Available online at www.sciencedirect.com
ScienceDirect
BioSystems 90 (2007) 623–635
BioSystems
www.elsevier.com/locate/biosystems

Taking the example of computer systems engineering for the analysis of biological cell systems

Tessa E. Pronk [a,b,*], Andy D. Pimentel [a], Marco Roos [b], Timo M. Breit [b]

[a] *Computer Systems Architecture group, Faculty of Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands*
[b] *Integrative BioInformatics Unit, Faculty of Science, University of Amsterdam, Kruislaan 318, 1098 SM Amsterdam, The Netherlands*

Received 22 November 2005; received in revised form 11 December 2006; accepted 13 February 2007

Abstract

In this paper, we discuss the potential for the use of engineering methods that were originally developed for the design of embedded computer systems, to analyse biological cell systems. 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. In contrast to the biology domain, the computer engineering domain has the opportunity to quickly evaluate design options and consequences of its systems by methods for computer aided design and in particular design space exploration. We argue that there are enough concrete similarities between the two systems to assume that the engineering methodology from the computer systems domain, and in particular that related to embedded systems, can be applied to the domain of cellular systems. This will help to understand the myriad of different design options cellular systems have. First we compare computer systems with cellular systems. Then, we discuss exactly what features of engineering methods could aid researchers with the analysis of cellular systems, and what benefits could be gained.

Keywords: Embedded computer systems; Design space exploration; Systemic metaphor; Interdisciplinary research; Systems biology

1. Introduction

With the uprise of systems biology, biologists find themselves at work in a field that is highly oriented to interdisciplinary research. In a systems approach, biologists work together with physicists, mathematicians, engineers, chemists and computer scientists. This development supplies biologists not only with new tools for research (Fields, 2001) but also with the inspiration to take a different perspective towards biological systems.

One of the central topics of systems biology is unravelling the networks and dynamics of living cells. Although much research effort has already been directed towards systems biology, it proves difficult to gain a complete picture of the complex dynamics and networks in a cell.

Currently, cells are analysed mainly by reverse engineering methods. In this approach signalling, metabolic or gene regulatory pathways are inferred from experimental data (Basso et al., 2005). A method that is often used, for instance, is a knock-out experiment in which (the functionality of) a gene is removed. In this way biologists try to deduce the connectivity and function of individual genes. Current *in silico* modelling efforts focus mainly on incorporating as much data as possible for constructing accurate and detailed models (e.g. Tomita et al., 1999; Snoep and Westerhoff, 2004). Especially high-throughput experiments provide

* Corresponding author. Tel.: +31 20 5257898; fax: +31 20 5257762.
E-mail address: tepronk@science.uva.nl (T.E. Pronk).

0303-2647/$ – see front matter © 2007 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.biosystems.2007.02.002

so much information (though still incomplete) that it easily becomes overwhelming when attempting to analyse and interpret these data.

With the increase in data availability a different approach becomes viable: one in which the system is analysed top-down on different scales of detail in order to reduce complexity. This concept is frequently used in systems engineering. Another well-known concept in systems engineering which can be especially useful is that often more is learned about existing system architectures through an attempted redesign than through analysis alone (Simpson, 2004).

Computer systems are engineered systems that share many features with living cells (as we will argue). An interesting feature of computer systems in general is that, although they are complex, just as any cell is, they are highly specified and their behaviour can be effectively characterized, at least on low levels of abstraction. For cellular systems this is not possible as yet (see also Lazebnik, 2002, for a discussion on this subject). It may not simply be the lack of data that inhibits major advances in cell research, but also the lack of an appropriate modelling framework (Paton, 1993). A forward engineering framework could help cellular systems research a step further.

One of the first stages in the forward engineering of a system is the exploration of the design space. The design space is the total of options of possible components needed to perform tasks that make the system functional, and their specific wiring. For computer systems this can be defined as a multi-dimensional, often discrete, space defined by alternatives for allocating resources and binding functionality to resources. For biological cell systems, there are also many different alternatives to perform a given task or function and it is often not clear why one particular option was adopted rather than another. For instance, in Eukaryotic cells, there are three types of RNA polymerases whereas in Prokaryotic cells there is only one type that takes care of transcription of genes (Alberts et al., 2002 pp. 309). This could have consequences for the efficiency of this process. Another example is the synthesis of a particular transcription factor complex which can be synthesised by 50 systemically independent pathways (Papin and Palsson, 2004).

We believe that the most promising methods for design space exploration that can be applied to cellular systems are those developed for a specific kind of computer system: embedded systems (Wolf, 2001) (see Box 1 for an explanation of these systems). Specifically, this is because embedded systems share some key global features with cellular systems. The most important feature is that an embedded system has strict trade-offs between performance, power, cost, and flexibility of the system (Vahid and Givargis, 2002; Gries, 2004; Pimentel et al., 2006). By this, we mean an increase in one of these characteristics will go at the direct expense of (one of) the other characteristics. There is a strong selection on all of these characteristics, created by the fierce competition for a share on the customer market. As a result of the different trade-offs, 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) (see top part Fig. 2), which possess different combinations of the trade-offs that suit the intended function. This results in a heterogeneous architecture where many different types of components must interact to create a functioning system. This heterogeneity in components is comparable between cellular and computer systems (Fig. 1).

The multiplicity of components and also their different wiring possibilities cause a myriad of possible

Box 1. Embedded system

Embedded systems (Wolf, 2001) are computer systems 'embedded' in specific devices such as remote controls, telephones (Fig. 2) and vehicles. In contrast to general purpose computer systems, embedded systems perform predefined tasks. An embedded system thus has a specific purpose, and it controls surrounding components to perform the tasks it is intended for. Another feature that distinguishes embedded systems from general purpose computers is that they deal with the physical world, so they have temporal constraints. In other words, they usually are real-time systems. Because of this feature, the time behaviour (e.g. 'open landing gear before touchdown') of their response to stimuli may be as important as the correctness of the response (Hylands et al., 2003). Embedded systems need to cope with multiple inputs and outputs. More and more frequently, embedded systems are composed of systems on chip (SoC). These are intricate networks of components (processors and memories) that allow for parallel computing, making the system fast and (power-) efficient.

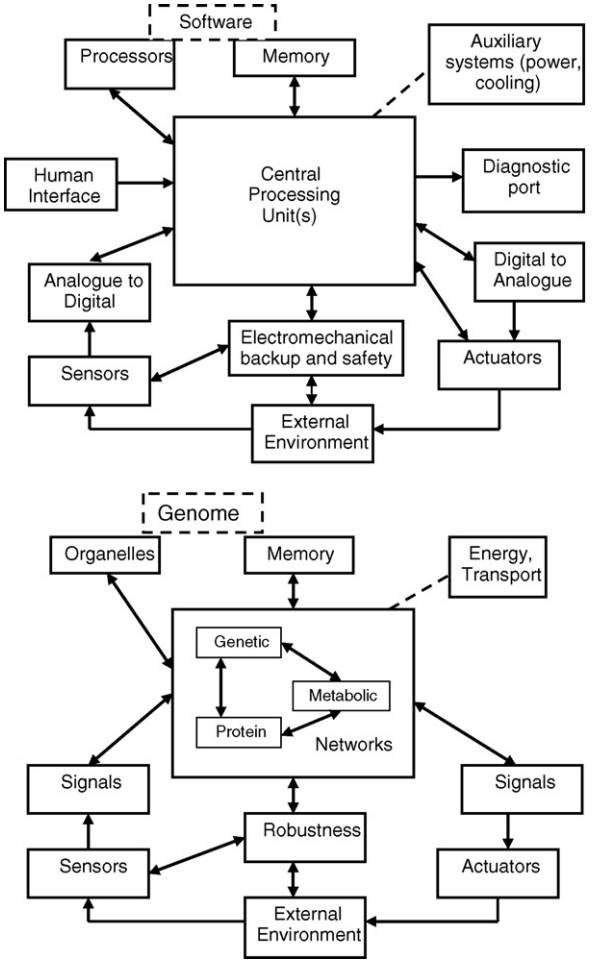

Fig. 1. Both cells and embedded computers are heterogeneous systems, consisting of different types of components that have to work together. Where possible, components with similar functions in both systems are put in the same location in the figure. 'Organelles' and 'Processors' can be seen as structures performing tasks. 'Software' and 'Genome' are the places where application is programmed. 'Energy and Transport' and 'Auxiliary systems (power, cooling)' are the elements that make it possible for the system to function. 'Central Processing Unit' and 'Networks' take care of the regulation of actions of the system. 'Sensors, Signals, Actuators and 'Digital to Analogue' converters interact with the external environment. 'Robustness' and electromechanical backup and safety' are all components to make the systems reliable and robust. Top part adapted from Koopman (1996).

designs (see Fig. 2 for an example). 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 modelling tools that facilitate this have been developed. Embedded systems are designed with the help of specific model(s) of computation that describe the behaviour and the interaction of the heterogeneous components.

For systems biologists similar design space exploration tools could speed up the understanding of underlying mechanisms of the design of cells. Of course, it is a fact that there are lots of physical differences between cellular and computer systems (Paton, 1993). These make that engineering methods for computer systems cannot be translated one-to-one to cellular systems. Establishing and describing fundamental differences and similarities of computer systems and cell systems will help to pinpoint the exact problems and possibilities for the use of methods from computer systems design in the understanding of cellular systems, and vice versa.

What is promising is that computer scientists are turning to biology to see how certain problems (adaptability, flexibility) are dealt with, and apply these solutions to computer systems (e.g. Yao and Higuchi, 1999). The future computer aided design (CAD) tools for computer engineering (Bryant et al., 2001; De Micheli, 1994) will have to deal with these new developments. This could mean that these tools will be increasingly suitable for use in the evaluation of design of biological systems in years to come. In the remaining text, when referring to CAD tools we mean those intended for (embedded) computer engineering.

In this paper, we argue that methods from the field of computer engineering can be used as an alternative framework to evaluate the architecture and functioning of cellular systems. We will highlight the similarities and differences between computer systems and cellular systems and also indicate the potential of using an engineering approach, such as used in (embedded-) computer systems design, in addition to the widely used deductive methods of cell biology.

2. Similarities and dissimilarities between computer systems and living cells

Computers and cells are arguably similar in many aspects. However, a computer circuit can be understood by any computer engineer, whereas the workings of a cell are still largely unknown (Lazebnik, 2002). Does this difference reflect a fundamental difference in the design of cellular systems and computers, or is it merely a reflection of the methods that researchers have deployed? It is a fact that both systems are investigated in opposite directions: whereas computer systems are assembled from scratch to create function, a cell is already functional and is disassembled to find where function originates from (Fig. 4). The latter is profoundly more difficult: an engineer is as unlikely to derive the exact circuit diagram of an

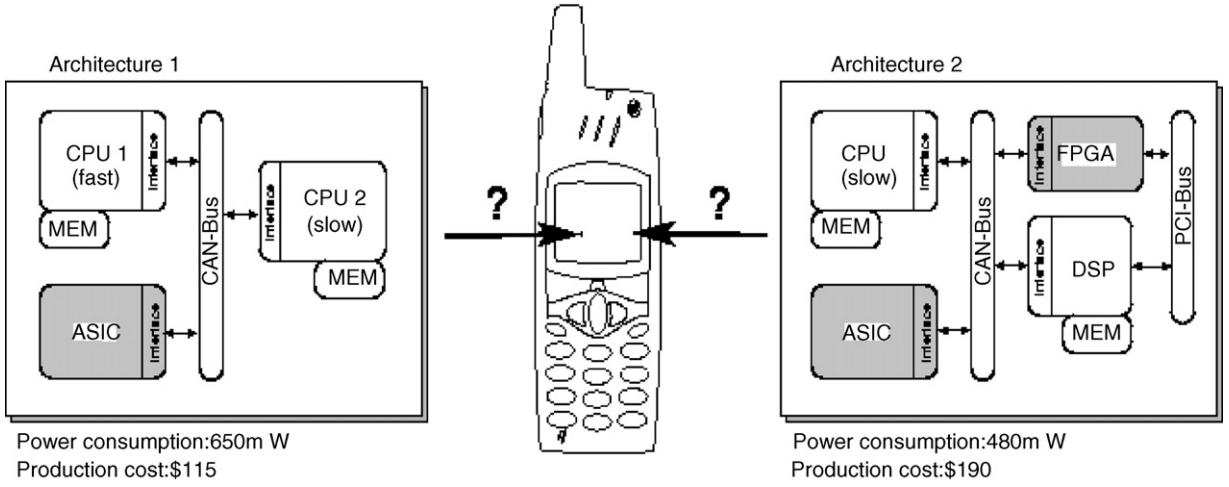

Fig. 2. Two possible instances of an embedded system. Left part: favorable with respect to cost. It has few components, two of which are cheap general purpose processors (CPU's). Right part: favorable with respect to power consumption and performance. It has two (partly-) dedicated processors (ASIC and DSP) and a reconfigurable processor (FPGA), all with low energy consumption. MEM: memory, CPU: central processing unit, ASIC: application specific hardware, CAN/PCI-bus: line of transport for information, DSP: digital signal processor, FPGA: field programmable gate array. Figure taken from the Locopos site: http://www.ecs.soton.ac.uk/~ms4/lopocos/lopocos_index.html.

unknown microprocessor component simply by correlating its outputs with its inputs as is a biologist to derive a network by the same method (Hartwell et al., 1999). On the other hand, the way in which computer systems are studied nowadays is in general very similar to the way complex biological cell systems are studied in general. In both fields, modelling is used to generate hypotheses, with (*in silico*) experiments and fine-tuning and validation on the basis of those experiments (Priami, 2004). Usually for biological systems (stochastic) average case models are used with average parameter settings. In CAD tools, worst case analysis (e.g. Jayaseelan et al., 2006) often is the focus of researchers, especially for hard-real-time embedded systems. This is because every individual system should function well, also under extreme circumstances. In the future, this can be used for engineered (in opposition to evolved) biological systems which in general also have to comply with such reliability issues.

Many authors hint at the conformity between cells and computers by comparing the electrical circuitry in computer systems with regulatory networks in cells (Savageau, 2001; Hasty et al., 2002; Kaern et al., 2003). There are, however, other useful parallels to be drawn between computer and cellular systems that illustrate the potential of applying computer-based engineering methods to study cellular systems.

2.1. Basic components

Although engineered computer systems and naturally evolved cellular systems are of different origin (purposely designed versus created by unintentional forces), there are many analogies between the two in terms of basic building blocks. If we consider a computer system in a classical (von Neumann) (Godfrey and Hendry, 1993) architecture, it consists of a *memory* for storing information such as instructions and data, a *central processing unit* (*CPU*) that executes instructions like computations, and a device that connects the different parts; the *data bus*. Information exchange with the environment comes from *input and output* (*I/O*) *units*. In Fig. 3, we compare these major components of a computer system to components in a cellular system.

One functional analogy between cellular and computer systems is the fact that both systems store

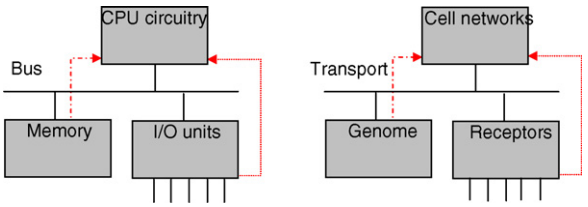

Fig. 3. Analogies between a computer (left part, from Williams, 2001) and a cellular system (right part). Interrupt request: ▪▪▪▪▪▪▪▪▪▪▪▪ Fetch-execute cycle: ▬ ▪ ▬ ▪ ▬ ▪ 'I/O units' and 'Receptors' communicate with the external environment. 'CPU circuitry' and 'Cell Networks' take care of the regulation of actions of the system. 'Memory' and 'Genome' store information. In the fetch-execute cycle, the pieces of information that are transported are electrical pulses from certain locations in the memory in computer systems and proteins stored in cell compartments in cellular systems. Interrupt requests come from the external environment in both systems.

information (Fig. 3). Where computers store their information with sequences of 1s and 0s in the memory, cells primarily store their information in the genome in linear sequences of four different types of monomers: A, T, C and G (Alberts et al., 2002). In a way, cells are even more efficient in storing information than computers. Whereas computers will be able to store 1 kilobyte of memory on 2 μm^2 of silicon, cells can fit a chromosome of 4.6 million base pairs in 2 μm^2, capable of storing some 9.2 megabyte of information (Simpson et al., 2004). On the other hand, the memory in cells, apart from mutations, is static (read only) and comparatively slow to read out because of time-consuming transcription and translation processes, whereas computers possess a flexible memory (information can be read from, and written to the memory) with much more effective read out. It depends on the viewpoint whether a cell or a computer is superior with regard to information storage.

Perhaps the most remarked-upon analogy in scientific literature is that networks within cells are roughly comparable with the logic in the CPU circuitry within computer chips (Fig. 3). In both electronic networks and cellular networks many functional circuits can be identified, such as feedback loops, memory devices, switches, threshold control, noise mitigators, etc. (Savageau, 2001; Hasty et al., 2002; Kaern et al., 2003). The biological version of circuits may be the more versatile of the two because there are many varieties of biomolecules, each with slightly different traits, which shape part of a circuit (Paton, 1993).

Transport of information (sequences of 0s and 1s) in computer systems is handled by communication media such as buses. In cellular systems the means of transport of information (e.g. proteins) comprises diffusion or directed transport by vesicles or carrier proteins. An electronic circuit has information that is represented by charge carriers, processed by the control of transport between circuit nodes. Electronic circuit nodes must be physically isolated to prevent uncontrolled transport of information from one node to the other. In contrast, isolation of molecular nodes in cellular systems is achieved by chemical specificity (Simpson et al., 2004). In the case of signal transduction via diffusion, the chemical potential (concentration) in cellular systems can be seen as an analogue to electrical potential between circuit nodes. Molecular concentration is, thus, the analogue of voltage, while synthesis, decay and molecular conversion are the analogues of current (Simpson et al., 2004).

Rather than regarding the whole cell as a machine, particular molecules can be regarded as separate machines that are capable of making computations. Especially DNA computing is a topic of research. Specific cases may be solved much more efficiently with DNA than with a traditional computer, we refer to Adleman (1994) for an example of such a case. Aside from the above mentioned examples, Paton (1993) also discusses other examples of (macro-) molecules in a cell, which can be viewed as computing machines.

2.2. Organization

The general principles that drive the design of both systems also share similarities. Just like computer systems, cells have been designed, although by evolution rather than on a drawing board. Both systems are shaped by optimisation of function. More precisely, biological systems are shaped by selective forces in the environment, computer systems by the selective forces of human demand. When optimising function, both systems are subject to trade-offs. The trade-offs in building computer systems are among others: power consumption of the system, costs of components, and speed of the system to perform the tasks that it was built for (Vahid and Givargis, 2002; Gries, 2004; Pimentel et al., 2006). In cells, trade-offs are comparable, e.g. energy requirement, building blocks such as metabolic compounds or minerals, and speed, for instance, to perform reactions or reproduce. In cells these trade-offs will have caused systems to evolve in different directions, depending on the environmental stresses. For instance, a cell might evolve to process matter quickly but have a high energy requirement, whereas in sparse environments it might evolve to be energy efficient but slow in the processing of matter. This will be effectuated with specialized components, just as in computers.

In computer systems, there are generic (processing) components that are shared for the processing of a wide range of applications. On the other hand, for instance, a discrete cosine transform performed by a dedicated hardware component is less universally applicable. A process that operates in a broad context in a cellular system is, for example, a multi-target regulation mechanism such as carbon catabolite repression. In contrast, a process that is very specific and applicable within a narrow context is a process like the induction/repression of individual pathways by their cognate substrate/product (Stelling et al., 2004). For both computer and biological systems will apply that components used for general processes can be expensive whereas components for specific processes must be cheap when they are less often used. These design options in cellular systems could possibly be modelled with design methodology for embedded systems (Edwards et al., 1997; Wolf, 2001; Gries, 2004).

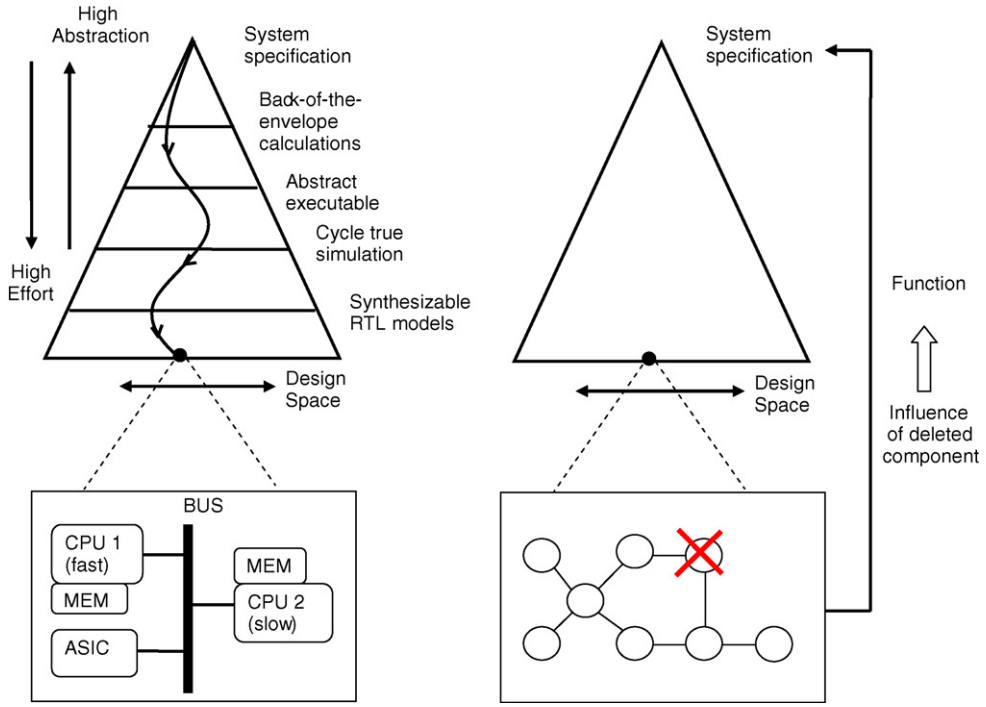

Fig. 4. Differences between the forward (computer system, left part of the figure) and the reverse (traditional biology, right part of the figure) engineering approach. Left part: for computer systems, the system is first specified at abstract level. At this level the alternatives in specifications are relatively few (top of the triangle). Possible implementations can be any on the bottom of the triangle. As the specifications are further refined to be accurate on the clock cycles of the computer and to register transfer level (RTL) models (these are implementable in hardware), the design space for implementations (bottom of the triangle) decreases in size, until the specification is so detailed that it represents a single implementation (black dot). The path shown is one possible design trajectory towards such a specific implementation. Right part: for biological cell systems, the specific 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 red cross). For abbreviations, see Fig. 2. Figure adapted from Pimentel et al. (2001).

In computer-engineered applications, design is generally carried out in an hierarchical or 'nested' fashion. This means that the behaviour of a system at a certain level is modelled and then refined to involve properties of lower space- or timescales (Edwards et al., 1997; Lauffenburger, 2000) in a top-down approach (Fig. 4). Because experimental studies performed on cellular systems typically provide information at many levels, the hierarchical representation of process knowledge is also important for the understanding of cellular systems (Peleg et al., 2002).

In engineering, understanding complex electronic networks is further facilitated by analysing them within a modular framework (Nasi, 2004). Modules in the computer-engineering domain nowadays are the IP (intellectual property) blocks: components such as co-processors or programmable microprocessors, manufactured by independent companies. They are ready-made standardized components. An IP block performs a particular task that can be very specific (e.g. decoding an MPEG video stream). The IP blocks have identifiable interfaces that enable them to be integrated easily in a system. Modularity also appears to be a feature of biological cell systems (Hartwell et al., 1999, Lipson et al., 2002). Modules can be defined as networks of molecules (protein, DNA, RNA, small molecules) that perform a given function largely independent of the context and that have identifiable interfaces to other modules (Csete and Doyle, 2002; Stelling et al., 2004; Alberghina et al., 2004; Hartwell et al., 1999). The isolation of modules to ensure their independence can be achieved in several ways. For instance, a ribosome acts to concentrate the reactions involved in making a polypeptide. A signal transduction system achieves its isolation through the specificity of initial binding of the chemical signal to receptor proteins and interactions between signalling proteins within a cell (Hartwell et al., 1999). This said, although examples of modules in biological systems are plentiful, they are in general harder to define because in these systems they usually are highly linked (Papin and

Palsson, 2004). In a practical sense, a high connectivity might make it difficult to analyse what a differential activity of a highly linked module means for the whole system. This might be a factor that makes cellular systems more difficult to analyse than computer systems.

Especially the view of the system as being hierarchical and modular in general helps to reduce complexity and therefore is useful for increasing the understanding of both cellular and computer systems (Alberghina et al., 2004).

2.3. Complexity

It might be that computers are less complex than cells, which makes computer systems better analysable than cell systems.

We can distinguish two kinds of complexity. Complexity may lie in the amount of detail; this is when there are many variables. It may also lie in the dynamics, which is the case in situations where cause and effect are subtle, such that the effects of interventions over a period of time are not obvious. Both types of complexity apply to cellular as well as to computer systems. Nevertheless, in computers, although effects are sometimes far from obvious, the functional behaviour can be fully analysed at various abstraction levels (Fig. 4), provided that computational power is sufficient. For cells this proves difficult, this could be caused by a number of factors.

Firstly, the functional behaviour of computer systems is easier to analyse as a result of having only a few non-linear relations between modules. This is simply because they can be – and are – omitted at construction. Nevertheless the performance (for instance the execution time) behaviour needs not be linear at all. In cellular systems there is no such 'omniscient power' that makes this type of conscious decision of linearity to ensure functional simplicity. On the contrary, cellular systems are renowned for their non-linear behaviour (Bruggeman et al., 2000) although processes have been reported that do enhance linearity between modules (Ihmels et al., 2004). Also crosstalk between modules is a problem because components made in one module may inadvertently influence the functioning of another module when it is used in both (Papin and Palsson, 2004). This makes the behaviour of cellular systems less predictable and testable. Interestingly, future CAD tools will have to deal with many of the same problems as the biology domain in terms of coping with crosstalk and reliability. Because computer systems become smaller and smaller, deep submicron (DSM) effects take place (Bryant et al., 2001). For instance, because of the proximity of transistors leakage of electrons can take place between neighbouring transistors (crosstalk). Also because the voltage difference between 0 and 1 diminishes, soft errors may occur by passing electrons. The biology and informatics domains can combine their expertise to solve such problems.

Secondly, much of the complexity of cells stems from redundancy and repair mechanisms that exist to cope with failure of its components. Although only a few hundred genes are essential for the basic functioning of *E. coli*, in reality it has about 4600 genes. This results in extra complex networks that may be intended to ensure robustness (Csete and Doyle, 2002). Attempts of cellular systems to reduce the frequency of failure probably cause much of the complexity of networks (Hartwell et al., 1999). In cells, components are broken down and constructed constantly (see Box 2), making them targets for mistakes every time. As an example, during the synthesis of mRNA from DNA 1 in 2000 amino acids is misread. About 1 in 3000 times, misread amino acids lead to the abortion of synthesis of the protein. As a result 1 in 12 proteins of a length of 500 amino acids remain carrying a flaw (Goodsell, 2000). Computer systems do not dynamically construct physical components and hence do not have to cope with such mistakes. In general, errors occur more often in the mechanical parts of computer steered systems than in the electrical circuitry. There are few failures of data transmissions or software executions that cannot be fixed simply by resetting the system, hence restoring its flow.

Thirdly, a cell must be able to respond to a myriad of perturbations from the outside environment and at the same time remain functional (Csete and Doyle, 2002). This kind of robustness is a key feature of biological cell systems (Stelling et al., 2004; Csete and Doyle, 2002; Kitano, 2002). A system is said to be robust if it is insensitive to the exact values of its biochemical parameters (Stelling et al., 2004; Alberghina et al., 2004). To ensure robustness, a module within a cell must be extended to accommodate, for instance, transport, and re-oxidation and feedback control. In computer systems, perturbations are less diverse. Therefore less protection needs to be incorporated in the design of computer systems and these systems can suffice with less complexity. Of course, the measure of robustness for computers will depend on their application. An airplane will have more fault-tolerance mechanisms to ensure robustness than a telephone.

2.4. Optimality

Another fundamental difference between biological and engineered systems lies in the concept of optimal-

Box 2. Biological cell system

A cellular system can be seen as a chemical factory. It takes in matter from its surroundings and uses it to make copies of itself (Alberts et al., 2002). A cellular system can exist consisting of several (differentiated) cells or exist as a single cell organism. An individual cell consists of a cell membrane that works as a barrier to the environment. Within the membrane there are molecular machines that carry out processes such as transport, energy conversion, storage, etc. Nearly all reactions within a cell are performed by proteins, which are encoded in the form of genes. Even a simple cell can exhibit thousands of different reactions. A typical cell contains more than 30.000 different proteins and large number of small molecules (Alberts et al., 2002). Specific parts of the cell (e.g. proteins, metabolic compounds, etc.), are replenished constantly or are newly produced in reaction to the environment by the system itself during its lifetime. This regulation of the internal state of the cell is done by several networks, which are organized into (more or less) separate functional modules (Csete and Doyle, 2002; Alberghina et al., 2004):

- Signal transduction networks (from signal to action)
- Gene expression networks (from gene to protein)
- Metabolic networks (from nutrients to building blocks)
- Energy conversion networks (often a part of metabolic networks)

Normally, a biological cell system originates by evolution. Recently, however, there has been a development towards synthetically altering and designing cells (Simpson, 2004; Endy, 2005).

ity. The essence of computer engineering is the capacity to engineer circuits that transform information from one form to another based on a set of rules. For individual engineered systems, the objective is to optimise and organize the functioning of the system directly with respect to one (or more) specific task(s) (Vilarroya, 2002) which is a multi-parameter optimisation problem. In biological cell systems, natural selection results in the emergent objective of a design that gives the highest fitness, i.e. a design that enables the organism to leave as many copies of itself as possible over time (or at least more than its direct competitors). In other words, the objective is the survival of the line. This implies that, in order to understand the full function of modules in nature (e.g. for a frog: a tongue to catch flies), we may require a measure of their effect on reproductive ability (Hartwell et al., 1999). The ultimate objective of a biological system thus is several degrees separated from the function of individual modules, whereas the modules in an engineered system contribute more directly to the objective function (Vilarroya, 2002). For biological cell systems this makes it more difficult to evaluate the functioning of individual components to execute a certain task, in terms of their efficiency.

Recently, however, computer systems also have to comply with new design goals such as flexibility and ease-of-deployment that cannot be quantified as easily. Expression in an objective function is difficult in this case, just as it is for biological systems. Future CAD tools will have to address these problems.

2.5. Adaptability

Events within in a cell are often stochastic, meaning that there is always a certain chance that an event (e.g. an enzymatic reaction) occurs, based for instance on the concentrations of involved compounds. This makes events and processes unreliable. Also fuzzy, probabilistic intermediates need to be refined first to give unique solutions. This is done by error-detection and error-correction mechanisms that work with 'trial and error' events. This can be seen as a design principle, especially suitable for modifications on evolutionary timescales (Hartwell et al., 1999). In computers, these phenomenon are not widely applied.

Another aspect of biological cell systems is that the problems that need to be coped with are often ambiguous (Diorio and Rao, 2000). Typically there is a lot of noise in the signals within cellular systems, so that continuous adaptation is needed to cope with problems. Furthermore, a cell can adapt to a change in circumstances. For instance, a cell can adapt its metabolism to the available substrate, but this works only if the transition from one substrate to the other is gradual because it takes time for a cell to become fully adapted. This design principle is not yet widely used in engineered computer systems.

Instead, in computer systems, behaviour is designed beforehand to cope with certain problems in a static way: it can either cope with certain problems or it cannot. This makes the behaviour more predictable and hence easier to analyse. In accordance with the differences in adaptability, computers are superior in fast calculations of large tasks, whereas cellular systems are superior in energy efficient processing of ill-posed problems (Diorio and Rao, 2000).

Nevertheless, in computer systems there is an increasing need for higher flexibility. Reconfigurable components are developed using, for example, field programmable gate arrays (FPGA) (Yao and Higuchi, 1999). This makes computer systems adaptable to different circumstances as well. This leads to so called evolvable computing. Future CAD tools will have to cope with such new design requirements and might therefore become increasingly suitable for evaluating biological systems.

3. An embedded systems' engineering approach to understand cellular systems

So far we have discussed general features of cells and computers. The similarities between both systems suggest that methodology for the evaluation and design of computer systems can be applied to cellular systems. An engineering approach for cells might give new insights and better understanding of cellular systems. We will now specifically discuss methods for the design space exploration (Edwards et al., 1997; Pimentel et al., 2001; Gries, 2004) of embedded systems (see also Box 1) and their useful features.

3.1. Useful features

Firstly, embedded computer systems are almost always evaluated beforehand by top-down engineering approaches (Pimentel et al., 2001) rather than afterwards with bottom-up reverse engineering approaches, such is currently mostly the case with cellular systems (Fig. 4). In a top-down approach, the most basic traits of components are taken into account, defined by some rough parameter specification. This means that the more detailed and complex behaviour that takes place at lower levels of abstraction is disregarded. This black-box method is very fast for roughly determining the basic design options. After determining the basic design options, refinements can be done to analyse the system in more detail (Edwards et al., 1997; Pimentel et al., 2006). The top-down approach enables working with gaps in the knowledge, or with components that are not fully specified. In cells, a top-down approach can be used, for instance, to omit the complex (and often unknown) kinetics at detailed cellular level (Brand, 1996).

Secondly, aside from working at different levels of hierarchy and abstraction, partitioning the system in different domains to study different aspects of the system is an approach used in embedded system design to reduce the complexity of a system (Jantsch, 2004). For instance, in computer systems engineering, separation of concerns is a principal concept. It is a much-used procedure to separate function (application) from architecture (the task-performing modules) (Keutzer et al., 2000; Pimentel et al., 2006). With separation of application and architecture, the architecture blocks can be interchanged while the application remains unaltered (Pimentel et al., 2006). This trait enables the programmer to change architecture independently from the functional application, reducing programming time. So, only a part of the system needs rebuilding when a designed system functions sub optimally, rather than the whole system. Also, computation (within modules) can be separated from communication (between modules) in embedded systems (Keutzer et al., 2000). IP blocks are engineered to perform a particular procedure and have identifiable interfaces. If the blocks are connected, communication between the blocks can take place. This communication is separate from the computation that occurs within the IP blocks and this increases their reusability. The separation of concerns can help for instance with the design of organisms with pharmaceutical or industrial applications. The field of synthetic biology (Endy, 2005) might facilitate this as we expect this field to provide clear-cut functional modules (Benner and Sismour, 2005) that can be seen as separate biological IP blocks. The field of synthetic biology aims at designing well-defined and functional modules, for instance by including those components that are needed for a particular task, thus omitting extra functionality for coping with various stresses which would be present in naturally evolved cells.

The functionality of an embedded system can be captured using a variety of conceptual specification models (Edwards et al., 1997). As result of separating a system in different domains (e.g. time, computation, communication between processes, and data) different models may be appropriate for the separate domains. Consequently, the focus in embedded systems design has been on coupling of heterogeneous models (Eker et al., 2003; Hylands et al., 2003; Jantsch, 2004; Pimentel et al., 2006). In biology, a similar development has taken place. In attempts to cover a whole cell, different data is generated (proteomic, metabolomic, transcriptomic,

and genomic data) which, for an overall understanding of the system, will have to be integrated. Especially in the embedded computer systems community the integration of heterogeneous information is more advanced and expertise and models from this field may be used for integrating the data of cellular systems.

Lastly, the most useful models of computation (Edwards et al., 1997) for embedded systems handle concurrency (the execution of multiple processes or operations simultaneously) and time, because the timing behaviour of the system is very important and components need to synchronize with each other and with the environment (Box 1). Where possible, concurrency in the embedded systems (as it would in any system) significantly increases the speed of the systems. This is relevant because embedded systems are usually real-time systems (see Box 1 for an explanation). The living cell is also a real-time system. Given the similarities between computer and cellular systems, the above mentioned methods and models of computation might be applied to modelling cellular systems.

3.2. Possible general applications

Applying methods for design space exploration of embedded systems to cellular systems might give new insights into design options of cellular systems. There are a few characteristics that a biological system must comply with for it to be purposely modelled by design space exploration methods. Sesame (Pimentel et al., 2001, 2006), for instance, can be used to evaluate designs for timed information processing systems. This excludes static designs such as protein shape, or design for mechanic properties such as rigidity but includes all process based functions. The evaluation of the design is in principle on the basis of efficiency in performance (how fast does a particular design perform the specified functions). However, other objectives can be included such as cost or power efficiency. Rather than striving for maximal fidelity to the actual present system, the design methods of embedded systems will generate null-hypotheses and mechanistic explanations on phenomena that would otherwise be difficult to derive from the complex interactions within the living cell. These will have to be confirmed with *in vivo* experiments. In the cases where *in silico* engineered systems can be reproduced *in vivo*, it is important to keep in mind that if model predictions deviate from the real-life measurements, this may indicate a gap in our knowledge. This will provide guidance for future experimentation (You, 2004).

An engineering approach, e.g. looking at the design options of cells, can be an appropriate method to contribute to the following issues:

1. Putative design options. For both cellular and embedded systems there are many design options that could yield a required functionality. In cells, it is often difficult to see why one particular implementation was adopted rather than another, seemingly equally appropriate alternative. As in design space exploration methods for embedded systems, there are possibilities to evaluate these options, and adopting such a system engineering approach will be useful for addressing such design issues in cells.
2. Address ‘white spots’ in cellular architecture. A top-down engineering method can be used in embedded systems to account for components that have not yet been identified, e.g. to determine candidate components by the restrictions and specific demands of the existing architecture. Filling of these gaps in our knowledge will be simplified by the introduction of abstract components that can be filled in and/or refined at a later stage of development.
3. Evaluate existing architectures of cellular systems on their optimality (e.g. is the configuration as we expected?). We could evaluate the robustness and fragility of a system in terms of bottlenecks and overloads and pinpoint the level at which the problem occurs (energy, communication, etc.). Additionally, the architecture can be evaluated on its ability to cope with different requirements or environments. For instance, how does a different connection (e.g. rewiring) between different modules affect behaviour and architecture of cells (Hartwell et al., 1999). When we compare different architectures with the actual implementation in cells, it can help us elucidate past evolutionary pressures.
4. Design parts of a cell or specific cellular mechanisms can help in identifying which components are essential for performing a task or function. The rest of the components in the real organism can be considered excess, for instance to ensure stability of output (Vohradsky, 2001).
5. Study the amount of parallelism in cells. Parallelism is an important feature of biological cell systems as well as computer systems (Paton, 1993). We can consider one of the core processes in cellular functioning: the expression of genes. The expression of genes is a highly parallel task. From one gene copy, several mRNA strings can be made in parallel and from each separate mRNA string; several proteins can be made in parallel. For performing such highly parallel tasks

either one component must be able to process them in parallel, as often processors in computers do, or there must be several of the same components that work simultaneously. The latter option often has evolved in biological cells. With methods originally developed for embedded systems we might calculate the required amount of parallelism in a cell in addition to finding the obligate sequential processes.

The methods and techniques used in the field of computer architecture design will, ideally, help us focus on understanding the design options and consequences within a cell. We might be able to gain insight in the most fundamental issues concerning cellular design. Taking into account restraints from available compounds, energy and time limits, we can find and characterize the best possible solution in terms of architecture for a given task within a cell.

3.3. Example: a design space exploration approach applied to cellular systems

As proof of principle, we demonstrate a possible application of a CAD tool to a biological case. We model a transcription/translation application using the Sesame approach (Pimentel et al., 2001, 2006). The Sesame approach is a specific example of a new and promising modelling framework from the computer engineering domain. This framework is originally intended for the exploration of design space of embedded systems architectures early in the design phase (i.e. abstract executable, see Fig. 4). We will shortly describe our ideas in this paper rather than give an in-depth and technical description.

In the transcription/translation application we have the following task: produce a protein from a piece of DNA strand. Suppose the protein consists of four identical parts. We could ponder about whether it is better to have programs (a) or (b):

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

We model this problem in the Sesame framework. An essential feature of the Sesame framework is the separation of application (functionality, tasks) from architecture (performing structures). During a simulation, the application emits traces (that represent a sequence of events) to the performing structures in the architecture which simulate their timing consequences. All traces combined represent the workload for the architecture. In Sesame the events within the traces are automatically scheduled in an appropriate order (i.e. to avoid deadlock) to their performing structures by an intermediate mapping layer. The amount and kind of events in traces emitted by the application in our proof of principle depend on the DNA sequence code that has to be put to expression and the function description, e.g. how many transcriptions/translations are needed. These are given functions. Each task (i.e. trace event) is performed by the architecture with a certain time cost (latency). In the application layer we have the tasks load/store ‘DNA’, ‘mRNA’, ‘ribonucleotide’, ‘amino acid’ and an execute task ‘move along DNA’. Suppose we would want to refine these tasks, for instance because of renewed insights. The task ‘load DNA’ in the application is left as it is. In the intermediate mapping layer, the task is disassembled into sub tasks such as ‘form holoenzyme’, ‘locate promoter’, and ‘unwind DNA’. At the same time in the architecture layer we have performing structures: one for all transcription tasks (named ‘RNA polymerase II’) one for al translation tasks (named ‘Ribosome’) and one for all tasks for the supply of amino acids (named ‘tRNA’). These performing structures can be refined to contain the different sub structures needed to perform the refined tasks. In this way, the performance of a structure can more accurately be determined.

Suppose we have a protein that consists of four identical subparts of three amino acids. Intuitively, case (a) seems simplest. However, for case (a) the process takes 706 time steps, whereas the production of the protein in case (b) was done in 406 time steps. We come to the conclusion that, given the architecture, the task in the format (b) can be processed most efficiently. Mainly because in case (a) many ribonucleotides have to be loaded in the transcription in comparison with case (b) (this proof of principle is not necessarily biologically realistic).

From another perspective, we derive from this proof of principle that it is more rewarding to re-engineer the performing structure ‘RNA polymerase II’ (be it by evolution or synthetically) towards more efficiency especially in case (a) for it is the limiting factor: it takes about 75% of the total time of the process (results not shown here). It would also be a candidate for refinement, to see what stalls the performance most in this performing structure.

To make design space exploration useful in general, there should be alternative options in the system under study to perform a certain task, with consequences for the performance of the system, without the choice of per-

forming structure having any effect on the application. In this way, the best possible architecture for a given (user specified) task or process within a cell can be found independent from the task or process itself. 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. Many tasks in a cell thus must be performed by 'dedicated' components. For instance, in cells, only RNA polymerase II is suitable to perform the transcription of a gene to mRNA. Design options, in this case, lie within the amount of RNA polymerase II or its location (e.g. cell compartment).

On the other hand, we do have tasks for which we know alternative performing structures exist. One much studied example is the different types of tRNA that can be used in translation. Every of the 20 possible amino acids that are present in living material is coded by a three nucleotide long codon. Every amino acid is coded for by one or several codons (Figure 6–50 in Alberts et al., 2002). These codons have to be recognized by the anti-codons of tRNA to supply the correct amino acid to the growing chain of peptides. In different organisms, there is variation in the amount and type of tRNA used to perform translation. With a design space exploration tool as Sesame we can hope to answer, for instance, the following question: Given a gene expression pattern, and given we know the behaviour of different tRNA types, what is the tRNA population that will produce the requested proteins most efficiently (e.g. fastest, cheapest)?

4. Conclusion

Between cellular and computer systems more analogies can be made than merely the obvious similarity between electrical circuits in computer systems and regulatory networks in cellular systems. The universality of architectural features in computers and cells is an indication that the knowledge and expertise from large and well-characterized non-biological systems can be used to analyse and characterize cellular systems (Barabasi and Oltvai, 2004). In computer systems design, there are computer aided design (CAD) methods for quickly reducing the design space and simulating and analysing these complex systems. Especially the methods for design of embedded systems seem appropriate for an analysis of cellular systems design. This is because these systems have to take different distinct trade-offs into account that are taken care of by different types of components, making them heterogeneous in nature, just as biological cells. In addition, they are concurrent systems that operate in real-time and, just like biological cells, they react to the physical environment. Importantly, future CAD tools will have to deal with much of the same problems that now only apply to biological systems which will make them increasingly appropriate for modelling cellular systems. We foresee that methods for design and analysis of complex systems like embedded systems will help scientists to unravel general principles that govern the structure and behaviour of cellular systems.

Acknowledgements

This work was carried out in the context of the Virtual Laboratory for e-Science project (www.vl-e.nl <http://www.vl-e.nl>). This project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ). We would like to thank Bob Herzberger for providing the opportunity to carry out this work. We would like to express our gratitude towards Feike Schieving and Jack Valentijn for lively discussions on biological cells versus computers. For critical reading of the manuscript and discussions on the contents we thank Scott Marshall, Christiaan Henkel, and Jenny Batson.

References

Adleman, L.M., 1994. Molecular computation of solutions to combinatorial problems. Science 266, 1021–1024.

Alberghina, L., Chiaradonna, F., Vanoni, M., 2004. Systems biology and the molecular circuits of cancer. Chembiochem 5, 1322–1333.

Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P., 2002. Molecular Biology of the Cell. Garland Science, New York.

Barabasi, A.L., Oltvai, Z.N., 2004. Network biology: understanding the cell's functional organization. Nat. Rev. Genet. 5, 101–113.

Basso, K., Margolin, A.A., Stolovitzky, G., Klein, U., Dalla-Favera, R., Califano, A., 2005. Reverse engineering of regulatory networks in human B cells. Nat. Genet. 37, 382–390.

Benner, S.A., Sismour, A.M., 2005. Synthetic Biology. Nat. Rev. Genet. 6, 533–543.

Brand, M.D., 1996. Top down metabolic control analysis. J. Theor. Biol. 182, 351–360.

Bruggeman, F.J., van Heeswijk, W.C., Boogerd, F.C., Westerhoff, H.V., 2000. Macromolecular intelligence in microorganisms. Biol. Chem. 381, 965–972.

Bryant, R.E., Cheng, K.T., Kahng, A.B., Keutzer, K., Maly, W., Newton, R., Pileggi, L., Rabaey, J.M., Sangiovanni-Vincentelli, A., 2001. Limitations and challenges of computer-aided design technology for CMOS VLSI. Proc. IEEE 89 (3), 341–365.

Csete, M.E., Doyle, J.C., 2002. Reverse engineering of biological complexity. Science 295, 1664–1669.

De Micheli, G., 1994. Computer-Aided Hardware-Software Codesign. IEEE Micro 14 (4), 10–16.

Diorio, C., Rao, R.P.N., 2000. Neural circuits in silicon. Nature 405, 891–892.

Edwards, S., Lavagno, L., Lee, E.A., Sangiovanni-Vincentelli, A., 1997. Design of embedded systems: formal models, validation, and synthesis. Proc. IEEE 83 (3), 366–390.

Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y., 2003. Taming heterogeneity—the ptolemy approach. Proc. IEEE 91 (1), 127–144.

Endy, D., 2005. Foundations for engineering biology. Nature 438, 449–453.

Fields, S., 2001. The interplay of biology and technology. Proc. Natl. Acad. Sci. USA 98, 10051–10054.

Godfrey, M.D., Hendry, D.F., 1993. The computer as von Neumann planned it. IEEE Ann. Hist. Comput. 15 (1), 11–21.

Goodsell, D.S., 2000. Biomolecules and nanotechnology. Am. Sci. Online 88 (3), 7–11.

Gries, M., 2004. Methods for evaluating and covering the design space during early design development. Integr. VLSI J. 38, 131–183.

Hartwell, L.H., Hopfield, J.J., Leibler, S., Murray, A.W., 1999. From molecular to modular cell biology. Nature 402, C47–C52.

Hasty, J., McMillen, D., Collins, J.J., 2002. Engineered gene circuits. Nature 420, 224–230.

Hylands, C., Lee, E., Liu, J., Liu, X., Neuendorffer, S., Xiong, Y., Zhao, Y., Zheng, H., 2003. Overview of the Ptolemy Project, Technical Memorandum University of California, Berkeley, pp. 1–36.

Ihmels, J., Levy, R., Barkai, N., 2004. Principles of transcriptional control in the metabolic network of Saccharomyces cerevisiae. Nat. Biotechnol. 22, 86–92.

Jantsch, A., 2004. Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation. Morgan Kaufmann Publishers, San Francisco, pp. 1–351.

Jayaseelan, R., Mitra, T., Li, X., 2006. Estimating the worst-case energy consumption of embedded software. Proc. 12th IEEE RTAS, 81–90.

Kaern, M., Blake, W.J., Collins, J.J., 2003. The engineering of gene regulatory networks. Ann. Rev. Biomed. Eng. 5, 179–206.

Keutzer, K., Malik, S., Newton, A.R., Rabaey, J.M., Sangiovanni-Vincentelli, A., 2000. System-level design: Orthogonalization of concerns and platform-based design. IEEE Trans. Computer-Aided Des. Circuits Syst. 19 (12), 1523–1543.

Kitano, H., 2002. Computational systems biology. Nature 420, 206–210.

Koopman, P., 1996. Embedded system design issues (the rest of the story). Proc. Int. Conf. Comput. Des. (ICCD96), 310–318.

Lauffenburger, D.A., 2000. Cell signaling pathways as control modules: Complexity for simplicity? Proc. Natl. Acad. Sci. USA 97, 5031–5033.

Lazebnik, Y., 2002. Can a biologist fix a radio? or, what I learned while studying apoptosis. Cancer Cell 2, 179–182.

Lipson, H., Pollack, J.B., Suh, N.P., 2002. On the origin of modular variation. Evolution 56, 1549–1556.

Nasi, S., 2004. From databases to modelling of functional pathways. Comp. Funct. Genomics 5, 179–189.

Papin, J.A., Palsson, B.O., 2004. Topological analysis of mass-balanced signaling networks: a framework to obtain network properties including crosstalk. J. Theor. Biol. 227, 283–297.

Paton, R.C., 1993. Some computational models at the cellular level. BioSystems 29, 63–75.

Peleg, M., Yeh, I., Altman, R.B., 2002. Modelling biological processes using workflow and Petri Net models. Bioinformatics 18, 825–837.

Pimentel, A.D., Erbas, C., Polstra, S., 2006. A systematic approach to exploring embedded system architectures at multiple abstraction levels. IEEE Trans. Comput. 55 (2).

Pimentel, A.D., Hertzberger, L.O., Lieverse, P., van der Wolf, P., Deprettere, E.F., 2001. Exploring embedded-systems architectures with Artemis. IEEE Comput. 34 (11), 57–63.

Priami, C., 2004. Computational systems biology—Preface. Theor. Comput. Sci. 325, 1–2.

Savageau, M.A., 2001. Design principles for elementary gene circuits: elements, methods, and examples. Chaos 11, 142–159.

Simpson, M.L., 2004. Rewiring the cell: synthetic biology moves towards higher functional complexity. Trends Biotechnol. 22, 555–557.

Simpson, M.L., Cox, C.D., Peterson, G.D., Sayler, G.S., 2004. Engineering in the biological substrate: information processing in genetic circuits. Proc. IEEE 92, 848–863.

Snoep, J.L., Westerhoff, H.V., 2004. The silicon cell initiative. Curr. Genomics 5, 687–697.

Stelling, J., Sauer, U., Szallasi, Z., Doyle, F.J., Doyle, J., 2004. Robustness of cellular functions. Cell 118, 675–685.

Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T.S., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter, J.C., Hutchison, C.A., 1999. E-CELL: software environment for whole-cell simulation. Bioinformatics 15, 72–84.

Vahid, F., Givargis, T., 2002. Embedded System Design: a Unified Hardware/Software Introduction. John Wiley and Sons, Inc.

Vilarroya, O., 2002. "Two" many optimalities. Biol. Philos. 17, 251–270.

Vohradsky, J., 2001. Neural model of the genetic network. J. Biol. Chem. 276, 36168–36173.

Williams, R., 2001. Computer Systems Architecture: a Networking Approach. Pearson Education Limited, Harlow, p. 50.

Wolf, W., 2001. Computers as Components, Principles of Embedded Computing System Design. Morgan Kaufman publishers, San Francisco.

Yao, X., Higuchi, T., 1999. Promises and challenges of Evolvable Hardware. IEEE Trans. Syst. Man Cybern.—Part C: Appl. Rev. 29 (1), 87–97.

You, L.C., 2004. Toward computational systems biology. Cell Biochem. Biophys. 40, 167–184.