Centrum voor Wiskunde en Informatica

REPORTRAPPORT

SEN

Software Engineering

Software ENgineering

Modelling coordination in biological systems

D.G. Clarke, D.F. de Oliveira Costa, F. Arbab

Report SEN-R0415 September 2004

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

Modelling coordination in biological systems

ABSTRACT

We present an application of the Reo coordination paradigm to provide a compositional formal model for describing and reasoning about the behaviour of biological systems, such as regulatory gene networks. Reo governs the interaction and flow of data between components by allowing the construction of connector circuits which have a precise formal semantics. When applied to systems biology, the result is a graphical model, which is comprehensible, mathematically precise, and flexible.

1998 ACM Computing Classification System: J.3, D.2.11, D.3.2, F.1.2
Keywords and Phrases: Coordination; Systems Biology; Reo

Modelling Coordination in Biological Systems

Dave Clarke David Costa Farhad Arbab
CWI
PO Box 94079, 1090 GB Amsterdam
The Netherlands
{dave,costa,farhad}@cwi.nl

September 30, 2004

Abstract

We present an application of the *Reo* coordination paradigm to provide a compositional formal model for describing and reasoning about the behaviour of biological systems, such as regulatory gene networks. *Reo* governs the interaction and flow of data between components by allowing the construction of connector circuits which have a precise formal semantics. When applied to systems biology, the result is a graphical model, which is comprehensible, mathematically precise, and flexible.

1 Introduction

Within a biological system, complex *biological pathways*, the interconnection between genes and proteins and other chemical reactions within organisms, form the basic fabric of life. Systems biology aims to integrate information about biological entities and their relationships with the aim of understanding the complex metabolic networks and the role of genes within them. Beginning with incomplete information about a system, biologists produce a mathematical model of their system, which is ultimately used to predict the behaviour of their system. By testing their model against experimental data, biologists can produce successively more accurate models. But as biologists study larger systems, their modelling technology is proving to be insufficient. What systems biologists need are formal techniques to describe reaction networks, giving, for example, their algebraic behaviour, enabling abstraction from molecular actions, if desired. The models should enable composition of larger models from smaller ones, and tools should be provided for analysing the behaviour of these models under a variety of boundary conditions [15]. A number of models have recently been proposed to fill this gap. Frequently, these models stem from the study of concurrency theory, which traditionally provide theoretical foundations for concurrent and distributed computer systems. Some models are based on process calculi [27, 10, 11, 12, 7, 26, 20], whereas others adopt circuit or network-based models [28, 19, 22, 23]. An immediate advantage of this approach to modelling is that the extensive theory that exists for reasoning about concurrency can be applied to biological systems [6, for example]. Furthermore, by providing formally precise, executable models of biological systems, these approaches represent a promising path toward understanding highly complex biological systems.

An alternative approach is to use a framework which abstracts away from the behaviour of agents and focus on their interaction both at a primitive level and across the entire system. Understanding biological systems requires understanding, for example, the gene networks which *regulate* protein production through the *activation* and *suppression* of various enzymes and other genes. Often mere *competition* for molecules is the means by which nature achieves organisation through distributed *control* [15]. In their most general setting, regulation, activation, suppression, competition, and control are studied within the inter-disciplinary field *Coordination Theory* [21].

The *Reo* coordination model for coordinating software components [2] falls within this general theory. Rather than focusing on what components (or processes) do in isolation, *coordination models*, including *Reo*, focus on the composition of components and their interaction, generally by governing the *flow* of data between components. The coordination layer prescribes/describes the interaction between the components. This change of focus enables one to understand a system at a higher level of abstraction, focusing on the interaction and control aspects rather than on the entities being controlled. It is from this perspective that we endevour to contruct models for use in systems biology. *Reo* coordinates components using connectors composed of primitive channels. These connectors can be seen as circuits that capture the interaction, data flow, synchronisation, and feedback among components. Circuits have a graphical representation and precise semantics [1, 4], and allow the composition of models of large systems out of smaller ones. This paper applies *Reo* to modelling biological systems.

2 Evidence of Coordination in Biological Systems

In biological systems the presence of coordination mechanisms is evident at different levels and in different ways [31, 21, 30, 14]. In general, coordination takes place in a highly distributed manner, but we can break it down into a number of categories: inter-cellular coordination, boundary coordination, intra-cellular coordination, and gene coordination.

Inter-cellular coordination. A cell is coordinated via interaction with its environment. This can include interaction with other cells. All cells receive and respond to signals from their surroundings. The simplest bacteria sense and swim toward high concentrations of nutrients, such as glucose or amino-acids. Many unicellular eukaryotes also respond to signalling molecules secreted by other cells, allowing cell-cell communication. It is, however, in multi-cellular organisms where cell-cell communication reaches its highest level of sophistication. The behaviour of each individual cell in multi-cellular plants and animals must be carefully regulated to meet the needs of the organism as a whole. The function of the many individual cells in a multi-cellular organism is integrated and coordinated via a variety of signalling molecules that are secreted on the surface of one cell and bound to a receptor present on another cell.

Boundary coordination. A cell's internal behaviour is stimulated or, more generally, regulated via interaction on its boundary. Most cell surface receptors stimulate target enzymes which may be either directly linked or indirectly coupled to receptors. A chain of reactions transmits signals from the cell surface to a variety of intra-cellular targets. The targets of such signalling pathways frequently include factors that regulate gene expression. Intra-cellular signalling pathways thus connect the cell surface to the nucleus, leading to changes in gene expression—the internal coordinator of cell behaviour—in response to extra-cellular stimuli. Changes in expression lead to different metabolic pathways.

Intra-cellular coordination. Intra-cellular reactions regulate all aspects of cell behaviour including metabolism, movement, proliferation, survival and differentiation. Metabolism is a highly integrated process. It includes *catabolism*, in which the cell breaks down complex molecules to produce energy, *anabolism*, where the cell uses energy to construct complex molecules and perform other biological functions, and more general *metabolic pathways* consisting of a series of nested and cascaded feedback loops which accommodate flexibility and adaptation to changing environmental conditions and demands. Metabolism is regulated through competition for resources, and by positive and negative feedback. Negative feedback (usually by end-product inhibition) prevents the over-accumulation of intermediate metabolites and contributes to maintaining homeostasis—a system's natural desire for equilibrium.

Gene coordination. Gene behaviour plays the most significant coodination rôle, ultimately being the central coordination mechanism within the entire cell. *Gene Regulatory Networks* are a model which represents and emphasizes the genes' rôle in all activities within a cell. Genes regulate and orchestrate the different phases that comprise a metabolic pathway. A gene regulatory network can be interpreted as a complex signalling network in which all coordination between different cell entities is dictated dynamically by the genes directly, or indirectly through coordination mechanisms such as *suppression/negative regulation* and *activation/positive regulation.* Furthermore, gene behaviour is itself regulated via the products of metabolism and via self-regulation/feedback.

3 Coordination in *Reo*

Coordination models and languages enable the control of the interaction behaviour of mutually anonymous components (or processes) from outside of those components. Rather than allowing components to communicate directly, a coordination model intervenes to regulate, inhibit, and direct the communication and cooperation of independent components. *Reo* is a powerful channel-based coordination model wherein complex coordinators, called *connectors*, are compositionally built out of smaller ones. Every connector in *Reo* imposes a specific coordination pattern on the entities that interact via the connector [2].

The most primitive connectors in *Reo* are *channels.* A channel has exactly two ends, each of which may be an *input* end, through which data enter, or an *output* end, through which data leave the channel. Channels may have an input end and an output end, or two input ends, or even two output ends. *Reo* places no restriction on their behaviour, so long as they support certain primitive operations such as I/O. This allows an open set of different channel types to be used simultaneously together in *Reo*, each with its own policy for synchronisation, buffering, ordering, computation, data retention/loss, etc. A number of basic channels are [2]:

Sync A synchronous channel is denoted $\longrightarrow$. A data item is transmitted through this channel when both a write on its input end and a take on its output end are pending.

LossySync A lossy synchronous channel is denoted $\dashrightarrow$. This channel behaves like a synchronous channel whenever a take on the output end is pending. However, if a write is performed to its input end and no take is pending, the input is performed, but data is lost and thus not transfered.

SyncDrain A synchronous drain is denoted $\rightarrow\!\!-\!\!\leftarrow$. A data item flows only when writes are pending on each of its two input ends. The effect is to synchronise the two writers. The data is lost.

SyncSpout A synchronous spout is denoted $\leftarrow\!\!\rightarrow$. A data item flows only when takes are pending on each of its two output ends. The effect is to synchronise the two readers.

FIFO1 A FIFO1 channel has buffer of size one. A write to its input end can succeed only if the buffer is empty, after which the written value is stored in the buffer. Otherwise the writer blocks. A take from its output end only succeeds if the buffer is full. A FIFO1 which is initially empty is denoted $-\!\!\boxed{}\!\!\rightarrow$, and one which is initially full is denoted $-\!\!\boxed{0}\!\!\rightarrow$.

A *Reo connector* is a set of channel ends and their connecting channels organized in a graph of *nodes* and edges such that: zero or more channel ends coincide on every node; every end coincides on exactly one node; and there is an edge between two (not necessarily distinct) nodes if and only if there is a channel whose two ends coincides on each of the nodes. The coincidence of channel ends at a node has specific implications on the data flow among and through those ends. There are three kinds of node, depending upon the kinds of coincident ends. *Input nodes* have only input ends, *output nodes* have only output ends, and *mixed nodes* have at least one of each. A component connected to an input node may write to that node, which then acts as a *replicator*, copying the data item to all ends coincident on the node. Similarly, a component connected to an output node can take data if data is available on any one of the output ends coincident on the node. Thus the node acts as a non-deterministic *merger* of the coincident output ends. A mixed node acts like a self-contained pumping station, combining the behaviour of an output node (merger) and an input node (replicator), by non-deterministically taking a suitable data item offered by any one of its coincident output ends and replicating it to all of its coincident input ends. This operation succeeds only when all the output ends attached to the node are able to accept the data. Graphically nodes are represented using "•".

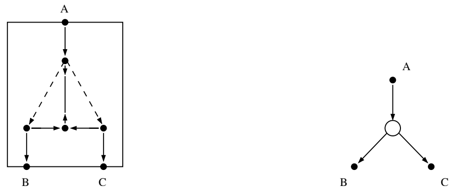


Figure 1: (a) Exclusive Router connector and (b) the Mnemonic for its instances

The last feature of *Reo* we mention is *encapsulation*. This enables abstraction of the details of a connector. It it representated as a box enclosing a circuit.

We now present an example of a *Reo* connector to illustrate the complex behaviour which can emerge through composition of simple channels. An *exclusive router* is depicted in Figure 1, along with the shorthand mnemonic "◦" which we will use subsequently to represent the instances of this connector. Each data item entering via node A will be synchronously passed to either node B or node C, but not both, depending upon which of B and C first makes a request for data. Ties are broken non-deterministically [1]. This behaviour emerges in an non-obvious manner simply by composing together a few simple channels. More examples are included in an appendix.

4 Case Study: Galactose Utilization in Yeast

When a biologist wants to understand a system, such as the role of various genes in regulating a metabolic pathway, they construct a model of the system. Initially, the model is a black box. Through successive refinements, based on experimental data, more accurate models are developed. The ultimate goal of this process is to find models which are accurate enough to be used in a predictive manner, giving an indication of how the system being modelled will behave under conditions outside those covered by existing experimental data.

In this section, we outline an approach to developing models using *Reo*, and apply it in a case study, the biological system *Galactose Utilization in Yeast* [18]. In particular, we focus on the process of metabolizing *Galatose* to *Glucose-6-P* (Figure 2). The system's behaviour can be described as following [18]:

> Yeast metabolizes galactose through a series of steps involving the *GAL2* transporter and enzymes produced by *GAL1*, *7*, *10*, and *5*. These genes are transcriptionally regulated by a mechansim consisting primarily of *GAL4*, *80*, and *3*. *GAL6* produces another regulatory factor thought to repress the GAL enzymes in a manner similar to *GAL80*.

In order to produce a model, we need to determine the level of abstraction at which the model will work. We require a definition:

Definition 4.1 *The finite set of* biological entities *(or just* entity*) that play a role in a biological system (or process) is called its* Domain.

Entities can be any cell organelle (e.g., mitochondrion, ribossome), suborganelle constituent (e.g., gene), cell component (e.g., membrane), chemical, etc.

The first step then is to determine the domain. For the present case study, the domain consists of *seven chemical substances* (*Gal-out*, *Gal-in*, *Gal-1-P*, *UDP-Glu*, *UDP-Gal*, *Glu-1-P*, *Glu-6-P*), *five chemical reactions* (*Gal-out*→*Gal-in* (actually a membrane crossing), *Gal-in*→*Gal-1-P*, *Gal-1-P*+*UDP-Glu*→*Glu-1-P*+*UDP-Gal*, *UDP-Gal*→*UDP-Glu*, *Glu-1-P*→*Glu-6-P*) and *nine genes* (*Gal1*, *Gal2*, *Gal3*, *Gal4*, *Gal5*, *Gal6*, *Gal7*, *Gal10*, *Gal80*).

The next step in the modelling process is to instantiate each domain element with a suitable *Reo* connector. This may be as simple as selecting the appropriate connector from a library, or it may require new models to be developed.

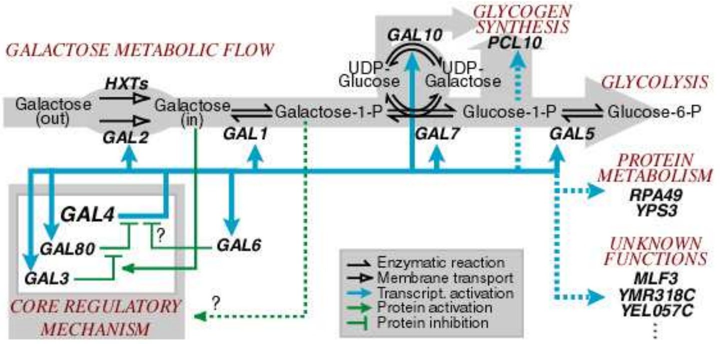
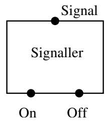


Figure 2: The Galactose System. The top gray section depicts the chemical reactions involved. The bottom part depicts the gene regulatory network coordinating the reactions. From [18].

Genes as Signallers The first step in modelling the behaviour of a gene as a *Reo* connector is to extract the relevant characteristics and behavioural properties that genes express in the biological system. Once this behaviour has been determined, up to the chosen level of abstraction, a *Reo* connector can be constructed to match that behaviour.

Genes are segments of DNA within chromosomes which cells transcribe into RNAs and translate, at least in part, into proteins. Genes affect behaviour within the cell through the proteins which are produced. Some of the resulting proteins will become enzymes which may regulate gene behaviour—though for modelling purposes, this indirection is a secondary issue.

We can abstract this behaviour using a *Reo* connector which we call a *signaller*. A signaller, see below, can alternate between two different states, *On* or *Off*. When the signaller is in the *On* state, an output can be emitted on the **Signal** channel end. In the *Off* state, no output is emitted on the **Signal** channel end. The state of a signaller is dictated by inputs on the channel ends labelled **On** and **Off**. An input on **On** switches the signaller to the *On* state, if the state of the signaller is *Off*, or is otherwise ignored. Similarly, an input on **Off** switches the signaller to the state *Off*, or is ignored. A signaller, with intial state *Off*, is represented using the following mnemonic:



If the connection to either the **On** or **Off** end of a signaller is not known, we omit the end from the diagram. A full *Reo* circuit implementing the signaller is given in an appendix.

The nine genes can thus be modelled by *signaller* connectors.[1]

Chemicals The five chemical substances that either act as reactants or products of reaction are modelled as *Reo* nodes. More precisely, we introduce a node into a connector and associate "data flowing through the node" with "the presence of the chemical." As we outline in Section 5, such nodes can be used to make observations about these entities within the system.

Modelling Chemical Reactions A chemical reaction can either be the synthesis of a substance from two or more individual molecules coming together, or the decomposition of a molecule into smaller molecules. A reaction may have multiple reactants and multiple products, and may require the presence of an enzyme.

[1] An **enzyme** is a protein whose presence can activate or suppress chemical reactions without being itself consumed. It can be modelled using a *signaller*. For simplicity, we have folded enzyme behaviour into that of the gene which produces it.

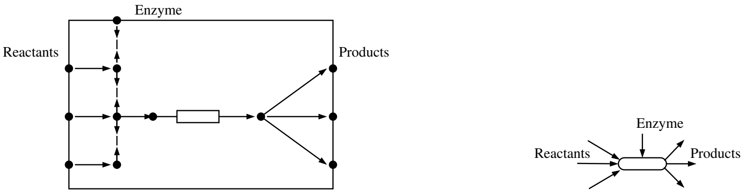


Figure 3: (a) Connector for Complex Reactions with Enzyme and (b) its Mnemonic

A chemical reaction can be modelled simply as a number of input channel ends (one for each reactant), a number of output channel ends (one for each product) and an input for the enzyme signal required to enable the reaction. A signal on one of the reactants channel ends signals the availablitiy of a reactant. The reaction proceeds whenever a signal is available on all reactants channel ends and the enzyme channel end. This simultaneous availability is imposed using a *SynchDrain* channel. A *FIFO1* channel is placed in the circuit after the reaction has occured to model the delay present in a chemical reaction.

Four chemical reactions (*Gal-out*→*Gal-in*, *Gal-in*→*Gal-1-P*, *UDP-Gal*→*UDP-Glu*, *Glu-1-P*→ *Glu-6-P*) need one reactant and output one product. The remaining reaction (*Gal-1-P*+*UDP-Glu*→*Glu-1-P*+*UDP-Gal*) needs two reactants and outputs two products. These are both modelled using the connector in Figure 3.

Now that we have a domain, D, and the behaviour of the entities which constitute it, we can start to build a *Reo* model. To do so, we must ask a second question: how de we construct models of complex systems which comprise multiple biological entities? To answer this question, we first define what a system is.

Definition 4.2 *A* system *is a tuple $S = (D, \rho, B)$, where D is a domain,* entity relation *ρ is a subset of $(D \times D)$, and B is a subset of D called the* boundary entities*. The set of* internal entities *is defined to be the set $I = (D - B)$.*

The entity relation ρ is a relation over $D \times D$ with the interpretation that for all pair of entities (i_1, i_2), the entity i_1 regulates the behaviour of entity i_2. This also includes basic connections between chemical reactions where the product of one is a reactant of another. A boundary entity in S is an entity whose behaviour does not depend on and which is not regulated by any other entities in S. For our example, the boundary entities include two elements *Gal-out* and *Glu-6-P*. An internal entity typically depends on or is regulated by the activity of other entities in S.

The entity relation ρ identifies the interactions between various entities in a system. Each of these interactions must be modelled in *Reo*. Usually, this consists simply of composing the connectors involved in the right manner. The relation ρ between the entities of D can be determined from Figure 2.

Composition: Activation, Suppression The action of a gene is to provide the proteins vital to a cell. An enzyme activates or suppresses chemical reactions, or, in some cases, plays a role in regulating (activating or suppressing) gene activity. The behavior of activation and suppression can be modelled by composing the signaller connector which models the gene or enzyme with the circuit modelling the entity which it regulates. To model activation (or suppresion) by one signaller on some target signaller, the **Signal** channel end of the first signaller is connected to the **On** (or **Off**) channel end of the target signaller, as depicted in Figure 4(a). Multiple sources of activation or suppression present in a system can simply be modelled by merging signals from various sources via a *Reo* node.

Composition: Self-regulation Self-regulation means that a biological entity directly or indirectly is regulated by itself. Products of the biological process influence, either positively or

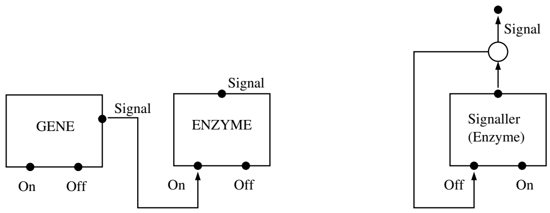


Figure 4: (a) A Gene activates an Enzyme. (b) Self-Regulation using Feedback

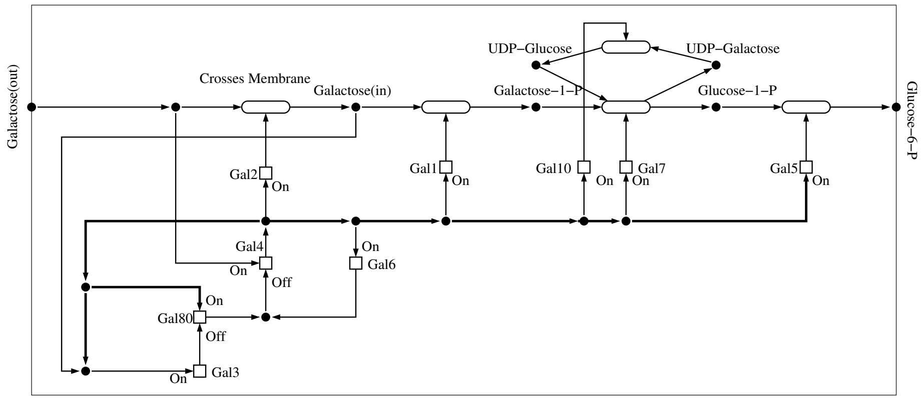


Figure 5: Galactose Metabolism modelled using *Reo*

negatively, the process which created those products. To model self-regulation, we use feedback in a *Reo* circuit. A simple example is presented in Figure 4(b). This figure models a gene whose products (the signal) cause it to turn itself off. For more complex situations where the regulation depends upon time, reaction rates, or concentration, may require timed *Reo* circuits to be more accurately modelled [3].

Composition: Reaction Pathways, Reversible Reactions Composition permits the building of chains of chemical reactions, in which the product of one reaction is the reactant of another. Reaction chains are the fundamental elements of biological pathways, as we discussed in Section 2. Reversible reactions can also be modelled by suitably composing the forward and reverse reactions together. The reactants involved can either become reactants in the reaction going in the other direction, or be used by another reaction in the pathway.

The Composed System We can now compose all of our entities together. Figure 2 informs whether the regulation from a gene is positive or negative. This information is used to determine how to connect genes together, as outlined above. Finally, the appropriate gene action is connected to each chemical reaction, and these reactions are chained together. The result is the circuit in Figure 5.

5 Reasoning About *Reo* Models

Having constructed a *Reo* model, we need techniques for reasoning about its behaviour. This will include checking whether the model fits the experimental data, and determining more general properties, such as the presence of stable states or cycles of states [16] within the model, and understanding the relationship (both causal and cooperation) between various entities. Our model permits reasoning about the relationship between the input and output behaviour of boundary entities. Various behaviours can be determined by *perturbing the boundary*—by setting up different boundary conditions, the behaviour at the remainder of the boundary can be determined. The idea can easily be extended to model internal behaviour also, simply by exposing the internal entities on the boundary.

Reo semantics have been defined in terms of two different formalisms: (timed) constraint automata and abstract behaviour types (not used here) [1, 4]. In addition, a number of modal logics and model checking algorithms have been developed for specifying and checking properties of *Reo* circuits. This machinery can be readily applied to biological models.

Constraint automata A constraint automaton [4] is an automaton which describes the sequence of possible observations on the boundary nodes of a *Reo* connector. Such an automaton is defined over a set of names $\mathcal{N} = \{A_1, \ldots, A_n\}$, which correspond to the input/output nodes. We present only "data-insensitive" automata. A constraint automaton is a tuple $\mathcal{A} = (\mathcal{Q}, \mathcal{N}, \longrightarrow, \mathcal{Q}_0)$, where $\mathcal{Q}$ is a finite set of states; $\mathcal{N}$ is a finite set of names; $\longrightarrow$ is a finite subset of $\mathcal{Q} \times 2^{\mathcal{N}} \times \mathcal{Q}$, called the transition relation of $\mathcal{A}$, written $q \xrightarrow{N} p$, with the constraint that $N \neq \emptyset$; and $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is the set of initial states.

An automaton starts in an initial state $q_0 \in Q_0$. If the current state is q, then the automaton waits until signals occur on some of the nodes $A_i \in N$. If signals are observed at nodes A_1 and A_2, for example, and at no other nodes at the same time, then the automaton may take a transition $q \xrightarrow{\{A_1,A_2\}} p$. A *run* of an automaton is a sequence of non-empty subsets of $\mathcal{N}$, $(N_0, N_1, N_2, \ldots)$ which corresponds to a series of signals occuring simultaneously at the nodes of the *Reo* connector which the automaton models.

There are two important constructions on constraint automaton. The first construction, *product*, denoted $\mathcal{A} \bowtie \mathcal{B}$, takes two constraint automata and produces an automaton which is the result of "joining" the actions on names shared between the two automata. This captures the composition of *Reo* circuits, though the details are too involved to go into here. The semantics of this operation is similar to the join operation in relational databases.

The second construction, *hiding*, denoted $\exists[C]\mathcal{A}$, takes a name, C, and a constraint automaton, $\mathcal{A}$, and produces an automaton where all behaviour at node C is internalised. Observations about C are no longer possible. Paths involving just label $\{C\}$ are compressed to avoid empty paths in the resulting automaton. The resulting automaton has the same behaviour on the other nodes.

Projection to Subsystem of Interest The first step when reasoning about a *Reo* circuit is to construct a constraint automaton for it by constructing the product of the automata which model its channels and nodes. Assume now that the resulting automaton, $\mathcal{A}$, has nodes $\{A, B, C, D, E\}$, where, $\{A, B, C\}$ are on the boundary and $\{D, E\}$ are internal. At this stage, the constraint automaton contains all the information which occurs on every node. Normally, hiding would be used to produce an automaton which models the behaviour on the boundary of the connector, i.e., $\exists[D, E]\mathcal{A}$. However, the fact that the automaton resulting from a product alone keeps the behaviour of the internal nodes exposed means that we can use the automaton to reason about the internal behaviour, simply by not hiding them. Thus if a different set of nodes is of interest to the reasoner, an different automaton can be produced containing only those nodes. For example, the automaton $\exists[A, B, E]\mathcal{A}$ can be used to understand the relationship between boundary node C and internal node D.

Causality and Cooperation Analysis A temporal logic called Time Scheduled-Data-Stream Logic (TSDSL) has been defined, along with model checking algorithms, for the timed version of constraint automata [3]. Scheduled-stream logic (SSL) is the obvious simplification of TSDSL, removing references to time and data. Its formulae over a node set $\mathcal{N}$ are given by the grammar:

$$\begin{array}{rcl}\phi & ::= & \mathsf{true} \mid \phi_1 \wedge \phi_2 \mid \neg\psi \mid \langle\langle\alpha\rangle\rangle\psi \mid \psi_1 \,\mathsf{U}\, \psi_2 \\ \alpha & ::= & N \mid \alpha_1 \vee \alpha_2 \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 ; \alpha_2 \mid \alpha^*\end{array}$$

N is a nonempty subset of $\mathcal{N}$. α is a so-called schedule expression, giving a regular expression for finite sequences of subsets of $\mathcal{N}$, corresponding to a sequence of "events". Thus $\langle\!\langle \alpha \rangle\!\rangle \psi$ states that each run has a prefix in the set described by α, with the suffix of the run satisfying ψ. Lastly, $\psi_1 \mathsf{U} \psi_2$ is the until modality from Linear Temporal Logic [9], stating that ψ_1 must hold up until the particular point which ψ_2 holds. This logic can express properties of runs of an automaton.

Biologists wish to pose a number of questions about a model. These often take the general form, *what happens if I press this button*? More concretely, they ask: what is the cause of gene *Gal80* being on? Does *Gal80* affect product *Galactose-1-P*? Do *Gal10* and *Gal7* cooperate to produce *Glucose-1-P*? Is this cooperation necessary? Can other entities produce *Glucose-1-P*? Can *Glucose-1-P* be produced without *Gal10*? Given a constraint automata at the appropriate level of abstraction, with nodes hidden to avoid "noise", such questions can be expressed in SSL as assertions over the visible node set. The validity of the assertions can be determined using model checking. Now if, $[\![\alpha]\!]\psi \equiv \neg\langle\!\langle \alpha \rangle\!\rangle \neg\psi$, $\mathsf{false} \equiv \neg\mathsf{true}$, and $\langle\!\langle \neg Gal4 \rangle\!\rangle \psi \equiv \langle\!\langle N_1 \vee N_2 \vee \cdots \rangle\!\rangle \psi$, where $N_1 \vee N_2 \vee \cdots$ is the set of nodes apart from *Gal4*, we can readily make assertions such as: $\langle\!\langle Gal4 \rangle\!\rangle \mathsf{true}$ – *Gal4* must signal; $[\![Gal4]\!]\mathsf{false}$ – *Gal4* cannot signal; $[\![Gal80]\!]([\![Gal4]\!]\mathsf{false}\, \mathsf{U} \langle\!\langle \textit{Gal-out} \rangle\!\rangle \mathsf{true})$ – after *Gal80* has signalled, it is not possible for *Gal4* to signal until *Galactose* has been detected on the outside of the cell; $[\![Gal4]\!]\langle\!\langle Gal80 \rangle\!\rangle \mathsf{true}$ — *Gal4* turns *Gal80* on; $[\![Gal80]\!]\mathsf{false} \Rightarrow \langle\!\langle \neg Gal4 \rangle\!\rangle [\![Gal80]\!]\mathsf{false}$ — only *Gal4* turns *Gal80* on; $[\![\textit{Gal-in} \wedge Gal1]\!]\langle\!\langle \textit{Gal-1-P} \rangle\!\rangle \mathsf{true}$ — *Galactose* detected inside the cell and the signalling of *Gal1* are required to produce *Galactose-1-P*; and $[\![Gal6 \vee Gal80]\!][\![Gal4]\!]\mathsf{false}$ — the presence of either *Gal6* or *Gal80* can stop *Gal4* from signalling. Many other questions regarding the states of a system and the pathways therein can be posed [6].

Generating Experiments from Models An important reason for having accurate models is to understand the cause and effects of disease and the effects of drugs used to treat them. Models of biological systems will typically, at least initially, be insufficiently accurate for the desired purpose. In order to refine a model, a biologist will need hypotheses to test experimentally. To generate experiments to test the behaviour of disease and drug treatment, in the case where they act at a genetic level, the behaviour of the faulty gene or drug needs to be modelled.

The first approach is to extract the consequent behaviour from an existing connector. This is done by first characterising the behaviour of the faulty gene or drug as streams of observations at some visible node in the model. The constraint automaton can then be reduced to one which contains only the streams of observations selected by the biologist, and can then be analysed using the techniques above to generate new hypotheses. This approach, however, is only capable of producing behaviour which is already present in the model.

The second approach is to modify the *Reo* model by replacing the connector corresponding to the entity of interest by a new connector which captures the behaviour of the gene malfunction or the action of the drug introduced into the system. The result is a new *Reo* model with the desired behaviour built-in.

In both cases, biologists can study a model by subjecting it to the reasoning techniques described above, and use it to generate new hypotheses. An advantage of using the first technique, in the event that the experimental data matches the generated hypothesis, is that the behaviour is already included within the model, thus no change to the model need be made. Otherwise, if either the experimental data and the model do not match, or the second technique is used, appropriate refinement of the model will be required.

Unsoundness, Incompleteness and Refinement We see modelling as an iterative process whereby a model is refined, based on experimental data produced by a biologist, to produce a more accurate model. A model can be inaccurate in two ways: it can be unsound or incomplete. Soundness is the requirement that any behaviour which the model exhibits is also exhibited by the biological system. Completeness is the other way around, namely, that any behaviour observed in the real system can be reproduced in the model. (These definitions are only relative to the level of abstraction at which the modeller is operating.) Unsound and incomplete behaviour can be discovered by both probing the model, and by testing hypotheses experimentally in the biological system. When the predictions of the model fail to match the results of the experiment, the model needs to be refined, that is, to be adapted to preserve all previous sound behaviour and also capture (or exclude) the new experimental observations. The refinement technique is related to software evolution in the presence of changing specifications. We do not yet have a clear approach to using the "evidence" of unsoundness or incompleteness in one model to produce a model which is more sound and complete.

Simulation The last technique available to biologists is simulation. We are at present developing a tool for simulating *Reo* circuits which is applicable here.

6 Related Work

A number of researchers have expressed the need for larger scale models of biological systems [15, 18, 24]. A number of different modelling approaches have emerged to meet this need. Many depart dramatically from the traditional, small-scale approaches based on differential equations and Monte Carlo simulation. In the remaining space, we can only give a taste of what these new approaches are.

A number of biological modeling languages are based on process calculi. The pi-calculus has been used for modeling general reaction pathways [27], variations of the ambient calculus have modeled systems involving membrane interactions [26, 5, 12], and special purpose calculi have been used to model protein-protein interactions [11]. In some cases, a stochastic element is added, making the models surprisingly accurate [25]. In general, this approach displays a lot of flexibility, tools for reasoning about the models exist or can be readily adapted from existing tools [6], and process calculi can be simulated.

Boolean Networks are one of the first models of gene regulatory network behaviour [16]. These consist of a number of boolean states, corresponding to whether genes are active or not, and a transition function which *lock-step* determines the next state. This process is iterated, and the network can be analysed for stable cycles of states, called *attractors*. These models have the advantages of simplicity and that they are readily simulated, but they are limited by their discrete nature and their lock-step evaluation. Some of the limitations have been overcome, by introducing probability to the model [29]. More advanced network models break away from the lock-step evaluation of Boolean networks, by incorporating continuous aspects into models, producing hybrid models which have both discrete and continuous factors. Such models include the circuits of McAdams and Shapiro [23] and models based on hybrid petri nets [22], and hybrid automata [8]. These models capture direct information flow within a biological system, and computational techniques often exist to determine indirect relationships and effects. Although more flexible than Boolean networks, computational effort is required to analyse these models.

Coordination is a buzzword commonly used when talking about (systems) biology [31], although, to our knowledge, this paper is the first attempt at applying a coordination model in this area. We expect to offer significant advantages because coordination, prevalent in Biology, is foremost in our model. Compared with approaches based on other conconcurrent formalisms, our model often works at a higher level of abstraction, because the coordination (synchronisation among multiple entities, for example) needs to be programmed in a process calculus model, whereas in *Reo* it comes for free, either directly in primitive channels or in more complex examples via composition. Indeed, one of our colleagues has demonstrated that it is trivial to embed an Elementary Petri-net into *Reo*, whereas the reverse embedding was much more difficult [17]. Although this embedding not been developed for hybrid Petri nets, or other circuit-based models, its existence indicates that *Reo* can often express more with less.

7 Conclusions and Future Work

Fontana and Buss highlighted the need for an algebraic semantics of behaviour suitable for biological systems [15]. We have proposed that Reo, and the coinductive semantics that underlies it, could provide a suitable framework. We believe this to be the case, because Reo is open-ended, enabling channels with arbitrary behaviour to be added, and it permits the compositional description of biological processes by providing a set of connectors modelling the behaviour present in biological systems. Furthermore, a number of formal tools have been developed to specify and reason about connector behaviour.

The following brief roadmap will guide our future work. We have a number of goals: developing a methodology both for building toolkits for modelling classes of biological systems and for their refinement; extensively studying and validating the approach underlying this methodology *in conjunction with biologists.* It would be great if biologists are willing to work with use to develop more accurate models, so that we can provide better tools. In particular, we would like to further apply the formal reasoning tools for *Reo* with time constraints, so that we can provide models which include metric data such as rates, concentration, delays [3].

References

[1] Farhad Arbab. Abstract behavior types: A foundation model for components and their composition. In *[13]*, pages 33–70, 2003.

[2] Farhad Arbab. Reo: A channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, June 2004.

[3] Farhad Arbab, Christel Baier, Frank de Boer, and Jan Rutten. Modeling and temporal logics for timed component connectors. In *IEEE International Conference on Software Engineering and Formal Methods (SEFM '04)*, Beijing, China, September 2004. Submitted.

[4] Farhad Arbab, Christel Baier, Jan J. M. M. Rutten, and Marjan Sirjani. Modeling component connectors in Reo by constraint automata. In *International Workshop on Foundations of Coordination Languages and Software Architectures (FOCSLA)*, ENTCS, Marseille, France, September 2003. Elsevier Science.

[5] Luca Cardelli. Brane calculi: Interaction of biological membranes. In *Computational Methods in Systems Biology (CMSB'04)*, LNCS, Paris, France, May 2004. To appear.

[6] Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, François Fages, and Vincent Schächter. Modeling and querying biomolecular interaction networks. *Theoretical Computer Science*, 325(1):25–44, September 2004.

[7] Bor-Yuh Evan Chang and Manu Sridharan. PML: Towards a high-level formal language for biological systems. In *BIO-CONCUR'03*, Electronic Notes in Theoretical Computer Science, Marseille, France, 2003.

[8] Kwang-Hyun Cho and Karl Henrik Johansson Olaf Wolkenhauer. A hybrid systems framework for cellular processes, 2004. submitted for publication.

[9] Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled. *Model Checking.* MIT Press, 1999.

[10] Vincent Danos and Jean Krivine. Formal molecular biology done in CCS-R. In *BIO-CONCUR'03*, Electronic Notes in Theoretical Computer Science, Marseille, France, 2003.

[11] Vincent Danos and Cosmino Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, September 2004.

[12] Vincent Danos and Sylvain Pradalier. Projective brane calculus. In *Computational Methods in Systems Biology (CMSB'04)*, LNCS, Paris, France, May 2004.

[13] F.S. de Boer, M.M. Bonsangue, S. Graf, and W.-P. de Roever, editors. *Formal Methods for Components and Objects*, volume 2852 of *LNCS*. Springer, 2003.

[14] USA Department of Energy. `http://www.doegenomestolife.org/`, June 2004.

[15] Walter Fontana and Leo W. Buss. The barrier of objects: From dynamical systems to bounded organizations. In J. Casti and A. Karlqvist, editors, *Boundaries and Barriers*, pages 56–116. Addison-Wesley, 1996.

[16] K. Glass and S. A. Kauffman. The logical analysis of continuous, nonlinear biochemical control networks. *J. Theoretical Biology*, 44:103–129, 1974.

[17] Juan Vincent Guillen-Scholten. A first translation from Reo to Petri nets and vice-versa, 2004. work in progress.

[18] Trey Ideker, Timothy Galitski, and Leroy Hood. A new approach to decoding life: systems biology. *Annual Review of Genomics and Human Genetics*, 2:343–72, September 2001.

[19] Hiroaki Kitano. A graphical notation for biochemical networks. *BIOSILICO*, 1(5):169–176, November 2003.

[20] Céline Kuttler, Joachim Niehren, and Ralf Blossey. Gene regulation in the pi calculus: Simulating cooperativity at the lambda switch. In *Workshop on Concurrent Models in Molecular Biology*, ENTCS. Elsevier, 2004. BIO-CONCUR workshop proceedings.

[21] T. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994.

[22] Hiroshi Matsuno, Atsushi Doi, Masao Nagasaki, and Satoru Miyano. Hybrid Petri net representation of gene regulatory network. In *Proc. Pacific Symposium on Biocomputing 5*, pages 341–352, 2000.

[23] Harley McAdams and Lucy Shapiro. Circuit simulation of genetic networks. *Science*, 269:650–6, 1995.

[24] Amitai Regev and Ehud Shapiro. Cells as computation. *Nature*, 419(26):343, September 2002.

[25] Aviv Regev. Representation and simulation of molecular pathways in the stochastic pi-calculus. In *2nd Workshop on Computation of Biochemical Pathways and Genetic Networks*, 2001.

[26] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. BioAmbients: An abstraction for biological compartments. *Theoretical Computer Science, Special Issue on Computational Methods in Systems Biology*, 325(1):141–167, September 2004.

[27] Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In *Pacific Symposium on Biocomputing*, volume 6, pages 459–470, 2001.

[28] M. A. Savageau. Rules for the evolution of gene circuitry. In *Pacific Symposium of Biocomputing*, pages 54–65, 1998.

[29] Ilya Schmulevich, Edward R. Dougherty, and Wei Zhang. From boolean to probabilistic boolean networks as models of genetic regulatory networs. *Proceedings of the IEEE*, 90(11), November 2002.

[30] Lubert Stryer. *Biochemistry.* Freeman, 1988.

[31] Olaf Wolkenhauer, Bijoy Ghosh, and Kwang-Hyun Cho. Control & coordination in biochemical networks (editorial notes). *IEEE CSM Special Issue on Systems Biology*, August 2004.

A Constructing a Signaller

We demonstrate that a signaller can be constructed using the primitive channels described in this paper, giving testament to the expressiveness of *Reo*. An alternative is to supply a signaller as a primitive with its behaviour defined directly using constraint automata.

Sequencer The sequencer connector [2] (for 2 elements) is depicted in Figure 6. This connector is used to alternate the behaviour at nodes **A** and **B**, given by the regular expression: $(\mathbf{AB})^*$. The implementation of the sequencer consists of a loop of *FIFO1* buffers, of which all but one are empty, which implements a token ring, enabling **A** and **B** to input data alternately. The token is used to indicate which synchronous channels may pass data. After **A** inputs data, the token moves to the other *FIFO1* buffer, enabling the **B** to input data. After **B** inputs, the sequencer returns to its initial state, accepting **A**.

SwitchConverter The token passing loop which forms a part of the sequencer can readily be adapted to act as the core of a switch that enables or inhibits the flow of data, toggled by alternating values from a third channel end (see the PreValve circuit, below). A more natural switch consists of two separate channel ends, one corresponding to *On*, the other corresponding to *Off* (repeated inputs in **On** or **Off** channel ends have no effect). The SwitchConverter connector (Figure 6) converts the latter kind of switch into the former (conversion in the other direction is also possible).

PreValve The PreValve connector is depicted in Figure 6. Its initial state can be either *On* or *Off*. When the connector is in the *On* state, data can be inputed continually through the **Flow** channel end. In the *Off* state, not data can be inputed. Data on the **Toggle** channel end toggles the connector between its *On* and *Off* states. Note that, this circuit is the core of the Valve connector in [1].

Signaller The detailed implementation of this circuit is given in Figure 8. For its description, see Section 4

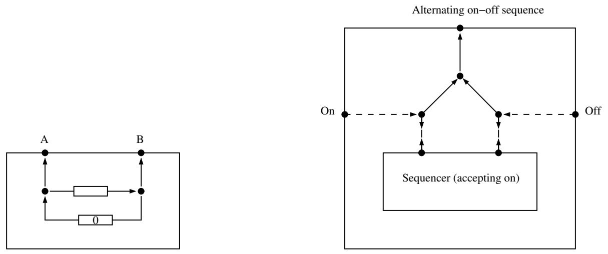


Figure 6: (a) Sequencer connector (**A** is enabled) (b) Switch Converter connector

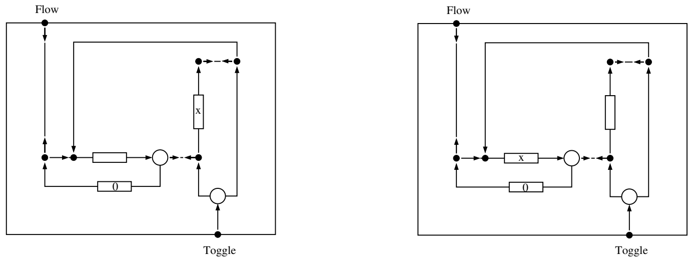


Figure 7: PreValve connector – State On and State Off

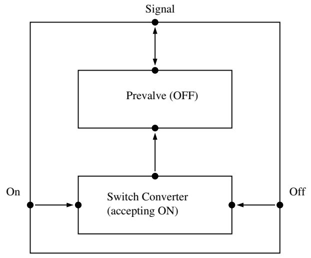


Figure 8: Signaller connector (Initial state is *Off*)