Towards 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 non-biological system design. One is the inter-object approach using the language of live sequence charts (LSCs) with the Play-Engine tool, and the other is the intra-object 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 *C. elegans* in LSCs, while 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**

I. 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

Manuscript received June 8, 2006. This work was supported in part by The John Von Neumann Minerva Center for the Development of Reactive Systems at the Weizmann Institute of Science.

A. Sadot, D. Barak, Y. Admanit, D. Harel are with the Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, 76100 Israel (e-mail: {avitals, dan.barak, yishai.admanit, david.harel}@ weizmann.ac.il).

J. Fisher is with the School of Computer and Communication Sciences, Swiss Federal Institute of Technology (EPFL), Lausanne, CH-1015, Switzerland (e-mail: jasmin.fisher@epfl.ch).

M.J. Stern is with the Department of Genetics, Yale University School of Medicine, New Haven, CT 06520-8005, USA (e-mail: michael.stern@yale.edu).

E.J.A. Hubbard is with the Department of Biology, New York University, New-York, NY 10003-6688, USA (e-mail: jane.hubbard@nyu.edu).

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", i.e., 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].

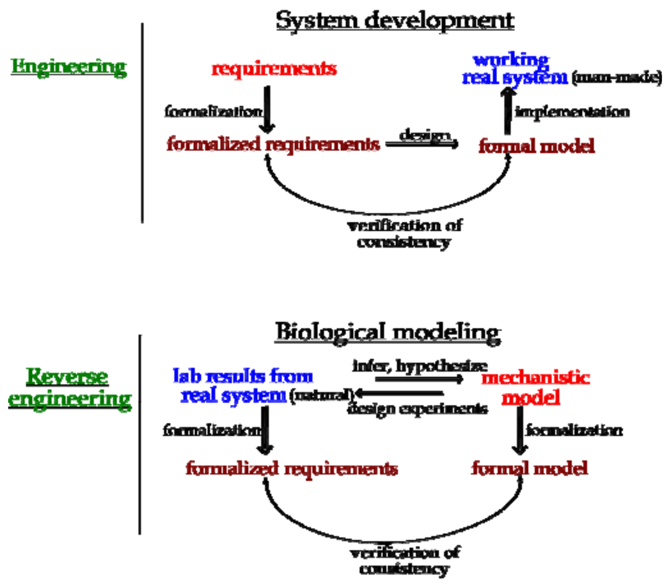


Fig 1: Building engineered systems vs. modeling and model verification of biological systems.

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 used techniques for carrying out verification. *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.

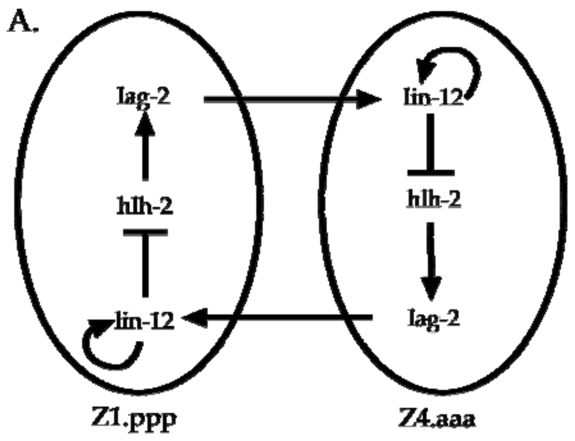
In building biological models, a process analogous to testing is the re-consideration of each of the laboratory-based results that were used to formulate the mechanistic model as a means to determine if the model is a sufficient reflection of the current understanding of the system [4]. Additional laboratory-based 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.

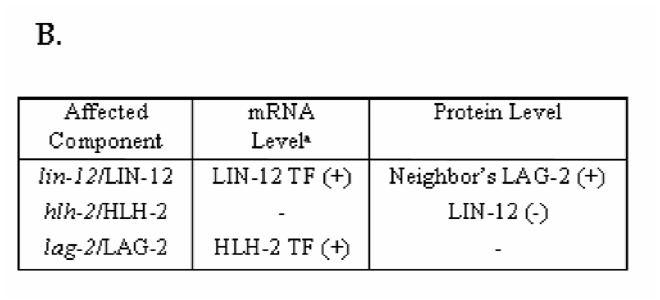
Another shared feature of biological and engineered systems is that both display functional modularity. A functional module is a distinct unit whose function is separable from those of other modules [6]. Several areas of research have highlighted the modularity within biology, particularly in evolution and development [6], [7], [8], [9]. 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 [10], Petri nets [11], process algebra and pi calculus [12], [13]. Different methodologies depend on the type of system being modeled and the questions being addressed. Our group utilizes two main methods: an intra-object, state-based approach using statecharts and Rhapsody and an inter-object, scenario-based approach using live sequence charts (LSCs) and the Play-Engine [3], [14], [15], [16], [17]. 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.

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 [18]. 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 (e.g., synthesis and degradation rates for components of the pathway). For testing, we generated LSCs to 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.

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 [19]. 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 [20]. 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 precursor cell (VU) [19]. The final conformation of the somatic primordium depends on the outcome of the cell fate determination process between these two cells.




B.

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 (+)	-

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 post transcriptionaly by LIN-12. Arrows represent positive regulation and bars represent negative regulation. Adapted from (Karp and Greenwald, 2003) and (Wilkinson et al., 1994). 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

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) [21], [22], [23], [24], [25]. 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% of the animals, Z4.aaa becomes the AC and Z1.ppp becomes a VU, and vice versa for the other 50%). 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*, while 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 [21], [23], [26], [27]. 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 post-transcriptionally down-regulated in the presumptive VU upon LIN-12 activation as part of the negative feed-back mechanism that leads to the termination of *lag-2* transcription [28] (Fig. 2A). 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).

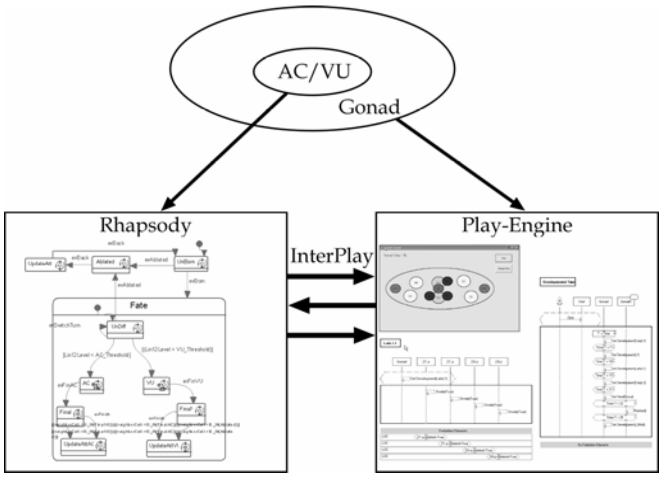


Fig. 3: Combining different tools for different modules. The AC/VU decision was modeled in the state-based formalism using Rhapsody, and the lineage of the somatic gonad was modeled in the scenario based formalism, using the Play-Engine. InterPlay was used to connect the two tools.

We further composed a set of LSCs representing a key sub-set 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).

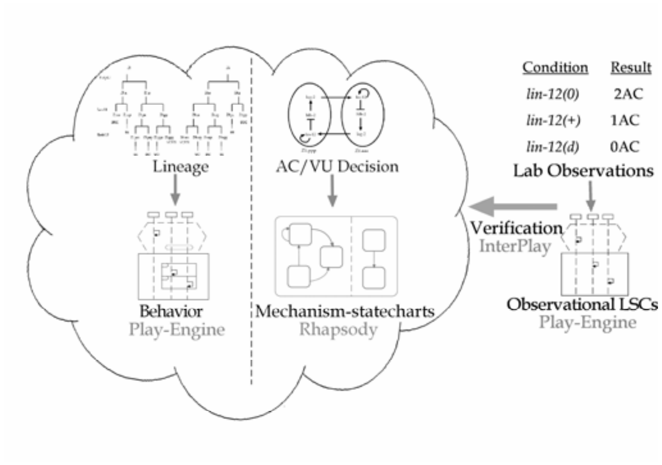


Fig. 4: Model verification. After we constructed the model using Rhapsody and the Play-Engine, we used the Play-Engine again to test the integrated model. We created a set of simple LSCs based on lab observations, and used InterPlay to run the combined Play-Engine-Rhapsody mechanistic model against them.

II. Methods

A. Statecharts

Statecharts is a visual formalism, developed in 1983 as a language for specifying reactive behavior [29]. In its object-oriented version [30], one uses statecharts to define the behavior of objects over time, in an intra-object 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 non-specialists.

B. Rhapsody

Rhapsody is a software tool for the design of statechart-based models [30] (http://www.ilogix.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.

C. LSCs

Live sequence charts (LSCs) constitute a visual formalism for specifying sequences of events and message passing between objects [31]. 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 anti-scenarios). There are two types of LSCs, universal and existential. Universal charts are more relevant for modeling, and are built of a pre-chart and main-chart. The relationship between the pre-chart and the main-chart can be viewed as a condition-result pair (see Fig. 5A): whenever the scenario in the pre-chart occurs (condition), the scenario in the main chart must follow (result) [31].

D. 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 inter-object behavior, and execute and simulate a modeled system [32]. 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. The Play-Engine also detects any violations of constraints or contradictions between scenarios if attempted. For more on LSCs, play-in/play-out and the Play-Engine see [32].

E. 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 [18]. 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 (to appear in an extended version of [18]).

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 which 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.

III. Results

A. 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 cells. This component is simulated using LSCs. The model of the lineage is constructed from two types of GUI objects – a Gonad object and the objects of the Somatic Cells. The GUI depicts the somatic gonad 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). 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 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). 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 the work of Karp and Greenwald (2003), the outcome of the AC/VU decision is influenced by the birth order of Z1.ppp and Z4.aaa. Therefore, for these two cells the LSC `Stochastic Event` decides randomly which of the two cells is born first (i.e. changes its `Born` property to true first) (See Supp. Fig. 3). 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.

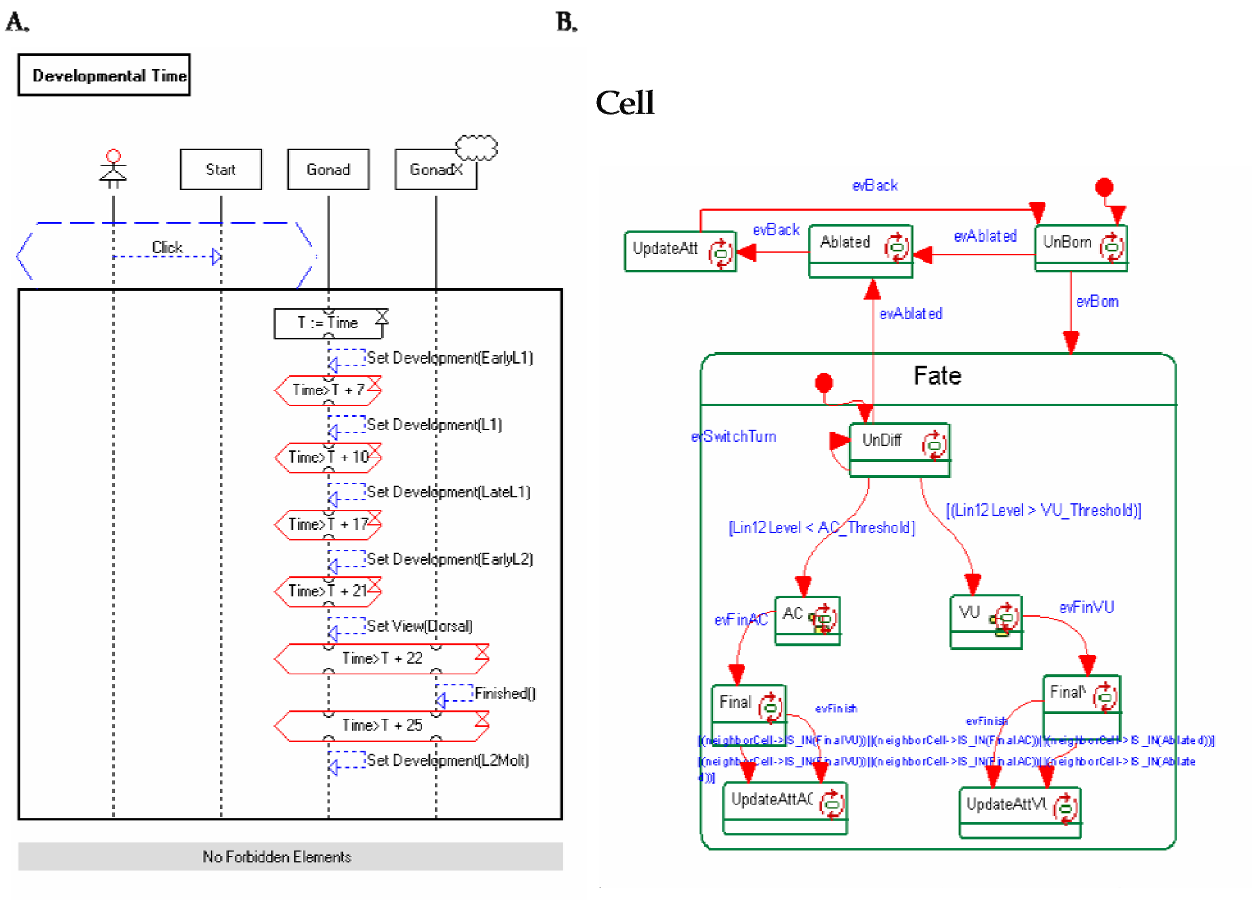


Fig. 5: Examples from the model. (A) An example LSC. This LSC follows developmental time and manages the development of the gonad. If the user clicks the Start button (the pre-chart, surrounded by a blue dashed line), then the Gonad object starts to measure time, and advances through the developmental 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.

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 (i.e. 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 [19], [26], [33], 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.

1) Gonad class statechart

The instantiation of the Gonad statechart starts in the `InitConditions` state (see Supp. Fig. 4). This state has a sub-statechart 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). The default initial condition is the wild type.

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 post-translational levels, representing the production of mRNA and protein products (Supp. Table 1). 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. 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 (Supp. Table 2).

For each of the genes, the calculation includes a basal transcription and degradation rate for the mRNA, and 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 Fig. 2A, B). 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 [28], 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 [34]. 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% and +5%) 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.

B. 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 (Supp. Tables 1, 2). 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.

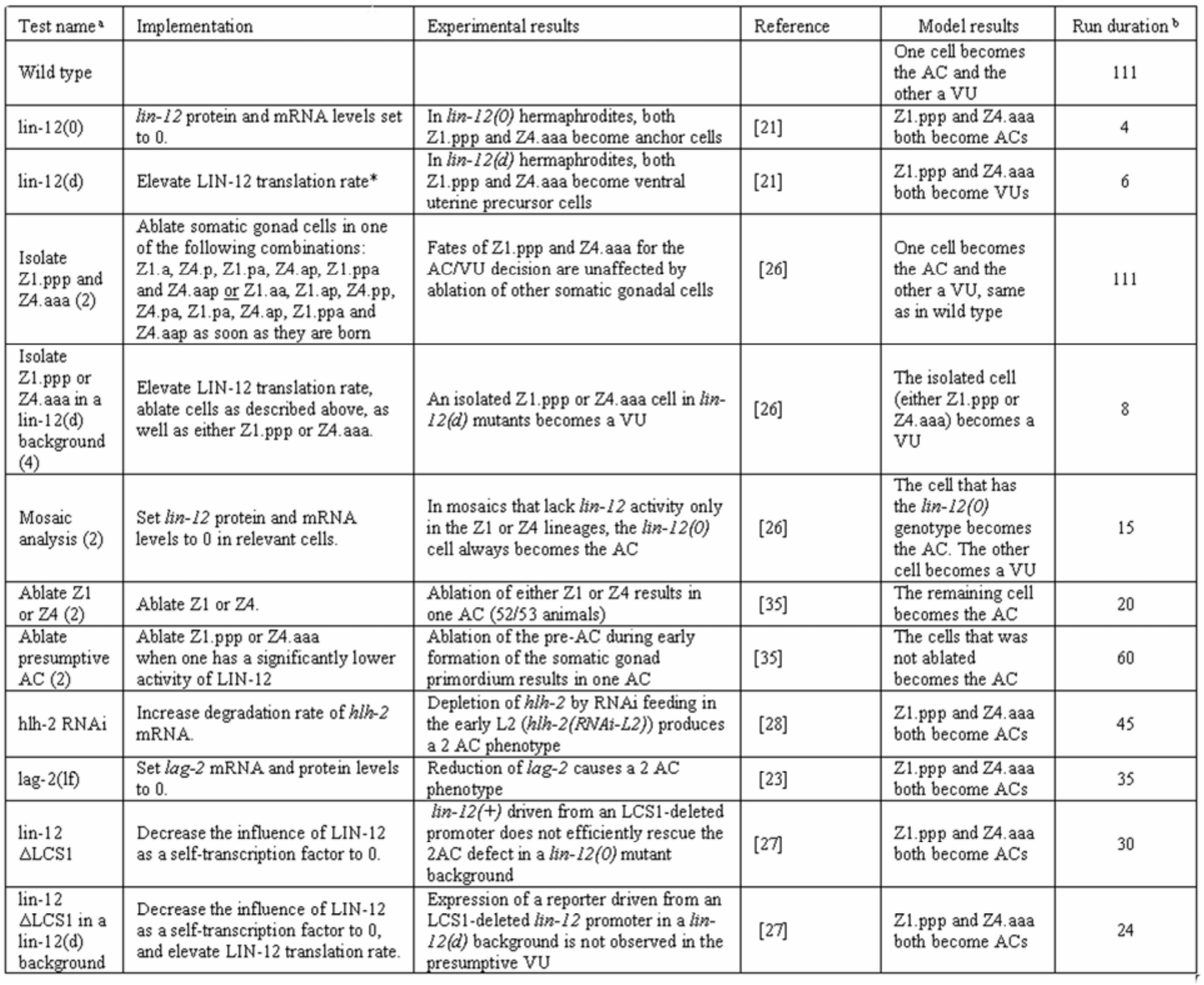
The Cell statechart then advances into the state `Fate` (Fig. 5B), in which the model represents the process of cell fate determination, including a set of two sub-statecharts to separate events that occur upon reaching the "decision threshold" of LIN-12 levels from the final stage of cell fate acquisition. 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 sub-statechart 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). 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 1-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 sub-statechart 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 (Kimble and Hirsh, 1979) – is set and displayed via the GUI (see Supp. Fig. 1C). An example of an outcome of a run of the model under normal, wild-type conditions can be seen in Supp. Fig. 7.

C. 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 pre-chart that states the initial condition of the experiment (for example *lin-12(0)* indicates *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). These LSCs do not contain any information about the mechanism that led to the result.

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 <u>or</u> 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

[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 activate-able (see text for further explanation).

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 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. Fig. 9-17). For laboratory observations in which the experimental outcomes were non-deterministic, 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 non-deterministic outcomes. Though the model and the tests thereof are not comprehensive – that is, these are but a small sub-set 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.

IV. 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, while 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 inter-object 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 intra-object 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 standalone 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 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 exhaustive testing. The modular approach is very convenient, since 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 [36]. 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 [37]. 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 [37]. 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 [36]. 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 [28]. 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 standalone, 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 [15], [17]. Another plausible expansion of this model is the construction of additional aspects of C. elegans gonadogenesis.

REFERENCES

[1] D. Harel and A. Pnueli, "On the Development of Reactive Systems", in *Logics and Models of Concurrent Systems* (K. R. Apt, ed.), NATO ASI Series, F13, New York: Springer-Verlag, pp. 477-498, 1985.

[2] D. Harel, "A Grand Challenge for Computing: Full Reactive Modeling of a Multi-Cellular Animal", *Bulletin of the EATCS* , European Association 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. Visual Languages and Formal Methods* (VLFM'01), part of *IEEE Symp. on 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), Lecture Notes in Bioinformatics, vol. 3082, Berlin: Springer-Verlag, 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] 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. C47-52, 1999.

[7] N. Kashtan and U. Alon, “Spontaneous Evolution of Modularity and Network Motifs”, *Proc Natl Acad Sci U S A*, vol. 102, no. 39, pp. 13773-13778, 2005.

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

[9] 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.

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

[11] J. Barjis and I. Barjis, “Formalization of the Protein Production by means of Petri Nets”, *International Conference on Information Intelligence and Systems (ICIIS'99)*, 1999.

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

[13] A. Regev, W. Silverman and E. Shapiro, “Representation and Simulation of Biochemical Processes Using the Pi-Calculus Process Algebra”, *Pac Symp Biocomput,* vol. 6, pp. 459-470, 2001.

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

[15] J. Fisher, N. Piterman, E.J.A. Hubbard, M.J. Stern and D. Harel, “Computational Insights into Caenorhabditis elegans Vulval Development”, *Proc Natl Acad Sci U S A*, vol. 102, no. 6, pp. 1951-1956, 2005.

[16] D. Harel, S. Efroni and I.R. Cohen, “Reactive Animation”, *Proc. 1st Int. Symposium on Formal Methods for Components and Objects* (FMCO 2002) (invited paper), Lecture Notes in Computer Science, vol. 2852, Springer-Verlag, pp. 136-153, 2003.

[17] 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” in *Proc. 1st Int. Workshop on Computational Methods in Systems Biology* (ICMSB 2003), Lecture Notes in Computer Science, Vol. 2602, Springer-Verlag, pp. 4-20, Feb. 2003.

[18] D. Barak, D. Harel and R. Marelly, “InterPlay: Horizontal Scale-Up and Transition to Design in Scenario-Based Programming” in *Lectures on Concurrency and Petri Nets* (J. Desel, W. Reisig and G. Rozenberg, eds.), Lecture Notes in Computer Science, Vol. 3098, Springer-Verlag, pp. 66-86, 2004.

[19] J. Kimble and D. Hirsh, “The Postembryonic Cell Lineages of the Hermaphrodite and Male Gonads in Caenorhabditis elegans”, *Dev Biol*, vol. 70, no. 2, pp. 396-417, 1979.

[20] J.E. Kimble and J.G. White, “On the Control of Germ Cell Development in Caenorhabditis elegans”, *Dev Biol*, vol. 81, no. 2, pp. 208-219, 1981.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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 Dev*, vol. 17, no. 24, pp. 3100-3111, 2003.

[29] D. Harel, “Statecharts: A Visual Formalism for Complex Systems”, *Science of Computer Programming*, vol. 8, pp. 231-274, 1987.

[30] D. Harel and E. Gery, “Executable Object Modeling with Statecharts”, *Computer*, vol. 30, no. 7, pp. 31-42, 1997.

[31] 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.

[32] D. Harel and R. Marelly, *Come, Let's Play - Scenario-Based Programming Using LSCs and the Play-Engine*, Springer-Verlag, 2003:

[33] 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.

[34] L.A. Segal, *Mathematical Models in Molecular and Cellular Biology*, Cambridge: Cambridge University Press, 1980,

[35] J. Kimble, “Alterations in Cell Lineage Following Laser Ablation of Cells in the Somatic Gonad of Caenorhabditis elegans”, *Dev Biol*, vol. 87, no. 2, pp. 286-300, 1981.

[36] I. Greenwald, “LIN-12/Notch Signaling: Lessons from Worms and Flies”, *Genes Dev*, vol. 12, no. 12, pp. 1751-1762, 1998.

[37] N.A. Lynch, *Distributed Algorithms*, San Francisco: Morgan Kaufmann Publishers, Inc, 1996.