Some Thoughts on the Semantics of Biocharts

David Harel[1] and Hillel Kugler[2]

[1] The Weizmann Institute of Science, Rehovot, Israel
dharel@weizmann.ac.il
[2] Microsoft Research, Cambridge, UK
hkugler@microsoft.com

Dedicated to the dear memory of Amir Pnueli: Friend, mentor, colleague, and a truly towering figure in computer science

Abstract. This paper combines three topics to which Amir Pnueli contributed significantly: the semantics of languages for concurrency, the semantics of statecharts, and reactive and hybrid systems. It is also no accident that the main motivation of our paper comes from biological systems: in recent years Amir became interested in these too. In [KLH10] we introduced Biocharts, a fully executable, two-tier compound visual language for modeling complex biological systems. The high-level part of the language is a version of statecharts, which have been shown to be successful in software and systems engineering. These statecharts can then be combined with any appropriately well-defined language (preferably a diagrammatic one) for specifying the low-level dynamics of the biological pathways and networks. The purpose of [KLH10] was to present the main concepts through a biological example and to illustrate the feasibility and usefulness of the approach. Here we discuss some of the questions that arise when one attempts to provide a careful definition of the semantics of Biocharts. We also compare the main requirements needed in a language for modeling biology with the way statecharts are used in software and system engineering.

1 Introduction

In recent years it is becoming clearer that understanding and predicting the behavior of complex biological systems requires developing and using new computational modeling languages and tools. In addition to the reductionist approach, which has been very successful in uncovering biological mechanisms, there is a need to integrate and synthesize the knowledge gained through reductionism to build system-level models that can explain and predict the behavior of a system as a whole and not only focus on very specific parts or aspects of the system behavior.

The language of *Statecharts*, which has proven to be very useful for specifying complex reactive software and systems, has been applied in the past ten years to modeling biological systems [KCH01, FPH+05, EHC07, SCDH08]. There are many aspects for which the statecharts approach is well suited to biological

Z. Manna and D. Peled (Eds.): Pnueli Festschrift, LNCS 6200, pp. 185–194, 2010.

modeling, but also several major challenges specific to biology that led us to introduce *Biocharts*, a variant of statecharts geared towards biological modeling. Here we explain to a non-biological reader some of the challenges Biocharts aims to address and outline a definition of the semantics of the language. We have not yet built a dedicated tool to support Biocharts (in [KLH10] we used Rhapsody for demonstrating the feasibility of the approach), and some of the semantic decisions are still to be made. For various issues we point out here different semantic options and their implications.

2 Biological Modeling with Differential Equations

The traditional mathematical approach for modeling biological systems has been to use tools from classical continuous mathematics, mainly differential equations, and despite its successful application in many cases, this approach suffers from several limitations. First, differential equations require numeric values for the coefficients in the equations, values which are often unknown and can be very difficult to determine experimentally. Second, the continuous assumption is sometimes not valid when we consider a small number of molecules or cells. Third, often the level of abstraction that biologists use when thinking about their systems is discrete; for example, considering a gene to be either on or off, a signal to be low, medium or high and the differentiated fate of a cell to be primary, secondary or tertiary. Biologists know that these discrete values are only an abstraction but since this way of thinking is useful to them, we deem it an important facet of any approach to modeling that the languages and tools support such abstractions in a natural way. Finally, our computational ability to handle large systems of differential equations is often extremely limited.

3 Challenges in Biological Modeling

There are many reasons why modeling biological systems is extremely challenging. First and foremost is the inherent complexity of biological artifacts. Even when one focusses "merely" on modeling the behavior of a single cell, the process is likely to result in a model that is more complex than most engineered software systems. There are certain aspects of biology that are best treated using a continuous approximation, or differential equations, and there are other aspects where discrete methods are more suitable. A combination of these approaches would lead to hybrid models, which may play an important role in biological modeling, but only if suitable languages and tools are developed to integrate the approaches in an intuitive and rigorous manner.

A biological system can be examined and modeled on many scales: the molecular scale, the cellular scale, the scale of a tissue or an organ, or the scale of an organism or an entire population. For certain purposes, focusing on one scale is sufficient, while for other purposes building system-level models that incorporate several or even all of these scales is a must.

Now, multi-scale models can be constructed using the same language for describing each of the scales, or using different languages to describe the behavior on each scale. Whenever several languages are used, the interaction between the different scales described in different languages and sub-models must be clearly defined and has to be well integrated in the supporting tools. These types of multi-scale models are also challenging due to the computational resources needed to run them effectively.

We now discuss in more detail some of the common aspects and differences between biology and reactive software and the influence they have on the appropriate task of language design.

4 Biology vs. Software

A fundamental observation regarding biology and software, which leads to the idea of using languages that have proven themselves effective in software design also for biological modeling, is the fact that both types of systems (in the case of software this applies to many of the most complex kinds of systems) are *reactive* in nature. Reactive systems [HP85] are those whose role is to maintain an ongoing interaction with their environment, rather than produce a final result upon termination. Both types of systems are composed of many different parts, that act together in order to maintain the desired interaction with the environment and to achieve some required high-level system goals.

At the heart of the statechart language, which was designed specifically to deal with the dynamic behaviors of reactive systems, are states and transitions and ways to describe them in an intuitive and concise manner. This turns out to be very important also in biological modeling. In fact, many biologists are already used to thinking, and informally describing, their systems in terms of states and changes between states in response to the occurrence of certain events. Thus, adopting statecharts for biological modeling is quite natural.

A main difference between the software and biology domains is that for the former the main goal is constructing a system that will satisfy a set of requirements whereas for the latter the main goal is to understand how an existing biological system works. This is really the difference between engineering and reverse-engineering. It is interesting to observe that the emerging field of synthetic biology [End05] aims to go a step forward and engineer new biological systems to achieve given goals, typically by modifying certain aspects of existing systems, yet even for this direction the ability to understand and predict the behavior of existing biological systems is crucial.

While modeling a biological system, the model can be considered as a theory aiming to explain the behavior of the system, so that one's confidence in the theory increases if the model can predict behaviors that have not been observed yet. Such behaviors are sometimes considered to be *emergent properties* of the model, since despite not being explicitly programmed they emerge as a result of the combined interactive behavior of many components. Such behaviors may be hard to predict and understand without a fully executable model.

All relevant and interesting predictions arising from the model must be verified experimentally, since even models that seem very plausible could well be wrong and thus the model and its result can only serve as a guide towards performing interesting experiments in the lab. Biological models are more valuable if in addition to their predictive capabilities they have the ability to explain the phenomena by uncovering hidden mechanisms and underlying principles. In general, the full correctness of a model can never be established; hence the main goal is to refute potential models and thus to try and rule out hypotheses about the principles of the system behavior. This is a very Popperian approach to modeling. See [Har05].

Building models for biological systems is also different from software construction due to the fact that there are many aspects of biology that we still do not understand, and many units and components whose role in certain processes in still largely unknown. Thus, almost any attempt at biological modeling must involve dealing with such "black boxes". In software modeling and design there is a much clearer understanding of the system being developed, including the requirements, architecture and implementation, yet it is still very helpful to use abstraction and "black boxes", which provide freedom from the bias of implementation and thus help develop more robust software. In software models one aims to simplify and to avoid redundancy, whereas biological systems have many inherent redundancies. Hence, when modeling biology any simplifications to the model should be done very carefully, considering the assumptions made in the model and their implications, since a model that can reproduce biological behavior is not a goal in itself. Rather, we are mainly interested in what can be learned from the model and from its predictive capabilities.

In both software and biology, the ability to involve the domain experts, the various stake-holders in a software project and experimental biologists in biological modeling, is key to the success of the effort. In this way, developing languages and tools that are useful and intuitive for non-programmers is essential, which appears to be one of the major strengths of both Statecharts and Biocharts.

5 Semantics Outline

Biocharts is a fully executable, two-tier compound visual language for modeling complex biological systems. The high-level part of our language is a version of statecharts, which can be combined with any appropriately well-defined language (preferably a diagrammatic one) for specifying the low-level dynamics of the biological pathways and networks. We now outline the semantics of the variant of statechart we propose to use in Biocharts, and discuss how it integrates with the lower level languages.

5.1 The Basics

The statechart itself is similar to the original description in [Har87], and to that of Statemate [HN96] and Rhapsody [HG97], in that there are three types of states:

OR-states, *AND-states* and *basic states*. The OR-states have *substates* related to each other by "exclusive or", AND-states have *orthogonal components* that are related by "and", while basic states have no substates, and are the lowest in the state hierarchy. When building a statechart there is an implicit additional state, the root state, which is the highest in the hierarchy. The *active configuration* is a maximal set of states that the statechart can be in simultaneously, including the root state, exactly one substate for each OR-state in the set, all substates for each AND-state in it and no additional states. The general syntax of an expression labeling a transition in a statechart is "$m[c]/a$" where m is the message that triggers the transition, c is a condition that guards the transition from being taken unless it is true when m occurs, and a is an action that is carried out if and when the transition is taken. All of these parts are optional.

5.2 Classes and Objects

For Biocharts we adapt some of the basic principles of the Rhapsody semantics of statecharts, as described in [HG97, HK04], especially the way statecharts are incorporated into an object-oriented framework. The motivation for this decision is that typical biological models require specifying many entities (e.g., cells) with the same specification but each one in a different active configuration. These entities (e.g., cells) can be born and may die during model execution, so the object oriented framework is a natural one for representing such models.

A system is composed of classes. A statechart describes the modal behavior of the class; that is, how it reacts to messages it receives by defining the actions taken and the new mode entered. A class can have an associated statechart describing its behavior. These classes are called *reactive classes*. During runtime there can exist many objects of the same class, called *instances*, and each can be in a different active configuration – a set of states in which the instance resides. Thus, a new statechart is "born" for each new instance of the class, and it runs independently of the others. When a new instance is created, the statechart enters its initial states by taking default transitions recursively until it is in an active configuration.

A new feature that we propose for Biocharts, building on our experience from previous biological projects, is to enable an object to dynamically create a new object of the same class in exactly the same active state as it is in at the moment of creation. This is useful in several biological contexts; for example, during cell division, where daughter cells typically inherit the state of the mother cell. In this case, the statechart of the daughter cell is "born" in an active configuration identical to its mother cell, and no default transitions are taken as part of this initialization.

5.3 Messages and Actions

As mentioned above, the general syntax of an expression labeling a transition in a statechart is "$m[c]/a$", for message m, condition c and action a. We now describe each of these parts in more detail. The message m is either an event

or a triggered operation. Consider a simple transition between states $S1$ and $S2$ labeled "$m[c]/a$". Regardless of whether m is an event or a triggered operation, if the statechart is in state $S1$, message m occurs and the condition c holds, state $S1$ is exited, the action a is performed and then state $S2$ is entered.

A practical question concerns the language in which the conditions and actions should be written. Statemate introduces a special action language for this purpose, whereas Rhapsody, which is geared towards software development, allows using the implementation language produced by the code generator, e.g., C++, as the action language. We leave this as an open decision for Biocharts. On the one hand, the main advantage of using an existing programming language for the action language is easy integration with other software modules and tools, which is an important aspect of Biocharts since one of the main ideas is that the lower level modules can be described in appropriate existing languages and tools. However, on the other hand, defining a special-purpose action language may be very beneficial, as it makes the models more language and platform independent. Also, by carefully restricting the expressive power of the special action language compared to a language like C++ , verification and analysis tools can be more effective, which is also an important consideration for biological modeling.

Events represent asynchronous communication, while triggered operations represent synchronous communication. Both are supported in Rhapsody and we suggest that they also be supported by Biocharts. Both an event and a triggered operation are invoked by a sender object that invokes the destination object. For an event a special event instance is created and is placed in an event queue. The sender object can then continue its work without waiting for the receiver to consume the event, which will take place later when the event reaches the top of the queue and is dispatched to the destination object, potentially triggering a transition. Even though an event queue does not seem to have any direct counterpart in biological systems, our current experience shows that events are a practical solution for modelers, since the event does not have to worry about the receiver being able to communicate. This fits well with the modeling view that considers entities in the model as autonomous agents.

A triggered operation is a synchronous operation, thus the sender object must wait until the receiver object responds to the invocation, possibly causing a transition. Triggered operations may return a value to the calling object, the value being specified on the transition. If no transition is triggered by invoking the triggered operation, a special value is returned and the sender object can proceed. Our current experience from several statechart-based models that used Rhapsody shows that events were used more often than triggered operations, although we believe both are useful for biological modeling.

Events can also have attributes (variables), which the sender object sets to concrete values when sending an event. This feature is useful for biological modeling where one needs to deal with more quantitative information about the strength of certain signalling events. Events can be sub-classed, a mechanism that can be used in order to add attributes. In particular, if event e' is derived from event e in this way, e' will trigger any transition that has e as a trigger.

5.4 Inheritance and Statechart Modification

A basic feature of object-oriented programs is inheritance, which allows class B to be a subclass of class A, thus inheriting its variables and methods, which can later be modified by the programmer of class B. Rhapsody deals with inheritance of reactive classes (ones that have a statechart), by copying the statechart of the superclass to the derived class, and allowing the modeler to perform certain restricted changes in the derived class. In particular, B inherits all As states and transitions, and these cannot be removed, though certain refinements are allowed.

Inherited states can be modified in three ways: 1) decomposing a basic state by Or (into substates) or by And (into orthogonal components); 2) adding substates to an Or state; and 3) adding orthogonal components to any state. Transitions can be added to the derived statechart, and certain modifications are allowed in the original inherited ones: the target state of an inherited transition can be changed, for example, and certain changes are also allowed in the guard and action.

Our current experience in biological modeling points towards allowing in Biocharts the modeler to perform any changes he would like to the derived statechart, including building it from scratch. In case the modeler decides to indeed perform only very well controlled changes in the derived call, it will be beneficial if the tool can provide traceability and show visually where exactly changes were made.

5.5 Dynamic Changes to a Statechart

We propose a new feature for Biocharts, which we believe will become useful for biological modeling but which was not supported in previous statechart semantics. This is the ability to change the statechart itself during runtime. The motivation for this feature is that in biology there is a stronger connection than in software between the structure of the system and its behavior, and a natural way to represent changes in the structure of the biological system is by adding or removing a transition, adding a new basic-state, or changing the target of a default transition.

We propose to support such a dynamic change on the object level or on the class level. If the change is on the object level, it can be invoked by calling, for example, the method $O.RemoveTransition(t)$ for object O that contains transition t, and the result at runtime would be that for object O transition t from this point onwards will not be considered. Invoking such a call on the class level, for example by calling $C.RemoveTransition(t)$, will remove transition t from all existing objects of class C, and future instances of the class will be created without this transition.

Although supporting this feature in an implementing tool, such as Rhapsody, will require overcoming various technical challenges, we believe it is a very natural representation of the dynamic changes in a biological system and will allow one to perform *in-silico* mutations and various perturbations in a natural and elegant way.

5.6 Steps

At the heart of statechart semantics is the precise definition of the effect of a step, which takes the system from one stable configuration to the next one. In general, we propose to adapt the definitions of the Rhapsody semantics; for a detailed description see [HK04]. However, we leave open two key issues, for which we suggest to carefully consider whether to adopt the Rhapsody semantics or the Statemate semantics [HN96].

These issues are these: (i) Should changes made in a given step take effect in the current step or only in the next step; (ii) does a step take zero time or can it take more than zero time.

In Rhapsody, changes made in a given step take effect in the current step, and a step can take more than zero time, whereas in Statemate changes made in a given step take effect only in the next step and a step takes zero time. In Rhapsody, as mentioned earlier, the action language is the programming language that serves as the target for the code generator, thus it would have been difficult to postpone to the next step the effects of changes caused by running part of an action in a given step. Also the zero time step assumption does not hold for such general action languages. However, if a special action language is defined for Biocharts, this opens the way to consider adapting the Statemate semantics for these two issues.

Another issue involves how to resolve conflicting transitions. Roughly speaking, Rhapsody gives priority to lower level source states, while Statemate gives priority to higher level ones. We propose to adopt the Rhapsody priority scheme, since it is intuitive in an object-oriented setting, in that it allows to "override" behavior in lower level states. Rhapsody tries to detect and disallow transitions that have the same trigger and guard and have the same priority, with the motivation that the generated code is intended to serve as a final implementation and for most embedded software systems such nondeterminism is not acceptable. For biological modeling nondeterminism is very common and useful, so we suggest to support it in Biocharts too.

We also think that Biocharts should include support for the *clear history* operator, which erases the history of a state so that the next time the history connector for this state is entered the default transition will be taken. The motivation for supporting this feature is that it corresponds to biological phenomena that may be modeled in an easier way with clear history. We also suggest to revisit the way null transitions are handled in Rhapsody to ensure that one can specify a null transition with a guard and it will be taken immediately when the guard becomes true. In Rhapsody, such guards were evaluated on entering the state, so later changes did not always take effect unless they were associated with a visible event.

5.7 Low Level Modules

We now explain how the low level part of the two-tier language of Biocharts, which will typically describe the dynamics of the biological pathways and networks, is integrated with the high-level statechart part.

A state in a Biochart's statechart can include a 'low-level' module, which is a program P. This P is activated on entering the state, by calling $P.Start$, and is stopped when the state is exited, by calling $P.Stop$. The program P has input variables $x_1, x_2, \cdots x_l$, output variables $y_1, y_2, \cdots y_m$ and local variables $z_1, z_2, \cdots z_n$. The input and output variables are part of the object's variables, so that, for example, another program P' activated in an orthogonal state can use an output variable of P as one of its input variables.

Input variables are accessed by the program P by calling $x_i.get()$ initially and at any stage of its computation. Similarly, P can set the value of the output variables by calling $y_j.set(val)$. A Biocharts framework for supporting complicated scenarios with shared variables will need to support locking and notification of value changes for shared variables. To support hybrid modeling some of the local variables z_i can be continuous, and their dynamics would then be determined by a set of differential equations.

6 A Pledge

We are fully aware of the fact that in this paper we have only touched upon some of the issues around the semantics of Biocharts. In fact, there are several issues about what to include in, or exclude from, the language itself prior to defining the semantics rigorously. Obviously, all this has to be done before the language can be viewed as a complete modeling medium for biological systems.

The truth is that not only do we deeply miss Amir Pnueli personally, but we are confident that he would have been the ideal colleague with whom to continue this line of work. His pioneering research on hybrid systems, his unparalleled understanding of semantic issues for reactivity, and his rare scientific wisdom, would all have made the future work on this topic far easier, and the results far better than we can ever expect them to become in his absence.

Nevertheless, we pledge to forge forward with this work, with whatever talents and abilities we can muster, and bring it to a state where it can be seriously evaluated and hopefully then adopted and used beneficially.

References

[EHC07] Efroni, S., Harel, D., Cohen, I.R.: Emergent Dynamics of Thymocyte Development and Lineage Determination. PLoS Computational Biology 3(1), 3–13 (2007)

[End05] Endy, D.: Foundations for engineering biology. Nature 438, 449–453 (2005)

[FPH+05] Fisher, J., Piterman, N., Hubbard, E.J.A., Stern, M.J., Harel, D.: Computational Insights into C. *elegans* Vulval Development. Proceedings of the National Academy of Sciences 102(6), 1951–1956 (2005)

[Har87] Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8, 231–274 (1987); Preliminary version: Technical Report CS84-05, The Weizmann Institute of Science, Rehovot, Israel (February 1984)

[Har05] Harel, D.: A Turing-Like Test for Biological Modeling. Nature Biotechnology 23, 495–496 (2005)

[HG97] Harel, D., Gery, E.: Executable object modeling with statecharts. Computer 30(7), 31–42 (1997); Also in: Proc.18th Int. Conf. Soft. Eng., Berlin, pp. 246–257. IEEE Press, Los Alamitos (March 1996)

[HK04] Harel, D., Kugler, H.: The RHAPSODY Semantics of Statecharts (or, On the Executable Core of the UML). In: Ehrig, H., Damm, W., Desel, J., Große-Rhode, M., Reif, W., Schnieder, E., Westkämper, E. (eds.) INT 2004. LNCS, vol. 3147, pp. 325–354. Springer, Heidelberg (2004)

[HN96] Harel, D., Naamad, A.: The STATEMATE semantics of statecharts. ACM Trans. Software Engin. Methods 5(4), 293–333 (1996)

[HP85] Harel, D., Pnueli, A.: On the development of reactive systems. In: Apt, K.R. (ed.) Logics and Models of Concurrent Systems, New York. NATO ASI Series, vol. F-13, pp. 477–498. Springer, Heidelberg (1985)

[KCH01] Kam, N., Cohen, I.R., Harel, D.: The immune system as a reactive system: Modeling t cell activation with statecharts. In: IEEE International Symposium on Human-Centric Computing Languages and Environments (HCC 2001), pp. 15–22. IEEE Computer Society Press, Los Alamitos (2001)

[KLH10] Kugler, H., Larjo, A., Harel, D.: Biocharts: A Visual Formalism for Complex Biological Systems. J. R. Soc. Interface (2010)

[SCDH08] Setty, Y., Cohen, I.R., Dor, Y., Harel, D.: Four-Dimensional Realistic Modeling of Pancreatic Organogenesis. Proceedings of the National Academy of Sciences (2008)