Combining State-Based and Scenario-Based Approaches in Modeling Biological Systems

Jasmin Fisher[1,⋆], David Harel[1,⋆⋆], E. Jane Albert Hubbard[2,⋆⋆⋆], Nir Piterman[1,†], Michael J. Stern[3,‡], and Naamah Swerdlin[1]

[1] Dept. of Computer Science and App. Math., Weizmann Institute, Rehovot 76100, Israel
`firstname.lastname@weizmann.ac.il`
[2] Dept. of Biology, New York University, New York, NY
`jane.hubbard@nyu.edu`
[3] Dept. of Genetics, Yale School of Medicine, New Haven, CT
`michael.stern@yale.edu`

Abstract. Biological systems have recently been shown to share many of the properties of reactive systems. This observation has led to the idea of using methods devised for the construction (engineering) of complex reactive systems to the modeling (reverse-engineering) of biological systems, in order to enhance biological comprehension. Here we suggest to combine the two formal approaches used in our group — the state-based formalism of statecharts and the scenario-based formalism of live sequence charts (LSCs). We propose that biological observations are better formalized in the form of LSCs, while biological mechanistic models would be more natural to specify using statecharts. Combining the two approaches would enable one to verify the proposed mechanistic models against the real data. The biological observations can be compared to the requirements in an engineered system, and the mechanistic model would be analogous to the implementation. While requirements are used to design an implementation, here the observations are used to motivate the invention of the mechanistic model. In both cases consistency of one with the other must be established, by testing or by formal verification.

1 Introduction

Experimental biology is an interplay between collecting data in experiments (observations), followed by the analysis of the data and suggestion for possible mechanistic models that would explain how the system under study works, and how it gives rise to the observed phenomena. Further experiments are often preformed to test the hypothetical

⋆ Supported by the Dov Biegun postdoctoral fellowship.
⋆⋆ Supported in part by NIH grant F5490-01 and ISF grant 287/02.
⋆⋆⋆ Supported in part by NIH grant R24-GM066969 and R01-GM61706.
† Supported by the John von-Neumann center for Verification of Reactive Systems.
‡ Supported in part by NIH grant R24-GM066969.

mechanism. Here we propose that the dichotomy between theses two aspects of biology calls for different formal methods.

Recently, the resemblance between reactive systems (systems that continuously interact with their environment) and biological systems has been noted [7, 11]. This observation has led to the idea of using methods devised for the construction of complex reactive systems to model biological systems. The first attempt to follow this path was a modest model of T-cell activation [11], which was followed by an extensive animated model of T-cell behavior in the thymus [4, 5]. At present there is an ongoing effort to model the vulval development in the nematode *Caenorhabditis elegans* [12].

Two approaches are used in our group to model biological behavior: an **intra-object** one based on the language of **statecharts** [6] and the **Rhapsody** tool [8, 10], and an **inter-object** one that uses the more recent language of **live sequence charts** (LSCs) [3] and the **Play-Engine** tool [9]. Both languages are visual, having a clear (and formal) syntax and semantics, and both approaches enable the construction of a formal model and its execution. In Rhapsody, a state-based transition diagram (statechart) of the system under study is constructed. The tool automatically generates executable C/C++/Java code from the statecharts. In contrast, LSCs specify scenarios of behavior between objects, with varying modalities (e.g., required, possible, and forbidden scenarios), and the Play-Engine executes these directly in a way that satisfies the modalities for each.

As typical biological data is available in the form of ‘condition-result’ scenarios, we believe that biological observations are best formalized in the form of LSCs, which take the following general form: if X (the prechart) occurs then Y (the main chart) should too, where X and Y consist of scenarios of behavior and can be simple or complex. Indeed, in an LSC we can formalize the terms of the experiment as the conditional prechart that enables an LSC, and we can formalize the result of the experiment as a sequence of happenings resulting from that condition. On the other hand, since many biological mechanistic models are ‘state-based’, we feel that in such cases it would be more natural to specify mechanistic models using statecharts, since they specify the behavior of the system based on its internal (stipulated) mechanism.

Here, we propose that these two approaches complete each other and should be used together in order to model the two aspects of a common biological system simultaneously. By using a scenario-based approach to formalize the behaviors of the biological system and a state-based approach to formalize the mechanism underlying these behaviors, one can formally verify that the mechanistic model reproduces the system’s real behavior. In this functional sense, the biological observations can be compared to the requirements in an engineered system, while the proposed mechanistic model would be analogous to the system’s design and implementation.[1] To carry this analogy further, in an engineered system the requirements are used to help in coming up with the design and implementation, and are then used a second time to build test suites for testing the implementation against the requirements. In biological systems the observations are used to motivate the construction of mechanistic models (that serve as working hypotheses),

[1] In another sense it is the other way around: the biological observations are directly related to the actual system as is the implementation of an engineered system, whereas requirements and biological mechanistic models are both invented by humans.

and our approach enables them to be used also in testing and verification, by simulation or, e.g., model-checking. These techniques may also yield interesting predictions that should be then corroborated experimentally in the biological system.

2 Engineering Computerized Systems Versus Reverse-Engineering Biological Systems

In **engineering**, we try to produce a **design** that satisfies a set of requirements. This set of requirements is determined by our ideas about how the system should eventually work. In biological modeling we do **reverse-engineering**, trying to construct a **mechanistic model** that explains how the biological system works. This model has to fit the experimental observations.

In engineering, we formalize the requirements (emanating from the system's concept) in a formal specification language (e.g., LSCs). This formal specification not only guides the construction of the design and implementation but is later used also to check the design's correctness. In formal modeling of biology, on the other hand, the situation is different. The biological system is already 'built'. In fact, there is a running exemplar of the system. Unfortunately, we are restricted in the way we can analyze it (in particular, we are unable to access its 'blueprint'). By experimenting with the biological system (i.e., testing it under different conditions) and recording the results of these experiments we can gain knowledge about its behavior. Once we formalize these observations in the form of LSCs we get a sort of 'requirements' specification that can be used in constructing the mechanistic model. See Figure 1.

	Computerized Systems (Engineering)	**Biological Systems (Reverse-engineering)**
Scenario-based	Requirements	Observations
State-based	Design & Implementation	Mechanistic Model

Fig. 1. Analogy between computerized systems and biological systems

The testing phases in both cases (engineering versus reverse-engineering) possesses an interesting duality. In engineering we test the design in order to improve assurance of its quality. In biology we probe the system, a process we can also call testing, get the results, and rerun the results on the mechanistic model. Thus, in the reverse-engineered process of modeling biology a test would be run twice: once on the biological system itself and once on the mechanistic model. As a result, once we check all the tests known at a given time-point, we get what we might call (borrowing from software engineering terminology) a complete **coverage** of the desired behavior of the mechanistic model. At that time-point, there are no more tests to preform on the system until more experiments are carried out. In contrast, when engineering man-made systems the problem of determining whether the requirements are sufficient is an interesting question in its own right.

3 Regression Testing and Model Checking

The fact that when we come up with a mechanistic model there is at hand a fixed given set of tests (the biological observations) suggests that we can use **regression testing** to get a higher type of assurance of the model's correctness. In engineering, regression testing is used to compare different versions of the same design, by running the old test suites on the new version to make sure that we haven't inadvertently changed previously decided-upon desired behavior. Thus, during the development of the design, we form a collection of tests and save the results we got when running them on the present version. Once a change is made, we run the same set of tests again and make sure that the new design produces the same results. In the case of reverse-engineering a biological system the comparison made by re-running the tests is not between an old and a new version of the system, but between the real biological system and the proposed mechanistic model thereof. Here, the collection of tests is already given, as the set of observations resulting from the performed experiments.

Both in engineering and reverse-engineering performing additional ad-hoc tests can produce interesting results. In the engineering world, arbitrary tests may produce behaviors that indicate the existence of bugs, causing the need to redo the design and implementation, whereas in reverse-engineering, ad hoc tests (such as running the formalized models on additional inputs or in different ways) may produce interesting predictions regarding the behavior of the biological system, or questions that need to be resolved by further experimentation.

In engineering, we would like to use verification techniques, such as **model-checking** (see, e.g., [2]), in order to acquire greater confidence in the correctness of the design. Model-checking is a method to formally verify that all the possible behaviors of the system satisfy a given requirement, and can be used with statecharts or LSCs. In reverse-engineering of biological systems we could use model-checking to get around the main disadvantage of model execution, which is its inability to cover all possible execution scenarios, which is particularly problematic for mechanistic models that are non-deterministic. This would be done by model-checking the specification (LSCs) against the mechanistic model (statecharts), and could provide a major additional boost to the validity of the latter. One of the results of the model-checking process could be interesting predictions regarding the behavior of the actual system.

4 Implementation

We have applied the suggestion made in this paper to the mechanistic model of [14] that explains parts of the formation of the vulva in *Caenorhabditis elegans*. We have formalized the mechanistic model in the form of statecharts, and the experimental observations (that led to the suggestion of this model) as existential LSCs. We then used the Rhapsody tool [10] and its testing component — the **TestConductor** [13]. In the TestConductor, tests are given in the form of combinations of existential LSCs. Each test can then be performed individually, or the tool can be asked to produce a report on the entire behavior of the model when checked versus all the tests. Running regression testing in different stages of the development of the statecharts model enabled us to fine-tune the model to reproduce all the behaviors on which this model is based.

An interesting aspect of this particular work is that our mechanistic model is completely deterministic. Thus, testing a scenario using simulation is sufficient to make sure that the mechanistic model reproduces the behavior depicted in the scenario. We thus have full assurance that our formalization of the mechanistic model completely reproduces the data. This fact improves our confidence in the correctness of the proposed mechanistic model. A detailed description of this modeling effort will be reported separately.

5 Concluding Remarks

Based on the above, we suggest that the state-based and scenario-based approaches complete each other. We propose that biological systems should be modelled using both approaches. Observations should be formalized by inter-object scenario-based methods (in our case, LSCs using the Play-Engine tool), while the mechanisms should be formalized by intra-object state-based methods (in our case using statecharts and the Rhapsody tool). Once this is done we can simulate all the experiments carried out in practice and use the state-based model to drive the simulation that the scenarios follow. Using regression testing, we can ensure that the mechanistic model reproduces all the behaviors observed in the living system.

A connection between Rhapsody and Play-Engine is currently under development, which will enable them to work on cooperation; see [1]. Such connection will enable to verify automatically that a biological mechanistic model is consistent with the experimental observations obtained by the system. We believe this would further facilitate our understanding of biological systems and help simulate and analyze their reactive nature.

References

1. D. Barak, D. Harel, and R. Marelly. Interplay: Horizontal scale-up and transition to design in scenario-based programming. To appear, 2004.
2. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
3. W. Dam and D. Harel. LSCs: Breathing life into message sequence charts. *FMSD*, 2001.
4. S. Efroni, D. Harel, and I. R. Cohen. Modeling and simulation of the thymus. *Multidisciplinary Approaches to Theory in Medicine*, 2002.
5. 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*, 2003.
6. D. Harel. Statecharts: A visual formalism for complex systems. *SCP*, 8:231–274, 1987.
7. D. Harel. A grand challenge for computing: Towards full reactive modeling of a multi-cellular animal. *Bulletin of the EATCS*, 81:226–235, 2002.
8. D. Harel and E. Gery. Executable object modeling with statecharts. *Computer*, 30(7), 1997.
9. D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.
10. I-logix,inc. http://www.ilogix.com.
11. N. Kam, D. Harel, and I. R. Cohen. The immune system as a reactive system: Modeling T-cell activation with statecharts. *Bull. Math. Bio.*, 2003.

12. 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 *1st Int. Conf. on Computational Methods in Systems Biology*, February 2003.
13. M. Lettrari and J. Klose. Scenario-based monitoring and testing of real-time UML models. In *4th Int. Conf. on the Unified Modeling Language*, October 2001.
14. P. W. Sternberg and H. R. Horvitz. The combined action of two intercellular signaling pathways specifies three cell fates during vulval induction in c. elegans. *Cell*, 58(4):679–93, 1989.