Towards Design Space Exploration for Biological Systems

Simon Polstra[†‡] , Tessa E. Pronk[†‡§] , Andy D.Pimentel[‡] , Timo M. Breit[§]
{spolstra, tepronk, 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
[†] Both authors contributed equally to this work

Abstract— **For both embedded systems and 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 could be applied to model 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 the reverse-engineering approach now used by default in this domain. 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 model components that allows for inclusion of components of which 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, thus reducing complexity. As a proof-of-principle, we show that we can make a statement regarding the design of the gene expression machinery of a cell to produce a protein, using such a method.**

I. 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 must cope with different trade-offs in energy, power, cost and flexibility. For coping with trade-offs, separate components that function within a system may 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 lead could be caused by several factors. Firstly, the two systems are investigated in opposite directions [1]: whereas computer systems are assembled from scratch to create function, the cell is already functional and is disassembled to discover where function originates (Figure 1). Secondly, sophisticated methods already exist for computer systems, specifically for evaluating 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' [2], [3]), there is a demand for such methods. There certainly is a trend in applying methods originally developed for other domains to the benefit of biology. Examples are frame-synchronization techniques [4], process calculi [5], model checking [6], and pathway logic [7].

In biology, the engineering approach as such is emerging as a tool for research. One application is the actual construction of simple regulatory networks to help find design principles (i.e. feedback loops, switches, oscillators) of cellular networks [2], [9], [10]. One important issue in understanding cellular design principles is to learn why, in a particular cell, options for coping with a problem or function have been adopted whilst there are many seemingly equally appropriate alternatives. During evolution, some core design issues have been highly conserved in cells, e.g. the tools for performing 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 have also adopted different ways of performing similar functions. For instance, if we compare different cells, there are several design options for coping with particular tasks, e.g. different networks to deliver a signal or produce a metabolic compound [11] or different components to handle osmotic stress actively. These different options may have evolved by differential selective pressure, possibly restricted by previously adopted solutions for other functions. It is difficult to backtrack exactly what the case is. If we were able to understand the consequences of adopted solutions, we could also ponder about their origin. It would be helpful to know the theoretical consequences of chosen alternatives in designing experiments on such biological systems. This would also be of help in the recent attempts to design cellular function [12]–[14]. So far, these attempts have been successful only on a small scale (i.e. simple networks) [15], which shows the

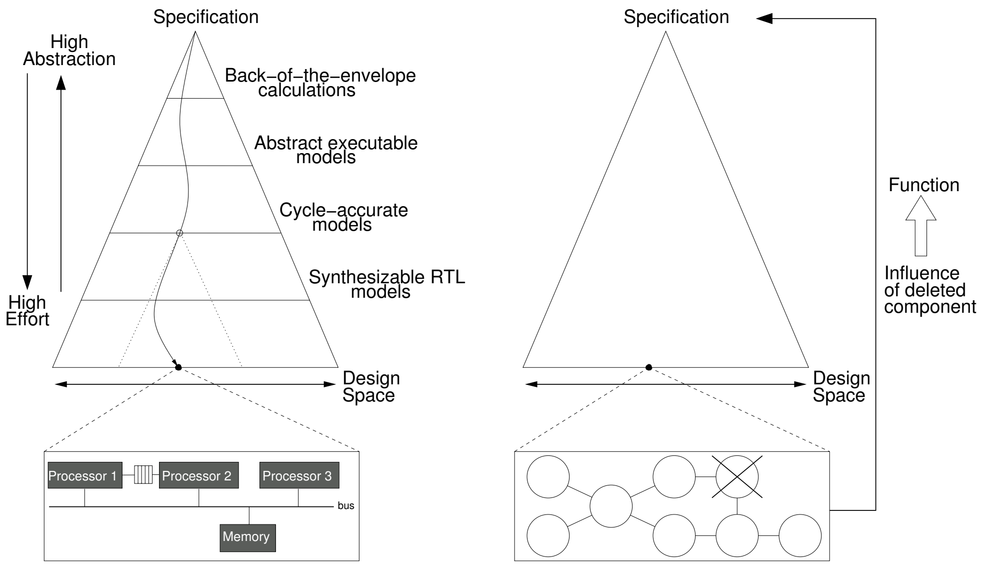

Figure 1. Difference in the direction of research between Computer Science and Biology. Left part: For computer systems, the system is first specified at abstract level. Going down in abstraction (meandering arrow), the design space of a specification (transparent dot) decreases in size (area in dotted 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 [8].

difficulty in engineering the cell to be more efficient when performing more elaborate tasks. This is where structured frameworks such as used in methods for design space exploration would come in useful. Whereas biological problems are traditionally tackled with a reverse engineering approach, much can be said to, as an alternative, use a forward engineering approach such as used in design space exploration for computer systems engineering (Figure 1). These methods are capable of evaluating design options quickly and formally on a larger (systems-) scale. We anticipate that methods from the domain of computer systems engineering may as such serve as an alternative framework to formally evaluate design of biological systems [9], [16]. In this way, general principles that govern the structure and behavior of cellular systems may be discovered [9]. The application of methods for design space exploration to biological systems has not been attempted before. 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.

II. THE COMMON GROUND BETWEEN BIOLOGICAL AND EMBEDDED SYSTEMS

An embedded system [17] is a computer-based system that has in particular much in common with biological cells. 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 subject to strict trade-offs [18]–[20]. For instance, the market position of an embedded system is hampered directly if it is too expensive, but also if it requires too much (battery-) power. The emphasis of this trade-off will depend on the requirements of the system. Also, in designing embedded systems the interactions of the heterogeneous components in the system have to be taken into account, as well as the fact that these systems usually have multiple simultaneous sources of stimuli. Additional criteria, such as real-time behavior and reactivity [21], robustness, and concurrency are of importance in all computations that embedded systems perform. These cause additional restrictions to the architecture. Cellular systems have similar issues to comply with. Functioning in a real world, cells must exhibit real-time behavior. Because a cell’s environment is rarely constant it also needs to take robustness issues and resource usage into account. A design that is not optimal will rapidly disappear by natural selection.

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. 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 create a myriad of possible designs. The heterogeneity in components is comparable between cellular and computer systems [22]. To model the consequences of each and every possible design would take much time and effort. Somehow the engineer must be able to scan a design space quickly 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 have been developed to facilitate it [19].

III. 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, so as to come quickly 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 [8], [20] (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.

In models for the design of computer systems, parts of the design process are quite often separated to reduce the complexity, a feature called separation of concerns [23]. This enables the analysis of 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) in an application from performance (how are these performed) in an architecture. With the separation of application and architecture, these 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 [20], [23]. The separation of application and architecture allows for rapid assessment of 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 hold 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. In the architecture model, components include 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 communication from computation

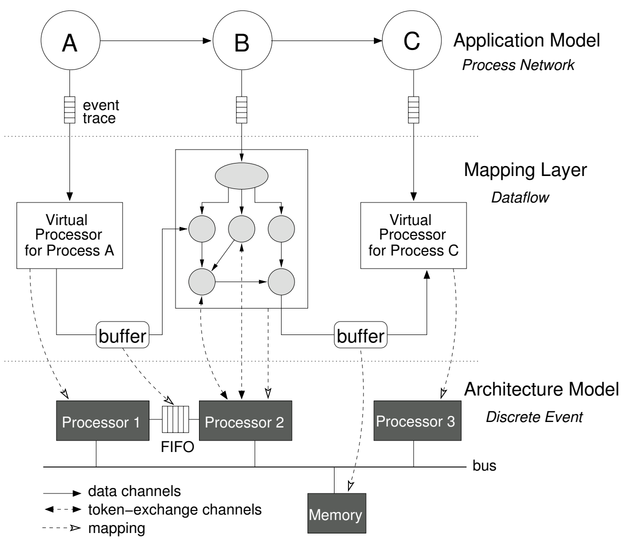

Figure 2. The Sesame workbench with its application model layer, architecture model layer and mapping layer that interfaces the application and architecture model layers.

by using the Kahn process network model of computation. In this model the work is separated into tasks, and communication is made explicit by allowing only 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 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. A (Kahn) process network model is used for the application model, as Sesame currently focuses on the modeling of multimedia (streaming) applications. For the architecture, a discrete event simulation is used. The intermediate mapping layer is modelled 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 redesign the same components each time. 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

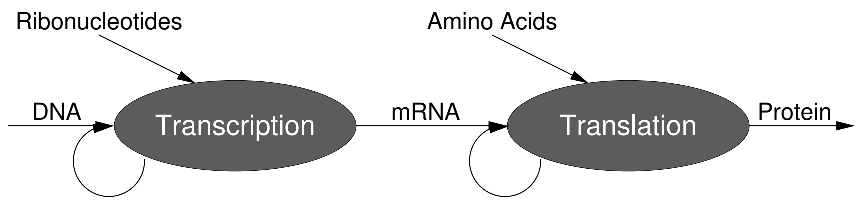

Figure 3. The process of gene expression. On the basis of a DNA template, ribonucleotides are linked in the process 'Transcription' to form an mRNA. With mRNA as a template, Amino Acids are linked in the process 'Translation' to form a protein. DNA and mRNA templates are reused.

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 both reduces the modeling effort and also enables simulation of the system early in the design cycle without detailed specifications. This will give coarse-grained predictions of the system behavior. Within the Sesame workbench, architectural components of interest that contribute significantly to the performance of the system can be refined to determine the performance more accurately. 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 [24]. 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 be modeled.

Lastly, integrating specific models in biology can be inspired by the way this is done in workbenches such as Sesame.

IV. Case study: Gene expression in Sesame

As proof-of-principle we now show how a simple biological process can be modeled and simulated in the Sesame workbench. We consider one of the core processes in cellular functioning: the expression of genes [25], see Figure 3. The expression of genes in cells is a highly (semi-) parallel task. From one gene copy, many mRNA copies can be made, although each copy is initiated with a short time-lag. Consequently, from each separate mRNA string many copies of proteins can be made (each copy also with a short time-lag). In Figure 3 we suggest the nodes for the application model. Within these nodes, C-like code models the behavior. In the application node 'Transcription', the DNA of a single gene is read and 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. The architecture simulates latencies for loading input, storing output and executing events. The combination of all traces that are emitted from the application model represents the workload for the architecture. The latencies are given properties of the architectural components, 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). The application node 'Transcription' is mapped to component 'RNA polymerase II (Pol II)', which simulates a latency for the trace events emitted by the transcription node. The application node 'Translation' is mapped to component 'Ribosome'. Because building blocks (see Figure 3) must remain somewhere, we introduce a cell environment in the architecture in which building blocks can be stored and retrieved. This component works similar to a memory in a computer-based system (Figure 2). All architecture components are linked to the cell environment component, as they store and retrieve building blocks from it. The experiment carried out with these specifications is described further on in this paper, but first we will discuss the modifications to Sesame that were needed in order to run it.

V. Modifications to Sesame

In our earlier conference paper [26] we modeled gene expression with an extremely simple architecture. For the current case study we wanted to model the executing expression machinery (i.e. Pol II, Ribosomes) more realistically as a network of parallel processors. The execution of tasks in biological systems has often evolved as being handled by multiple identical, dedicated components. For our biological case study we therefore needed to parallelize the sequential application trace for parallel execution at the architecture layer. Normally, on the high level at which embedded systems are modeled in Sesame, it suffices to abstract away from massive parallelism by capturing it in a single component with parameters (i.e. when a component represents ten processors instead of one, execution is ten times faster) or to use a more detailed level of modeling where a parallel component is modeled as a network of components. In the latter case, the traces are scheduled to the architecture by a more detailed virtual processor at the mapping layer. This virtual processor recognizes the fixed event pattern which can be executed in parallel. Such a recognized sequence of events, in Sesame, triggers the activation of a chain of mapping layer components that essentially replays the sequence of events thereby generating architecture workload for the parallel

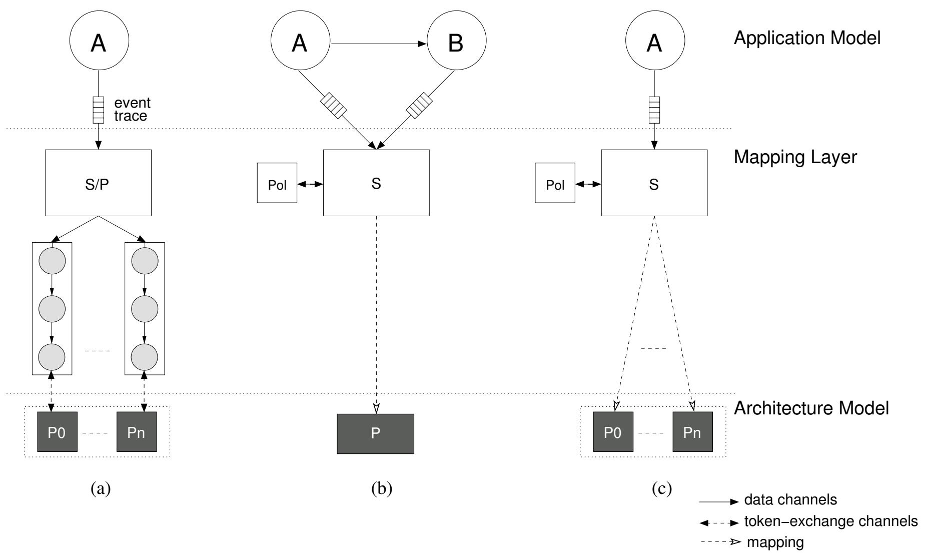

Figure 4. Different methods to schedule application traces in the mapping layer. (a) The scheduling of a sequential application trace to parallel components (P0-Pn) by pattern recognition (used in Sesame). (b) The scheduling of traces of multiple application tasks to a single processor (P) (used in Sesame). (c) Parallel scheduling of a single application task to multiple processors (P0-Pn). This scheduling method is based on (b) and implements (a). A and B are application processes. 'S/P' in (a) is the scheduling component, called 'S' in (b) and (c). 'Pol' can be any scheduling policy.

component. A chain of mapping layer components is represented in Figure 4a by a box containing gray circles.

However, the approach to model parallelism in Sesame as described above proved to be too inflexible to handle the scheduling and synchronization that was required for our biological case study. This was partly because of the massive parallelism required and partly because of the different requirements of the application model. We will elaborate on these problems below.

Firstly, the patterns to be executed in parallel in our biological case are not static, as they depend on the input given in the application, i.e. the gene length. Therefore we need a way to recognize these flexible patterns.

Secondly, the application requires the flexibility to read the data of a specific type without having to be concerned from which process that data came. For instance, mRNA can come newly constructed from the transcription process but is also re-used multiple times within the translation process. This is similar to a non-deterministic select operation in Kahn process networks, which is not implemented natively in Sesame.

Thirdly, we needed to retain the ability to let the data contents influence the performance (e.g. Ala is translated faster than Met but both are of the same type: amino acids). In Sesame the performance model only receives the size of the tokens that are transferred, not the actual data.

Finally, the synchronization between two communicating parallel virtual processors was difficult because the data token produced by one parallel virtual processor needed to be sent to the correct, chain of mapping layer components of the other parallel virtual processor. The parallel scheduling of the sequential application trace into the parallel chains (done by the scheduler/pattern recognizer 'S/P' in Figure 4a) of a virtual processor dictates the distribution of the synchronization tokens. Because there is no way to discern this schedule from outside of the virtual processor it was impossible to schedule/distribute these synchronization tokens in a way that results in maximal parallelism.

To overcome these problems, we tailored the Sesame workbench to fit our requirements. We needed to do some extensive modifications to the original Sesame workbench to model general biological cases to our satisfaction. However, we did not need to sacrifice the main merits of the approach, which are the separation of concerns, the possibility to model at different levels of abstraction and using different models of computation.

Firstly, we made a new component (Figure 4c) for scheduling by modifying a custom scheduling component developed for scheduling multiple tasks on a single processor (Figure 4b). The original scheduler was used to schedule multiple tasks onto a single processor by merging their traces into a sequential trace that can be executed on that processor. The policy for scheduling could be changed by simply plugging in a different policy. In contrast, the new scheduling component (Figure 4c) takes a sequential application trace and schedules it to multiple parallel processors on a pattern basis. In the application task we explicitly annotate the code with an 'end of pattern' notation. This normally is the last annotation of the main loop of the expression task but can in principle occur anywhere in the code. Because the recognised trace patterns are not used to trigger a fixed chain of mapping layer components that generate architecture workload as was the case in the original Sesame, we have the ability to recognize patterns of variable length and contents dynamically.

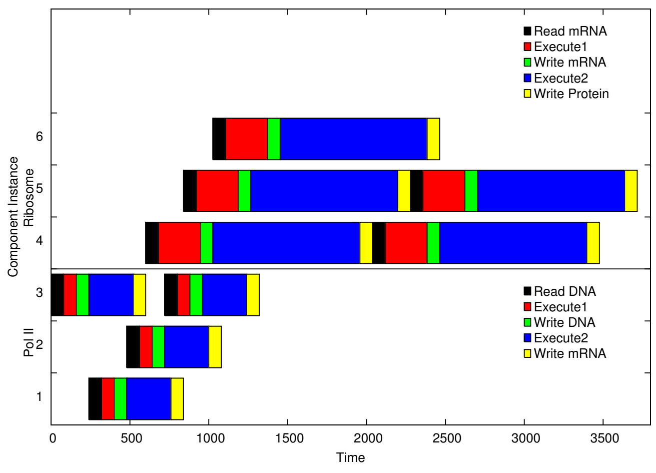

Figure 5. Example of the performance of components in the architecture functioning in parallel to perform the application tasks to construct four mRNA's and five proteins. On the x-axis are the individual components: Pol II (bottom three) and Ribosome (top three). On the y-axis are the tasks of the individual components. Components can be respectively attaching to DNA/mRNA ('Read'); occupying the start of DNA/mRNA ('Execute 1'); transcribing/ translating ('Execute 2'); storing the product mRNA/protein ('Write') or be idle. Each individual Pol II or Ribosome executes the tasks in this order and with each 'write' an mRNA or Protein is created respectively. Time is in units, a hundred units correspond to one second.

Secondly, to be able to use this new scheduling component we moved the synchronization of data between tasks from the synchronization layer to the architecture layer of Sesame. By removing the synchronization via token channels at the mapping layer we solved the problem of the distribution of synchronization tokens, discussed as the last problem in the previous paragraph. In this solution the memory component now needs to keep track of the data in order to provide synchronization. The memory component is split in multiple sub-components, each synchronizing for a single data type. Moving the synchronization from the dataflow model of computation to the more flexible discrete event simulator of the architecture layer allowed us to model massive parallelism easily.

The last modification to the Sesame workbench was to change the model of computation of the application from the dedicated Kahn process network simulator 'PNRunner' into the same discrete event simulation model as the architecture is modeled in. However, we kept the semantics of Kahn process networks (blocking reads, non-blocking writes). This change of the model of computation allows use of the flexible 'select' semantics that we need for our biological case study.

VI. Gene expression experiment

In the case study, the application model generates the following communication events (Figure 5): 'Read DNA', 'Write DNA', 'Write mRNA', 'Read mRNA', and 'Write protein', and the following computational events: 'Execute1' (occupy a stretch of respectively DNA or mRNA to initiate transcription or translation) and 'Execute2' (respectively transcribe the rest of the DNA into mRNA or translate the rest of the mRNA into protein). All trace events have specific latencies. The amount of trace events (communication or computation) emitted by the application in our proof-of-principle depend on the DNA sequence code (input data) that must be expressed and the function description, i.e. how many transcriptions/translations and the functions inherent to them are needed. Figure 5 shows how an architecture with three Pol II and three Ribosomes divides the workload to produce four mRNA's and five proteins. As the processivity of an individual Pol II is about seventy percent higher than that of a Ribosome [25], the Pol II are ready for a next transcription much faster than the Ribosomes are for a next translation. While a Pol II occupies the initial part of the DNA ('Execute 1' in lower part of Figure 5), no other Pol II can initiate transcription. This forces Pol II to be idle. Ribosomes are less affected by the initial occupancy of mRNA ('Execute 1' in upper part Figure 5), since they have multiple mRNA's to share amongst themselves (Figure 5). The Ribosomes can make use of the mRNA's stored from the transcription process as well as those restored after initial occupancy in the translation process.

For a larger experiment we simulate the production of twelve mRNA's from a piece of DNA strand which, in turn, serve as templates for the production of fifty proteins. For this application, we investigate the efficiency of architectural components (Pol II and Ribosomes) in generating the proteins. Figure 6a shows the performance of different combinations of Pol II and Ribosomes in the execution of the application tasks. The combination of three Pol II and thirty-five Ribosomes is the quickest to make the proteins with the help of twelve mRNA's. The performance statistics of this assembly are depicted in Figure 6b. This Figure shows that for the whole application, Pol II and Ribosomes have similar statistics. Pol II have idle time because they are restricted by the fact that they have to share a single DNA, but the main cause is that they have such a high processivity that the task that was mapped to them, transcription, is finished early on in the application. Ribosomes have much slower processivity. In our case, the architectures that perform well compensate for the slow processivity of Ribosomes by having more copies. In this way the translation takes about half of the total busy time of the whole application (Figure 6a). According to our model, a cell would need far fewer Pol II than Ribosomes. In real biological systems, it is probably useful to generate many mRNA's in a short amount of time with few Pol II because the Pol II are quickly available to generate mRNA's for other proteins, if necessary. Of course, having many copies of any component generates a cost in energy and building blocks to make them. Many Ribosomes are needed for an equal performance (in terms of time) of the translation and transcription processes. From an engineering perspective, it would be rewarding to re-engineer the component 'Ribosome' (be it by evolution or synthetically) towards more efficiency because this would speed up the whole process of gene expression while requiring less Ribosomes. This would be especially useful when resources for the generation of Ribosomes are low. The logical step in Sesame would be to refine the component Ribosome to have a clearer knowledge of where its low processivity stems from.

Another determinant of the process speed is the number of mRNA's generated by Pol II. If less mRNA's are made, for instance five, the time for the expression of our experiment of fifty proteins would increase drastically -about tenfold- from 4888 to 44878 time units. In this case the optimal combination of Pol II and Ribosomes is three and seventeen. Here, individual Ribosomes are forced to make more proteins. More Ribosomes would not speed up the process because the scarce mRNA's must be shared. Having low numbers of mRNA would logically be a good way for the cell to control the speed of gene expression while keeping the potential of Ribosomes for the translation of other proteins. The optimal number of Pol II largely depends on the length of the DNA that needs transcription. A longer stretch will be able to hold more Pol II at a time.

VII. Discussion

The similarities between embedded systems and biological systems are a clue that methods from computer systems engineering can be used for evaluation of biological systems. As a proof-of-principle we simulated a biological process, the expression of genes, in the Sesame workbench. The Sesame workbench is developed for design space exploration. It is fast and flexible because it simulates and evaluates design alternatives on a high level of abstraction. It is a quick way to evaluate the design space of systems that perform complex tasks. It adds to the current modeling technology by using the principle of separation of concerns and allowing co-simulation of components with different levels of abstraction.

In general, 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 such as protein shape or mechanics. Time should be involved in processing the information. It is not suitable for modeling systems that adjust their function according to the performance of architecture components 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 were otherwise, different architectures could not be compared under an equal application workload. Lastly, to make the exploration useful, there should be alternative architectural options for performing a certain task in the system under study. 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; it's one of Sesame's merits. This however, limits the application of Sesame to biological systems because in cells, in contrast to embedded computer systems, there is generally a tight mapping between application tasks and the components that perform the tasks. This means that many tasks in a cell are performed by 'dedicated' components. For instance, in cells, only RNA polymerase II (Pol II) is suitable for performing the translation of a gene to mRNA. In this case, design options lie within the amount of Pol II or its location (i.e. cell compartment). Possibly, in the future, biological components that perform suboptimally can be re-engineered in vivo, which would give Sesame the added value of pointing out bottlenecks in performance as candidates for re-engineering. For now, in biological cell systems, the design options found will generally concern 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 concerns the different types of tRNA that can be used in translation. Every of the twenty possible amino acids present in living material is coded by a three nucleotide long codon. As there are four types of nucleotides (A, T, C, and 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, the codons recognized by one tRNA type should logically code for the same amino acid [25]. 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 for modeling different metabolic routes by seeing them as standardized (architectural) components with different traits. In cell systems there are often many routes to come to a specific compound [11]. Another application could be to evaluate the different versions of cellular components (i.e. RNA Polymerases) through evolutionary time to increase the understanding of why these components were adapted.

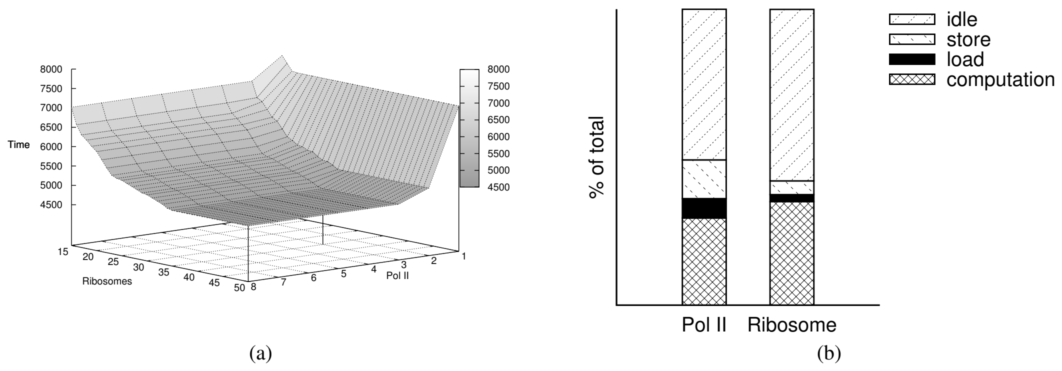

Figure 6. Architecture performance for a range of component assemblies for gene expression. (a) Depicted on the z-axis is the performance in units of time of different combinations of Pol II (y-axis) and Ribosome (x-axis) components to generate fifty identical proteins, using twelve mRNA's. The best performing architecture (three Pol II and thirty-five Ribosomes) is marked with a line going from the plot to the x-y plane. One hundred time units corresponds to one second. (b) Performance statistics of the best performing architecture instance in (a). For the biological equivalents of store and load see Figure 5. The two 'Execute' tasks in Figure 5 constitute the computation part in the performance statistics.

Although there is a remarkable fit between simulation in biological and embedded system tasks, the Sesame workbench had to be adjusted for a meaningful simulation of biological tasks. In its original form, different tasks were executed within the application, different components could process the tasks concurrently or a component in Sesame could process them in parallel single-handedly when refinement was carried out on that component. For simulating a biologically realistic case, we had to provide the opportunity for many components to work simultaneously on the same, identical task because this option has evolved quite often in biological cells [25]. For instance, to translate a gene to mRNA's, many copies of Pol II are available for performing this task in parallel. Likewise, many Ribosomes work concurrently per mRNA in translating it to protein. For the simulation of biological applications we were therefore forced to create a new component that allows for massive parallelism. This new component acts as a scheduler that regulates the occupancy of the separate concurrent components. This new component can of course also be used to model the parallelism in massively parallel computer systems. The mapping layer in its present status was unable to handle the communication between many components performing a single task in the way this is done in cells where there are not the typical one-to-one producer-consumer relations as found in embedded systems, so we had to change it to fit our needs. Lastly, similar to other types of biological simulation models, much time is spent on the application model. However, in the future, we will extend the available library of modules. Such a library of application nodes and architectural components will ease the modeling process as these can be reused when performing design space exploration. This is much like using IP (intellectual property) blocks such as used in computer science. For the library we will investigate and implement other relevant biological processes in the modified Sesame workbench, such as energy conversion processes and processes for transporting building blocks or waste material, as they occur in biological cells. A next step is to link several of these modules of different biological processes from the library together to create a more integral model of a biological cell, which can be done at multiple levels of abstraction as this is supported by the Sesame workbench. In this way we work towards the integral model of cell systems that is needed in Systems Biology.

We showed that the methods for design space exploration from computer systems engineering 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 can be addressed by adjusting the methods for design space exploration. It would certainly be worthwhile to asses useful modeling practices from other methods for design space exploration in the embedded systems domain.

Acknowledgment

We would like to thank Mark Thompson for his ideas on technical developments in the Sesame workbench and Jenny Batson for critically reading the manuscript.

References

[1] Y. Lazebnik, "Can a biologist fix a radio?–or, what I learned while studying apoptosis," *Cancer Cell*, vol. 2, no. 3, pp. 179–182, Sep 2002.

[2] H. Kitano, "Computational Systems Biology." *Nature*, vol. 420, no. 6912, pp. 206–210, Nov 2002. [Online]. Available: http://dx.doi.org/10.1038/nature01254

[3] L. Alberghina, F. Chiaradonna, and M. Vanoni, "Systems Biology and the molecular circuits of cancer," *ChemBioChem*, vol. 5, no. 10, pp. 1322–1333, 2004.

[4] J. Weindl and J. Hagenauer, "Applying Techniques from Frame Synchronization for Biological Sequence Analysis." *To appear in: Proc. IEEE International Conference on Communications (ICC 2007)*, 2007.

[5] C. Kuttler, "Gene regulation in the pi calculus: Simulating cooperativity at th e lambda switch," *Lecture notes in computer science*, vol. 4230, pp. 24–55, 2006.

[6] T. Pronk, E. de Vink, D. Bosnacki, and T. Breit, "Stochastic modeling of codon bias with prism," *Proc. MTCoord07*, pp. 1–15, 2007.

[7] C. Talcott, *Symbolic modeling of signal transduction in Pathway Logic.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.
[8] A. D. Pimentel, P. Lieverse, P. van der Wolf, L. O. Hertzberger, and E. F. Deprettere, "Exploring embedded-systems architectures with Artemis," *IEEE Computer*, vol. 34, no. 11, pp. 57–63, Nov 2001.
[9] L. H. Hartwell, J. J. Hopfield, S. Leibler, and A. W. Murray, "From molecular to modular Cell Biology," *Nature*, vol. 402, pp. C47–C52, Dec 1999, 10.1038/35011540. [Online]. Available: http://dx.doi.org/10.1038/35011540
[10] J. Stelling, U. Sauer, Z. Szallasi, F. Doyle, and J. D. e, "Robustness of cellular functions," *Cell*, vol. 118, pp. 675–685, 2004.
[11] 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*, vol. 227, no. 2, pp. 283–297, 2004.
[12] M. A. Savageau, "Design Principles for Elementary Gene Circuits: Elements, Methods, and Examples," *Chaos*, vol. 11, pp. 142–159, Mar 2001.
[13] J. Hasty, D. McMillen, and J. J. Collins, "Engineered gene circuits," *Nature*, vol. 420, pp. 224–230, Nov 2002, 10.1038/nature01257. [Online]. Available: http://dx.doi.org/10.1038/nature01257
[14] M. Kaern, W. J. Blake, and J. Collins, "The engineering of gene regulatory networks," *Annual Review of Biomedical Engineering*, vol. 5, pp. 179–206, 2003.
[15] M. L. Simpson, "Rewiring the cell: Synthetic Biology moves towards higher functional complexity." *Trends Biotechnol*, vol. 22, no. 11, pp. 555–557, Nov 2004.
[16] A.-L. Barabasi and Z. N. Oltvai, "Network Biology: Understanding the cell's functional organization," *Nature Reviews Genetics*, vol. 5, pp. 101–113, Feb 2004. [Online]. Available: http://dx.doi.org/10.1038/nrg1272
[17] W. Wolf, *Computers as Components: Principles of Embedded Computing System Design.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.
[18] F. Vahid and T. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction.* New York, NY, USA: John Wiley & Sons, Inc., 2001.
[19] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integr. VLSI J.*, vol. 38, no. 2, pp. 131–183, 2004.
[20] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Transactions on Computers*, vol. 55, no. 2, pp. 99–112, 2006.
[21] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng, "Overview of the ptolemy project," University of California, Berkeley, Technical Memorandum UCB/ERL M03/25, Jul 2003.
[22] T. Pronk, A. Pimentel, M. Roos, and T. Breit, "Taking the example of computer systems engineering for the analysis of biological cell systems." *BioSystems doi:10.1016/j.biosystems.2007.02.002*, 2007.
[23] 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*, vol. 19, no. 12, Dec 2000. [Online]. Available: citeseer.ist.psu.edu/keutzer00system.html
[24] A. Finkelstein, J. Hetherington, L. Li, O. Margoninski, P. Saffrey, R. Seymour, and A. Warner, "Computational challenges of Systems Biology," *IEEE Computer*, vol. 37, no. 5, pp. 26–33, May 2004.
[25] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, *Molecular Biology of the Cell*, fourth edition ed. New York: Garland Science, 2002.
[26] T. Pronk, S. Polstra, A. Pimentel, and T. Breit, "Evaluating the design of biological cells using a computer workbench," *Proc. ANSS 07*, no. 40, pp. 88–98, 2007. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ANSS.2007.18

Simon Polstra received his MSc degree in Computer Science from the University of Amsterdam. Currently he is a member of the Computer Systems Architecture group of the University of Amsterdam. He has previously been active as an engineer at the National Aerospace Lab in the Netherlands. His research interests include computer architecture and abstract system modeling.

Tessa E. Pronk is currently a post-doc at the Universty of Amsterdam. She received her MSc degree in Biology from Wageningen University and her PhD degree in Biology from the University of Utrecht. Her current research interests include the consequences of design, and the use of methods from Computer Science for Biology.

Andy D. Pimentel received the MSc and PhD degrees in Computer Science from the University of Amsterdam, where he currently is an assistant professor in the Department of Computer Science. He is a member of the European Network of Excellence on High-Performance Embedded Architecture and Compilation (HiPEAC). His research interests include computer architecture, computer architecture modeling and simulation, system-level design, design space exploration, performance analysis, embedded systems, and parallel computing. He is a senior member of the IEEE and a member of the IEEE Computer Society.

Timo M. Breit did his PhD studies on molecular genetic mechanisms of gene rearrangements (Erasmus University Rotterdam). He worked as a post-doc on gene expression studies (University of Utrecht) and later for several years in Fred Alt's lab (Harvard Medical School, Boston, USA). He continued as a group leader of the Micro-Array Department & Integrative Bioinformatics Unit (University of Amsterdam). Here he collaborated in the national (bio)informatics programs on e-Science (Virtual Laboratory for e-Science, VL-e), bioinformatics research (BioRange), and bioinformatics support & infrastructure (BioAssist). Within the framework of the VL-e project, his group recently created the e-BioLab, an interactive and creative environment for multi-disciplinary collaboration and e-BioScience domain interaction.