Toward Verified Biological Models

Avital Sadot, Jasmin Fisher, Dan Barak, Yishai Admanit, Michael J. Stern, E. Jane Albert Hubbard, and David Harel

Abstract—The last several decades have witnessed a vast accumulation of biological data and data analysis. Many of these data sets represent only a small fraction of the system's behavior, making the visualization of full system behavior difficult. A more complete understanding of a biological system is gained when different types of data (and/or conclusions drawn from the data) are integrated into a larger scale representation or model of the system. Ideally, this type of model is consistent with all available data about the system, and it is then used to generate additional hypotheses to be tested. Computer-based methods intended to formulate models that integrate various events and to test the consistency of these models with respect to the laboratory-based observations on which they are based are potentially very useful. In addition, in contrast to informal models, the consistency of such formal computer-based models with laboratory data can be tested rigorously by methods of formal verification. We combined two formal modeling approaches in computer science that were originally developed for nonbiological system design. One is the interobject approach using the language of live sequence charts (LSCs) with the Play-Engine tool, and the other is the intraobject approach using the language of statecharts and Rhapsody as the tool. Integration is carried out using InterPlay, a simulation engine coordinator. Using these tools, we constructed a combined model comprising three modules. One module represents the early lineage of the somatic gonad of *Caenorhabditis elegans* in LSCs, whereas a second more detailed module in statecharts represents an interaction between two cells within this lineage that determine their developmental outcome. Using the advantages of the tools, we created a third module representing a set of key experimental data using LSCs. We tested the combined statechart-LSC model by showing that the simulations were consistent with the set of experimental LSCs. This small-scale modular example demonstrates the potential for using similar approaches for verification by exhaustive testing of models by LSCs. It also shows the advantages of these approaches for modeling biology.

Index Terms—*C. elegans*, modeling, statecharts, verification.

1 Introduction

Biological systems are fundamentally similar to reactive engineered systems. By definition, a reactive system continuously interacts with its environment [1], which also describes the functioning of biological systems [2], [3]. This analogy can be extended to a comparison between building engineered systems and the process of modeling and model verification of biological systems [4]. The design of an engineered system begins with the definition of a set of requirements determined by a concept of how the system should eventually work. These requirements not only guide the construction and implementation of the system but are also used in verifying the system's correctness, that is, that the system acts as it should with respect to the requirements. Building biological models, on the other hand, involves reverse engineering, where mechanistic models are built to represent how the biological system works based on information known about the system [4] (Fig. 1). By testing an existing biological system under different conditions, we increase our knowledge of its behavior. The observed results from these tests can then be formalized to generate a set of behavioral "requirements." The inferred rules governing the system's behavior can then be used to construct a mechanistic model. As in engineered systems, these "requirements" can also be used to verify the mechanistic model's "correctness," that is, its consistency with the laboratory observations on which it was based [4] (Fig. 1). The biological model will optimally be based on all the available relevant information but may also include educated assumptions about how the system functions [5].

In computer science, there are a number of accepted terms used for ways to show that a system satisfies its requirements. *Testing* is used mainly to describe the fact that one runs, or executes, the system or a model thereof on some inputs. In most cases, the set of inputs is infinite or impractical. The theory of testing is concerned with finding a "good" set of tests, which serve to somehow cover most important cases. *Verification* is used to describe a rigorous mathematical and/or algorithmic process, whereby one proves that the program satisfies the specification. Successful verification leaves no doubts as to the system's correctness relative to the precise specification of the requirements. Model checking is one of the most widely

- *A. Sadot and D. Harel are with the Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: {avitals, dharel}@weizmann.ac.il.*
- *D. Barak can be reached at 12/4 Ha'Midron Street, Herzliya, Israel, 46541. E-mail: dan.barak@gmail.com.*
- *J. Fisher is with Microsoft Research, 7 J J Thomson Ave, Cambridge CB3 0FB, UK. E-mail: jasmin.fisher@microsoft.com.*
- *Y. Admanit can be reached at POB 316, Elkana, postal code 44814, Israel. E-mail: yishai.admanit@gmail.com.*
- *M.J. Stern is with the Department of Genetics, Yale University of School of Medicine, I-354 SHM, PO Box 208005, New Haven, CT 06520-8005. E-mail: michael.stern@yale.edu.*
- *E.J.A. Hubbard is with the Department of Biology, New York University, 1009 Silver Center, 100 Washington Square East, New York, NY 10003-6688. E-mail: jane.hubbard@nyu.edu.*

Manuscript received 8 June 2006; revised 18 Dec. 2006; accepted 14 Feb. 2007; published online 27 Mar. 2007.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-0124-0606.
Digital Object Identifier no. 10.1109/TCBB.2007.1076.

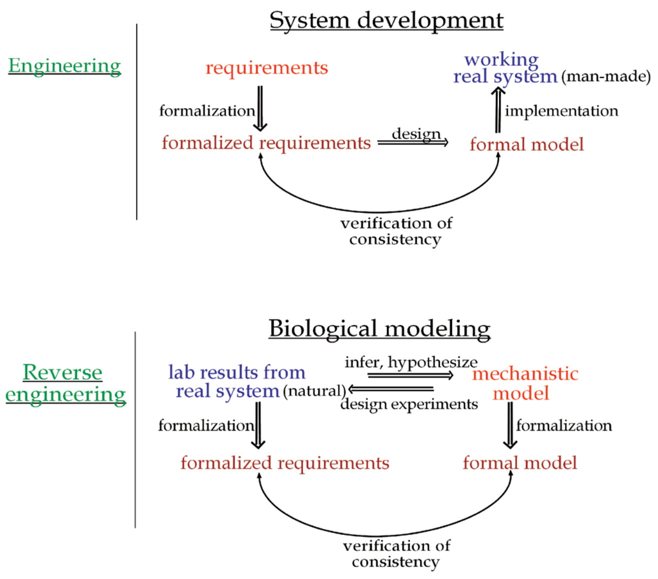


Fig. 1. Building engineered systems versus modeling and model verification of biological systems. The color code emphasizes the analogy between the "real" man-made system and the lab results from the actual biological system (both in blue) and the analogy between the requirements of the design of the man-made system and the hypothesized mechanistic model of the biological system (both in red). The formalized stages of both processes are shown in brown.

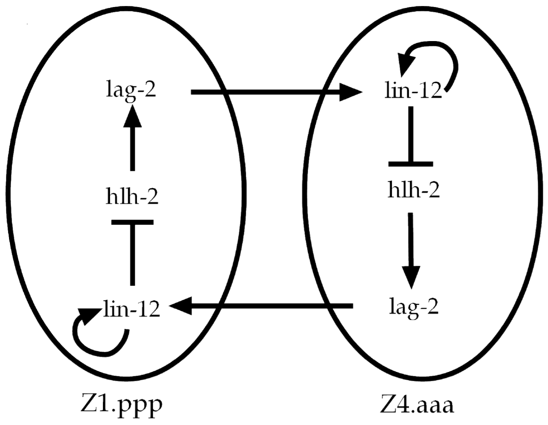
used techniques for carrying out verification [6]. *Exhaustive testing* is the term used for the case where the testing actually covers all possibilities, so that the result is the same as that for verification. This can only apply when the number of possibilities is finite and practically small and is therefore not normally possible and is rarely done, except in relatively small-scale cases. Thus, when building engineered systems, testing and/or verification are used to make sure that the system behaves in the desired fashion.

In building biological models, a process analogous to testing is the reconsideration of each of the laboratory-based results that were used to formulate the mechanistic model as a means to determine if the model is a sufficient reflection of the current understanding of the system [4]. Additional laboratory-based experiments are then designed to test hypotheses generated by the model. The fact that there is a finite set of tests (the laboratory data) from which mechanistic models are initially formulated, suggests the possibility that such a model can potentially be verified by exhaustive testing, that is, its compatibility with the data set used to generate it can in fact be determined. Having validated a biological model for a specific data set, new experimental results can be added to further challenge the validity of the model, which can be appropriately updated. Provided adequate means to formally represent 1) the biology represented in a mechanistic model and 2) the experiments that generated the model, the informal process of biological mechanistic-model building and testing can be made amenable to formal verification methods.

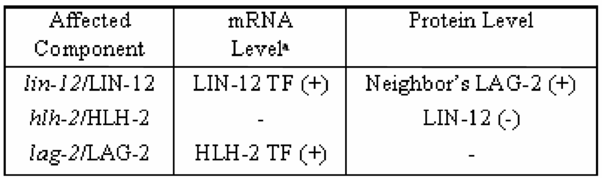
Another shared feature of biological and engineered systems is that both display functional modularity. A functional module is a distinct unit whose function is separable from those of other modules [7]. Several areas of research have highlighted the modularity within biology, particularly in evolution and development [7], [8], [9], [10]. Because of the growing volume and complexity of biological data, the synthesis of the various aspects of biological analysis into a complete systemic model and the illustration of its functional modules have come to rely more and more on mathematical and computational methodologies.

The challenge of developing computational models of biological systems has been the focus of many studies. Different groups use different modeling methodologies, including differential equations [11], Petri nets [12], and process algebra and pi calculus [13], [14]. The work presented in [15] shows an example of representing formal executable models of biology using the rewriting logic language Maude. Another approach offers models based on Markov chains and the use of a continuous stochastic logic and the probabilistic symbolic model checker PRISM [16]. BIOCHAM is a programming environment for modeling biochemical systems, carrying out simulations, and querying the model in temporal logic. It also provides an interface to the symbolic model checker NuSMV [17]. Other work use hybrid systems to model biological phenomena [18], [19]. Different methodologies depend on the type of system being modeled and the questions being addressed. Our group utilizes two main methods: an intraobject, state-based approach using statecharts and Rhapsody, and an interobject, scenario-based approach using LSCs and the Play-Engine [3], [20], [21], [22], [23]. These methods have proven applicable to biology, including the field of developmental genetics that, because of its reliance on genotype-phenotype relationships as opposed to more quantitative measures such as reaction diffusion kinetics, is relatively refractory to quantitative modeling methodologies. The modeling formalisms used in our group can be equally well implemented using other methods, for example, state-based models can be described using Petri Nets instead of statecharts, and scenario-based models can be portrayed in temporal logic instead of LSCs.

Another aspect of modularity in biology stems from the fact that different biological systems are investigated using different experimental approaches and raise different kinds of questions. Accordingly, we and others have proposed that distinct aspects of the modeling strategies may be amenable to distinct computational approaches. We further proposed that experimental observations can be formalized and then used to verify that a formalized proposed mechanistic model is consistent with the data upon which it was based [4]. The present paper implements these ideas by creating a modular model that integrates the scenario-based and the state-based approaches using the simulation engine coordinator InterPlay [24]. We used both LSCs and statecharts to create a mechanistic model that focuses on a well-characterized cell fate decision that is part of the development of the nematode *Caenorhabditis elegans (C. elegans)* hermaphrodite somatic gonad—the anchor cell/ventral uterine cell (AC/VU) decision. The mechanistic model includes both inferences from available genetic data (a genetic interaction pathway), as well as previously unmeasured quantitative features of the pathway (for example, synthesis and degradation rates for components of the pathway). For testing, we generated LSCs to



(a)



Affected Component	mRNA Level[a]	Protein Level
lin-12/LIN-12	LIN-12 TF (+)	Neighbor's LAG-2 (+)
hlh-2/HLH-2	-	LIN-12 (-)
lag-2/LAG-2	HLH-2 TF (+)	-

(b)

Fig. 2. (a) Gene interaction in the AC/VU decision. The interaction between Z1.ppp and Z4.aaa is mediated by the receptor LIN-12 and the ligand LAG-2. During the AC/VU decision, *hlh-2* is required for *lag-2* transcription and is down regulated posttranscriptionaly by LIN-12. Arrows represent positive regulation and bars represent negative regulation. Adapted from that in [34] and [33]. (b) Activity level at which regulation is modeled. All components have basal rates of production and degradation for both the mRNA and protein in addition to these specific regulatory effects. [a]TF, transcription factor; (+), activation; (−) inhibition.

represent the results of genetic and anatomical perturbation experiments, from which many of the key aspects of the mechanistic model were originally derived. We then used these LSCs to verify computationally that the assumptions and hypotheses in the model are consistent with the biological observations. We divided the biological data between the two formalisms in a way that seemed to us to be the most natural and gave rise to a more intuitive model construction and execution. Thus, we modeled the behavioral aspect of the biological system in the scenario-based formalism and the mechanism that underlies this behavior in the state-based formalism.

The development of the *C. elegans* somatic gonad starts from two founder cells present at hatching, Z1 and Z4, which divide two to three times during the first larval stage (L1) to produce the 12 cells of the gonad primordium [25]. Ten cells of the primordium have invariant fates: eight precursors that later generate uterine and sheath/spermatheca cells and two terminally differentiated distal tip cells (DTCs), which lead the growth of the elongating gonad and play critical roles during germline development [26]. Two other cells of the primordium, named Z1.ppp and Z4.aaa, have naturally variable fates: In the unperturbed worm, one cell will become the terminally differentiated anchor cell (AC), and the other will become a ventral uterine (VU) precursor cell [25]. The final conformation of

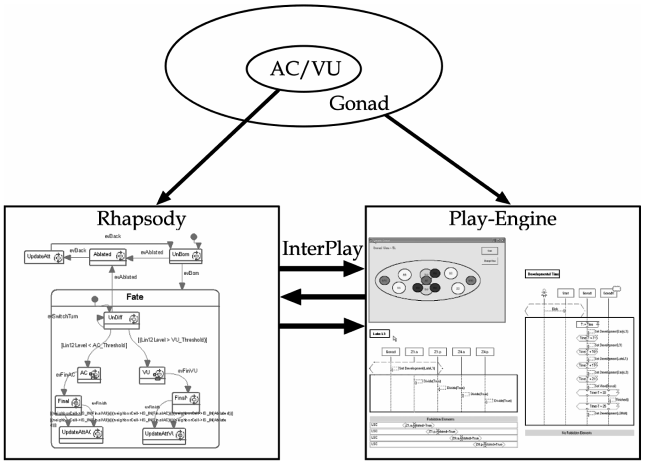


Fig. 3. Combining different tools for different modules. The AC/VU decision was modeled in the state-based formalism using Rhapsody, and the lineage of the somatic gonad was modeled in the scenario-based formalism using the Play-Engine. InterPlay was used to connect the two tools.

the somatic primordium depends on the outcome of the cell fate determination process between these two cells.

The AC/VU decision occurs during the L2 stage and depends on cell-cell interactions between Z1.ppp and Z4.aaa that are mediated by LIN-12, a receptor of the LIN-12/Notch family, and LAG-2, a ligand of the Delta-Serrate-LAG-2 (DSL) family (Fig. 2a) [27], [28], [29], [30], [31]. Z1.ppp and Z4.aaa have the potential to acquire either the AC or the VU fate, but in wild-type animals, only one becomes the AC, and the other becomes a VU (in 50 percent of the animals, Z4.aaa becomes the AC, and Z1.ppp becomes a VU, and vice versa, for the other 50 percent). Initially, Z1.ppp and Z4.aaa both express *lin-12* and *lag-2*. The current understanding of the mechanism whereby these two cells acquire stable mutually exclusive fates is that an initially small difference in *lin-12* activity between the cells triggers the amplification of ligand and receptor expression such that the cell with slightly higher *lin-12* activity continues to transcribe *lin-12* and ceases to transcribe *lag-2*, whereas the cell with lower *lin-12* activity continues to transcribe *lag-2* and ceases to transcribe *lin-12*. Ultimately, the cell expressing high levels of *lin-12* becomes a VU, and the *lag-2* expressing cell becomes the AC [27], [29], [32], [33]. In addition, *hlh-2* promotes *lag-2* transcription during the AC/VU decision. Based on the different patterns of expression of a *hlh-2* transcriptional reporter versus HLH-2 protein, it has been proposed that HLH-2 is posttranscriptionally downregulated in the presumptive VU upon LIN-12 activation as part of the negative feedback mechanism that leads to the termination of *lag-2* transcription [34] (Fig. 2a).

Other examples of modeling Delta-Notch type decisions include using PDEs [35], gene networks [36], hybrid automata [18], and Petri nets [37].

The modular model we present here simulates the early lineage of the somatic gonad cells in the scenario-based formalism and focuses on the AC/VU decision in the state-based formalism. We used InterPlay to integrate the two formalisms (Fig. 3).

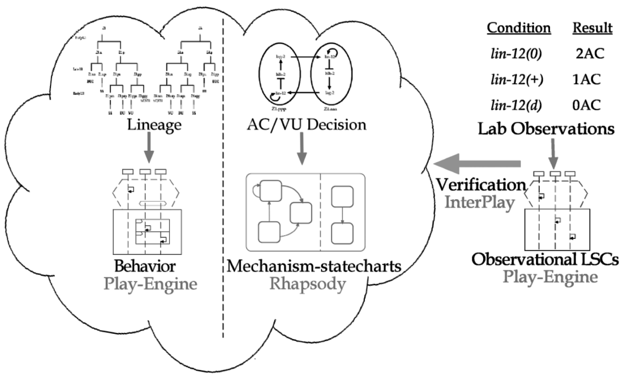


Fig. 4. Model verification. After we constructed the model using Rhapsody and the Play-Engine, we used the Play-Engine again to test the integrated model. We created a set of simple LSCs based on lab observations and used InterPlay to run the combined Play-Engine-Rhapsody mechanistic model against them.

We further composed a set of LSCs representing a key subset of the experimental data upon which the mechanistic model was based, and again, via InterPlay, we used them to verify that the integrated model produced the expected outcomes (Fig. 4).

2 Methods

For more elaborate explanations of the modeling formalisms, please see Appendix A.

2.1 Statecharts

Statecharts is a visual formalism, developed in 1983 as a language for specifying reactive behavior [38]. In its object-oriented version [39], one uses statecharts to define the behavior of objects over time, in an intraobject fashion, based on the various states that an object can be in over its lifetime and the events that cause it to move from one state to another. Statecharts describe both how objects communicate and collaborate and how they carry out their own internal behavior under different circumstances. Thus, states are actually abstract situations in an object's life cycle. The language is well structured and hierarchical and is thus relatively easy to deal with even by nonspecialists.

2.2 Rhapsody

Rhapsody is a software tool for the design of statechart-based models [39] (http://modeling.telelogic.com). It can automatically translate a statechart model into executable C, C++, or Java code. Once the model (or some part thereof) is constructed and translated into executable code, Rhapsody can execute it so that one can observe its progress, in animated versions of the model's statecharts. During animation, active states and transitions are shown in a different color. The code can also be linked up with a graphical rendition of the system being specified, a so-called GUI, so as to obtain a realistic simulation of the system in operation.

2.3 LSCs

LSCs constitute a visual formalism for specifying sequences of events and message passing between objects [40]. The behaviors are specified as scenarios of events and actions, with a variety of possibilities including scenarios that may occur, scenarios that must occur and scenarios that are forbidden (called antiscenarios). There are two types of LSCs, universal and existential. Universal charts are more relevant for modeling and are built of a prechart and main chart. The relationship between the prechart and the main chart can be viewed as a condition-result pair (see Fig. 5a): Whenever the scenario in the prechart occurs (condition), the scenario in the main chart must follow (result) [40].

2.4 Play-Engine

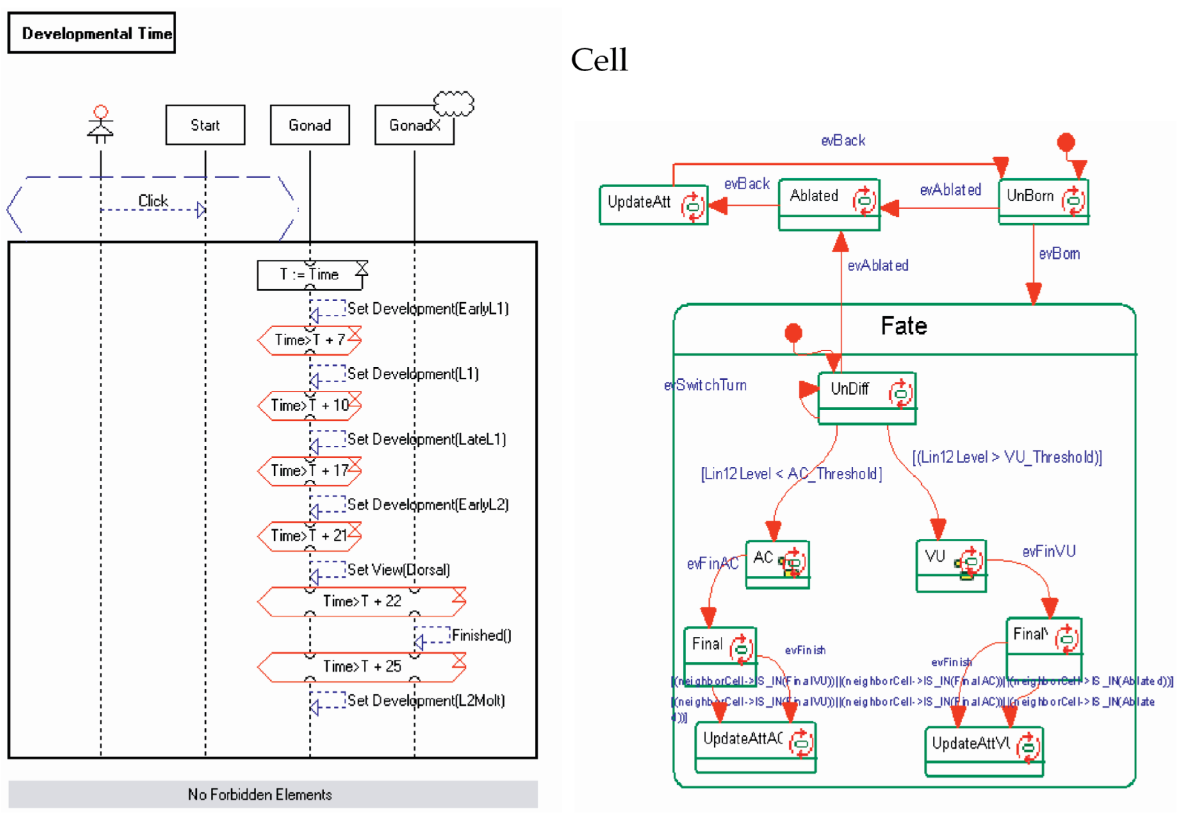
The Play-Engine is a recently developed tool that supports modeling and model execution with LSCs. The Play-Engine supports the play-in/play-out methodology, in which one can easily represent interobject behavior and execute and simulate a modeled system [41]. Using play-in, LSCs that specify system behavior can easily be generated with a user-friendly mechanism. The user first builds a graphical user interface (GUI) of the system, with no behavior integrated in it and then "plays" the GUI by clicking the graphical control elements in an intuitive manner. In this way, the Play-Engine constructs the corresponding LSCs, which determine the sequences of events and actions, and how the system should respond to them. Thus, play-in is analogous to writing programs that determine system behavior.

Accordingly, play-out is analogous to running these programs, in that it allows execution of the LSCs. Using play-out, the user simply plays the GUI as he or she would have done while executing a real system (for example, clicking the "On" button on a computer). As this is happening, the Play-Engine interacts with the GUI and uses it to exhibit the system's response over time. Events occurring in the LSCs that govern system behavior during play-out are represented in the GUI, so that the user may view the full modeled behavior of the system operating simultaneously. For more on LSCs, play-in/play-out, and the Play-Engine see [41].

2.5 Interplay

InterPlay is a simulation engine coordinator that supports cooperation and interaction of multiple simulation and execution tools, thus helping in the scale-up process of designing large reactive systems [24]. Among other things, InterPlay enables the connection of the Play-Engine and Rhapsody and can be used in distributing large systems into their parts while retaining the ability to execute them in tandem [24].

The connection of the Play-Engine with Rhapsody via InterPlay is done through external non-GUI objects. External objects are mirror images in the local system of objects in a remote system, serving as an interface to it. As such, their structure (properties and methods) is known by the local system, as are the elements of their behavior that are relevant to this system. Behavioral changes in the remote objects are reflected to the mirror images, and vice versa. The technical aspects of this reflection are carried out by InterPlay. It is important to stress that the behavior of the external objects is driven only by the remote system.



(a)

(b)

Fig. 5. Examples from the model. (a) An example LSC. This LSC follows developmental time and manages the development of the gonad. If the user clicks the Start button (the prechart, surrounded by a blue dashed line), then the Gonad object starts to measure time, and advances through the developmental stages accordingly (the main chart, surrounded by a black rectangle). The external `Gonad` object, which is intended for communication with InterPlay, is indicated with a small cloud on the right upper corner. (b) The statechart of the Cell class. The cell starts from the `unborn` state, and from there, it can advance either to the `Ablated` state or to the `Fate` state. In the `Ablated` state, the cell's participation in the decision is terminated. In the `Fate` state, the process of fate determination takes place, at the end of which, the cell either becomes an AC or a VU. Green rectangles represent states. There could be hierarchy among the states as seen in the `Fate` state, which contains a number of inner states. Arrows at the right corner of states mean that there is a code written within them. Red arrows represent transitions between states. Filled circle arrow heads represent transitions into a default state. Blue represents events or conditions. This figure shows examples from the separate statecharts and LSC models and obviously does not depict the information flow between them.

3 Results

3.1 Model Structure

Since our model is modular and involves the continuous interaction of different modeling tools, we will describe its structure by following the progress of an execution.

The model starts at the beginning of the lineage of the somatic gonad. This component is simulated using LSCs. The model of the lineage is constructed from two types of GUI objects—a Gonad object, which represents the somatic gonad as a separate entity, and the objects of the Somatic Cells, each representing a specific cell that belongs to the lineage that eventually composes the gonad. The GUI depicts the gonad, and the cells in cartoon form that reflects the activity of the underlying LSC events and that approximates their relative size and positions during development, as well as their lineal relationships and divisions (see Supp. Fig. 1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). All of the Cells objects belong to the same class and, therefore, have the same properties and methods. The LSCs that describe the lineage of the somatic gonad advance according to developmental time. Every Play-Engine clock tick represents one hour of the actual developmental time. The LSC `Developmental Time` (Fig. 5a) acts as a "manager" LSC that monitors the time and consequently sets the developmental stage of the Gonad object. This part of the model describes the lineage of the somatic gonad only up to the L2 molt. For each developmental stage that the gonad reaches, there is a suitable LSC in which the relevant cells change their `Divide` property to true (see Supp. Fig. 2, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). For each cell that divides, there is an LSC in which the divided cell sends its two daughter cells the method `Divided`. Another LSC states that each cell that receives the method `Divided` from another cell changes its `Born` property to true. As suggested in [34], the outcome of the AC/VU decision is influenced by the birth order of Z1.ppp and Z4.aaa. In the said work [34], the authors examined the Z1 and Z4 lineages of 13 worms and found that in 50 percent of the worms, Z1.ppp is born first, and in the other 50 percent, Z4.aaa is born first. Therefore, for Z1.ppp and Z4.aaa, the LSC `Stochastic`

TABLE 1
Testing

Test name[a]	Implementation	Experimental results	Reference	Model results	Run duration[b]
Wild type				One cell becomes the AC and the other a VU	111
lin-12(0)	*lin-12* protein and mRNA levels set to 0.	In *lin-12(0)* hermaphrodites, both Z1.ppp and Z4.aaa become anchor cells	[21]	Z1.ppp and Z4.aaa both become ACs	4
lin-12(d)	Elevate LIN-12 translation rate*	In *lin-12(d)* hermaphrodites, both Z1.ppp and Z4.aaa become ventral uterine precursor cells	[21]	Z1.ppp and Z4.aaa both become VUs	6
Isolate Z1.ppp and Z4.aaa (2)	Ablate somatic gonad cells in one of the following combinations: Z1.a, Z4.p, Z1.pa, Z4.ap, Z1.ppa and Z4.aap or Z1.aa, Z1.ap, Z4.pp, Z4.pa, Z1.pa, Z4.ap, Z1.ppa and Z4.aap as soon as they are born	Fates of Z1.ppp and Z4.aaa for the AC/VU decision are unaffected by ablation of other somatic gonadal cells	[26]	One cell becomes the AC and the other a VU, same as in wild type	111
Isolate Z1.ppp or Z4.aaa in a lin-12(d) background (4)	Elevate LIN-12 translation rate, ablate cells as described above, as well as either Z1.ppp or Z4.aaa.	An isolated Z1.ppp or Z4.aaa cell in *lin-12(d)* mutants becomes a VU	[26]	The isolated cell (either Z1.ppp or Z4.aaa) becomes a VU	8
Mosaic analysis (2)	Set *lin-12* protein and mRNA levels to 0 in relevant cells.	In mosaics that lack *lin-12* activity only in the Z1 or Z4 lineages, the *lin-12(0)* cell always becomes the AC	[26]	The cell that has the *lin-12(0)* genotype becomes the AC. The other cell becomes a VU	15
Ablate Z1 or Z4 (2)	Ablate Z1 or Z4.	Ablation of either Z1 or Z4 results in one AC (52/53 animals)	[35]	The remaining cell becomes the AC	20
Ablate presumptive AC (2)	Ablate Z1.ppp or Z4.aaa when one has a significantly lower activity of LIN-12	Ablation of the pre-AC during early formation of the somatic gonad primordium results in one AC	[35]	The cells that was not ablated becomes the AC	60
hlh-2 RNAi	Increase degradation rate of *hlh-2* mRNA.	Depletion of *hlh-2* by RNAi feeding in the early L2 (*hlh-2(RNAi-L2)*) produces a 2 AC phenotype	[28]	Z1.ppp and Z4.aaa both become ACs	45
lag-2(lf)	Set *lag-2* mRNA and protein levels to 0.	Reduction of *lag-2* causes a 2 AC phenotype	[23]	Z1.ppp and Z4.aaa both become ACs	35
lin-12 ΔLCS1	Decrease the influence of LIN-12 as a self-transcription factor to 0.	*lin-12(+)* driven from an LCS1-deleted promoter does not efficiently rescue the 2AC defect in a *lin-12(0)* mutant background	[27]	Z1.ppp and Z4.aaa both become ACs	30
lin-12 ΔLCS1 in a lin-12(d) background	Decrease the influence of LIN-12 as a self-transcription factor to 0, and elevate LIN-12 translation rate.	Expression of a reporter driven from an LCS1-deleted *lin-12* promoter in a *lin-12(d)* background is not observed in the presumptive VU	[27]	Z1.ppp and Z4.aaa both become ACs	24

Z1, Z4, Z1.a, Z4.p, and so on are names of specific cells in the somatic gonad lineage.
[a] *Numbers in parenthesis indicate the number of variant LSCs per test name if greater than 1.*
[b] *Run duration is measured in the average number of total rounds that the model goes through to reach completion (that is, until it reaches the* `FinalAC` *and* `FinalVU` *states).*
* *All protein produced is assumed to be active or activateable (see text for further explanation).*

`Event` decides randomly which of the two cells is born first (that is, changes its `Born` property to true first, see Supp. Fig. 3, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). This LSC gives equal chances to both cells to be the first born cell. The change to true of the `Born` property of Z1.ppp and Z4.aaa is sent to Rhapsody via InterPlay, and this triggers the cells in the Rhapsody model to start the cell fate determination process.

The model of the AC/VU decision, which is simulated in Rhapsody, is composed of a Gonad class and a Cells class. The Gonad class aids in the initialization of the simulation (that is, setting the initial conditions for a particular execution). The Cell class consists of two instances of the same statechart—one for Z1.ppp and one for Z4.aaa. Consistent with their biological behavior as an equivalence group (that is, a group of cells with the same developmental potential) [25], [32], [42], Z1.ppp and Z4.aaa start the process in the wild type (here, implying the absence of genetic or anatomical perturbations) with the same initial conditions. During the run, however, their gene and protein levels change as a result of their interactions. Below, we describe the statecharts of the various classes, starting with that of the Gonad.

3.1.1 Gonad Class Statechart

The instantiation of the Gonad statechart starts in the `InitConditions` state (see Supp. Fig. 4, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). This state has a substatechart that specifies the variety of initial conditions, according to the different perturbations used to run and test this model (Table 1) (see Supp. Fig. 5, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). The default initial condition is the wild type.

3.1.2 Cell-Class Statechart

The Cell class describes the modeled processes that take place within and between the two cells. *lin-12*, *lag-2*, and *hlh-2* are represented at transcriptional, translational, and posttranslational levels, representing the production of mRNA and protein products (see Supp. Table 1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). Although exact levels of these molecules have not been measured experimentally, we assigned relative values that were adjusted during model building to produce the desired behavior. There is also a representation of mRNA and protein degradation. This set of values produced a result of one cell becoming an AC and, the other, a VU every time we run the system. If we change one or several of these values, the system can oscillate. However, there could be other sets of values for which the ratio between them will produce similar results. We assume that all produced protein is active or able to be activated: In the case of the wild-type LIN-12 receptor, the activity of the protein is contingent on the neighboring cell producing the ligand. All of these properties are depicted as attributes of the class Cell. The processes of transcription and translation and the interaction between the cells are implemented in C++ as operations (see Supp. Table 2, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076).

For each of the genes, the calculation includes a basal transcription and degradation rate for the mRNA and, also, a basal translation and degradation rate for the protein. Additional terms are included in the operations to account for the known regulatory steps in this system (see Figs. 2a and 2b). These include 1) the effect of LIN-12 as a self-activating transcription factor on the *lin-12* mRNA level, 2) the effect of LAG-2 of the neighbor cell as a signal that activates LIN-12, and 3) the effect of HLH-2 as a transcription factor on the *lag-2* mRNA level. In addition, since LIN-12's negative effect on HLH-2 is postulated to be post-translational [34], the protein level of HLH-2 is computed not only as a factor of the protein's basal translation and degradation rates, but also as a factor of the influence of activated LIN-12 as a negative regulator.

The strength of the effect of a transcription factor on the transcription rate of its target gene is described by a monotonic s-shaped function [43]. This function is used each time there is a calculation of a transcriptional regulation interaction between two elements in the system (see Fig. 2b). It determines the extent of the influence one element has over the activity level of another element, taking into consideration the current activity level of the influencing element. We also introduce into the system a random level of noise (between -5 percent and $+5$ percent) each time we update both mRNA and/or protein levels of each of the three key components. This simulates the natural biological fluctuations of the efficiency of these processes.

3.2 A Run of the Model

At each execution of the model, there are two active copies of the statechart for the class Cell—one for Z1.ppp and one for Z4.aaa. Each copy of the statechart has its own list of attributes and operations that change during the run (see Supp. Tables 1 and 2, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). The statechart of the Cell class starts at the `UnBorn` state (Fig. 5b). Once Z1.ppp and Z4.aaa are born according to the lineage modeled in the Play-Engine, the message `evBorn` is received by Rhapsody via InterPlay.

The Cell statechart then advances into the state `Fate` (Fig. 5b), in which the model represents the process of cell fate determination, including a set of two substatecharts to separate events that occur upon reaching the "decision threshold" of LIN-12 levels from the final stage of cell fate acquisition. The Fate state starts at the `UnDiff` (for Undifferentiated) state (Fig. 5b). In this state, the level of the genes and proteins are updated. The fate of the cell is determined when the level of LIN-12 protein in a cell reaches one of two critical levels—either a lower-bound AC threshold or an upper bound VU threshold. If the level of LIN-12 drops below the AC threshold, the cell advances to the `AC` state (Fig. 5b), in which a substatechart ensures that the `FinalAC` fate is not reached until the `AC` state is stable, that is, has reached an additional (arbitrary) 1 unit below the AC threshold (see Supp. Fig. 6, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). In this state, a loop updates the LIN-12 level within the `maintainAC` state by iterating the same calculation method that was used in the `UnDiff` state. Once this one-unit difference is achieved, the cell advances to the `FinalAC` state. Similarly, if at the end of the `UnDiff` state the level of LIN-12 protein in the cell exceeds the VU threshold, the cell advances to the `VU` state (Fig. 5b). Again, this state has a substatechart that assures that this level of LIN-12 is maintained before the cell advances to the `FinalVU` state.

When the cells in the Rhapsody model reach their final fate, the result is sent back to the Play-Engine, again through InterPlay, and the appropriate final conformation of the somatic gonad primordium—5R or 5L [25]—is set and displayed via the GUI (see Supp. Fig. 1c, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). An example of an outcome of a run of the model under normal, wild-type conditions can be seen in Supp. Fig. 7, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076.

3.3 Model Testing

We created a set of 18 LSCs (Table 1) to test the model, each of which represents a condition-result experiment. These LSCs consist of a prechart that states the initial condition of the experiment (for example, *lin-12(0)* indicates that *lin-12* is homozygous for a null allele) and a main chart that describes the end result of that condition (Z1.ppp and Z4.aaa both become ACs, for example, see Supp. Fig. 8, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). These LSCs do not contain any information about the mechanism that led to the result. Methods of an object in the Play-Engine model are mapped to events of its mirror object in the statecharts model. Thus, when such a method is called, the appropriate event is triggered; the object moves to the indicated state, and the initial conditions are updated accordingly.

In the Rhapsody component of the model, the statechart of the Gonad initializes the conditions for each test. For example, for *lin-12(0)*, the mRNA and protein levels of *lin-12* in both Z1.ppp and Z4.aaa are set to zero (Table 1). When running the model, the user can choose which test to conduct, either through Rhapsody or through the Play-Engine. The user may also define new tests by constructing the suitable LSCs and/or states. The appropriate events are sent via Inter-Play, and they consequently trigger the progression of the model. Once the model finishes its execution, the test-LSCs are either satisfied or not satisfied. If they were satisfied, this implies that the outcome of the simulation driven by the mechanism depicted in the model is consistent with the laboratory observations represented by the LSCs.

We used the set of experimental observations shown in Table 1 and tested, one by one, the combined statechart-LSC model. In the course of applying these tests, we had to make small adjustments to the parameters of the mechanistic model and to check again if it displayed the desired behavior. Eventually, all of the test-LSCs were satisfied, meaning that the mechanistic model we built was consistent with the laboratory observations tested (see Supp. Figs. 9-17, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2007.1076). For laboratory observations in which the experimental outcomes were nondeterministic, our model is deterministic and therefore produces the most frequent results. That is, where the biological outcome was incomplete penetrance of a given phenotype, our model only produces the most penetrant phenotype. Future modeling work will address a more realistic representation of nondeterministic outcomes. Though the model and the tests thereof are not comprehensive—that is, these are but a small subset of the relevant experiments reported in the literature that relate to the AC/VU decision—this set is sufficient to illustrate the process, utility, and further potential of this style of modeling and model verification.

4 Discussion

Complex systems are built by combining together simpler parts of the system. The process of modeling biological systems requires the integration of the mechanistic rules by which the smaller pieces operate. In this paper, we combined two formal modeling approaches in computer science that were originally developed for the systems design field. These were used to construct a model for certain aspects of the development of the somatic gonad of *C. elegans*. In particular, we focused on the AC/VU decision. We then used one of the approaches to formally test the integrated model using a defined set of biological condition-result experiments.

As biological processes are studied, the relevant data frequently have distinct features. For instance, some of the data are observations of normal behavior, whereas other data are obtained after specific perturbations of the normal system. The mechanisms underlying the behavior under normal conditions are often inferred from the results of experiments conducted under perturbed conditions. In addition, data from many different aspects of the biology are often combined into a mechanistic model. For example, the behavior of a condition that alters the activity of a given gene can be combined with information about the identity of the protein encoded by the gene and from relevant biochemical experiments. These combined inferences are collected into a mechanistic model from which testable hypotheses are derived and then lead to additional experiments. It was previously suggested [4] that by using a scenario-based approach (LSCs) to formalize the observed behaviors and experimental perturbations of a biological system and a state-based approach (statecharts) to formalize the mechanisms underlying these behaviors, one can formally verify that the mechanistic model reproduces the system's known behavior.

Here, we have followed this idea: We used different computational approaches to model different aspects of the system. As the lineage of the somatic gonad cells is more intuitively depicted in the form of scenarios, we chose to describe it using the interobject approach of LSCs. We also used an LSC-based approach to represent condition-result laboratory experiments and their outcomes. The AC/VU decision, however, is a continuous process, consisting of feedback loops among key components that influence the states of the two cells, Z1.ppp and Z4.aaa. Moreover, based on additional more general knowledge about genetic information transfer and the dynamic behavior of the mRNA and protein components of the system, we incorporate into the statecharts model additional quantitative features, some of which have not yet been measured directly in the lab. We chose to model this part of the system using the intraobject approach of statecharts to represent the interactions between three of the crucial components that regulate this cell fate decision and their molecular dynamics. Thus, our mechanistic model includes quantitative aspects of the system that may provide additional insights as future laboratory measurements are made. We used InterPlay as an interface connecting the LSC and statecharts-based aspects of the model (Fig. 3).

This approach has several more broadly applicable advantages. Each module is a stand-alone model. Thus, we can choose to explore different aspects of the systems separately by looking at each module by itself or to investigate the complete system by looking at the integrated model. This flexibility is useful on several levels: When building the model, one can concentrate on developing a single component without influencing other components of the system. It is also possible to distribute the modeling work between several investigators/developers, each responsible for a single module. Then, for the complete system, all modules can be connected. Another advantage is that investigators interested in diverse facets of the modeled system can look at the processes that interest them on their own or as a part of the full system.

We used the same modular approach for the verification of the model. The statecharts-based model incorporates inferences from a wide set of studies. Using a small set of core behaviors, we were able to demonstrate that this model can reproduce these fundamental behaviors. To do this, we summarized this key set of previously published laboratory observations in the form of LSCs and used them to test that the mechanistic model we constructed was consistent with these laboratory observations (Fig. 4). Thus, we allowed the Play-Engine to follow the combined Rhapsody and Play-Engine model execution and ensured that our mechanistic model matches these experimental observations. This approach demonstrates the potential for model verification. The modular approach is very convenient, because it enables us to test the components of the system either separately or combined. Further tests can be used to help develop a more complete model of this system. Since our mechanistic model also includes previously unmeasured quantitative aspects of the system, this type of modeling can serve to simulate experiments to determine these important values.

The AC/VU decision is a process of cell fate determination between two initially equivalent cells. This process is mediated by members of the Delta/Notch gene family—LIN-12, a receptor from the LIN-12/Notch family, and LAG-2, a member of the DSL (Delta-Serrate-LAG-2) family. Members of the Delta/Notch family are involved in such processes in various organisms [45]. The proposed mechanism for this cell fate determination is similar to leader-election algorithms in computer science. These algorithms are designed to solve a problem in which a leader needs to be chosen in a network of initially identical elements. A natural observation is that if all elements are identical, the problem cannot be solved deterministically, and one unique leader cannot be elected [46]. This implies that the only way to solve the leader-election problem is to somehow break the symmetry. The assumption made in some of the algorithms designed to solve this problem is that each element in the network starts the process with some unique identifier (sometimes chosen at random), which distinguishes the elements and makes it possible to break the symmetry. The elements in the network then communicate with each other and send their identifiers across the network. Eventually, a leader is elected according to the nature of these identifiers [46]. The change in symmetry in the biological interactions is presumed to be due to some kind of stochastic event, which gives one of the cells the advantage in adopting the leader fate [45]. In the AC/VU decision, this event is biased by the birth order, which enables one of the cells to start accumulating LIN-12 before the other [34]. The LIN-12 activity level could act as the unique identifier in the algorithm by which the leader is eventually chosen.

The modular nature of our modeling approach makes it easily expandable. One can simply connect additional modules of the system to the existing configuration using any computational tool desired. Furthermore, since every component of the model is stand alone, it is possible to choose just one of the components and incorporate it into another system. Thus, this model can be integrated into the ongoing efforts in our group to model *C. elegans* vulval development [21], [23]. Another plausible expansion of this model is the construction of additional aspects of *C. elegans* gonadogenesis.

Appendix A
Modeling Formalisms

Parts of this appendix were adapted from that in [3], [24], and [47].

A.1 The State-Based Formalism

In our work, we use the language of statecharts [38] and the Rhapsody tool [39] (http://www.ilogix.com) to implement the state-based specification. Statecharts are naturally suited for the specification of objects that have clear internal behavior, an approach that has been called *intraobject* [40]. Together with object model diagrams, they provide a graphical representation of the dynamics of objects using states, transitions, events, and conditions [39].

Object model diagrams [39] show the static structure of a system. They describe the types of objects in the system, the attributes and operations that belong to those objects, and the static relationship that can exist between classes.

Statecharts define the behavior of objects [38], [39], including the various states that an object can enter into over its lifetime and the messages or events that cause its transition from one state to another. A statechart attached to a class specifies the behavior of all instances thereof. The language makes it possible to visualize the behavior of an object in a way that emphasizes the elements in its life cycle.

Objects can communicate by exchanging messages between their statecharts. In order to communicate, two objects must be related by some kind of association. Via these associations, as defined in the object model diagram, one object can refer to its associated objects. The communication between objects can be effected either by event generation or by invoking triggered operations.

Rhapsody is a model-driven development environment supporting statecharts and object model diagrams [39], (http://www.ilogix.com). In addition to providing a computerized visual design environment for performing object-oriented modeling, Rhapsody is capable of automatically translating any syntactically legal model into an executable code (in C, C++, or Java). Once the application is built, Rhapsody can animate the running application so that one can observe its progress in animated versions of the model's statecharts. Animation shows the current states and transitions by coloring them uniquely. In Rhapsody, the transitions between graphical diagrams and executable code are bidirectional: During the compilation stage, the object model diagram and the statecharts are translated into executable code, and while animating the application, the executable code is translated back into diagrams.

A.2 The Scenario-Based Formalism

We use the language of LSCs [40], [41] and the Play-Engine [41] tool to implement the scenario-based formalism. LSCs are scenario-based and *interobject* in nature and are particularly suited for describing behavioral requirements. LSCs extend classical message sequence charts (MSCs) with logical modalities, depicted as hot and cold elements in the charts. The language thus achieves far greater expressive power than MSCs and is comparable to that of temporal logic [48]. In particular, LSCs can specify possible mandatory and forbidden scenarios and can be viewed as specifying multimodal restrictions over all possible system runs.

LSCs have two types of charts, universal and existential. Universal charts are used to specify restrictions over all possible system runs and, thus, constrain the allowed behaviors. A universal LSC typically contains a prechart and a main chart. The semantics is that if the scenario in the prechart executes successfully, then the system is forced to satisfy the scenario given in the main chart. Existential charts, on the other hand, specify sample interactions between the system and its environment and are required only to be satisfied by at least one system run. Thus, existential LSCs do not force the application to behave in a certain way in all cases, and simply illustrate longer (nonrestricting) scenarios that provide a broader picture of the behavioral possibilities to which the system gives rise.

LSCs may contain two types of conditions—hot and cold. Hot conditions are mandatory and must always be true; if not, the requirements are violated and the system aborts. However, when dealing with time, the system simply waits until the specified condition holds. On the other hand, if a cold condition is false, the surrounding subchart is exited. An LSC can also contain scoped forbidden elements, listed in a separate area beneath the main chart. For example, a hot forbidden condition that becomes true within its scope causes the requirements to be violated and the system aborts, whereas a cold one becoming true causes the chart or subchart which is its scope to be exited.

The Play-Engine is the tool built to support LSCs [41]. It enables the system designer to capture behavioral requirements by "playing in" the behavior of the target system and to execute the specific behavior by "playing out." The play-in process requires that the user first build a GUI of the system, with no behavior built into it. The user then "plays" the GUI by clicking the graphical control elements and thus giving the engine sequences of events and actions and teaching it how the system should respond to them. As this is being done, the Play-Engine continuously constructs the corresponding LSCs automatically.

While play-in is the analog of writing programs, play-out is the analog of running them. Here, the user simply plays the GUI as he would have done when executing the real system (for example, clicking buttons). As this is going on, the Play-Engine interacts with the GUI and uses it to reflect the system's state at any given moment. The scenarios played in, using any number of LSCs, are all taken into account during play-out, so that the user gets the full effect of the system with all its modeled behaviors operating correctly in tandem. All specified ramifications entailed by an occurring event or action will immediately be carried out by the engine automatically, regardless of where in the LSCs it was originally specified.

Acknowledgments

This work was supported in part by The John Von Neumann Minerva Center for the Development of Reactive Systems, and by a grant from the Kahn Fund for Systems Biology, both at the Weizmann Institute of Science, and National Institutes of Health Grants F5490-01 (to D.H.) and R24-GM066969 (to E.J.A.H., M.J.S. and D.H.).

References

[1] D. Harel and A. Pnueli, "On the Development of Reactive Systems," *Logics and Models of Concurrent Systems,* NATO ASI Series F, K. R. Apt, ed., pp. 477-498, 1985.

[2] D. Harel, "A Grand Challenge for Computing: Full Reactive Modeling of a Multi-Cellular Animal," *Bull. of the EATCS, European Assoc. for Theoretical Computer Science,* no. 81, pp. 226-235, 2003.

[3] N. Kam, I.R. Cohen, and D. Harel, "The Immune System as a Reactive System: Modeling T Cell Activation with Statecharts," *Proc. IEEE Symp. Visual Languages and Formal Methods (VLFM '01), part of the IEEE Symp. Human-Centric Computing (HCC '01),* pp. 15-22, 2001.

[4] J. Fisher, D. Harel, E.J.A. Hubbard, N. Piterman, M.J. Stern, and N. Swerdlin, "Combining State-Based and Scenario-Based Approaches in Modeling Biological Systems," *Proc. Computational Methods in Systems Biology (CMSB '04),* pp. 236-241, 2004.

[5] D. Harel, "A Turing-Like Test for Biological Modeling," *Nature Biotechnology,* vol. 23, no. 4, pp. 495-496, Apr. 2005.

[6] E.M. Clarke, O. Grumberg, and D. Peleg, *Model Checking.* MIT Press, 1999.

[7] L.H. Hartwell, J.J Hopfield, S. Leibler, and A.W. Murray, "From Molecular to Modular Cell Biology," *Nature,* vol. 402, no. 6761 Suppl., pp. C47-C52, 1999.

[8] N. Kashtan and U. Alon, "Spontaneous Evolution of Modularity and Network Motifs," *Proc. Nat'l Academy of Sciences,* vol. 102, no. 39, pp. 13773-13778, 2005.

[9] P.J. Roy and Q. Morris, "Network News: Functional Modules Revealed during Early Embryogenesis in *C. elegans,*" *Developmental Cell,* vol. 9, no. 3, pp. 307-308, 2005.

[10] J.M. Stuart, E. Segal, D. Koller, and S.K. Kim, "A Gene-Coexpression Network for Global Discovery of Conserved Genetic Modules," *Science,* vol. 302, no. 5643, pp. 249-255, 2003.

[11] S.Y. Shvartsman, C.B. Muratov, and D.A. Lauffenburger, "Modeling and Computational Analysis of EGF Receptor-Mediated Cell Communication in Drosophila Oogenesis," *Development,* vol. 129, no. 11, pp. 2577-2589, 2002.

[12] J. Barjis and I. Barjis, "Formalization of the Protein Production by Means of Petri Nets," *Proc. Int'l Conf. Information Intelligence and Systems (ICIIS '99),* 1999.

[13] D.D. Errampalli, C. Priami, and P. Quaglia, "A Formal Language for Computational Systems Biology," *Omics,* vol. 8, no. 4, pp. 370-380, 2004.

[14] A. Regev, W. Silverman, and E. Shapiro, "Representation and Simulation of Biochemical Processes Using the Pi-Calculus Process Algebra," *Proc. Pacific Symp. Biocomputing,* vol. 6, pp. 459-470, 2001.

[15] C. Talcott and D.L. Dill, "The Pathway Logic Assistant," *Proc. Computational Methods in Systems Biology (CMSB '05),* pp. 228-239, 2005.

[16] M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton, "Analysis of Signaling Pathways Using the PRISM Model Checker," *Proc. Computational Methods in Systems Biology (CMSB '05),* pp. 179-190, 2005.

[17] L. Calzone, F. Fages, and S. Soliman, "BIOCHAM: An Environment for Modeling Biological Systems and Formalizing Experimental Knowledge," *Bioinformatics,* vol. 22, no. 14, pp. 1805-1807, 2006.

[18] R. Ghosh, A. Tiwari, and C. Tomlin, "Automated Symbolic Reachability Analysis with Application to Delta-Notch Signaling Automata," *Proc. Hybrid Systems: Computation and Control (HSCC '03),* pp. 233-248, 2003.

[19] R. Alur, C. Belta, F. Ivancic, V. Kumar, M. Mintz, G. Pappas, H. Rubin, and J. Schug, "Hybrid Modeling and Simulation of Biomolecular Networks," *Proc. Hybrid Systems: Computation and Control (HSCC '01),* pp. 19-32, 2001.

[20] S. Efroni, D. Harel, and I.R. Cohen, "Toward Rigorous Comprehension of Biological Complexity: Modeling, Execution, and Visualization of Thymic T-cell Maturation," *Genome Research,* vol. 13, no. 11, pp. 2485-2497, 2003.

[21] J. Fisher, N. Piterman, E.J.A. Hubbard, M.J. Stern, and D. Harel, "Computational Insights into *Caenorhabditis elegans* Vulval Development," *Proc. Nat'l Academy of Sciences,* vol. 102, no. 6, pp. 1951-1956, 2005.
[22] D. Harel, S. Efroni, and I.R. Cohen, "Reactive Animation," *Proc. First Int'l Symp. Formal Methods for Components and Objects (FMCO '02)* , invited paper, pp. 136-153, 2003.
[23] N. Kam, D. Harel, H. Kugler, R. Marelly, A. Pnueli, E.J.A. Hubbard, and M.J. Stern, "Formal Modeling of *C. elegans* Development: A Scenario-Based Approach," *Proc. First Int'l Workshop Computational Methods in Systems Biology (ICMSB '03),* pp. 4-20, Feb. 2003.
[24] D. Barak, D. Harel, and R. Marelly, "InterPlay: Horizontal Scale-Up and Transition to Design in Scenario-Based Programming," *IEEE Trans. Software Eng.,* vol. 32, no. 7, pp. 467-485, 2006, earlier version in *Lectures on Concurrency and Petri Nets,* LNCS 3098, Springer, pp. 66-86, 2004.
[25] J. Kimble and D. Hirsh, "The Postembryonic Cell Lineages of the Hermaphrodite and Male Gonads in *Caenorhabditis elegans*," *Developmental Biology,* vol. 70, no. 2, pp. 396-417, 1979.
[26] J.E. Kimble and J.G. White, "On the Control of Germ Cell Development in *Caenorhabditis elegans*," *Developmental Biology,* vol. 81, no. 2, pp. 208-219, 1981.
[27] I.S. Greenwald, P.W. Sternberg, and H.R. Horvitz, "The lin-12 Locus Specifies Cell Fates in *Caenorhabditis elegans*," *Cell,* vol. 34, no. 2, pp. 435-444, 1983.
[28] S.T. Henderson, D. Gao, E.J. Lambie, and J. Kimble, "lag-2 May Encode a Signaling Ligand for the GLP-1 and LIN-12 Receptors of *C. elegans*," *Development,* vol. 120, no. 10, pp. 2913-2924, 1994.
[29] E.J. Lambie and J. Kimble, "Two Homologous Regulatory Genes, lin-12 and glp-1, Have Overlapping Functions," *Development,* vol. 112, no. 1, pp. 231-240, 1991.
[30] F.E. Tax, J.J. Yeargers, and J.H. Thomas, "Sequence of *C. elegans* lag-2 Reveals a Cell-Signaling Domain Shared with Delta and Serrate of Drosophila," *Nature,* vol. 368, no. 6467, pp. 150-154, 1994.
[31] J. Yochem, K. Weston, and I. Greenwald, "The *Caenorhabditis elegans* lin-12 Gene Encodes a Trans-Membrane Protein with Overall Similarity to Drosophila Notch," *Nature,* vol. 335, no. 6190, pp. 547-550, 1988.
[32] G. Seydoux and I. Greenwald, "Cell Autonomy of lin-12 Function in a Cell Fate Decision in *C. elegans*," *Cell,* vol. 57, no. 7, pp. 1237-1245, 1989.
[33] H.A. Wilkinson, K. Fitzgerald, and I. Greenwald, "Reciprocal Changes in Expression of the Receptor lin-12 and Its Ligand lag-2 Prior to Commitment in a *C. elegans* Cell Fate Decision," *Cell,* vol. 79, no. 7, pp. 1187-1198, 1994.
[34] X. Karp and I. Greenwald, "Post-Transcriptional Regulation of the E/Daughterless Ortholog HLH-2, Negative Feedback, and Birth Order Bias during the AC/VU Decision in *C. elegans*," *Genes and Development,* vol. 17, no. 24, pp. 3100-3111, 2003.
[35] J.R. Collier, N.A. Monk, P.K. Maini, and J.H. Lewis, "Pattern Formation by Lateral Inhibition with Feedback: a Mathematical Model of Delta-Notch Intercellular Signaling," *J. Theoretical Biology,* vol. 183, no. 4, pp. 429-446, 1996.
[36] G. Marnellos, G.A. Deblandre, E. Mjolsness, and C. Kintner, "Delta-Notch Lateral Inhibitory Patterning in the Emergence of Ciliated Cells in Xenopus: Experimental Observations and a Gene Network Model," *Proc. Pacific Symp. Biocomputing,* vol. 5, pp. 329-340, 2000.
[37] H. Matsuno, R. Murakami, R. Yamane, N. Yamasaki, S. Fujita, H. Yoshimori, and S. Miyano, "Boundary Formation by Notch Signaling in Drosophila Multicellular Systems: Experimental Observations and Gene Network Modeling by Genomic Object Net," *Proc. Pacific Symp. Biocomputing,* pp. 152-163, 2003.
[38] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming,* vol. 8, pp. 231-274, 1987.
[39] D. Harel and E. Gery, "Executable Object Modeling with Statecharts," *Computer,* vol. 30, no. 7, pp. 31-42, 1997.
[40] W. Damm and D. Harel, "LSCs: Breathing Life into Message Sequence Carts," *Formal Methods in System Design,* vol. 19, no. 1, pp. 45-80, 2001.
[41] D. Harel and R. Marelly, *Come, Let's Play—Scenario-Based Programming Using LSCs and the Play-Engine.* Springer, 2003.
[42] H.R. Horvitz and I. Herskowitz, "Mechanisms of Asymmetric Cell Division: Two Bs or Not Two Bs, That Is the Question," *Cell,* vol. 68, no. 2, pp. 237-255, 1992.
[43] L.A. Segal, *Mathematical Models in Molecular and Cellular Biology.* Cambridge Univ. Press, 1980.
[44] J. Kimble, "Alterations in Cell Lineage Following Laser Ablation of Cells in the Somatic Gonad of *Caenorhabditis elegans*," *Developmental Biology,* vol. 87, no. 2, pp. 286-300, 1981.
[45] I. Greenwald, "LIN-12/Notch Signaling: Lessons from Worms and Flies," *Genes and Development,* vol. 12, no. 12, pp. 1751-1762, 1998.
[46] N.A. Lynch, *Distributed Algorithms.* Morgan Kaufmann, 1996.
[47] N. Kam, "The Immune System As A Reactive System: Modeling T-Cell Activation Using Statecharts," master's thesis, Weizmann Inst. of Science, 2000.
[48] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.

Avital Sadot received the MSc degree in bioinformatics from the Weizmann Institute of Science in 2003, with an MSc thesis on comparative genomics. She is currently working toward the PhD degree in the Department of Computer Science and Applied Mathematics, Weizmann Institute of Science. Her research interests include using computational tools for modeling and model verification of complex biological systems.

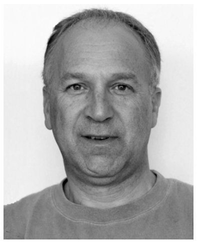
Jasmin Fisher received the PhD degree in neuroimmunology from the Weizmann Institute of Science and was previously a postdoctoral fellow in the Department of Computer Science at the Weizmann Institute, and then a postdoctoral researcher in the School of Computer Science at the EPFL. She is a group leader in the Computational Biology Division at Microsoft Research Cambridge. Her research focuses on the applications of formal methods to biological modeling, as well as on the development of novel formalisms and tools to better understand complex biological systems. She is mainly interested in processes of cell fate determination and signaling networks operating during normal animal development and cancer. Her web page is http://research.microsoft. com/~jfisher/.

Dan Barak received the BA degree in computer science from the Israel Institute of Technology (Technion) in 1998 and the MSc degree, with an MSc thesis on the design and implementation approaches in software engineering, particularly scenario-based modeling and analysis and its connections to the state-based approach, from the Weizmann Institute of Science in 2005. His research interests include hands-on software development and system and project management.

Yishai Admanit studied in Yeshivat Har Etzion, Alon Shevut (a Rabbinical college) in 1995-2000 and integrated an army service in the IDF during this period. He received a degree in computational biology from Bar-Ilan University in 2003 and the MSc degree in computational biology from the Weizmann Institute of Science in 2006, with a thesis on advanced algorithms for inferring cell lineage trees. He is currently with Ceva, Herzelia, Israel.

Michael J. Stern is an associate professor and the director of graduate studies for genetics at Yale University. His research focuses on understanding the fundamental mechanisms of development using *C. elegans* as a model system. His current research interests include mechanisms of FGF signal transduction, cell migration guidance, muscle development, and the computational modeling of developmental systems.

E. Jane Albert Hubbard received the PhD degree in genetics and development from Columbia University in 1993. She was a postdoctoral researcher in the Department of Biochemistry and Molecular Biophysics, Columbia University, studying genes that functionally interact with *lin-12* in *C. elegans*. Since 1998, she has been an assistant professor of biology at New York University, where her laboratory research focuses on the molecular-genetic aspects of gonadogeneis in *C. elegans*.



David Harel received the PhD degree from the Massachusetts Institute of Technology in 1978. Since 1980, he has been at the Weizmann Institute of Science in Israel, where he was the department head for five years and was the dean of the Faculty of Mathematics and Computer Science for seven years. He was also the cofounder of I-Logix Inc. and has spent time at IBM Yorktown Heights and sabbaticals at Carnegie Mellon University, Cornell University, and the University of Edinburgh. In the past, he worked mainly on theoretical computer science (logic, computability, and automata theory), and now, he works on software and systems engineering, modeling biological systems, and the synthesis and communication of smell.