Hindawi Publishing Corporation
Computational Biology Journal
Volume 2016, Article ID 4106329, 12 pages
http://dx.doi.org/10.1155/2016/4106329

Review Article

Survey of Engineering Models for Systems Biology

Gregory T. Reeves[1] and Curtis E. Hrischuk[2]

[1]*Department of Chemical and Biomolecular Engineering, North Carolina State University, Raleigh, NC 27606, USA*
[2]*ACM, Morrisville, NC 27709, USA*

Correspondence should be addressed to Curtis E. Hrischuk; hrischukc@acm.org

Received 31 August 2015; Revised 18 December 2015; Accepted 20 December 2015

Academic Editor: Rituraj Purohit

In recent years, the field of systems biology has emerged from a confluence of an increase both in molecular biotechnology and in computing storage and power. As a discipline, systems biology shares many characteristics with engineering. However, before the benefits of engineering-based modeling formalisms and analysis tools can be applied to systems biology, the engineering discipline(s) most related to systems biology must be identified. In this paper, we identify the cell as an embedded computing system and, as such, demonstrate that systems biology shares many aspects in common with computer systems engineering, electrical engineering, and chemical engineering. This realization solidifies the grounds for using modeling formalisms from these engineering subdisciplines to be applied to biological systems. While we document several examples where this is already happening, our goal is that identifying the cell as an embedded computing system would motivate and facilitate further discovery through more widespread use of the modeling formalisms described here.

1. Introduction

Science progresses by developing predictive models of reality to refine and validate hypotheses. Systems biology uses models to learn how systems behave, develop hypotheses about that behavior, contrast hypotheses with each other, or compare experimental results with a model's predicted results [1]. A repeated aspiration is for systems biology to become more like an engineering discipline [1–4]. While lacking a design element, the science of systems biology does benefit from taking an engineering approach, as it is often used to reverse-engineer complex biological systems [5–12]. Regardless, it would be helpful to identify an engineering discipline that is most like systems biology so that the related engineering processes and tools can be assessed for use within systems biology. But what is the process to do this? This paper is a step forward in answering this question.

Just as engineering consists of more than one discipline, there are also many different definitions for systems biology comprising cycles of modeling, prediction, and experiment (see [4, 13–16]). Models associated with systems biology can be categorized in different ways, such as (i) operational descriptions versus mathematical equations [1, 17]; (ii) time descriptions and concurrent behavior [1]; and (iii) macro- versus micromodeling [18]. Developing and applying the breadth of model types is challenging because there is no underlying model paradigm or relationships between model types. On the other hand, engineering has dealt with this by developing best practices around modeling paradigms and frameworks that are mathematical. Therefore, here we consider that systems biology could reuse some of these engineering best practices if the right engineering (sub)discipline is determined.

Selecting the (sub)discipline of engineering that is most like systems biology begins with the definition of the problem space. Following Brenner [19], a starting level of abstraction is the cell because it is the fundamental unit of structure, function, and organization of living systems. However, there are several information processing taxonomies in the computational sciences, and thus the one that best characterizes the cell needs to be selected. Given that the cell's self-preservation and replication processes are control systems, we deduce that an embedded computer system architecture is the appropriate architectural model. Further examination reveals an isomorphism between the concurrent, embedded, and reactive properties of an embedded system in comparison with the cell.

We surmise that an embedded system serves as the cell's exemplar computer system architecture and survey the available, applicable engineering tools for embedded system validation and dynamic analysis. In particular, three different modeling formalisms are considered: (1) analog control theory models; (2) process models; and (3) actor-based models. A review of these modeling formalisms shows that they have the expressive power to capture the essence of an embedded system: contemporary or biological. The modeling frameworks are differentiated by their purpose in the context of embedded systems, such as the ability to model steady state, dynamic, or transient behaviors. In general, a model is most helpful when dynamics are involved, such as time-based cellular behavior.

We find that there are instances where such embedded system modeling formalisms are already being used in systems biology research and cite these specific examples. This validates that embedded system engineering best practices and modeling techniques are applicable to systems biology. With the identification of the cell as an embedded system, our ultimate goal with this review paper is to provide the rigorous framework to encourage further modeling efforts such as these.

As this paper is bridging the world of computation and cellular biology, there will be references to things that are similar in both domains so it is necessary to easily identify which domain is being referred to. Throughout the paper, we use the term *contemporary* to refer to those computational things with which we are familiar, while the term *cellular* refers to things that are like contemporary computing elements but the context is within the cell.

2. Biological Computing

The intuition that the cell has computing infrastructure has been acknowledged by various investigators. For example, both contemporary computers and cells use a one-dimensional code to operate a multilayered system [20]. DNA satisfies the criteria of Turing's theoretical computing engine [21, 22] and has been compared to both a super-parallel computer and a storage medium, complete with intelligent heuristics and rewritable addressable data [23–25]. However, the complexity of the cell and its corresponding analogies to contemporary computing systems run deeper than simply DNA. The cell *as a whole* has also been viewed as a computational machine similar to a multiprocessing cluster, with a combination of a centralized instruction set and distributed computing elements [26, 27].

These analogies motivate a deeper examination of the computing elements that exist in the cell. In particular, we ask "*What contemporary computing system architecture bears the closest resemblance to the cell's computing system*?" Therefore, in this section, we will further explore the analogies between components of the cell and contemporary computers. Next, we will compare and contrast the types of contemporary computing machines with what we find in the cell. Finally, we describe in detail the identity between the cell and an *embedded computer system*.

2.1. Computing Elements of the Cell. The central dogma of molecular biology states that the information flow among DNA, RNA, and protein can only move along very specific channels: DNA → DNA (replication), DNA → RNA (transcription), RNA → DNA (reverse transcription, unusual), and finally RNA → protein (translation) [28]. Remarkably, these channels of information transfer and transformation have direct analogies in communication theory [29] and computational algorithms [30]. Furthermore, DNA, RNA, and proteins are computing elements that can be used to execute computational solutions [31]. Thus, it is clear that cellular components can act as computational hardware; this section examines these components. The three main elements that are examined are the executing control program, CPU processing elements that executes the control program, and a temporary storage medium (i.e., RAM).

DNA acts as the cell's executable program stored on a hard drive like memory: most of the instructions needed to carry out the basic functions of the cell are encoded in the DNA. The DNA instructions must also be faithfully copied as the cell produces daughter cells. However, the analogy of DNA with contemporary computing elements does not end there: DNA can also be thought of as a control center. DNA controls the cell's self-preservation and replication processes. These processes are algorithms where proteins react with other proteins to advance the algorithm until its completion. DNA is the locus of control for the protein manufacturing, coordination, and destruction. It is not simply an analogy that DNA is a control program, as there is a proof by example that DNA is a computational system since there is an entire research area where DNA software is programmed and executed to produce a computational result [25].

Just as silicon transistors are the building blocks of contemporary computers, proteins are building blocks of the cell's computer system. Proteins provide the input and output functions of a conventional computer. From an execution viewpoint, a protein is the equivalent of a contemporary CPU instruction even if its construction has multiple steps. The contemporary CPU analog of this is as follows: a single CPU instruction (i.e., protein) is composed of several microcode instructions (i.e., one or more genes and noncoding RNA). In this fashion, the transcription and translation processes are the equivalent of contemporary CPU execution decoding step, where the *cis*-regulatory code, as well as the spliceosome, instructs the manufacture of proteins to advance a cellular algorithm through protein execution. Therefore, DNA is the lowest level of software—the CPU microcode—in a hierarchically layered, biological computing system. Much prior research characterizes this layering, so a summary is provided in Table 1 (largely from Wang, Degeng, Gribskov 2005, and others as noted), mapping contemporary computing elements to a biological equivalent. Table 1 illustrates how structures of proteins build higher level computational functions such as forming logic gates, flip-flops for memory, timing mechanisms such as ring oscillators, and other computational elements.

Furthermore, the cell's memory architecture is also hierarchical, as described in Table 1. It is organized with primary memory for the fastest access, secondary memory for slower

Table 1: Multilayered computing architecture overview.

Level	Function	Implementation in computer	Implementation in a cell
Digital logic	Processing engine	Gate (organized into circuit) as basic element	Amino acid as basic element Cytoplasmic protein as a logic or signal processor [21]
	Information code	Binary with a base-2 system	Quaternary (Qbits) representing a base-4 system
	Primary (main) memory	Gate (organized into register) as basic element	Ribonucleotide as basic element Engineered gene circuits [32]
	Latch	Flip-flop	
	Timer/oscillator	Ring oscillator	Repressilator for the circadian rhythm [32]
	Secondary (working) memory	Miscellaneous, depending on media	Deoxyribonucleotide as basic element RNA as a temporary register memory [21]
Microarchitecture	Basic computing structure	Data path (the ALU, a circuit, connected with a number of registers)	Protein
Information system architecture	Instruction specification	Predetermined formatting	The transcribed region of a gene; a gene per CPU instruction [20]
	Instruction function	Controlling the data path	Enabling a chemical reaction Similarly, a protein, proposed as basic computational elements in living cells [33], is often involved in controlling a step of biochemical reaction through catalysis [20]
	Data specification	Type and format	Substrate specificity
	Memory addressing	Numerical	Promoter and enhancer binding
	Randomly accessed persistent storage	Memory bank or disk drive	Chromosomes [24, 26] Ribonome (RNA space) [20]
	Low level memory layout	RAM chip	
	High level data formatting	Drive formatting	The histone code and its control mechanisms [26]
	Cache (secondary memory) management	Software algorithm to implement a replacement policy	Cellular gene expression regulation: catalysis abundance and how active the catalysis is Protein degradation [20] Proteome (protein space) to computer cache memory
Operating system	Memory management	Swapping information in and out of primary and secondary memory	Gene transcription and RNA degradation
	Step in a process	Segment of a computer program (e.g., software function)	Many genes of a biochemical pathway are organized into one operon in prokaryote cells [20]; in eukaryotes, functionally related genes share common indexes (transcription factor binding sites)
	Process management	Shared resource such as semaphore	Shared route by pathways
		Interprocess signal	Pathway cross-talk
		CPU time sharing	Pathway bandwidth management
	File system	Continuous or indexed	Discontinuous indexed (transcription factor binding)

access, persistent memory to keep data around for long time, and hard drive-type bulk storage. RNA acts as an intermediate, transitory message exchange between DNA and proteins and thus is analogous to random access memory (RAM). And while the proteome may also fill this analogy, Cavin et al. specifically suggest that RNA is part of a memory interface and can also function as a memory buffer (as is the case in DRAM or SRAM) and as the component that mediates interactions between logic, memory, and input/output operations [21].

These are all familiar elements of contemporary computer systems. However, the cell also possesses rare computational properties. For example, if a protein is the equivalent of a CPU instruction, then the cell therefore possesses an untold amount of parallelism since there can be many copies of the same protein available. Note also that the parallelism in

memory-reading is also massive, with multiple copies of the RNA polymerase protein complex reading and transcribing DNA loci simultaneously [21]. This has an equivalent in contemporary computers where two different CPU architectural aspects are merged together:

(i) A single instruction (i.e., multiple instances of the same protein) operates on multiple data entities (parallel data computation using either vector processors or Very Long Instruction Word Processors (VLIW)).

(ii) Asynchronous processors do not use a clock to order the steps of the execution [34].

Thus, the cell's computational hardware is incredibly advanced compared to the contemporary version. With that in mind, we nonetheless proceed to describe a top-down approach to better frame the discussion.

2.2. Computer System Architecture Taxonomy. A *computer system architecture* (CSA) is a holistic pattern for how hardware and software components are used to solve a specific class of problem. There are several different CSA types and the selection of an appropriate CSA is based upon the type of problem to be solved. A taxonomy is as follows:

(i) *General Computing.* This is the most familiar CSA where input data (e.g., expense receipts) is transformed into some fashion (e.g., a spreadsheet tabulates the receipts) and output produced in another form (e.g., print an expense report). The focus is on flexibility. Examples of this class of computer system would be a laptop, a cell phone, or mainframe.

(ii) *Scientific (High Performance) Computing.* It is tailored for fast numerical computation to solve equations or perform simulations (e.g., fluid dynamics). There may be more than one processing element. The focus is on efficient, parallel execution of an algorithm on a predefined data set [35].

(iii) *Distributed Applications.* They extend general computing to include more than one server that distributes the processing of a flow of work, such as a client-server (2 layers) or client-application-database (3 layers) application structure. The focus is on general computing with parallel execution, high availability, or high throughput.

(iv) *DNA Computing.* It solves NP complete problems by the brute force of massive parallelism. Errors in the solution set are possible [36] and compensated for. The focus is on massive concurrency.

(v) *Embedded Systems.* These are computing systems that are embedded within specific physical products, such as the onboard computer in a car, autopilot in a plane, or the guidance system in a missile. These systems have real-time constraints, as well as dependability (robustness) and efficiency constraints [37]. The focus is on real-time interaction with a physical system to achieve prescribed objectives.

Selecting which CSA best approximates a cell is done by seeing which architecture best characterizes the two key cellular processes of self-preservation and replication. These two processes are achieved through (i) algorithmic control of distributed chemical processes; (ii) continuous time feedback where time is an important factor, such as ensuring that adequate levels of material (proteins) are available or that processes start (or stop) at the right time; and (iii) the ability to interface with protein sensors or transducers to interact with systems internal or external to the cell. While this is an incomplete list, it is sufficient to identify that the *embedded system* is the appropriate CSA for the cell.

2.3. The Cell as an Embedded Computer System. People use several embedded systems every day without realizing it. An embedded system is one or more custom computers that interact with physical systems, including GPS products, the autopilot in a plane, the computer system in a car, or the manufacturing robots in a factory. As such, contemporary embedded systems must have the following operational properties:

(i) *Environmental Interaction.* They must measure properties of their environment via multiple types of sensors and use this information to interact with physical processes, via control elements (actuators) [38]. This interaction is software controlled but mediated through actuators, servomotors, machines, or partially autonomous robots. At the lowest level, this may involve mechanical, electrical, or chemical mechanisms. This is a defining characteristic of embedded systems.

(ii) *Concurrence.* They must handle multiple independent stimuli simultaneously [38], so parallel operation is a necessity.

(iii) *Reactiveness.* They must respond in a timely manner to avoid system failures. As there is continual interaction with the external (physical) environment, the execution of instructions must keep pace with timescales determined by the environment [39].

(iv) *Liveness.* Their critical processes must not terminate or stall.

(v) *Robustness.* They must have the capability of adapting to changing conditions, including internal failures. Service demands, computing resources, and sensors may appear and disappear during the life of the embedded system.

(vi) *Heterogeneity.* Their processing capabilities must span various computational styles and implementation technologies. Indeed, heterogeneity is integral to computation in embedded systems [38]. The custom processors are known as ASIC (Application Specific Integrated Circuits) [40]. These processors are tailored for specific purposes and may have custom instruction sets, multiple processing units, high performance, very low power needs, and so forth.

Aside from these operational properties, an embedded system has environmental and structural features that differentiate it from other CSAs. Since it is embedded into the environment that it is interacting with, the embedded system must manage energy and resource (e.g., computational and memory) usage very efficiently. This is exemplified in the cell, with the use of thermal energy to drive the chemical algorithms, in conjunction with adenosine 5′-triphosphate (ATP), to act as the energy transfer mechanism that powers the protein complex. In general, the hardware and software operate for maximal resource efficiency at a required performance level.

Albertos et al. point out that another key differentiator is that an embedded system must function properly if resources are unavailable for periods of time, when data is delivered with variable delay or even if data is missing [41]. The system may operate in a degraded state but it will continue to operate. Ideally, the embedded system would heal itself, which is not possible with contemporary computer systems but is normal for cellular systems. Hardware and data redundancy provide fault tolerance, which is also built into the cellular system.

A review of the cell's processes, including self-preservation and replication, shows that the cell does indeed have the properties of an embedded system. The cell has a multitude of sensors/transducers for extracellular stimuli, such as those for pH, light (UV and visible), and many other applications, which are necessary for the cell to survive. As a specific example, in the bacterial chemotaxis system, sensors (in the form of transmembrane receptors) are clustered at the front of the cell and measure the local concentrations of various chemicals [42–44]. Each receptor is specific to a particular kind of chemical, such as amino acids, sugars, and molecular oxygen [45]. The concentration is measured by the extent of binding of the ligand to the receptor.

The example of receptors is not limited to bacterial chemotaxis. Plants sense visible light, and certain plants use this information to orient themselves towards light. Recently discovered rhodopsin molecules within human skin cells sense UV light and begin the pigmentation (tanning) process before DNA damage occurs [46]. It is also evident that most animals can sense light through photoreceptors in their eyes and aromatic chemicals through olfactory receptors.

Furthermore, cells require protein pathways that act as actuators to interact with and manipulate the physical world. Returning to the example of bacterial chemotaxis, flagellar motors, which are screw-like protein assemblages on the exterior of the cell, act as propellers to allow the bacterium to swim through the medium [35, 36]. Some cells use cilia—whip-like structures analogous to (but smaller than) flagella—to stir the local medium to improve feeding efficiency or even to create local flows for right/left symmetry breaking [47, 48].

In the chemical processing industry, final control elements (actuators) are most frequently valves to control flow to and from a process [49]. Valves are ubiquitous as final control elements in biology as well. For example, ion channels act as specialized valves to regulate internal pH and osmotic pressure; small molecules can enter the cell through still other specialized valves aided by transport enzymes. Perhaps one of the most striking examples of a biological actuator is a class of proton channels such as cytochrome C. This remarkable ensemble of proteins acts as a valve to control proton flow, coupled to an electrical motor, to generate useable energy for the cell. While the examples given here of transducers and actuators are clearly not exhaustive, they are sufficient to show that the cell satisfies the embedded property.

Returning to the cell's replication and self-preservation processes, there are a range of different proteins involved in those processes, satisfying the concurrent and heterogeneous properties. Finally, the reactive, live, and robust properties evidently must exist as part of the cell's self-preservation processes to maintain viability. Therefore, the cell can be classified as an embedded system, as it has the defining properties.

If the cell is an embedded system, then can the modeling tools used to engineer embedded systems also be used to understand the cell? In the next section we show that the cellular embedded system paradigm can be used to construct predictive models capable of comparing dynamical (i.e., time-based) experimental results with model predictions and then refining hypotheses/predictions in further research. We explore three types of time-based embedded models applied to the cell and cite several examples. While these modeling approaches have not yet been extensively validated (indeed, some are speculative), they all use the embedded system paradigm as the basis for the mathematical models, whether the researchers originally intended to or not.

3. Modeling Formalisms

We have shown that the cell is a very specific type of computing system: an embedded system. In this section, we will explore the different modeling formalisms for an embedded system that can be applied to the cell.

3.1. Control System Modeling. A contemporary embedded system controls a physical system based on control theory and engineering principles. Control theory uses ordinary differential equation-based models to study and predict transient and steady state behavior of physical systems. These ordinary differential equations (ODEs) model how the variables change in time (but not space). A car's cruise control is a canonical example of control theory implemented by an embedded system. Other more advanced applications are pH control [50] and paper manufacturing [51].

The generic control theory block flow diagram is shown in Figure 1 [49, 52], where feedback is introduced to keep the output within a particular range. The feedback system consists of the comparator, feedback controller, actuator, sensor, and processing element. The controller makes adjustments to the process input (through the actuator) in real time to ensure that the output value is maintained at or near a set point. These adjustments can be made in proportion to the error from set point and often take into account the time history (i.e., integral) of the error as well. In some cases, the controller also takes into account how fast the error is changing (i.e., derivative control). Other controller algorithms besides proportional-integral-derivative (PID) control are common as well. Transducers or sensors of process variables (such as temperature, pressure, flow rates, levels, density, and pH)

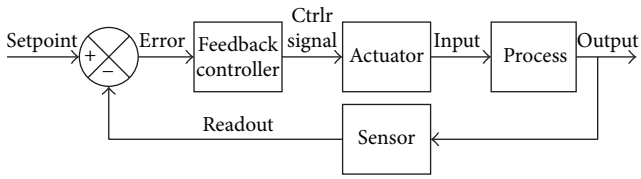

Figure 1: Elements of a generic controller.

measure process outputs that are fed back to adjust the behavior [53].

An embedded system is an embodiment of this generic control theory model controller. It is a computer that is *embedded* within the physical system whose hardware or algorithm is flexible enough to perform the multiple functions of difference, derivative, and integration operations. If the embedded system is designed for concurrent and reactive responses, then it is even more economical because more than one controller can be executed.

In the cell, a controller has been shown to be implemented in an analog fashion where the signal is analog (i.e., not quantized into a digital format, such as a silicon-based computing system) and the data input stream is continuous time (i.e., not broken into steps governed by a global clock, such as a modern CPU). Contemporary controllers of this form were referred to as analog computers and have a long history prior to the digital revolution [54]. The ODEs are then modeled in the form of an analog computer with appropriate signal conditioning applied to highlight the key signal used in the processing. Biological examples of this type of controller can be found in [55–57].

An analog computing controller is well suited for processes modeled by ODEs, such as those that are nonconcurrent, continuous time, always on, and which require relatively fast responses. However, if part of the feedback involves turning on and off output or intermediate processes, then a purely analog solution is not possible because there must be some aspect of *if-then-else* logic at work. This type of cellular promotion or inhibition of analog controllers can be found in genetic regulatory networks, in which approximate logic gates control the expression of critical controller components [58]. The contemporary view of this is called a Programmable Logic Controller (PLC), which is a digital computer that controls some physical process. The cellular embedded system exhibits the qualities of a PLC.

A control system can also be implemented as a purely digital controller once the input signal is quantized. While this digital controller structure requires the input signal to be quantized, it does not need to be the familiar contemporary binary digital format; any discrete format will do. For example, it may be a nonuniformly distributed form of quantization (e.g., logarithmic quantization; [59]) which, by its nature, would embed a signal conditioning activity through the quantization. It may be a stretch to think of the cell as having this sophisticated type of processing capabilities; however, surprise is nothing new when examining the complexities of the cell. Although speculative, an instance of where this type of digital control may occur is in the gene expression

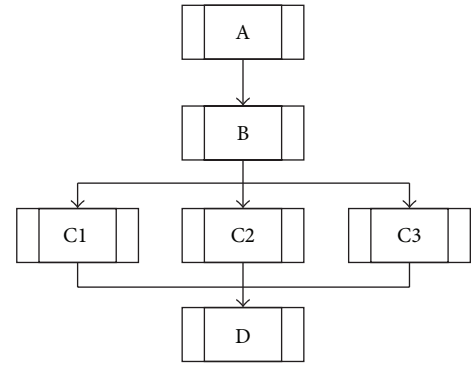

Figure 2: Simple biological process.

response to the oscillatory behavior of transcription factor activity found in some eukaryotic cells. Transcription factors are proteins that regulate gene expression and thus must be present in the nucleus to be active. The nuclear localization of some transcription factors is oscillatory in time, and the frequency of these oscillations has been shown, in some cases, to determine the nature of the gene expression response (e.g., see [60, 61]). This type of digital computing controller is well suited for processes modeled by ODEs, nonconcurrent, discrete time, always on, which does not require a fast response.

This control model approach is well known and is commonly implemented in chemical engineering disciplines. The next section discusses a computational model that does not include ODEs.

3.2. Process Modeling of the Cell. Most computer inspired examinations of the cell's bioinformatic system focus solely on the computational hardware (i.e., mechanisms). In this section, we consider an alternative approach that takes an end-to-end *process* view, emphasizing *what must* be done and not *how it will* be done. Within this *process modeling formalism*, a sequential algorithm of several steps is considered to be a simple process. More sophisticated processes can possess concurrency, communication, information exchange, and other sophisticated attributes to control behaviors. Here we describe the attributes of the process modeling formalism, cite examples of this formalism in contemporary systems, and conclude with examples from the biological literature.

In the process modeling formalism, the focus is on algorithmic processes that proceed in steps. There may be many steps that occur both in sequence and in parallel, with communication occurring between and across steps and with indeterminate choice of the next step. For illustrative purposes, Figure 2 is a simple process with four states that can be in {A, B, {C1, C2, C3}, D}, where one of the states has three concurrent, simultaneous internal states to reflect that three inputs are needed to move it to the next state D. This process is an algorithm in the sense that it properly describes what must be done and not what components are involved. Each arrow represents a computational step in the process, which, in the biological context, may represent a protein-catalyzed reaction that moves a chemical process closer to its final output. If each box is a simple state that does not

have internal subprocesses (i.e., substates), then the simplest transition is due to a single CPU step (contemporary) or protein reaction (cellular). Clearly each box could itself have internal processes that are more complex and descriptive than this. For the purpose of this paper, it is enough to recognize that moving from step A to B requires some activity, such as executing a single CPU instruction, a software function, or a message interaction that moves the process to B. In cellular terms, state A may represent the translation of a protein that is manufactured to move from state A to B, which would mean that A has subprocesses internal to it that are hidden.

It should be noted that the process modeling formalism allows simple models to form larger models by aggregating or combining them together. In reverse, higher level, coarse grained models can be recursively decomposed to add more details as warranted. Although Figure 2 is a simple example, it illustrates how recursive decomposition can be used to describe a much more complex process.

A processes modeling formalism can be described independent of time elements to characterize behavior. Such untimed process model languages have formal descriptions in computer science [62–64]. They include operators such as process state change, deterministic choice (if-then-else), nondeterministic choice, concurrency, synchronization between states, abstraction of parallel operations into a single state, abstraction of entire processes into a single state, composition of new processes by combining processes, replication, timed delay [65], communication path changes [66], interactions with the environment [67], resource constraints and priorities [68], and restrictions on state changes. These operators, in conjunction with recursive decomposition, provide a powerful modeling formalism. These modeling formalisms are grounded in mathematics so that properties of the process can be proven (like a theorem is proven) or the model is *executed* to produce a quantitative result that can be validated using an experiment.

The property (formula) proving aspects of this modeling formalism is useful when designing embedded systems because it is used to validate that the modeled system has required or expected properties, such that there are no deadlocks in the way processes behave. This has been used to validate deadlock avoidance, checking for timeout conditions [65, 69], ensuring that the timely execution of activities is achieved even in the presences of resource contention and synchronization [68, 70].

The modeling formalism can also use simulated execution to verify the process behavior. For example, Broenink and Hilderink discuss developing embedded control software through stepwise refinement of physical system models and control laws, using simulation to verify each step [71]. Prior to implementation, ten Berge et al. simulate a complete distributed control system including system delays and unreliable message communication of physically distributed computing elements [72]. The process modeling formalism is also used as the implementation paradigm for process control software [73, 74]. Surprisingly, even distributed web applications can make use of the process model formalism [75].

Some process modeling formalisms characterize time by adding stochastic attributes. One example of a timed, stochastic process model is called Stochastic Process Algebras (SPAs) [76]. These models include time and probabilistic factors, where time is a continuous function and the state transitions are discrete operations. Some process models use exponential distributions or continuous time Markov chains, which are simplifying assumptions to make the solution analytically tractable but sacrifice some fidelity to the real world. For example, SPAs have been used to estimate performance values for contemporary computer systems [76].

As already seen, this formalism fits nicely into the modeling of computer procedures and programs, yet it also can be directly applied to biochemical pathways and networks [20]. For example, in signal transduction, enzyme/substrate interactions are strung together to form pathways, which are then joined together to form networks. Indeed, recent work has applied SPAs to model and analyze biochemical signaling pathways [77], such as the ERK pathway, which conveys mitogenic and differentiation signals from the cell membrane to the nucleus.

Aside from the ability to describe processes in a concrete way, the process modeling formalisms have other benefits. As mentioned, they can make predictions by executing the model. Also the mathematical foundation can identify errors by enabling complex model validation to make sure that the state and operator combinations are sensible. For example, finding a process execution deadlock in a model invalidates the embedded system's *liveness* requirement, pointing to the need for model revision. Such an error has been found in the previously mentioned ERK signaling model which led to revisions of the model [77]. However, the disadvantage of SPAs is that they make simplifying assumptions to support validation. Some limitations are (i) assuming Poisson (exponential) distributions; (ii) that the process is memoryless which is not true since protein consumption changes the state of the system; and (iii) that the solution state space increases geometrically with the number of states so that larger models may take a very long time to execute. It may be possible to adapt process formalisms to cellular modeling by extending them, such as working with populations of proteins instead of single proteins or enhancing SPAs to better model biological (chemical) realities, such as diffusion.

The process model formalism is similar to that of Discrete Event System (DES) models which are used in control theory [78]. A DES model has one or more controllers that, at a high level, model a state machine that responds to inputs by changing states and producing outputs. So a single process could be viewed as a single DES controller and, by extension, multiple processes could be viewed as multiple asynchronous DES controllers. A DES model is limited by state space explosion so only very small models can be analyzed. An example of a DES modeling a gene regulatory network is [79]. The process model formalism also has a richer set of higher level operators.

3.3. Actor Modeling in the Cell. A process model is well suited when there is a dominant chemical algorithm that can be described in a simple directed graph with well-defined states and transitions. However, there are *distributed chemical algorithms* in the cell that are difficult to characterize by

Table 2: List of some actor robots in the cell.

Cell component	Description
Dynein, myosin, and kinesin	Protein machines employ lever arms, ratchets, and gear-like mechanisms ferry cellular cargo along tracks inside the cell
Ribosome	The ribosome is a gigantic molecular machine made up of proteins and RNA that manufactures proteins in an assembly-line process.
Ubiquitin E3 ligase	Ubiquitin E3 ligase recognizes malformed or obsolete proteins and tags them for degradation
Lysosomes, proteasomes	Proteasomes are large protein machines that destroy other defective protein machines. Lysosomes are organelles that ingest larger debris particles inside the cell and break them down so that their components can be recycled
Endoplasmic reticulum	The endoplasmic reticulum is a network of membranes that contain an internal channel where proteins are processed and prepared in an assembly-line fashion for incorporation into the cell membrane and secretion into the extracellular environment or packed into membrane bound organelles like lysosomes
Peroxisomes	These organelles contain proteins that oxidize cellular debris, cleaning up the cell's interior
Cell nucleus	This large organelle houses chromosomes which contain the information needed to make protein machines. It also contains the biochemical machinery that regulates the production of the protein machines and regions that make key components of ribosomes which, in turn, are the machines that make protein machines
Spliceosome and introns	These regions of the DNA molecule (introns) are variably excised by the spliceosome and spliced together to form a variety of protein machines from the same region of the DNA molecule

process models since there are a very large number of participant elements and no centralized control. Such a distributed chemical algorithm occurs when populations of proteins of different types interact to produce a result. The process is distributed and formed by the concentrations of proteins along with the gradient of protein diffusion, which is itself controlled by the rate of protein synthesis and the lifetime of the individual proteins. Here each protein serves as a fit-for-function, robot-like, computational processing element that interacts with other proteins—this is the *embedded* property. These proteins are autonomous and act concurrently. Such a distributed chemical algorithm is an embedded system, but it uses peer-to-peer, distributed control instead of a centralized process. A handful of the more well-known proteins that behave this way are shown in Table 2.

This type of distributed chemical algorithm has a corresponding theoretical computational framework called the *actor model* [62, 80, 81]. (Note that the actor model is more theoretical than the *agent based model* of Jennings [82] but, for the most part, agents and actors are considered interchangeable here.) The actor model is well suited to highly concurrent, indeterminate behavior with no centralized control or common time reference. The global behavior emerges from the behavior of the individual actors.

An *actor* is an autonomous entity with internal memory that receives asynchronous stimuli (messages). The behavior of the actor is described by a set of rules. Each stimulus causes a given rule to execute within the actor. These rules can be conditional (if-then-else) decisions, actor replication, stochastic selection based on a distribution [83], and choosing which other actor to interact with and when. The behavior of agents with the same rule set varies because of differences in local conditions and internal memory. The use of internal memory means that an agent's behavior can change over time. There may be more than one class of actors with each class having its own internal rule set. Actors can contain other actors as an act of composition or be indivisible.

The actor model formalism is used in the development of contemporary embedded systems to understand the emergent behavior of an ensemble of independent computing elements. It has detected deadlock in feedback loops, where actors cannot execute because they are waiting for input data from each other [84]. It has been used in the formal design of distributed embedded control systems [85]. Other embedded applications are the design for flying autonomous multivehicle control [86] and high-energy particle physics data acquisition systems [87]. Lastly, it is used in the design of industrial intelligent manufacturing systems of distributed robotic manufacturing [88].

Actor models characterize transient processes by simulation. A simulation step occurs where time progresses when an actor fires a rule in response to a message, which in turn may send out messages to other actors. The ensemble of actors sending and receiving messages at each step produces the corporate behavior as a function of time. The firing of a rule can be logical (i.e., a step with no temporal unit) or have a time duration. If stochastic assumptions are employed, the model can be solved by analytic techniques provided the process can be assumed to be Markovian [89].

There has been recent work using the actor model to investigate intercellular and extracellular processes. Some examples are endotoxin signaling at the cellular response level [83], how stochastic intracellular events affect cellular motility (i.e., bacterial chemotaxis) [90], and how vesicle patterns are formed via an agent-based model of intracellular transport inside a single cell [91]. A thought provoking result is that this actor model characterized *a bistable global result* that emerged from the ensemble of individual actors [83]. Reproducing nonergodic (e.g., bistable) systems is difficult to do, so this is an important result.

There are several benefits to an actor model. First, because actor models are designed to handle spatial inhomogeneity, they intuitively match many biological phenomena [83]. The model can be built independent of global knowledge by

Table 3: Taxonomy of time-based modeling of cellular behavior.

Model type	Research maturity	Formalism	Time	Concurrency	Model structure	Spatial parameters	Ergodic
Analog control system	Well known	Ordinary differential equations (math)	Steady state, with some transient	Ensemble view	Time based	No	Yes
Digital control system	Speculative	Ordinary differential equations (math)	Steady state, with some transient	Ensemble view	Time based	No	Yes
Process	Initial results	Process language	Simulated or stochastic	Medium	Top-down view	Yes	No
Actor	Initial results	Distributed autonomous	Simulated or stochastic	High	Bottom-up view	Yes	No

using only knowledge of the individual participants [92]. The protein actors may be described on an individual basis or as a population. An actor model can incorporate heterogeneity of behavior (i.e., actors have different rules for behavior) and spatial features of an environment (i.e., rule results may depend upon the position of an actor). Furthermore, the model is naturally stochastic in that the interactions can be based upon probabilities and some of the agent dynamics can be highly random. Molecule distributions, reaction rate constants, and structural parameters can be adjusted separately in the simulation allowing a comprehensive study of individual effects in the context of a realistic cell environment [93].

4. Summary

Systems biology takes a systems engineering approach to characterize biological processes. As such, systems biology necessarily relies on using modeling formalisms to synthesize data into a consistent mathematical framework. In this paper, we show that the cell is an embedded computing system and, as such, several modeling formalisms are appropriate for use in systems biology contexts. Furthermore, this reveals that the engineering disciplines of computer systems engineering, electrical engineering, and chemical engineering are relevant for development of systems biology models.

The cell is an embedded computer system because it has the properties of an embedded computer system architecture. It operates concurrently with various chemical algorithms and autonomous actors acting in parallel. It is reactive and live because an overly slow response will abort chemical algorithms, possibly resulting in the death of the cell. It is robust because it is capable of adapting to changing conditions. Lastly, the system is heterogeneous in that each protein constitutes a unique type of processing element. We acknowledge that a cell is likely more than an embedded system, but it certainly is not less than an embedded system.

The study of embedded systems belongs to the engineering discipline of embedded computer system engineering. This engineering discipline has proven techniques for designing and, more importantly, understanding the behavior of embedded systems. The modeling frameworks used to engineer embedded systems have been reviewed and found to apply to understanding cellular behavior. The application of these model types to studying cellular behavior has been validated because examples are provided of applying each model type to researching cellular behavior. The resulting inference is that embedded computer system engineering provides methods and tools for direct use in systems biology and this cross-discipline interaction is an area of further exploration. For this reason, we encourage the use of the modeling formalisms described here when modeling biological processes.

The three different embedded system models that were reviewed are summarized in Table 3. Selecting a model type depends upon the intended investigation. The important factors in selecting a model are (i) concurrency, which is the manner in which entities are described and interact; (ii) the way in which the system is characterized (model structure); (iii) spatial parameters to indicate if the volume being analyzed is part of the model; and (iv) ergodicity (a property about the type of distribution being modeled). Each of these properties is important to consider when selecting the model formalism.

Of course a model of a single cellular feature is useful; however, from the standpoint of systems biology, connecting several models to provide a more complete behavior description is more useful. This suggests two things. First there is the need to standardize the specification, storage, and retrieval of systems biology models. Systems Biology Markup Language (SBML) is useful here since it is a standard for representing and exchanging the essential aspects of a model between tools [94, 95]. Since SBML is XML based, new syntactical and semantic extensions can be made as needed, such as incorporating formalism specific concepts. The second need is the ability to cross the semantic and simulation gaps of different model types so that larger models can be built through aggregation. Thankfully this has already been done by an embedded system modeling tool called Ptolemy [96]. Ptolemy enables larger models to be composed from smaller models of different model types. These aggregated models are simulated to generate behavioral descriptions (it solves all of the model types of Table 3). This aggregation may be recursive in that aggregated models can themselves be aggregated to form larger and more complete models of cellular behavior.

Abbreviations

CPU: Central processing unit
RAM: Random access memory
DRAM: Dynamic RAM
SRAM: Static RAM

CSA: Computer system architecture
ODE: Ordinary differential equation
SPA: Stochastic Process Algebra.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

During this work, Gregory T. Reeves was partially supported by NSF CAREER Award CBET-1254344.

References

[1] J. Fisher and T. A. Henzinger, "Executable cell biology," *Nature Biotechnology*, vol. 25, no. 11, pp. 1239–1249, 2007.

[2] O. Wolkenhauer, "Systems biology: the reincarnation of systems theory applied in biology?" *Briefings in Bioinformatics*, vol. 2, no. 3, pp. 258–270, 2001.

[3] J. Stelling, "Mathematical models in microbial systems biology," *Current Opinion in Microbiology*, vol. 7, no. 5, pp. 513–518, 2004.

[4] H. Kitano, "Computational systems biology," *Nature*, vol. 420, no. 6912, pp. 206–210, 2002.

[5] K. Basso, A. A. Margolin, G. Stolovitzky, U. Klein, R. Dalla-Favera, and A. Califano, "Reverse engineering of regulatory networks in human B cells," *Nature Genetics*, vol. 37, no. 4, pp. 382–390, 2005.

[6] M. E. Csete and J. C. Doyle, "Reverse engineering of biological complexity," *Science*, vol. 295, no. 5560, pp. 1664–1669, 2002.

[7] C. J. Tomlin and J. D. Axelrod, "Understanding biology by reverse engineering the control," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 12, pp. 4219–4220, 2005.

[8] T. J. Perkins, J. Jaeger, J. Reinitz, and L. Glass, "Reverse engineering the gap gene network of *Drosophila melanogaster*," *PLoS Computational Biology*, vol. 2, no. 5, article e51, 2006.

[9] A. J. Hartemink, "Reverse engineering gene regulatory networks," *Nature Biotechnology*, vol. 23, no. 5, pp. 554–555, 2005.

[10] L. A. Johnson, Y. Zhao, K. Golden, and S. Barolo, "Reverse-engineering a transcriptional enhancer: a case study in Drosophila," *Tissue Engineering—Part A*, vol. 14, no. 9, pp. 1549–1559, 2008.

[11] K. Becker, E. Balsa-Canto, D. Cicin-Sain et al., "Reverse-engineering post-transcriptional regulation of gap genes in *Drosophila melanogaster*," *PLoS Computational Biology*, vol. 9, no. 10, Article ID e1003281, 2013.

[12] P. R. LeDuc, W. C. Messner, and J. P. Wikswo, "How do control-based approaches enter into biology?" *Annual Review of Biomedical Engineering*, vol. 13, pp. 369–396, 2011.

[13] R. Breitling, "What is systems biology?" *Frontiers in Physiology*, vol. 1, article 9, 2010.

[14] T. Ideker, T. Galitski, and L. Hood, "A new approach to decoding life: systems biology," *Annual Review of Genomics and Human Genetics*, vol. 2, pp. 343–372, 2001.

[15] T. Ideker, "Systems biology 101—what you need to know," *Nature Biotechnology*, vol. 22, no. 4, pp. 473–475, 2004.

[16] H. Kitano, "Systems biology: a brief overview," *Science*, vol. 295, no. 5560, pp. 1662–1664, 2002.

[17] C. Priami, "Algorithmic systems biology," *Communications of the ACM*, vol. 52, no. 5, pp. 80–88, 2009.

[18] A. M. Uhrmacher, D. Degenring, and B. Zeigler, "Discrete event multi-level models for systems biology," in *Transactions on Computational Systems Biology I*, C. Priami, Ed., vol. 3380 of *Lecture Notes in Computer Science*, pp. 66–89, Springer, Berlin, Germany, 2005.

[19] S. Brenner, "Sequences and consequences," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 365, no. 1537, pp. 207–212, 2010.

[20] D. Wang, "Discrepancy between mRNA and protein abundance: insight from information retrieval process in computers," *Computational Biology and Chemistry*, vol. 32, no. 6, pp. 462–468, 2008.

[21] R. K. Cavin, P. Lugli, and V. V. Zhirnov, "Science and engineering beyond moore's law," *Proceedings of the IEEE*, vol. 100, pp. 1720–1749, 2012.

[22] A. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society, Series 2*, vol. 42, pp. 230–265, 1936.

[23] C. M. Bogard, E. C. Rouchka, and B. Arazi, "DNA media storage," *Progress in Natural Science*, vol. 18, no. 5, pp. 603–609, 2008.

[24] J. Bonnet, P. Subsoontorn, and D. Endy, "Rewritable digital data storage in live cells via engineered control of recombination directionality," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 109, no. 23, pp. 8884–8889, 2012.

[25] A. Ekka and B. Sahoo, "A DNA computing approach to solve Task Assignment problem in Real Time Distributed computing System," in *Ptroceedings of the National Conference on Methods and Models in Computing*, pp. 1–10, New Delhi, India, December 2007.

[26] D. J. D'Onofrio and G. An, "A comparative approach for the investigation of biological information processing: an examination of the structure and function of computer hard drives and DNA," *Theoretical Biology and Medical Modelling*, vol. 7, article 3, 2010.

[27] J. A. Shapiro, "Genome informatics: the role of DNA in cellular computations," *Biological Theory*, vol. 1, no. 3, pp. 288–301, 2006.

[28] F. Crick, "Central dogma of molecular biology," *Nature*, vol. 227, no. 5258, pp. 561–563, 1970.

[29] H. Yockey, "Origin of life on earth and Shannon's theory of communication," *Computers & Chemistry*, vol. 24, no. 1, pp. 105–123, 2000.

[30] G. J. Chaitin, "Algorithmic information theory," *IBM Journal of Research and Development*, vol. 21, no. 4, pp. 350–359, 1977.

[31] A. Narayanan and S. Zorbalas, "DNA algorithms for computing shortest paths," in *Proceedings of the 3rd Annual Conference on Genetic Programming (GP '98)*, pp. 718–723, Madison, Wis, USA, July 1998.

[32] J. Hasty, D. McMillen, and J. J. Collins, "Engineered gene circuits," *Nature*, vol. 420, no. 6912, pp. 224–230, 2002.

[33] D. Bray, "Protein molecules as computational elements in living cells," *Nature*, vol. 376, no. 6538, pp. 307–312, 1995.

[34] T. Werner and V. Akella, "Asynchronous processor survey," *Computer*, vol. 30, no. 11, pp. 67–76, 1997.

[35] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2010.

[36] J. Chen and D. H. Wood, "Computation with biomolecules," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 97, pp. 1328–1330, 2000.

[37] P. Marwedel, *Embedded System Design*, Springer, Dordrecht, The Netherlands, 2011.

[38] E. A. Lee, "Embedded software," *Advances in Computers*, vol. 56, pp. 55–95, 2002.

[39] A. Guerrouat and H. Richter, "A component-based specification approach for embedded systems using FDTs," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 2, article 14, 2006.

[40] J. Henkel and S. Parameswaran, "Application-specific embedded processors," in *Designing Embedded Processors*, J. Henkel and S. Parameswaran, Eds., pp. 3–23, Springer, Dordrecht, The Netherlands, 2007.

[41] P. Albertos, A. Crespo, M. Vallés, and I. Ripoll, "Embedded control systems: some issues and solutions," in *Proceedings of the 16th IFAC World Congress*, Prague, Czech Republic, 2005.

[42] J. Adler, "Chemotaxis in bacteria," *Science*, vol. 153, no. 3737, pp. 708–716, 1966.

[43] J. Adler, "Chemotaxis in bacteria," *Annual Review of Biochemistry*, vol. 44, pp. 341–356, 1975.

[44] J. S. Parkinson, "Signal transduction schemes of bacteria," *Cell*, vol. 73, no. 5, pp. 857–871, 1993.

[45] J. S. Parkinson, P. Ames, and C. A. Studdert, "Collaborative signaling by bacterial chemoreceptors," *Current Opinion in Microbiology*, vol. 8, no. 2, pp. 116–121, 2005.

[46] N. L. Wicks, J. W. Chan, J. A. Najera, J. M. Ciriello, and E. Oancea, "UVA phototransduction drives early melanin synthesis in human melanocytes," *Current Biology*, vol. 21, no. 22, pp. 1906–1911, 2011.

[47] S. Nonaka, Y. Tanaka, Y. Okada et al., "Randomization of left-right asymmetry due to loss of nodal cilia generating leftward flow of extraembryonic fluid in mice lacking KIF3B motor protein," *Cell*, vol. 95, no. 6, pp. 829–837, 1998.

[48] S. Nonaka, H. Shiratori, Y. Saijoh, and H. Hamada, "Determination of left-right patterning of the mouse embryo by artificial nodal flow," *Nature*, vol. 418, no. 6893, pp. 96–99, 2002.

[49] J. B. Riggs and N. Karim, *Chemical and Bio-Process Control*, Prentice Hall, New York, NY, USA, 2008.

[50] F. Greg Shinskey and W. S. E. Levine, "Control of pH," in *The Control Handbook*, pp. 1205–1218, CRC Press, 1996.

[51] W. L. Bialkowski and W. S. E. Levine, "Control of the pulp and paper making process," *The Control Handbook*, CRC Press, pp. 1219–1242, 1996.

[52] W. S. Levine, *The Control Handbook*, CRC Press, Boca Raton, Fla, USA, 1996.

[53] C. J. Chesmond, *Control System Technology*, Edard Arnold, London, UK, 1984.

[54] B. J. Maclennan, "Analog computation," in *Computational Complexity*, R. A. Meyers, Ed., pp. 161–184, Springer, New York, NY, USA, 2012.

[55] T.-M. Yi, Y. Huang, M. I. Simon, and J. Doyle, "Robust perfect adaptation in bacterial chemotaxis through integral feedback control," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 97, no. 9, pp. 4649–4653, 2000.

[56] A. D. Lander, K. K. Gokoffski, F. Y. M. Wan, Q. Nie, and A. L. Calof, "Cell lineages and the logic of proliferative control," *PLoS Biology*, vol. 7, no. 1, Article ID e1000015, 2009.

[57] M. Cloutier and P. Wellstead, "The control systems structures of energy metabolism," *Journal of the Royal Society Interface*, vol. 7, no. 45, pp. 651–665, 2010.

[58] H. Bolouri and E. H. Davidson, "Modeling transcriptional regulatory networks," *BioEssays*, vol. 24, no. 12, pp. 1118–1129, 2002.

[59] C. Zhang and G. E. Dullerud, "Finite gain stabilization with logarithmic quantization," in *Proceedings of the 46th IEEE Conference on Decision and Control*, pp. 3952–3957, IEEE, New Orleans, La, USA, December 2007.

[60] L. Ma, J. Wagner, J. J. Rice, W. Hu, A. J. Levine, and G. A. Stolovitzky, "A plausible model for the digital response of p53 to DNA damage," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 40, pp. 14266–14271, 2005.

[61] L. Cai, C. K. Dalal, and M. B. Elowitz, "Frequency-modulated nuclear localization bursts coordinate gene regulation," *Nature*, vol. 455, no. 7212, pp. 485–490, 2008.

[62] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular ACTOR formalism for artificial intelligence," in *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pp. 235–245, 1973.

[63] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall International, 2004.

[64] R. Milner, *A Calculus of Communicating Systems*, Springer, 1980.

[65] F. Moller and C. Tofts, "A temporal calculus of communicating systems," in *CONCUR '90 Theories of Concurrency: Unification and Extension: Amsterdam, The Netherlands, August 27–30, 1990 Proceedings*, vol. 458 of *Lecture Notes in Computer Science*, pp. 401–415, Springer, Berlin, Germany, 1990.

[66] R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes, I and II," *Information and Computation*, vol. 100, no. 1, pp. 1–40, 1992.

[67] W. C. Rounds and H. Song, "The φ-calculus: a language for distributed control of reconfigurable embedded systems," in *Hybrid Systems: Computation and Control: 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3–5, 2003 Proceedings*, vol. 2623 of *Lecture Notes in Computer Science*, pp. 435–449, Springer, Berlin, Germany, 2003.

[68] H. Ben-Abdallah, J.-Y. Choi, D. Clarke, Y. S. Kim, I. Lee, and H.-L. Xie, "A process algebraic approach to the schedulability analysis of real-time systems," *Real-Time Systems*, vol. 15, no. 3, pp. 189–219, 1998.

[69] J. Parrow, "Verifying a CSMA/CD-protocol with CCS," in *Proceedings of the 8th IFIP Symposium on Protocol Specification, Testing and Verification*, S. Agarwal and K. Sabnani, Eds., pp. 373–384, North-Holland Publishing, Atlantic City, NJ, USA, June 1988.

[70] A. N. Fredette and R. Cleaveland, "RTSL: a language for real-time schedulability analysis," in *Proceedings of the Real-Time Systems Symposium*, pp. 274–283, December 1993.

[71] J. F. Broenink and G. H. Hilderink, "A structured approach to embedded control system implementation," in *Proceedings of the IEEE International Conference on Control Applications (CCA '01)*, pp. 761–766, September 2001.

[72] M. H. ten Berge, B. Orlic, and J. F. Broenink, "Co-simulation of networked embedded control systems, a CSP-like process-oriented approach," in *Proceedings of the IEEE Conference on Computer Aided Control System Design, IEEE International Conference on Control Applications, and IEEE International Symposium on Intelligent Control*, pp. 434–439, IEEE, Munich, Germany, October 2006.

[73] P. Welch, "Process oriented design for java: concurrency for all," in *Computational Science—ICCS 2002*, vol. 2330 of *Lecture Notes in Computer Science*, p. 687, Springer, Berlin, Germany, 2002.

[74] G. Hilderink, A. Bakkers, and J. Broenink, "A distributed real-time Java system based on CSP," in *Proceedings of the 3rd*

IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 400–407, IEEE Computer Society, Newport Beach, Calif, USA, March 2000.

[75] G. Salaün, L. Bordeaux, and M. Schaerf, "Describing and reasoning on web services using process algebra," *International Journal of Business Process Integration and Management*, vol. 1, no. 2, pp. 116–128, 2006.

[76] J. Hillston and M. Ribaudo, "Stochastic process algebras: a new approach to performance modeling," in *Modelling and Simulation of Advanced Computer Systems*, chapter 10, Gordon and Breach, Amsterdam, The Netherlands, 1998.

[77] M. Calder, S. Gilmore, and J. Hillston, "Formal methods for biochemical signalling pathways," in *Formal Methods State of the Art and New Directions*, P. Boca, J. P. Bowen, and J. Siddiqi, Eds., pp. 185–215, Springer, London, UK, 2010.

[78] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[79] N. A. Neogi, "Dynamic partitioning of large discrete event biological systems for hybrid simulation and analysis," in *Hybrid Systems: Computation and Control: 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25–27, 2004. Proceedings*, vol. 2993 of *Lecture Notes in Computer Science*, pp. 463–476, Springer, Berlin, Germany, 2004.

[80] G. Agha, "Concurrent object-oriented programming," *Communications of the ACM*, vol. 33, no. 9, pp. 125–141, 1990.

[81] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin, "Actor-oriented design of embedded hardware and software systems," *Journal of Circuits, Systems and Computers*, vol. 12, no. 3, pp. 231–260, 2003.

[82] N. R. Jennings, "On agent-based software engineering," *Artificial Intelligence*, vol. 117, no. 2, pp. 277–296, 2000.

[83] X. Dong, P. T. Foteinou, S. E. Calvano, S. F. Lowry, and I. P. Androulakis, "Agent-based modeling of endotoxin-induced acute inflammatory response in human blood leukocytes," *PLoS ONE*, vol. 5, no. 2, Article ID e9249, 2010.

[84] Y. Zhou and E. A. Lee, "A causality interface for deadlock analysis in dataflow," in *Proceedings of the 6th ACM & IEEE International Conference on Embedded Software (EMSOFT '06)*, pp. 44–52, ACM, Seoul, Republic of Korea, October 2006.

[85] C. Angelov, K. Sierszecki, and Y. Guo, "Formal design models for distributed embedded control systems," in *Proceedings of the 1st ESWC Workshop on Inductive Reasoning and Machine Learning on the Semantic Web*, CEUR Workshop Proceedings, pp. 43–57, Heraklion, Greece, June 2009.

[86] T. Koo, J. Liebman, C. Ma, and S. Sastry, "Hierarchical approach for design of multi-vehicle multi-modal embedded software," in *Embedded Software*, pp. 344–360, Springer, Berlin, Germany, 2001.

[87] J. Ludvig, J. Mccarthy, S. Neuendorffer, and S. R. Sachs, "Reprogrammable platforms for high-speed data acquisition," *Design Automation for Embedded Systems*, vol. 7, no. 4, pp. 341–364, 2002.

[88] L. Wang, S. Balasubramanian, and D. Norrie, "Agent-based intelligent control system design for real-time distributed manufacturing environments," in *Proceedings of the Agent-Based Manufacturing Workshop*, pp. 152–159, Minneapolis, Minn, USA, 1998.

[89] T. A. B. Snijders, G. G. van de Bunt, and C. E. G. Steglich, "Introduction to stochastic actor-based models for network dynamics," *Social Networks*, vol. 32, no. 1, pp. 44–60, 2010.

[90] T. Emonet, C. M. Macal, M. J. North, C. E. Wickersham, and P. Cluzel, "AgentCell: a digital single-cell assay for bacterial chemotaxis," *Bioinformatics*, vol. 21, no. 11, pp. 2714–2721, 2005.

[91] M. Birbaumer and F. Schweitzer, "Agent-based modeling of intracellular transport," *European Physical Journal B*, vol. 82, no. 3-4, pp. 245–255, 2011.

[92] A. Borshchev and A. Filippov, "From system dynamics and discrete event to practical agent based modeling: reasons, techniques, tools," in *Proceedings of the the 22nd International Conference of the System Dynamics Society*, vol. 22, Oxford, UK, July 2004.

[93] M. T. Klann, A. Lapin, and M. Reuss, "Agent-based simulation of reactions in the crowded and structured intracellular environment: influence of mobility and location of the reactants," *BMC Systems Biology*, vol. 5, article 71, 2011.

[94] M. Hucka, A. Finney, H. M. Sauro et al., "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.

[95] M. Hucka, L. Smith, D. Wilkinson et al., "The systems biology markup language (SBML): language specification for level 3 version 1 core," *Nature Precedings*, 2010.

[96] J. Eker, J. W. Janneck, E. A. Lee et al., "Taming heterogeneity—the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–143, 2003.