metabolism - updating bounds between simulation steps.
- DNA damage/repair - accessing shared chromosome state and dealing with randomness.
- Cytokinesis - dealing with randomness.
- Integration - variable sharing.

Integration Team Goals

- Coordinate the development of the interfaces for each process.
- Determine how SED-ML can be used to express the simulation of each process, as well as, the complete model.
- Determine a scheme for variable sharing between the processes.

Whole Cell Model Simulation



Simulation Algorithm

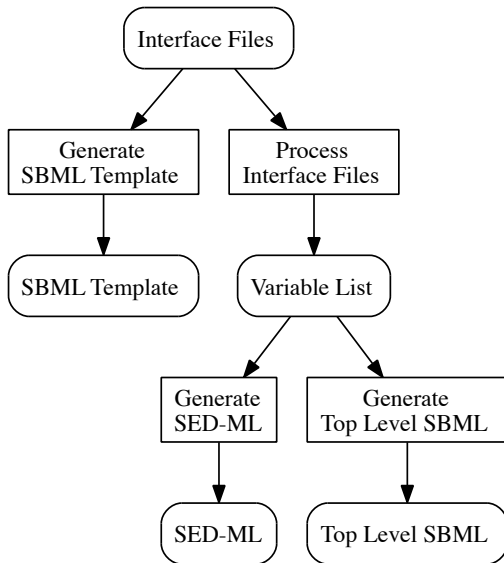
```
function RunSimulation( $S[0]$ )  
  for  $t \leftarrow 1..tMAX$  do  
     $S[t] \leftarrow S[t - 1]$   
    for  $P_k \leftarrow P_1$  to  $P_p$  do  
       $S_k[t] \leftarrow P_k.copyFromState(S[t])$   
       $Req_k \leftarrow P_k.require(S_k[t])$   
    end for  
    for  $P_k \leftarrow P_1$  to  $P_p$  do  
       $R_k \leftarrow Split(R[t], Req_k, Req_1, Req_2, \dots, Req_p)$   
    end for  
    for  $P_k \leftarrow Permute(P)$  do  
       $S_k[t] \leftarrow P_k.copyFromState(S[t])$   
       $(V_k[t] || R_k) \leftarrow P_k.evolve((V_k[t] || R_k))$   
       $V[t] \leftarrow P_k.copyToState(V_k[t])$   
    end for  
     $S[t] \leftarrow (V[t] || Merge(R_1, R_2, \dots, R_p))$   
  end for  
  return  $S$   
end function
```

Shared Variables

Three types of shared variables:

- Metabolites - distributed based upon process requirements.
- Proteins/variables - updated by a few processes.
- Chromosome - requires mutually exclusive access.

Integration Workflow



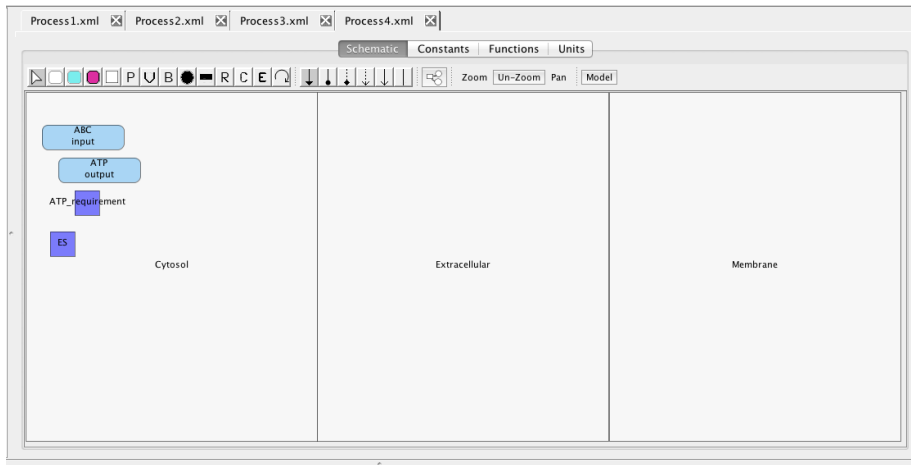
Interface Files

- For each species, must indicate type of access needed:
 - Read-only
 - Read/write
 - Requirement
 - Read-only and requirement
- For each parameter, also must indicate type of access needed:
 - Read-only
 - Read/write
 - Mutually exclusive

Interface Files Example

Process1						
Type	Id	RO	RW	REQ	ROREQ	ME
Metabolite	ATP		X			
Protein	ABC		X			
Parameter	ES		X			
Process2						
Type	Id	RO	RW	REQ	ROREQ	ME
Metabolite	ATP	X				
Protein	ABC		X			
Parameter	ES	X				
Process3						
Type	Id	RO	RW	REQ	ROREQ	ME
Metabolite	ATP			X		
Protein	ABC	X				
Parameter	ES	X				
Process4						
Type	Id	RO	RW	REQ	ROREQ	ME
Metabolite	ATP				X	
Protein	ABC					
Parameter	ES					

SBML Template Example



Variable List Example

species,ATP,Process1(rw),Process2(ro),Process3(req),Process4(roreq)
species,ABC,Process1(rw),Process2(rw),Process3(ro)
param,ES,Process1(rw),Process2(ro),Process3(ro)

SED-ML (no variable sharing)

```
<?xml version="1.0" encoding="UTF-8"?>
<sedML xmlns="http://sed-ml.org/sed-ml/level1/version2" level="1" version="2">
  <listOfSimulations>
    <oneStep id="sim_1" step="1.0">
      <algorithm kisaoID="KISA0:0000035"/>
    </oneStep>
  </listOfSimulations>
  <listOfModels>
    <model id="Process1" language="urn:sedml:language:sbml" source="Process1.xml"/>
    <model id="Process2" language="urn:sedml:language:sbml" source="Process2.xml"/>
    <model id="Process3" language="urn:sedml:language:sbml" source="Process3.xml"/>
    <model id="Process4" language="urn:sedml:language:sbml" source="Process4.xml"/>
  </listOfModels>
  <listOfTasks>
    <task id="tsk_1" modelReference="Process1" simulationReference="sim_1"/>
    <task id="tsk_2" modelReference="Process2" simulationReference="sim_1"/>
    <task id="tsk_3" modelReference="Process3" simulationReference="sim_1"/>
    <task id="tsk_4" modelReference="Process4" simulationReference="sim_1"/>
    <repeatedTask id="main_loop" resetModel="false" range="repeat">
      <listOfRanges>
        <uniformRange id="repeat" start="0" end="1000" numberOfPoints="1000" type="linear" />
      </listOfRanges>
      <listOfSubTasks>
        <subTask order="1" task="tsk_1" />
        <subTask order="2" task="tsk_2" />
        <subTask order="3" task="tsk_3" />
        <subTask order="4" task="tsk_4" />
      </listOfSubTasks>
    </repeatedTask>
  </listOfTasks>
</sedML>
```

SED-ML (variable sharing)

```
<?xml version="1.0" encoding="UTF-8"?>
<sedML xmlns="http://sed-ml.org/sed-ml/level1/version2" level="1" version="2">
  <listOfSimulations>
    <oneStep id="sim_1" step="1.0">
      <algorithm kisaoID="KISA0:0000035"/>
    </oneStep>
  </listOfSimulations>
  <listOfModels>
    <model id="Process1" language="urn:sedml:language:sbml" source="Process1.xml"/>
    <model id="Process2" language="urn:sedml:language:sbml" source="Process2.xml"/>
    <model id="Process3" language="urn:sedml:language:sbml" source="Process3.xml"/>
    <model id="Process3" language="urn:sedml:language:sbml" source="Process4.xml"/>
  </listOfModels>
  <listOfTasks>
    <task id="tsk_1" modelReference="Process1" simulationReference="sim_1"/>
    <task id="tsk_2" modelReference="Process2" simulationReference="sim_1"/>
    <task id="tsk_3" modelReference="Process3" simulationReference="sim_1"/>
    <task id="tsk_4" modelReference="Process4" simulationReference="sim_1"/>
    <repeatedTask id="main_loop" resetModel="false" range="repeat">
      <listOfRanges>
        <uniformRange id="repeat" start="0" end="1000" numberOfPoints="1000" type="linear" />
      </listOfRanges>
    </repeatedTask>
  </listOfTasks>
</sedML>
```

SED-ML (variable sharing)

```
<listOfChanges>
  <setValue target="/sbml:sbml/sbml:model[@id='Process1']/sbml:listOfSpecies/sbml:species[@id='ATP']"
    modelReference="Process1">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <ci> 0 </ci>
    </math>
  </setValue>
  <setValue target="/sbml:sbml/sbml:model[@id='Process2']/sbml:listOfSpecies/sbml:species[@id='ATP']"
    modelReference="Process2">
    <listOfVariables>
      <variable id="sp_1" modelReference="Process1" target="/sbml:sbml/sbml:model[@id='Process1']/
        sbml:listOfSpecies/sbml:species[@id='ATP']"/>
      <variable id="sp_3" modelReference="Process3" target="/sbml:sbml/sbml:model[@id='Process3']/
        sbml:listOfSpecies/sbml:species[@id='ATP']"/>
      <variable id="sp_4" modelReference="Process4" target="/sbml:sbml/sbml:model[@id='Process4']/
        sbml:listOfSpecies/sbml:species[@id='ATP']"/>
    </listOfVariables>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply><plus/><ci>sp_1</ci><ci>sp_3</ci><ci>sp_4</ci></apply>
    </math>
  </setValue>
  ...
</listOfChanges>
<listOfSubTasks>
  <subTask order="1" task="tsk_1" />
  <subTask order="2" task="tsk_2" />
  <subTask order="3" task="tsk_3" />
  <subTask order="4" task="tsk_4" />
</listOfSubTasks>
</repeatedTask>
</listOfTasks>
</sedML>
```

Metabolite Sharing

The screenshot displays the iBiosim software interface. The main window shows a metabolic model with a transition 't0' (a green square) and a place 'p1' (a circle). Transition 't0' is connected to place 'p1' and has an outgoing arrow to place 't1'. There are also two blue boxes labeled 'Req_1' and 'Req_2' on the left. On the right, there are three blue boxes labeled 'Metabolite_1 output', 'Metabolite_2 output', and 'Metabolite output'. The top of the window has tabs for 'Process1.xml', 'Process2.xml', 'Process3.xml', 'Process4.xml', and 'require'. Below the tabs are buttons for 'Schematic' and 'Constants'. A toolbar with various icons is visible above the main workspace.

The 'Transition Editor' dialog is open on the right. It contains the following fields and options:

- ID: t0
- Name: (empty)
- Is Mapped to a Port: ☐
- Enabling condition: true
- Enabling is persistent: ☐
- Fail transition: ☐
- Delay: (empty)
- Priority: (empty)
- SBOL DNA Component: Associate SBOL
- List of Assignments:
 - Metabolite := Metabolite_1 + Metabolite_2;
 - Metabolite_1 := floor((Metabolite_1 + Metabolite_2) * Req_1 / (Req_1 + Req_2));
 - Metabolite_2 := floor((Metabolite_1 + Metabolite_2) * Req_2 / (Req_1 + Req_2));
- Buttons: Add Assignment, Remove Assignment, Edit Assignment, Cancel, OK

Species/Parameter Sharing

The screenshot displays the iBioSim software interface. The main window shows a Petri net diagram with places p0, p1, and t0, and transition t1. Place p0 is a grey circle, p1 is a white circle, and t0 is a green square. Transition t1 is a white rectangle. Arrows indicate flow from p0 to t0, p0 to t1, and p1 to t1. To the right of the diagram are three blue rounded rectangles labeled 'S output', 'S1 output', and 'S2 output'. The top of the window has tabs for 'Process1.xml', 'Process2.xml', 'Process3.xml', 'Process4.xml', and 'requirements.xml'. Below the tabs are buttons for 'Schematic' and 'Constants'. A toolbar with various icons is located below the buttons. A 'Transition Editor' dialog box is open on the right, showing fields for ID (t0), Name, Is Mapped to a Port, Enabling condition (true), Enabling is persistent, Delay (0), Priority, SBOL DNA Component, and List of Assignments. The List of Assignments contains the text $S := S + (S-S1) + (S-S2);$. Buttons for 'Add Assignment', 'Remove Assignment', 'Edit Assignment', 'Cancel', and 'OK' are at the bottom of the dialog.

iBioSim

Process1.xml Process2.xml Process3.xml Process4.xml requirements.xml

Schematic Constants

S output

S1 output

S2 output

p0

t0

p1

t1

Transition Editor

ID: t0

Name:

Is Mapped to a Port: ☐

Enabling condition: true

Enabling is persistent: ☐ Fail transition: ☐

Delay: 0

Priority:

SBOL DNA Component: Associate SBOL

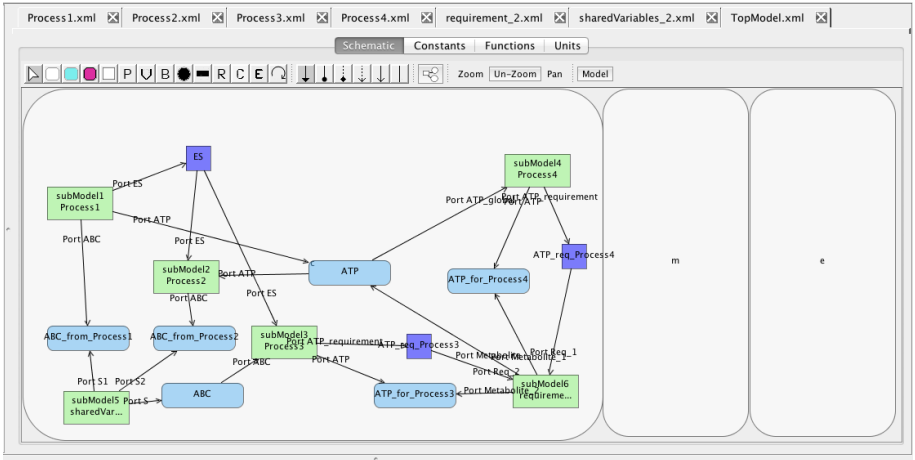
List of Assignments:

$S := S + (S-S1) + (S-S2);$

Add Assignment Remove Assignment Edit Assignment

Cancel OK

Top Level SBML



Remaining Integration Tasks

- Develop SBML for mutually exclusive accessed shared variables.
- Finish debugging the scripts.
- Obtain missing interface files.
- Share SBML templates with the other teams.
- Generate top-level SED-ML.
- Generate top-level SBML.
- Test with models provided from other teams.

Lessons Learned by Jonathan Karr

- Lot of work to do to make models more transparent.
- SBML can be more easily human readable.
- SBML more clearly expresses biochemistry, but not if rules are expanded out.
- Stochastic modeling (Gillespie) very similar to many sub-models except far more computationally expensive.
- New thoughts on how to generalize state allocation to chromosome.

SBML Challenges Observed by Jonathan Karr

- Unable to efficiently represent large state spaces.
- Unable to compactly represent template reaction rules.
- Semi-standard because not all packages implemented by all tools.
- SBML files must be paired with specific programs.
- SBML could improve with:
 - Expansion of language to include 1D spatial representation, new construct for chromosome, and other higher-level constructs.
 - New rule-based GUI to enable higher-level description.
 - New simulator to handle large scale models efficiently.

SED-ML/SBGN Observations by Jonathan Karr

- SED-ML useful, but care should be taken in avoiding encoding the model in SED-ML.
- SBGN useful for documentation, but it does not typically produce biologically intuitive diagrams for publication.