Evaluating the Design of Biological Cells Using a Computer Workbench

Tessa E. Pronk[‡§] , Simon Polstra[‡] , Andy D. Pimentel[‡] , Timo M. Breit[§]
{tepronk, spolstra, andy, breit}@science.uva.nl
[‡] Computer Systems Architecture Group, Informatics Institute, University of Amsterdam
[§] Integrative BioInformatics Unit, Swammerdam Institute for Life Sciences, University of Amsterdam

Abstract

For embedded systems as well as for biological cell systems, design is a feature that defines their identity. The assembly of different components in designs of both systems can vary widely. Given the similarities between computers and cellular systems, methods and models of computation from the domain of computer systems engineering might be applied to modeling cellular systems. Our aim is to construct a framework that focuses on understanding the design options and consequences within a cell, taking an in silico (forward-) engineering approach rather than a reverse engineering approach that is used in this domain as a default now. We take our ideas from the domain of embedded computer systems. The most important features of our approach, as taken from this domain, are a variable abstraction level of components that allows for addition of components when detailed information is lacking, and a separation of concerns between function and performance by components in the design. This allows for efficient and flexible modeling. Also, there is a strict separation between computation within- and communication between components, reducing complexity. As a proof of principle, we show that we can make a statement regarding the design of the gene expression machinery of the cell to produce a protein, using such a method.

1. Introduction

Although engineered computer systems and naturally evolved cell systems have different origins, many analogies exist between the two. In a metaphysical context, both are complex systems that have to cope with different trade-offs in energy, power, cost and flexibility. For coping with trade-offs, separate components that function within a system can have their own specific solution. As a result, both systems have a huge design space, i.e. possible components to choose from to make a system that performs a given function. These possibilities are restricted by the fact that all combinations of components should work together as best as possible.

Computer engineers are more advanced in simulating the complex behavior of the combined components of their systems than biologists. This could be caused by several factors. Firstly, both systems are investigated in opposite direction [12]: whereas computer systems are assembled from scratch to create function, the cell is already functional and is disassembled to find where function originates (Figure 1). Secondly, for computer systems, sophisticated methods exist especially to evaluate the best architecture for a system. In biological systems these sorts of methods were probably not developed in the past because systems take shape 'automatically' by natural selection. However, with the recent quest for an integral understanding of the behavior and drive of biological systems (collectively addressed by the term 'systems biology' [11, 1]), there is a demand for such methods.

One important issue in understanding the cellular design is to know why, in a particular cell, options to cope with a problem or function have been adopted whilst there are many seemingly evenly appropriate alternatives. Some core design issues have been highly conserved in cells during evolution, such as the tools to perform basic metabolism (the disassembly of nutrients to supply useful compounds in the cell), and transcription/translation (the processes that express information from genes to make proteins). Nevertheless, somewhere in the evolution, cells also have adopted different ways to perform similar functions. For instance, if we compare different cells, there are several design options to cope with particular tasks; such as different networks to deliver a signal or produce a metabolic compound [13] or different components to actively handle osmotic stress. These different options could have evolved by differential selective pressure, possibly restricted by previously adopted solutions for other functions. What is the case exactly is difficult to backtrack. If we could understand the consequences of adopted solutions, we can also ponder about their origin. To know the theoretical consequences of chosen alternatives will be helpful in designing experiments on

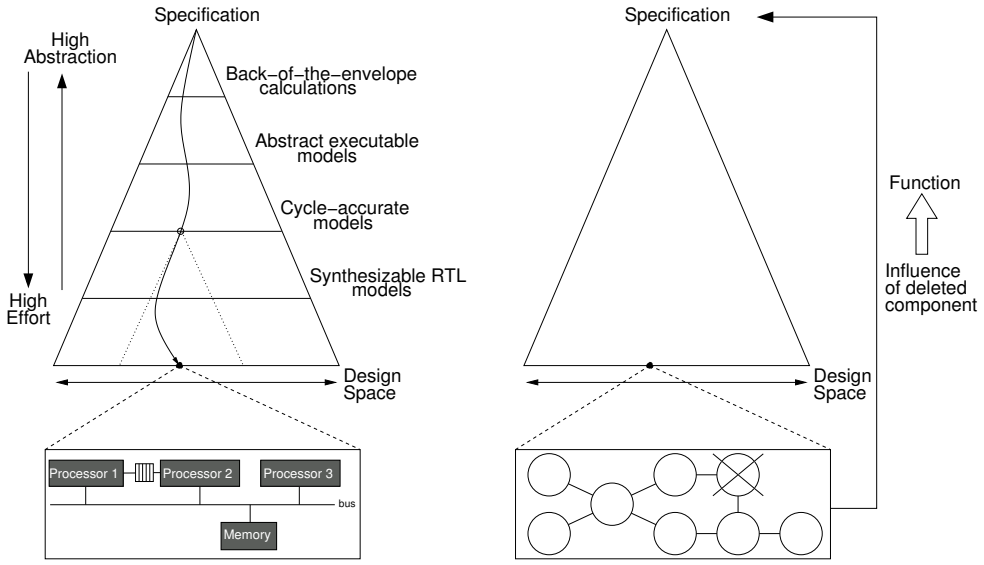

Figure 1. Difference in the direction of research between Computer Science and Biology. Left part: For computer systems, the system is first specified at abstract level. Going down in abstraction (meandering arrow), the design space of a specification (transparent dot) decreases in size (area in dotted triangle). The ultimate end product at the lowest level of abstraction is one instance of a design (black dot). Right part: For biological cell systems, the instance of the design (black dot) is the starting point and scientists try to derive the function of separate components by assessing their influence on higher level system function, for instance by inactivation or manipulation (indicated by the cross). Figure adapted from [15].

such biological systems. Also, it will be of help in the recent attempts to design cellular function [16, 7, 9]. These attempts, until now, are successful only on a small scale [17], which shows the difficulty in engineering the cell to efficiency in performing more elaborate tasks.

We think that the methods from the domain of computer systems engineering may serve as an alternative framework to formally evaluate design of biological systems [6, 3]. In this way, general principles that govern the structure and behavior of cellular systems may be discovered [6]. Our aim is to propose a framework that focuses on understanding the design options and consequences within a cell. As proof of principle, we simulate a biological process to exemplify the use and benefit of such methods for biology. We take our ideas from the domain of computer systems engineering, with a special interest in embedded computer systems.

2. The common ground between Biological and Embedded systems

A computer based system that has particularly much in common with biological cells is an embedded system [19]. Embedded systems, unlike general purpose personal computers, perform pre-defined tasks. They are ”embedded” in airplanes, security systems, telephones, medical instruments etc.

Embedded systems have very particular design requirements. They are more constrained in terms of timing, power, area, memory and other resources than general-purpose computers. As a result they are subjected to strict trade-offs [18, 5, 14]. For instance, the market position of an embedded system is hampered directly if it is too expensive, but it is hampered also if it requires too much (battery-) power. The emphasis of this trade-off will depend on the requirements of the system. Also, the interactions of the heterogeneous components in the system have to be taken into account in designing embedded systems, as well as the fact that these systems usually have multiple simultaneous sources of stimuli. The same goes for cellular systems. Additional criteria, such as real-time behavior and reactivity [8], robustness, and concurrency are of importance in all the computations that embedded systems perform. These cause additional restrictions to its architecture. Cellular systems have similar issues to comply with. Functioning in a real world, cells must exhibit real time behaviour. Because a cell’s environment is rarely constant it also needs to take into account robustness issues and resource usage. A design that is not optimal will rapidly be out selected by natural se-

lection.

The embedded systems engineer has many components at his disposal (e.g. number and type of micro- or dedicated processors, hard- or software components, input/output devices and memories), depending on whether it is more important for the system to be cheap, versatile, fast, low power, or a combination of all. This results in a heterogeneous architecture where many different types of components must interact to create a functioning system. The multiplicity of components for computer based systems and also their different wiring possibilities cause a myriad of possible designs. The heterogeneity in components is comparable between cellular and computer systems. To model the consequences of each and every possible design would take much time and effort. Somehow, the engineer must be able to quickly scan a design space and distinguish the most probable designs with a minimum of effort. As this problem of design space exploration is central to the embedded systems engineering domain, sophisticated modeling tools that facilitate this have been developed [5].

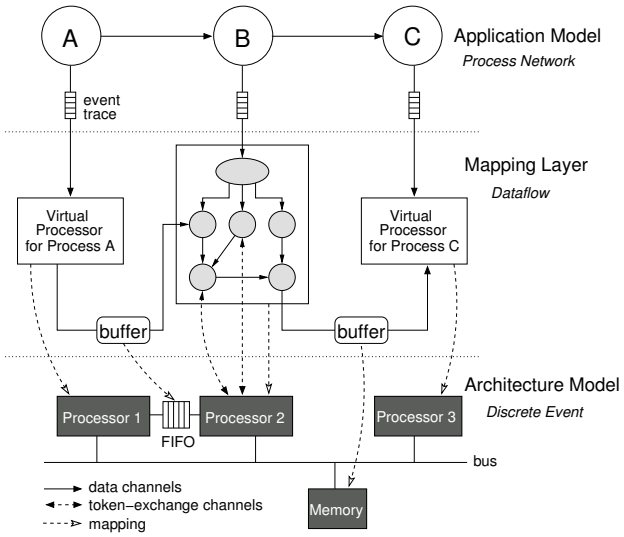

Figure 2. The Sesame workbench with its application model layer, architecture model layer and mapping layer that interfaces the application and architecture model layers.

3. An example computer workbench: The Sesame workbench

One of the first steps in designing embedded computer systems is the exploration and reduction of the design space, to quickly come to some promising design options that can be further tested. A specific example of a new and promising modeling framework is provided by the Sesame workbench [15, 14] (Figure 2). This workbench is intended for the evaluation of embedded systems architecture on a high level of abstraction. This is an especially efficient method to evaluate the best suitable architecture regarding performance (system throughput), but also cost and power consumption, i.e. the exploration of design space. We will use this modeling framework to exemplify the use of design exploration methods for biology.

Firstly, in models for embedded systems, parts of the design process are quite often separated to reduce the complexity, a feature called separation of concerns [10]. It is the ability to analyze different domains independently from each other. For instance, systems are divided into processors (computation components) and communication components (for communication between processors) which effectively separates these functionalities. Another method is to separate function (system specifications) from performance (how are these performed). With the separation of application and architecture, architecture and application can be changed independently of each other. Earlier, engineers modeled and simulated application and architecture in a monolithic manner. When architecture was changed (for instance, changing the hardware-software partitioning in the system), the whole model needed rebuilding. It is now widely recognized that for an efficient system-level exploration of design space, such a 'separation of concerns' is of paramount importance [10, 14]. The separation of application and architecture allows for rapidly assessing different application-to-architecture mappings as well as various hardware-software partitionings.

The Sesame workbench deploys separate models for application and architecture behavior (Figure 2). Application models holds the specifications for function ('what needs to be done'), while architecture models only simulate performance ('the consequences of how something is done'). The application model consists of processes that interact with each other through channels. Architecture models are composed of components that can 'execute' the processes in the application model. Components in the architecture model are processing elements (programmable cores, dedicated hardware blocks and/or reconfigurable hardware), interconnection components (such as buses, FIFO channels or crossbars), and different types of memories.

Sesame also separates the communication from the computation by using the Kahn process network model of computation. In this model the work is separated into tasks, and the communication is made explict by only allowing the passing of data between tasks via FIFO channels.

An application model in Sesame is mapped onto an architecture model by means of event traces. These event traces are emitted from the processes in the application and mapped as workloads on components in the architecture. The mapping is facilitated by an intermediate mapping

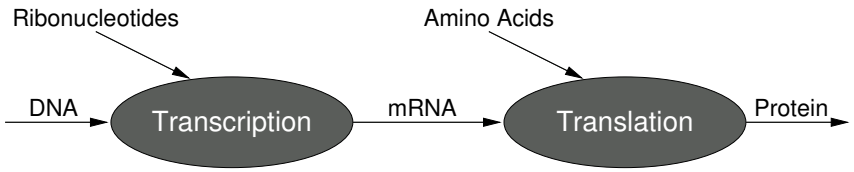

Figure 3. The process of gene expression. On the basis of a DNA template, ribonucleotides are linked in the 'Transcription' to form an mRNA. With mRNA as a template, Amino Acids are linked in 'Translation' to form a protein.

layer. The performance consequences of the event traces (the modeled workload) are simulated by the underlying architecture model. By changing the structure of the architecture model or changing the performance parameters of its components, the performance of different architectures can be compared and thus a suitable architecture can be selected.

The second feature that makes the Sesame workbench efficient is that each model layer has its own modeling method, fitting the modeling task at hand. For the application model, a (Kahn) process network model is used as Sesame currently focuses on the modeling on multimedia (streaming) applications. For the architecture, a discrete event simulation is used. The intermediate mapping layer is modeled with a dataflow model. All separate models communicate with each other. As each modeling method is chosen especially to represent the specific functionality in the model, the simulations are efficient and simulation time and effort is reduced.

Thirdly, abstraction of the system away from the details and re-use of standardized components at all these levels of abstraction is very important because it avoids the need to design the same components each time over. In Sesame, components can be modeled at varying levels of detail, from coarse grained to detailed. At the highest level, architectural components are modeled as black boxes with only a few parameters. These parameters can specify characteristics such as cost, energy usage and speed to process the event traces that are emitted by the application model. This high level of abstraction reduces both the modeling effort, and also enables simulation of the system early in the design cycle without detailed specifications. This will give coarse-grained predictions on the system behavior. Within the Sesame workbench, architectural components of interest that contribute significantly to the performance of the system can be refined to more accurately determine the performance. This allows for mixed level simulation, in which some components are refined while others remain operating at the higher level of abstraction. This refinement is facilitated by the intermediate mapping layer that brings the application model abstraction level in agreement with the (partly) refined abstraction level of the architecture model.

The above mentioned elements, namely separation of concerns, abstraction from details and model integration, are also important in systems biology [4]. Biological systems are so complex that a model based on a complete understanding will not readily be feasible. Top down approaches, where problems are modeled in an exploratory way first and later in more detail, will give information that is coarse grained but just. Separation of concerns will reduce complexity in the systems to model. Because certain problems are tackled with specific models in biology, these need to be integrated consequently for an integral understanding of a complete system.

4. Case study: Gene expression in Sesame

As proof of principle we now show how a biological process could be modeled and simulated in the Sesame workbench. We consider one of the core processes in cellular functioning: the expression of genes [2], see Figure 3. The expression of genes in cells is a highly parallel task. From one gene copy, many mRNA copies can be made in parallel and, consequently, from each separate mRNA string many copies of proteins can be made in parallel.

We model a simpler version of this problem in Sesame. In Figure 4 we suggest some nodes for the application model. Within these nodes, C++ code models the behavior. In the application node 'Transcription', DNA is read and it is used as a template for the production of mRNA. In the node 'Translation', the mRNA is used as a template for the production of proteins. In the node 'get Amino Acid' the appropriate Amino acids are actively delivered to shape the protein. All traces combined that are emitted from the application model represent the workload for the architecture. The traces need to be scheduled to the architecture. This is done in the intermediate mapping model (Figure 4). The mapping model, in principle, is capable of scheduling the many concurrent traces that have to be executed by the architecture. The architecture (Figure 4) simulates latencies for loading input, storing output and executing events. The latencies are given properties of the architectural compo-

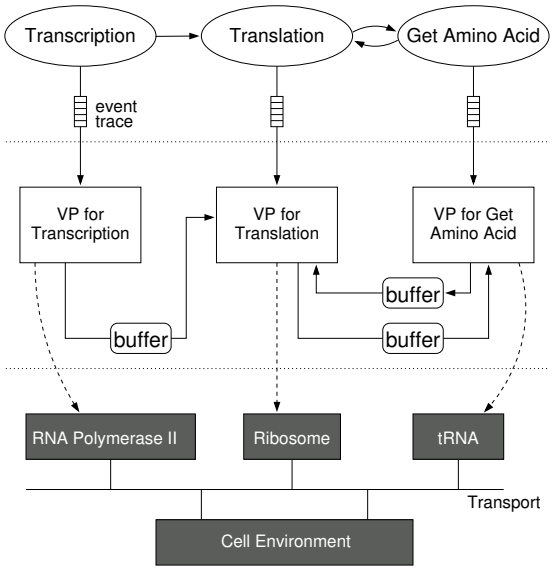

Figure 4. The expression of a gene to proteins mapped to the Sesame workbench. For a description of the biological process, see Figure 5. For the Sesame workbench, see Figure 2.

nents, but can also be made dependent on, for instance, the concentrations of products or building blocks. Each action (i.e. trace event) is performed by the architecture with a certain time cost (latency). Figure 4 shows how the application node 'Transcription' is mapped to component 'RNA polymerase II', which simulates a latency for the trace events emitted by the transcription node. The application node 'Translation' is mapped to 'Ribosome'. The application node 'Get Amino Acid' is mapped to 'tRNA' as this delivers the Amino Acids to the Ribosome in real biological cells. Because the building blocks (e.g. ribonucleotides, amino acids, see Figure 3) have to come from somewhere, we introduce a cell environment in the architecture in which building blocks can be stored and retrieved (Figure 4). This component works similar to a memory in a computer-based system. In a more elaborate version of the model, the presence of building blocks can be represented by a variable. All three architecture components are linked to the cell environment component, as they store and retrieve building blocks from it.

In the case study the application model can generate the following communication events: load/store 'DNA', load 'ribonucleotide', store 'mRNA' for the 'Transcription' node; load/store 'mRNA', load 'amino acid' and store 'protein' for the 'Translation' node; load 'codon information' (coding for amino acid) and store 'amino acid' for the 'Get Amino Acid' node and the following computational events: 'move along DNA' for the 'Transcription' node; and 'Bind Amino Acid' for the 'Translation' node. All trace events have specific latencies. The amount and kind (communication or computation) of traces emitted by the application in our proof of principle depend on the DNA sequence code (input data) that has to be put to expression and the function description, i.e. how many transcriptions/translations and the functions inherent to them are needed. These are given parameters.

We simulate the production of a protein from a piece of DNA strand. We investigate how the efficiency of architecture components is in the execution of all the actions in the application. Suppose that the protein consists of four parts of identical sequences.

We could ponder about whether it is better to have programs (a) or (b) (see Figure 5):

(a) Have a long gene consisting of four identical parts, transcribe it once, and translate it once to the protein.

(b) Have a short gene consisting of one of the identical parts, transcribe it once, translate it four times and merge the pieces later.

At the end of the simulation of program (a) and (b) a protein is made that consists of four identical subparts of three amino acids. Intuitively, to get to this result, program (a) seems simplest. However, for program (a) the process takes 714 time steps, whereas the production of the protein in program (b) was done in 423 time steps, as shown in Figure 6. We come to the conclusion that, given the architecture, the task with program (b) can be processed most efficiently according to our simple model. Mainly because in program (a) many ribonucleotides have to be loaded in the transcription in comparison with program (b).

From another perspective, we derive from this proof of principle that it is more rewarding to re-engineer the component 'Ribosome' (be it by evolution or synthetically) towards more efficiency in program (b), for it is the limiting factor: it is busy in about 61% of the total busy time of the process and would benefit most from a decrease in load, store, or execute time (Figure 6). In program (a) it would be best to re-engineer RNA polymerase II.

5. Discussion

The similarities in embedded systems and biological systems are a clue that methods from Computer Science could be used for evaluation of biological systems.

As a proof of principle we simulated a biological process in the Sesame workbench. The Sesame workbench is suitable for design space exploration. It is fast and flexible because it simulates and evaluates design alternatives on a high level. It is a quick way to evaluate the design space

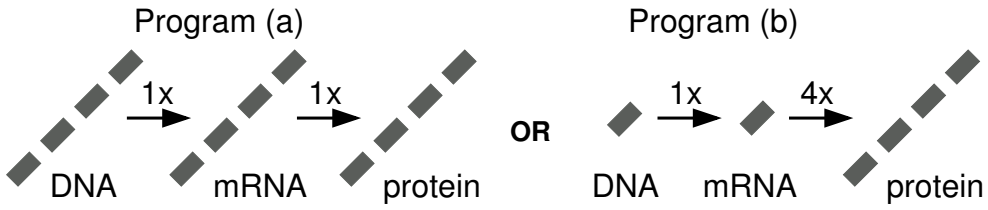

Figure 5. Two simulated applications in the Sesame workbench to produce a protein that consists of four identical stretches of Amino Acids. In program (a), the DNA is transcribed once to mRNA, which is translated once to produce the protein. In program (b), DNA is transcribed once to mRNA and this is translated four times to produce the protein.

of systems that perform complex tasks. It adds to the current modeling technology by the principle of separation of concerns and co-simulation of components with different levels of abstraction. Also, it uses the most suitable model of computation for each of the separated concerns.

A biological case study to simulate in the Sesame workbench should comply with a few constraints. Sesame can be used to evaluate designs for information processing systems, which excludes static cases like protein shape, or mechanics. Time should be involved in processing the information. It is not suitable to model systems that adjust their function according to the performance of architecture components. This is because in Sesame alternative architectures have consequences for the performance of the system, without the choice of component having any effect on the application. If it would be otherwise, different architectures could not be compared. Lastly, to make the exploration useful, there should be alternative architectural options in the system under study to perform a certain task. This will provide the opportunity to compare and find the best possible solution in terms of architecture for a given (user specified) task within a cell; its one of Sesames merits. This however, limits the application of Sesame to biological systems because, in contrast to embedded computer systems, in cells there is mostly a tight mapping between application tasks and the components that have to perform the tasks. Thus, many tasks in a cell are performed by 'dedicated' components. For instance, in cells, only RNA polymerase II is suitable to perform the translation of a gene to mRNA. Design options, in this case, lie within the amount of RNA polymerase II or its location (i.e. cell compartment). Possibly, in the future, biological components that perform sub optimally can be re-engineered in vivo and this would give Sesame the added value of pointing out bottle necks in the performance as candidates for re-engineering. For now, generally in biological cell systems, the design options that can be found will lie in the amount, location and assemblages of components.

On the other hand, we do have tasks in biological cells for which we know alternative components exist to perform them. One much studied example is the different types of tRNA that can be used in translation. Every of the twenty possible amino acids that are present in living material is coded by a three nucleotide long codon. As there are four types of nucleotides (A,T,C,G) there are 64 different codons. As a consequence, every amino acid is coded for by one or several codons. These codons have to be recognized by the anti-codons of tRNA to supply the correct amino acid to the growing chain of peptides. One possibility is that every codon is recognized by one exclusive tRNA anti-codon, resulting in 64 different tRNA types. Another possibility is that each tRNA anti-codon can recognize several codons. To avoid mistakes in the amino acid sequence, logically the codons recognized by one tRNA type should code for the same amino acid. Theoretically this would result in a lower bound of 20 different tRNA types. In different organisms, there is variation in the amount and type of tRNA used to perform translation. Sesame should also be suitable to model different metabolic routes by seeing them as standardized (architectural) components with different traits. In cell systems sometimes there are up to 77 routes to come to a specific compound.

Although there is a remarkable fit between the simulation in biological and embedded system tasks, the Sesame workbench has to be adjusted further for simulation of biological tasks. For instance, in its present form Sesame abstracts away from massive parallelism by capturing this within a single architectural component by its parameters. When different tasks are executed within the application, different components can process the tasks concurrently or a component in Sesame can process them in parallel singlehandedly when refinement is done on that component. For simulating a biological realistic case, there must be many components that work simultaneously on the same, identical task because this option has evolved quite often in biological cells [2]. For instance, to translate a gene to mRNA's there are thousands of RNA polymerases that perform this task in parallel. Therefore we are forced to create a new

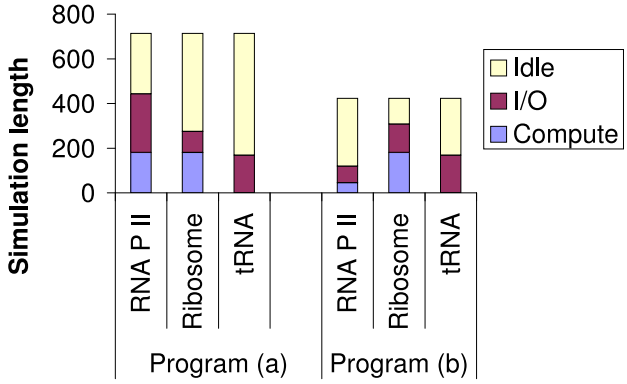

Figure 6. Performance of program (a) and program (b), see Figure 5, as simulated in Sesame (see Figure 4). In Sesame, statistics are user defined, thus other statistics can be calculated and visualized if needed.

component that allows for massive parallelism to use for simulation of biological applications. This new component will act as a global scheduler in the synchronisation layer that regulates the occupancy of the separate concurrent components. This new component can then also be used to model the parallelism in massively parallel computer systems. Also, the mapping layer is at its present status not suitable to handle communication scheduling between many components performing a single task in the way this is done in cells: these are not the typical one-to-one producer-consumer relations as found in embedded systems. For simulation of biological systems this has to be adjusted so the many tasks executed by the many parallel components can be scheduled in parallel in a more straightforward manner. This could mean that the SDF model of computation currently used in the synchronisation layer has to be replaced with a model that allows for a more flexible way of scheduling and handling the synchronisation. Another drawback of Sesame for the simulation of biological systems is that much time is spent on the application model. However, in the future, a library of components of functions and performing architectural components will ease the modeling process.

We showed that the methods for design space exploration from Computer Science can, in principle, be used for the simulation of specific biological processes. Nevertheless, both systems have their differences and there is a limitation to the kind of biological problems that can be modeled. These problems will have to be addressed by adjusting the methods for design space exploration to allow for more flexibly simulating processes that occur in biological systems, such as massive parallelism.

References

[1] L. Alberghina, F. Chiaradonna, and M. Vanoni. Systems Biology and the molecular circuits of cancer. *ChemBioChem*, 5(10):1322–1333, 2004.

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, New York, fourth edition edition, 2002.

[3] A.-L. Barabasi and Z. N. Oltvai. Network Biology: Understanding the cell's functional organization. *Nature Reviews Genetics*, 5:101–113, Feb 2004.

[4] A. Finkelstein, J. Hetherington, L. Li, O. Margoninski, P. Saffrey, R. Seymour, and A. Warner. Computational challenges of Systems Biology. *IEEE Computer*, 37(5):26–33, May 2004.

[5] M. Gries. Methods for evaluating and covering the design space during early design development. *Integr. VLSI J.*, 38(2):131–183, 2004.

[6] L. H. Hartwell, J. J. Hopfield, S. Leibler, and A. W. Murray. From molecular to modular Cell Biology. *Nature*, 402:C47–C52, Dec 1999. 10.1038/35011540.

[7] J. Hasty, D. McMillen, and J. J. Collins. Engineered gene circuits. *Nature*, 420:224–230, Nov 2002. 10.1038/nature01257.

[8] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng. Overview of the ptolemy project. Technical Memorandum UCB/ERL M03/25, University of California, Berkeley, Jul 2003.

[9] M. Kaern, W. J. Blake, and J. Collins. The engineering of gene regulatory networks. *Annual Review of Biomedical Engineering*, 5:179–206, 2003.

[10] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System level design: Orthogonalization of concerns and platform-based design. *IEEE transactions on Computer-Aided Design*, 19(12), Dec 2000.

[11] H. Kitano. Computational Systems Biology. *Nature*, 420(6912):206–210, Nov 2002.

[12] Y. Lazebnik. Can a biologist fix a radio?–or, what I learned while studying apoptosis. *Cancer Cell*, 2(3):179–182, Sep 2002.

[13] J. A. Papin and B. O. Palsson. Topological analysis of mass-balanced signaling networks: a framework to obtain network properties including crosstalk. *Journal of Theoretical Biology*, 227(2):283–297, 2004.

[14] A. D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers*, 55(2):99–112, 2006.

[15] A. D. Pimentel, P. Lieverse, P. van der Wolf, L. O. Hertzberger, and E. F. Deprettere. Exploring embedded-systems architectures with Artemis. *IEEE Computer*, 34(11):57–63, Nov 2001.

[16] M. A. Savageau. Design Principles for Elementary Gene Circuits: Elements, Methods, and Examples. *Chaos*, 11:142–159, Mar 2001.

[17] M. L. Simpson. Rewiring the cell: Synthetic Biology moves towards higher functional complexity. *Trends Biotechnol*, 22(11):555–557, Nov 2004.

[18] F. Vahid and T. Givargis. *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[19] W. Wolf. *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.