Technical Report: Defining a textual representation for SBGN Process Diagrams and translating it to Bio-PEPA for quantitative analysis of the MAPK signal transduction cascade

Laurence Loewe[1], Stuart L. Moodie[1] and Jane Hillston[2,1]

[1] Centre for System Biology at Edinburgh,
The University of Edinburgh, Edinburgh EH9 3JU - Scotland
Laurence.Loewe@ed.ac.uk, Stuart.Moodie@ed.ac.uk
[2] Laboratory for Foundations of Computer Science,
The University of Edinburgh, Edinburgh EH8 9AB, Scotland
Jane.Hillston@ed.ac.uk

Technical Report EDI-INF-RR-1334
School of Informatics, University of Edinburgh
http://www.inf.ed.ac.uk/publications/report/1334.html
24 July 2009

Abstract. Biologists have used visual representations of biochemical networks for a long time to gain a quick overview of important structural properties. Recently SBGN, the Systems Biology Graphical Notation, has been developed to standardise the way in which such graphical maps are drawn in order to facilitate the exchange of information. However, SBGN at this stage includes neither an explicit textual representation nor a defined way of including the quantitative details that are needed for computing quantitative properties of the underlying network. Here we define a possible mapping of SBGN Process Diagrams to a textual representation to facilitate the automatic parsing of SBGN-PD maps. We then define attributes for SBGN-PD glyphs to allow the specification of the quantitative information that is needed for quantitative analyses. We implemented SBGNtext2BioPEPA, a tool that demonstrates how such quantitative details can be used to automatically generate working Bio-PEPA code from our textual representation. Bio-PEPA is a process algebra that was designed for implementing quantitative models of concurrent biochemical reaction systems. We use this approach to compute the expected delay between input and output in deterministic and stochastic simulations of the MAPK signal transduction cascade. The scheme developed here is general and can be easily adapted to other output formalisms.

1 Introduction

Biologists are constantly searching for strategies that help them to understand the complexity of life. Navigating the functional molecular interactions within cells has proven to be an increasing challenge since molecular biological research is filling databases with detailed knowledge about the molecular mechanics of life. A wide variety of schemes has been developed to represent such knowledge, ranging from textual representations that resemble chemical reactions (e.g. Dizzy [1]) or reaction rules (e.g. BioNetGen, kappa [2, 3]) through XML-based standards like SBML [4] to graphical notations (e.g. [5–10]). Graphical maps of biochemical reaction networks are proving to be powerful tools for facilitating an overview of the interactions of particular molecules. Recently the Systems Biological Graphical Notation (SBGN) has emerged as a standard for drawing such reaction diagrams [10]. The objective is to provide molecular systems biologists with an intuitive description of the system by generating consistent maps across different editing tools (e.g. CellDesigner [11], Cytoscape [12], Edinburgh Pathway Editor [13], JDesigner [14]).

If quantitative analysis is to be carried out, further information must be captured that is not needed for graphical overviews. For example, equations and parameters have to be defined and the names of biochemical species (molecular entities) and reactions (processes) must be easy for a computer to parse. Implementing such models in code for quantitative analyses often requires considerable effort from modelling specialists. Once the code is properly implemented, quantitative analyses can be quick and flexible, especially if a modelling environment is as versatile as the Bio-PEPA tools [15, 16] that support multiple algorithms for multiple analysis methods like stochastic simulations, ordinary differential equations and continuous time Markov Chain analyses.

In this work we aim to build a bridge between these two worlds. Currently, SBGN is a standard without a formal textual representation. To facilitate automated translation to quantitative analysis code, we define a possible textual representation for SBGN Process Diagrams (SBGN-PD) that can be parsed by following the rules of an EBNF grammar (Section 2). This textual representation needs to include quantitative information to minimise tedious manual interactions with downstream code. To allow the capture of such quantitative information in graphical editors we define a minimal set of attributes for the various SBGN-PD glyphs (Section 3). We then show how SBGN-PD glyphs can be mapped to a quantitative analysis framework, using the Bio-PEPA modelling environment [16] as an example (Section 4). However, our work is by no means tied to Bio-PEPA and we discuss the various internal mechanisms and data structures that are needed for the translation process to facilitate the adaptation to other modelling platforms (Section 5). As an example of how the translation process works, we apply our new translation tool "SBGNtext2BioPEPA" to a simple model of the MAPK signalling cascade [17], which we automatically translate into Bio-PEPA, where we analyse the stochastic behaviour of the time needed for the cascade to be switched from "off" to "on". We end by reviewing related work and providing some perspectives for further developments.

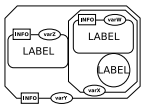
SBGN-PD glyph	EPNType	class type	comment
	`Unspecified`	material	EPN; use if specifics unknown
	`SimpleChemical`	material	EPN
	`Macromolecule`	material	EPN
	`NucleicAcidFeature`	material	EPN
	-	material	EPN; specified by `Cardinality`
	`Complex`	container	EPN; arbitrary nesting allowed
	`Source`	conceptual	external source of molecules
	`Sink`	conceptual	removal from the system
	`PerturbingAgent`	conceptual	external influence on a reaction

Table 1. Mapping SBGN-PD glyphs for EPNs and conceptual entities to SBGNtext.

2 A textual representation of SBGN

Here we present a potential 1:1 textual representation for the current definition of SBGN Process Diagram Level 1 (Release 1.0) [10]. We refer to this visual standard as "SBGN-PD" and to our textual representation as "SBGNtext" if needed.

Core overview. SBGN-PD was developed to represent biochemical reaction networks. Therefore reactions and the biochemical species that take part in them feature prominently in SBGN-PD, where reactions and other processes are represented by *"process nodes"* and biochemical species and other entities by *"entity pool nodes"*. SBGN-PD uses *"arcs"* to indicate which role a particular entity pool node is playing in a particular process. Here we focus on the many different types of entity pool nodes, processes and arcs that are used in SBGN-PD to specify information about a reaction concisely. Additional features like compartments, submaps, logic gates and conceptual entity nodes are also discussed briefly (see [18] for source code and further developments).

2.1 Entity pool nodes (EPN) and conceptual (non-EPN) nodes

An `EntityPoolNode` in SBGNtext represents a corresponding pool of functionally equivalent entities, like molecules in the system, for example. These molecules may be reactants, products, reaction rate modifiers or any combination of these three. An EPN can be as simple as a `SimpleChemical` or as complex as an arbitrarily nested `ComplexNode`. Table 1 provides an overview of the various nodes that can be connected to processes, including various conceptual types that are treated similarly to EPNs in

SBGNtext keyword	contribution to EntityPoolNodeID
EPNType	encoded as controlled list of abbreviations
EPNName	SBGN-PD name made unique after removing special chars
Cardinality	if multimer, give number of monomers; omit if '1' (monomer)
MaterialType	controlled abbreviations can distinguish DNA, RNA, Proteins, etc.
ConceptualType	controlled abbreviations can distinguish roles (gene, mRNA, etc.)
EntityState	modifications as specified in SBGN-PD (e.g. P for phosphorylation)
Compartment	full CompartmentID of entity; (omit if default)
ParentComplex	ID; only needed for components of a complex EPN

Table 2. SBGN-PD features that contribute to the ID of an EPN if specified and different from the default. EntityInformation and CloneMarker related information do not contribute here. Where possible, ConceptualType and EntityState should make use of vocabulary controlled by the Systems Biology Ontology as specified by SBGN-PD.

SBGNtext but will not be discussed further. All EPNs share the following features, which are represented in the syntax of SBGNtext:

EntityPoolNodeID. The EntityPoolNodeID is used to unambiguously identify an EPN in SBGNtext and in the code produced for quantitative analysis. This ID is synthesised from the elements given in Table 2. Note that details about *clone markers* (CloneMarkerIsOn, CloneMarkerName) and arbitrary SBGN-PD *units of information* (EntityInformation) do not contribute to the uniqueness of an ID. Since EntityPoolNodeIDs play a prominent role in Bio-PEPA, they must be globally unique and as short as possible. Thus all default information is omitted from the IDs.

EPNName. This is the label of an EPN specified by the user on the screen. Names allow for special characters that enhance readability and cannot be used in an ID. As EPNName is always enclosed in double quotes, double quotes cannot be used in labels, but single quotes, spaces and "=+-*/|\.,:;_()[]" are available to facilitate maximal freedom in labeling entities. Thus it is possible to specify chemical nomenclature names.

Compartment. All entities are located in some compartment, even if it is the default compartment. Compartments are referred to by their CompartmentID.

ParentComplex. All entities are either free within their compartment or they are bound together in a complex that forms a "super-EPN". Since complexes can be nested to arbitrary depth, each graphical entity element needs to know which encapsulating ParentComplex it belongs to. The corresponding parental ID is given, where free stands for no parental complex. This approach can result in very long IDs for elements of deeply nested complexes, but these IDs are never used in quantitative analyses, since reactions only involve free complexes.

EPNType. EPNs and non-EPN nodes belong to one of the types listed in Table 1. Multimers are not specified separately in SBGNtext despite different glyphs, since this information is encoded in the Cardinality already. EPNTypes can be either (i) simple material types or (ii) complex container types or (iii) conceptual types.

Simple material types and complexes are EPNs in the proper sense that they are nodes that represent a pool of entities. Conceptual entities such as `Source`, `Sink` and `PerturbingAgent` are not EPNs in the strict sense, as they do not represent pools. Here they are treated as EPNs because their functionality in the context of quantitative modelling is not different from EPNs and thus generating a separate category for them seems to be unnecessarily complicated. The conceptual entity `Observable` has an even more complicated position in SBGN-PD; while it is listed as a conceptual entity on p.9+p.67 of the SBGN-PD standard [10], technically it is defined as Controllable (p.43, [10]) and ModulatableNode (p.42, [10]), just like a Process. Thus we currently treat it as a Process of type `Observable` in SBGNtext and do not allow for an EPNType `Observable` .

`Cardinality` specifies the number of monomers in multimer EPNs. The default value 1 indicates that the entity is not a multimer (in this case the specification of cardinality can be omitted). SBGN currently defines multimers as non-covalent multimers (p.12 [10]), thus excluding polymers such as DNA, RNA and proteins, which are represented as "macromolecules" or "nucleic acid features". There is a grey area between these categories since, for example, different amino acid chains can form multimeric complexes with covalent disulfide bonds. Also, the cardinality of non-covalent multimers of type `SimpleChemical` can be difficult to determine, for example, if one considers a "multimer" of membrane molecules connected by hydrophobic bonds. These issues have to be dealt with at the SBGN-PD level, where the user specifies either an exact cardinality or the EPNType to be `Macromolecule` or where a simple EPN is deemed to be a reasonable representation of the pool of individual molecules, as their number constantly changes and their spatial positions are irrelevant. SBGNtext merely takes the result of the decision and stores the resulting integer number that denotes cardinality in an EPN with the help of the keyword "Cardinality". Since multimers of a different cardinality can play different roles, cardinality alone is enough to distinguish one entity from another. Thus cardinality is included in the `EntityPoolNodeID` if it is different from the default of 1.

`CloneMarker.` Each visual representation of an SBGN-PD entity is either unique in the diagram or has a clone marker to indicate that other identical copies exist. While this is irrelevant for the quantitative analysis of the network, it can accelerate the visual understanding of diagrams. SBGNtext stores each visual representation of an entity with a clone-marker switch and a clone-marker-label that is empty if the clone marker is of the "simple" type. This 1:1 representation of SBGN-PD facilitates extending SBGNtext for storing graphical information.

`EPNState.` Some entities can be chemically modified on different sites, affecting the functional role of the entity. To represent this, a list of *state variables* is used, where each entry describes a *variable* (site) that can adopt various *values* (covalently bound modifications). While SBGN encourages the use of the Systems Biology Ontology [35] vocabulary for describing the values of EPNStates (see p.8, [10]), it is also possible to freely define state variables. SBGNtext is indifferent to the corresponding semantics to facilitate maximal flexibility.

`EPNInformation` is a list of labels like `EPNState`. In SBGN-PD it can contain important fundamental or any other information about the entity. The two EPNInformation labels `MaterialType` and `ConceptualType` are reported separately from

other `EPNInformation` in SBGNtext, because these two are the only ones that can distinguish entities. Due to their fundamental importance, the content of these two labels is also included in the `EntityPoolNodeID` and SBGN encourages the use of the Systems Biology Ontology [35] vocabulary for describing their values. The purpose of additional `EPNInformation` in SBGNtext is to record any user defined information that does not distinguish entities.

SBGNtext does not conceptually distinguish between non-composed entities and entities that are complexes of other entities (the latter only carry a reference to their parent entity). This is despite huge differences in the complexity of the underlying structures that are required to represent complexes and simple types. This approach allows even the most complex entities to be described in detail, while the removal of such complexity remains simple for the purpose of generating quantitative analysis code. For formalisms like Bio-PEPA the internal structure of entities is irrelevant, as long as different entities can be distinguished. The latter purpose is served well by considering only the `EntityPoolNodeID` of all EPNs that are not part of a complex.

2.2 Process nodes

Each process is uniquely identified by its `ProcessNodeID`, which is manually entered by the user, to increase the readability of automatically derived Bio-PEPA code (see Section 3). Its main purpose is to provide a unique point of reference for use in the arcs. Processes can be of different `ProcessType` (Table 3), but if they are to be included in quantitative analyses, they will all have to provide the same details discussed in Section 3. In the context of the set of all arcs, each process defines a separate list of arcs for (i) reactants (input), (ii) products (output) and (iii) enzymes, catalysts or other modifiers of reaction rates (modulating). The processes do not store these lists of arcs; instead each arc points to a process. Compiling a classical chemical reaction requires walking over the list of all arcs, finding those that point to the given process and sorting them according to their type into one of the three lists above. Since each arc also points to an EPN, one only needs to follow the pointers to compile the list of EPNs involved in the reaction.

In addition, there is a list of LogicOperatorIDs; some of them ultimately point to processes as the final results of a cascade of logical operators that determines how the relative presence or absence of various EPNs will influence a given reaction.

Processes are not associated with any compartment in SBGN-PD to facilitate the representation of (simplified) transport processes that might cross several compartments or that involve entities that are in different compartments (e.g. complexes that are outside, cross a membrane and are also inside). Thus processes are also not associated with any compartment in SBGNtext.

Reversible processes. SBGN allows processes to be reversible if they are symmetrically modulated (p.28 [10]). In a reversible process the products of the forward process are consumed in the backward process, thus `Consumption` and `Production` arcs can no longer be as clearly separated as in unidirectional processes. To resolve this, SBGN-PD distinguishes the left-hand side from the right-hand side of a process and uses only arcs that look like `Production` arcs to indicate the double role

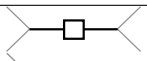
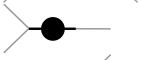
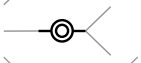
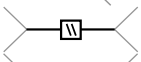
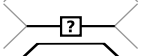
SBGN-PD glyph	ProcessType	meaning
	Process	normal known processes
	Association	special process that builds complexes
	Dissociation	special process that dissolves complexes
	Omitted	several known processes are abstracted
	Uncertain	existence of this process is not clear
	Observable	this process is easily observable

Table 3. Mapping SBGN-PD glyphs for Processes to SBGNtext.

(p.32 [10]). In SBGN-PD reversible process nodes are easy to recognise visually by the absence of `Consumption` arcs on both sides. To represent all such arcs either as `Consumption` arcs or as as `Production` arcs in SBGNtext would lose the information of which arc is on which side of the process node. Thus we define two new arc types that are only used for products and reactants in the context of reversible processes: `LeftHandSide` and `RightHandSide`. `LeftHandSide` arcs indicate that they are consumption arcs in the forward process (and production arcs in the backward process), where as `RightHandSide` arcs are the corresponding opposite. Thus the definition

$$\texttt{LeftHandSide} \longleftrightarrow \text{process} \longleftrightarrow \texttt{RightHandSide}$$

can be broken down as follows:

$$\begin{gathered}\texttt{LeftHandSide} \rightarrow \text{ForwardProcess} \rightarrow \texttt{RightHandSide} \equiv \\ \texttt{Consumption} \rightarrow \text{ForwardProcess} \rightarrow \texttt{Production} \\ \text{and} \\ \texttt{LeftHandSide} \leftarrow \text{BackwardProcess} \leftarrow \texttt{RightHandSide} \equiv \\ \texttt{Production} \leftarrow \text{BackwardProcess} \leftarrow \texttt{Consumption}\end{gathered}$$

To support reversible processes the visual editor needs to identify reversible processes and assign the corresponding arc types `LeftHandSide` and `RightHandSide` to the arcs (e.g. by testing for the property that reversible processes have *only* `Production` arcs and using graphical layout information for distinguishing left and right). In addition a forward and a backward propensity function needs to be stored to facilitate breaking up a bidirectional process into two unidirectional processes.

2.3 Arcs

While processes are at the heart of all action, arcs represent most of the biological knowledge in a SBGN-PD map. Each arc connects an EPN or non-EPN node with a

given EntityPoolNodeID and a process with a given ProcessNodeID, while providing additional information such as stoichiometry and the role of the node in the reaction. Ordinary arcs can be of type Consumption, Production, Modulation (most generic influence on reaction), Stimulation (catalysis or positive allosteric regulation), Catalysis (special case of stimulation, where activation energy is lowered), Inhibition (competitive or allosteric) or NecessaryStimulation (process is only possible, if stimulation is active, i.e. has surpassed some threshold). The glyphs are shown in Table 4, where their mapping to Bio-PEPA is discussed. Stoichiometry that deviates from the default value ('1') is denoted as a cardinality label for Consumption and Production arcs in SBGN-PD; similarly stochiometric relations for modulating entities might be indicated as cardinalities on modulating arcs.

To allow for the representation of reversible reactions, the Production arcs that characterise such reactions have to be replaced by LeftHandSide and RightHandSide arcs as explained above in section 2.2

All connecting arcs can be identified by an automatically generated ID that combines their type with a counter; this facilitates referencing them after replacing the ManualEquationArcIDs in the propensity functions in process nodes.

Logic arcs are not stored in the same format as ordinary arcs, since they are intimately tied to the logical operators and thus more intuitively stored there. Likewise, equivalence arcs are stored in submaps together with the tags they connect to. Logic arcs and equivalence arcs are discussed below.

2.4 Logical operators

Sometimes the biological function of a process can be simplified to a simple on/off logic that can be represented by Boolean operators. SBGN-PD supports this simplification by providing the logical operators of the types "AND", "OR" and "NOT", represented by a common dedicated structure in SBGNtext. Each logical operator has a unique LogicalOperatorID that is used for referencing in logical arcs.

Each LogicalOperator stores its type, a list of InputLogicArcs and the equivalent properties of one OutputLogicArc with an attached "modulation" arc that is usually of type NecessaryStimulation . Since each operator has only one result, there is only one output arc, which could either point to an input arc of another logic operator or be a necessary stimulation arc that affects a process node. Output is digital, where the high and low values that correspond to 1 and 0 are specified as properties in the graphical SBGN editor. Input is more differentiated, as it can consist of several signals that may assume any value that an EPN count can take. For "NOT" only one input is possible, everything else is an error. However, both "AND" and "OR are well defined for arbitrary inputs > 0. Thus SBGNtext allows an arbitrary number of inputs, where AND will expect all of them to be "on" to give an "on" result, while OR will require at least one of them to be "on". This definition allows graphical tools the possibility to collapse an apparently complex logical cascade with many operators into a single operator, hence reducing visual clutter.

2.5 Compartments

Compartments in SBGN are not conceptually nested, although they can be drawn as nested. They only contain entities, but not processes. These are all given a CompartmentID that is automatically derived from the visual label provided by the user. The CompartmentID is referenced in entities to indicate which compartment an entity belongs to. Compartments can contain UnitsOfInformation that further specify properties of the comparment (see discussion of EPNInformation above).

2.6 Submaps, tags and equivalence arcs

Submaps are a powerful tool for organizing complex biochemical network maps. They are ideal to hide the complex details of a series of biochemical reactions in high-level overviews, if a few entities represent the complete input and output of that submap (i.e. the submap has a comparatively simple interface).

For example, the MAPK signalling pathway depicted in Figure 2 could be abstracted in a submap. Thus a higher-level overview map merely needs to specify E1 as input and MAPK-PP as output, removing the need to specify any other details.

The namespaces for entities, compartments, processes, arcs and logical operators are the same between all maps at all levels. This facilitates the use of one big namespace in SBGNtext. Thus identifiers that are part of the inner workings of a submap will also be visible at the higher level map or in other submaps. However, syntactic and semantic visibility of names does not imply a recommendation to use such visibility, which could lead to "hacks", where interactions between different maps are specified without using the proper SBGN-PD glyphs. Thus, this visibility might not be desirable, as it could allow the introduction of unwanted side-effects of a submap. For example, a "local" entity in a submap (e.g. MAPKK in Figure 2) shares its name with some "external" entity in another submap, resulting in corresponding quantitative interactions in an automatically derived Bio-PEPA model, even though this interaction is not specified by corresponding tags in SBGN. Graphical SBGN editors can support the construction of maps by highlighting such potential side-effects and possibly enforcing their resolution.

These side-effects can be avoided by automated enforcing of the explicit definition of "input terminals" or "ports", which mediate the access to a submap. A port represents an EquivalenceArc that is connected to an existing entity on the same map level and a Tag that is part of the Submap definition. To facilitate a textual representation of ports, SBGNtext differentiates between "InputTerminalPort" (an arc in the higher-level map that points to the Tag in the box that represents the submap) and "OutputTerminalPort" (an equivalence arc on the lower-level submap that highlights an entity that plays a role in the wider context of the higher-level map). Both types of ports can have entities flow in and out through them. Graphical SBGN editors could check the consistency of the representation by comparing the ports on the higher-level map with the ports on the lower-level map.

Tags (listed as Entity Pool Nodes in the SBGN-PD reference card on p.67 in [10]) are defined as the other end of an equivalence arc that points to an EPN. Their purpose is to highlight EPNs of a given map level that also play a role in higher-level maps or lower-level maps.

2.7 Further uses of SBGNtext: Storing graphical details and error checks

SBGNtext in its present form can facilitate the storing of graphical details of SBGN-PD diagrams and their automated analysis for potential high-level errors (see Appendix).

3 Extensions for quantitative analysis

While SBGN process diagrams were not designed for quantitative analysis, the semantics of their notation is based on an underlying process flow abstraction (p.40 in [10]). In essence the notation describes the flow of entities between entity pools, with the rate of flow regulated by processes. A useful analogy is one of fluid flow, where EPNs represent tanks of fluid, consumption and production arcs are pipes with different thicknesses (stoichiometry), and processes are valves that can be regulated by the level of fluid in other tanks.

Thus SBGN process diagrams can easily be quantified by adding the following attributes to the notation. These attributes are stored as strings in our textual representation of SBGN-PD and are attached to the corresponding glyphs by a graphical SBGN-PD editor. They do not require a visual representation that compromises the visual ease-of-use that SBGN-PD aims for. To facilitate efficient access, the graphical editor could open a dedicated window for reading or editing the attributes of a particular glyph or support a special view of the whole network, where all attributes of each node are directly visible and editable if necessary. Next we discuss the various attributes that are necessary for the various glyphs of SBGN-PD to support quantitative analysis. There is no need to discuss auxiliary units, submaps, tags and equivalence arcs here, as they do not require extensions for supporting quantitative analysis.

3.1 Extensions of EntityPoolNode

For quantitative analysis, each unique EPN requires an `InitialMoleculeCount` to unambiguously define how many entities exist in this pool in the starting state. We followed developments in the SBML standard in using counts of molecules instead of concentrations, since SBGN-PD also allows for multiple compartments, which makes the use of concentrations very cumbersome (see section 4.13.6, p.71f. in [4]).

For entities of the type `PerturbingAgent`, the `InitialMoleculeCount` is interpreted as the numerical value associated with the perturbing agent, even though its technical meaning is not a count of molecules. For entities of the type `Source` or `Sink`, the `Initial- MoleculeCount` is likely to be a dummy variable in most cases; it could however still be interpreted as some quantity that could be referred to in the propensity functions of processes (e.g. some limitation on the number of source molecules that are available). Apart from the combination of unique `EntityPoolNodeID` (see definition above) and `InitialMoleculeCount`, no other information on entities is required for quantitative analysis.

3.2 Extensions of Arcs

Arcs link entities and processes by storing their respective IDs and the `ArcType`. The simplest arcs are of type `Consumption` or `Production` and do not require numerical information beyond the `Stoichiometry` that is already defined in SBGN-PD as a property of arcs that can be displayed visually in graphical SBGN-PD editors that do not support the quantitative extensions described here. Logic arcs and equivalence arcs will be discussed below. All other arcs modulate the process they point to, like some enzyme that catalyses a reaction, for example. Modulation is governed by parameters or other important quantities for the given process, like Michaelis-Menten-constants, for example.

SBGNtext stores process parameters in a list of `QuantitativeProperties` in the corresponding arc to allow for maximal flexibility and ease of use. This is equivalent to seeing the parameters of a reaction as something that is specific to the interaction between a particular modulator and the process it modulates. Other approaches are also possible, but lead to less elegant implementations. Storing parameters in equations requires frequent and possibly error-prone changes (e.g. many different Michaelis-Menten equations). One could also argue that the catalytic features are a property of the enzyme and thus make parameters part of EPNs; however this either forces all Michaelis-Menten reactions of an enzyme to happen at the same speed or requires cumbersome naming conventions to manage different affinities for different substrates.

To refer to parameters we give the `ManualEquationArcID` of an arc and then the name of the parameter that is stored in the list of `QuantitativeProperties` with its value. This scheme reduces clutter by limiting the scope of the relevant namespace (only few arcs per process exist; IDs only need to be unique within that immediate neighbourhood). Thus parameter names can be brief, since they only need to be unique within the arc. `ManualEquationArcID` is different from `ArcID`, a globally unique identifier that is automatically generated by the graphical editor.

The `ManualEquationArcID` allows for user-defined generic names that are easy to remember, such as 'Km' and 'vm' for Michaelis-Menten reactions. It should be easily accessible within the graphical editor, just as the parameters that are stored within an arc.

3.3 Extensions to ProcessNodes

For quantitative analyses, a ProcessNode must have a unique name and an equation that computes the corresponding propensity, which is proportional to the probability that this process occurs next, based on the current global state of the model.

Since the `ProcessType` is not required for quantitative analyses, it does not matter whether a process is an ordinary `Process`, an `Uncertain` process or an `Observable` process, for example. For all these processes, graphical editors need to support the manual specification of a `ProcessNodeID`, and a `PropensityFunction` for `ProcessNodes`. If support for bidirectional reactions is desired, then they need to facilitate entering a propensity function for the backward process as well. SBGNtext stores the forward propensity function under `PropensityFunction` or `ForwardPropensityFunction` and the backward propensity function under `BackwardPropensityFunction`. These

functions compute the propensity of a unidirectional process to be the next event in the model and can be used directly by simulation algorithms and solvers [19].

The ProcessNodeID is a meaningful, globally unique name that is reused in the numerical analysis software to help interpret results. The ProcessNodeID should ideally have a "speaking name" that will make it easy for the interpreting scientist to recognise the process. Such ProcessNodeIDs are best specified manually, as automatic algorithms for generating ProcessNodeIDs from a prefix and a running number would effectively obfuscate the model description and make the subsequent interpretation of numerical results cumbersome. If ProcessNodeIDs were only generated on the fly while saving SBGNtext files, it would be impossible to use ProcessNodeIDs from simulation output to identify a given process in the SBGN diagram (as ProcessNodeIDs are not part of the diagram). Thus the graphical editor needs to provide a dedicated attribute for editing and displaying the ProcessNodeIDs.

To instantiate the propensity function, all aliases need to be replaced by their true identity. We use the following syntax for a parameter alias that is substituted by the actual numeric value (or a globally defined parameter) from the corresponding arc:

<par: ManualEquationArcID.QuantitativePropertyName>

While translating to Bio-PEPA this would be simply substituted with a corresponding parameter name. The parameter is then defined elsewhere in the Bio-PEPA code by the numerical value stored in the corresponding property of the arc.

To allow the numerical analysis tool to access an EPN count at runtime we replace the following entity alias by the EntityPoolNodeID that the corresponding arc links to:

<ent: ManualEquationArcID >

This is shorter than the EntityPoolNodeID and allows the reuse of propensity functions if kinetic laws are identical and ManualEquationArcIDs follow the same pattern.

It is desirable that there is no need to specify the EntityPoolNodeID. The EntityPoolNodeID is fairly long and generated automatically to reflect various properties that make it unique. It would be cumbersome to refer to in the equation and it would require a mechanism to access the automatically generated EntityPoolNodeID before a SBGNtext file is generated. Also any changes to an entity that would affect its EntityPoolNodeID would then also require a change in all corresponding propensity functions, a potentially error-prone process.

Properties of compartments are accessed within equations by using a term with the syntax:

<comp: CompartmentID.QuantitativePropertyName>

In addition to these aliases, functions use the typical standard algebra rules and operators that are merely handed through to the analysis tool.

3.4 Extensions to Compartments

All compartments can have a list of quantitative properties that might be accessed from within propensity functions, but there is no mandatory parameter in this list and thus it

can be missing. It is not necessary to specify the size of a compartment, if and only if all kinetic parameters and all `IntitialMoleculeCount`s are specified as explicit numbers that are independent of compartment volume. Whether a given list of compartment properties is specified as a list within one attribute or whether each entry in the list is entered into a separate attribute, will depend on the details of each visual SBGN-PD editor.

3.5 Logical operators and logic arcs

To facilitate the use of logical operators in quantitative analyses one needs to convert the integers from the relevant molecule counts of the EPNs involved to a binary signal amenable to boolean logic. To this end SBGNtext supports "incoming logic arcs" that connect a SourceEntity or SourceLogicalResult with a DestinationLogicOperator and apply an InputThreshold to decide whether the source is above the threshold ("On") or below the threshold ("Off"). To this end the graphical editor needs to support the InputThreshold as a numerical attribute that the user can enter; all other information recorded in incoming logic arcs is already part of an SBGN diagram.

Once all signals are boolean, they can be processed by one or several logical operators, until the result of this operation is given in the form of either 0 ("Off") or 1 ("On"). This result then needs to be converted back to an integer or float value that can be processed in a propensity function. To this end the graphical editor needs to support the attributes OutputDefinitionLOW and OutputDefinitionHIGH.

4 Mapping SBGN-PD elements to Bio-PEPA

In this section we explain how to map the semantics of SBGN-PD to a formalism for the quantitative analysis of biochemical systems. We are using Bio-PEPA as an example, but our approach is general and can be applied to many other formalisms.

4.1 The Bio-PEPA language

Bio-PEPA is a stochastic process algebra which models biochemical pathways as interactions of distinct entities representing reactions of chemical species [15, 16]. A process algebra model captures the behaviour of a system as the actions and interactions between a number of entities, where the latter are often termed "processes", "agents" or "components". In PEPA and Bio-PEPA these are built up from simple *sequential components* [15, 20, 21]. Different process algebras support different modelling styles for biochemical systems [21]. Stochastic process algebras, such as PEPA [20] or the stochastic π-calculus [22], associate a random variable with each action to represent the mean of its exponentially distributed waiting time. In the stochastic π-calculus, interactions are strictly binary whereas in Bio-PEPA the more general, multiway synchronisation is supported. The syntax of Bio-PEPA is defined as [15] :

$$S ::= (\alpha, \kappa)\ \mathsf{op}\ S \mid S + S \mid C \qquad P ::= P \underset{\mathcal{L}}{\bowtie} P \mid S(x)$$

where S is a sequential *species component* that represents a chemical species (is also called "process" in other process algebras and "EntityPoolNode" in SBGN-PD), C is a constant pointing to an S, P is a *model component* that describes the set $\mathcal{L}$ of possible interactions between species components (these "interactions" or "actions" correspond to "processes" in SBGN and can represent chemical reactions). A count of molecules or a concentration of S is given by $x \in \mathbb{R}_0^+$. In the prefix term "(α, κ) op S", κ is the *stoichiometry coefficient* and the operator op indicates the role of the species in the reaction α. Specifically, op $= \downarrow$ denotes a *reactant*, $\uparrow$ a *product*, $\oplus$ an *activator*, $\ominus$ an *inhibitor* and $\odot$ a generic *modifier*. The operator "+" expresses a choice between possible actions. Finally, the process $P \underset{\mathcal{L}}{\bowtie} Q$ denotes the synchronisation between components: the set $\mathcal{L}$ determines those activities on which the operands are forced to synchronise. We can define a Bio-PEPA system as follows:

Definition 1. *A Bio-PEPA system $\mathcal{P}$ is a 6-tuple $\langle \mathcal{V}, \mathcal{N}, \mathcal{K}, \mathcal{F}_R, Comp, P \rangle$, where: $\mathcal{V}$ is the set of compartments, $\mathcal{N}$ is the set of quantities describing each species (includes the initial concentration), $\mathcal{K}$ is the set of all parameters referenced elsewhere, $\mathcal{F}_R$ is the set of functional rates that define all required kinetic laws, Comp is the set of definitions of species components S that highlight the reactions a species can take part in and P is the system model component.*

We make the following assumptions (see [15] for more details):

- only irreversible reactions are considered: reversible reactions can be seen as the union of a pair of forward and inverse reactions;
- the reactants of the reaction can only decrease their concentration whereas the products can only increase it. Enzymes and inhibitors do not change;
- the same species in different situations (e.g. phosphorylated, free, bound, different compartments, ...) are regarded as different species and represented by distinct Bio-PEPA components;
- compartments are static and do not play an active role in reactions. Throughout this paper we assume that all reactions take place within a single compartment.

A variety of analysis techniques can be applied to a single Bio-PEPA model, facilitating the easy validation of analysis results when the analyses address the same issues [23] and enhancing insight when the analyses are complementary [24]. Currently supported analysis techniques include stochastic simulation at the molecular level, ordinary differential equations, probabilistic model checking and numerical analysis of continuous time Markov chains [15, 16] (Figure 1).

Additional analysis techniques are facilitated by compositional reasoning, which allows the automated extension of elementary proofs of qualitative features to complex models. Examples for such qualitative analyses include deadlock and livelock detection and simple model checking of a model against a logical formula.

In Bio-PEPA a "reagent-centric" style of modelling is adopted, which means that different species components denote different species (and not different molecules that could belong to the same species, as in the so called "pathway-centric" style of modelling [36]). This approach reduces the problem of state-space explosion.

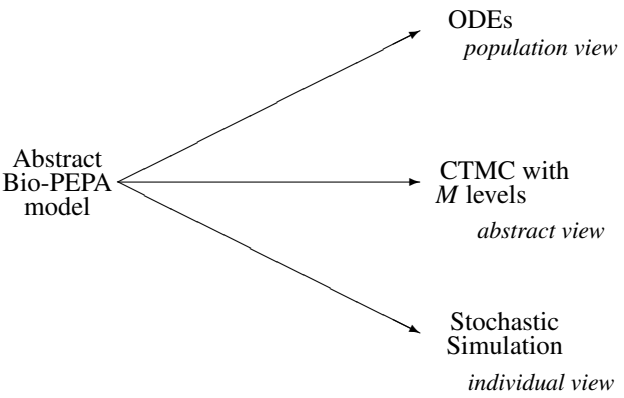


Fig. 1. Alternative modelling approaches: a single Bio-PEPA description of a system may be used to derive alternative mathematical representations offering different analysis possibilities.

4.2 SBGN-PD mapping

Here we map the core elements of SBGN-PD to Bio-PEPA (see [18] for an implementation).

Entity Pool Nodes Due to the rich encoding of information in the `EntityPoolNodeID`, Bio-PEPA can treat each distinct `EntityPoolNodeID` as a distinct species component. This removes the need to explicitly consider any other aspects such as entity type, modifications, complex structures and compartments (see Table 2 for the full list). To define the set $\mathcal{N}$ of a Bio-PEPA system requires the attribute `InitialMoleculeCount` for each EPN as defined in Section 3.

Processes All SBGN-PD `ProcessTypes` are simply represented as a reaction in Bio-PEPA. Compiling the corresponding set $\mathcal{F}_R$ relies on the attribute `ReactionPropensityFunction` and a substitution mechanism that makes it easy to define these functions manually. To help humans understand references to reactions in the sets $\mathcal{F}_R$ and $Comp$ requires recognizable names for SBGN-PD `ProcessNodeID`s that map directly to the corresponding identifiers in Bio-PEPA. Thus `ProcessNodeID`s need to be entered as attributes in a graphical Editor.

Reversible processes. The translator supports reversible SBGN-PD processes by dividing them into two unidirectional processes for Bio-PEPA. The translator reuses the manually assigned identifier and augments it by "_F" for forward reactions and "_B" for backward reactions. These two unidirectional processes are entered into the corresponding TreeMap instead of one unidirectional process after parsing the relevant process information.

When compiling the species components in Bio-PEPA, every time a `LeftHandSide` arc is found, the translator assumes that the corresponding forward and backward

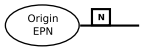
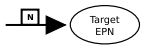
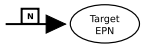
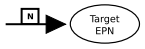
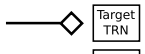
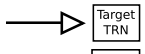
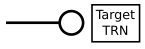
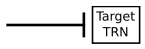
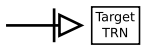
SBGN-PD glyph	prefix	keyword	Bio-PEPA symbol	code
	cn	Consumption	↓	<<
	pr	Production	↑	>>
	lh	LeftHandSide	↓ and ↑	<< and >>
	rh	RightHandSide	↑ and ↓	>> and <<
	mo	Modulation	⊙	(.)
	st	Stimulation	⊕	(+)
	ct	Catalysis	⊕	(+)
	ih	Inhibition	⊖	(-)
	ns	NecessaryStimulation	⊙	(.)

Table 4. Mapping SBGN-PD types of connection arcs to Bio-PEPA species components. The "prefix" can be used to increase readability of the ArcIDs denoted by the "keyword". The "Bio-PEPA symbols" are frequently used in Bio-PEPA papers, while the "code" column gives the corresponding notation in the current implementation of the Bio-PEPA Eclipse-plugin [16].

processes have been defined and will augment the process name by "_F" for forward reactions and "_B" for backward reactions, while adding the corresponding Bio-PEPA operator for reactant and product. `RightHandSide` arcs are handled in the same way. Thus the production arc glyph in SBGN-PD has three distinct meanings as shown in Table 4.

Arcs The arcs in SBGN-PD define which entities interact in which processes. Thus arcs play a pivotal role in defining the species components in Bio-PEPA. Since arcs can store kinetic parameters, they are also important for defining parameters in Bio-PEPA. As kinetic law definitions in Bio-PEPA frequently refer to such parameters, we use the `ArcID` that is automatically generated by the graphical editor to substitute the local manual arc references in propensity functions by globally unique parameters names (see Section 3). The type of an arc indicates both the role of the connected entity in the process (consumed reactant, product or modifier of rate) and the chemical nature of the reaction (catalysis, stimulation, inhibition, necessary stimulation or the most generic modification). Thus the type of an arc can be mapped directly to the operator "`op`" described in the Bio-PEPA syntax shown in Table 4.

Logical operators Logical operators require the conversion of integers of the relevant molecule counts of the EPNs to a binary signal and after some boolean logic processing back to a low and a high integer value. As evident from the implementation scheme above, the use of all quantitative properties culminates in the correct formulation of

the corresponding propensity functions that determine the probability that the corresponding process will be the next to occur. Thus an implementation of logical operators requires that their results be included in the corresponding propensity functions.

The current scheme of implementing propensity functions relies heavily on substituting the various components into the final equation, so that Bio-PEPA will ultimately only see one formula per propensity function. In this context the implementation of logical operators requires the insertion of a formula in the propensity function that computes the result of the boolean operations from their integer input. An arbitrarily complex logic operator network can be constructed from the following basic building blocks:

- Convert from integers or double floats to boolean values. This is best done by a specially defined mathematical function that takes an integer or float signal and compares it to a specified threshold, giving back either 0 (signal < threshold) or 1 (signal ≥ threshold). The definition of such a function in Bio-PEPA is not complicated and is available in the Bio-PEPA Eclipse Plug-in (version 0.1) and later.
  Alternatively this can be accomplished by the help of the "signum" function that returns -1 for negative values, +1 for positive values and 0 for 0. The boolean value B can then be computed from

  $$B = signum((signum(signal - threshold) + 1)$$

  if the threshold itself is to be included with positive results. If not,

  $$B = signum((signum(signal - threshold) - 1).$$

  If the signum function is not locally defined (e.g. Java before version 1.5) and if the signals to be converted to boolean values have a reduced range of values, then the signum function can be reduced to $signum(x) = x/abs(x)$ as long as x cannot become 0. However, it is cumbersome, risky and hence not recommended to use only basic arithmetic and $abs(x)$ to implement a true threshold function equivalent to the one described above. Implementing a dedicated function is much more elegant.
- AND operator. Multiply all boolean input to arrive at output.
- NOT operator. 1- *input* gives *output*
- OR operator. Summing over all input (0/1) and test if it is greater than 1 using the threshold / signum function.

Support for a useful implementation of logical operators in SBGNtext2BioPEPA has become possible with the release of the latest Bio-PEPA Eclipse Plug-in (version 0.1).

Submaps Since all submaps contribute globally unique EPNs and processes to the one big common namespace, their implementation does not require special attention, as all Bio-PEPA code is generated from this common namespace regardless whether it was populated by reading in one or several submaps.

5 Converter implementation and internal representation

We chose Java as implementation language for the converter described above, due to the good portability of the resulting binaries. Also, the tools at both ends of the workflow are implemented in Java: the Edinburgh Pathway Editor [13] that is being developed to write SBGNtext files and the Bio-PEPA Eclipse Plug-in [16] that is being developed to serve as analysis tool. We defined a grammar for SBGNtext in the Extended Backus-Naur-Form (EBNF) and augmented it by interspersed Java code that links semantic actions to the parsing process as supported by ANTLR [25]. ANTLR automatically compiles the Java sources for the corresponding parser that stores all important parsing results in a number of coherently organised internal TreeMaps. To compile a working Bio-PEPA model three main loops over these TreeMaps are necessary: (i) over all entities, (ii) over all processes and (iii) over all parameters.

The **loop over all entities** compiles the species components as well as the model description required by Bio-PEPA. The latter is a list of all participating EntityPool-NodeIDs separated by the cooperation operator "$< * >$" that automatically synchronises all necessary actions (equivalent to "$\underset{\mathcal{L}}{\bowtie}$" with the right actions included in $\mathcal{L}$). Please note that this simplification depends on all processes in SBGN-PD having (i) unique names and (ii) fixed lists of reactants with no mutually exclusive alternatives in them. The first condition can be enforced automatically by the tools that produce the code, the second is met by the very nature of how processes are described in SBGN-PD[3].

For each species component a loop over all arcs finds all connecting arcs that store all relevant `ProcessNodeID`s. The same loop determines the respective role of the component (as reflected by the choice of the Bio-PEPA operator in Table 4).

The **loop over all processes** compiles the kinetic laws by substituting the aliases for parameters and EPNs in the propensity functions specified in the graphical editor. Each function is handled separately by a dedicated function parser for isolating all aliases that are then resolved by querying the relevant TreeMaps that were generated when parsing the SBGNtext file.

The **loop over all quantitative properties** of the model defines the parameters in Bio-PEPA.

It is possible to avoid this step by inserting the direct numerical values into the equations processed in the second loop. However, this would substantially reduce the readability of the equations in the Bio-PEPA code and would make it difficult for third party tools to assist in the automated generation of parameter combinations. Thus we defined a scheme that automatically generates parameter names that maximise the readability of equations by combining:

[3] In SBGN-PD it is not possible to describe a *single* reaction called "bind", which states that A "binds" *either* B *or* C to produce D. Bio-PEPA can describe such competitive binding models by referring to a single action only, but then the species components
`A=(bind,1)<<A; B=(bind,1)<<B; C=(bind,1)<<C; D=(bind,1)>>D;`
require the model description `A` $< * >$ `(B ||  C)` $< * >$ `D` to prevent the system from demanding the presence of both B *and* C, before the interaction can occur. To describe the same model in SBGN-PD requires two reactions with different names (`A+B→D; A+C→D;`). This is then translated into the correct Bio-PEPA model if only "$< * >$" is used.

`ArcID_SpeakingQuantitativePropertyID`

This is much shorter than the alternative

`SpeakingProcessID_SpeakingEntityID_SpeakingQuantitativePropertyID`,

where the `ProcessID` and the `QuantitativePropertyID` are directly specified by the user and can thus be chosen to maximise readability.

To facilitate walking over the various collections specified above, the TreeMaps are organised in four sets, one for entities, one for processes, one for arcs and one for quantitative properties. Each of these sets is characterised by a common key to all maps within the set. This facilitates the retrieval of related parse products for the same key from a different map. Since all this information is accessible from Java code, it is easily conceivable to use the sources produced in this work for reading SBGNtext files in a wide variety of contexts. The system is easy to deploy, since it involves few files and the highly portable ANTLR runtime library. Our converter is called SBGNtext2BioPEPA and sources are available [18].

6 Example: Stochasticity in the MAPK cascade

Here we illustrate our translation by applying SBGNtext2BioPEPA [18] to a real-world example. We implemented a simple model of the Mitogen-Activated Protein Kinase (MAPK) signal transduction cascade [27, 17] in order to investigate the time needed for the cascade to switch from "off" to "on". The general pattern of the MAPK cascade is well conserved across many biological taxa and triggers a highly varied range of molecular responses [17]. Such a broad conservation suggests not only important functionality (maintained by purifying selection), but also an astonishing flexibility in how a MAPK cascade might be implemented (it has to work in a wide range of contexts). For example, MAPK cascades operate in oocytes of the frog *Xenopus* and in yeast cells, despite their 5 million fold differences in size [17, 28] and substantial differences in kinetic constants [17, 26].

Figure 2 shows a SBGN-PD diagram of the basic structure of a MAPK cascade. It accepts "input" in the form of a change in the concentration of the enzyme E1 and then propagates the signal by changing the concentrations of the various intermediate substrates and enzymes until the "output" enzyme MAPK_PP is affected. If the input is off (low E1), the output is off too (no MAPK_PP). If the input is on (E1 above threshold), output is reliably on as well (high MAPK_PP). Important properties for signalling cascades include the reliability of transmitting a signal and the speed with which this happens. Building on the model in [17], a recent study explored the expected percentage of active output and the time until all output is activated [27].

Here we automatically translate our SBGNtext model of the MAPK cascade via SBGNtext2BioPEPA into a Bio-PEPA model (see [18] for code), which we analyse with the Bio-PEPA Eclipse Plugin [16]. We report the results of stochastic simulations using

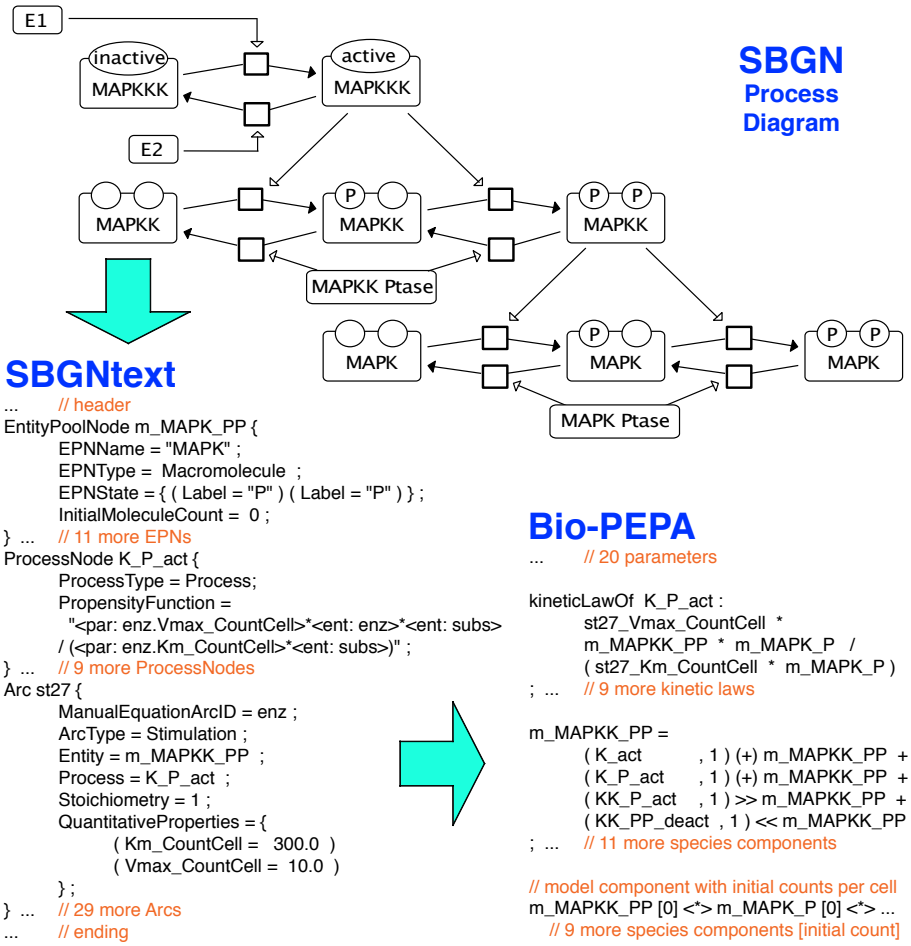


Fig. 2. A possible SBGN-PD representation of a MAPK cascade with fragments from the corresponding SBGNtext code and the automatically generated Bio-PEPA code. The code excerpts focus on EPN MAPK-PP and the reaction it catalyses (`'K_P_act'`). All parameters are scaled to represent counts of molecules per cell (including the Michaelis-Menten constants). For the full code and newer versions see [18]. Biologically, the input signal is a change in `E1`, the output signal a change in `MAPK_PP`. The back reactions on each of the three levels can be thought of as a "conveyer belt" that constantly turns active molecules (on the right) into passive ones (on the left). `E1` must overcome the conveyer belt on the first level for sequentially overcoming the other conveyer belts too. In many such systems either active or inactive molecules are essentially absent.

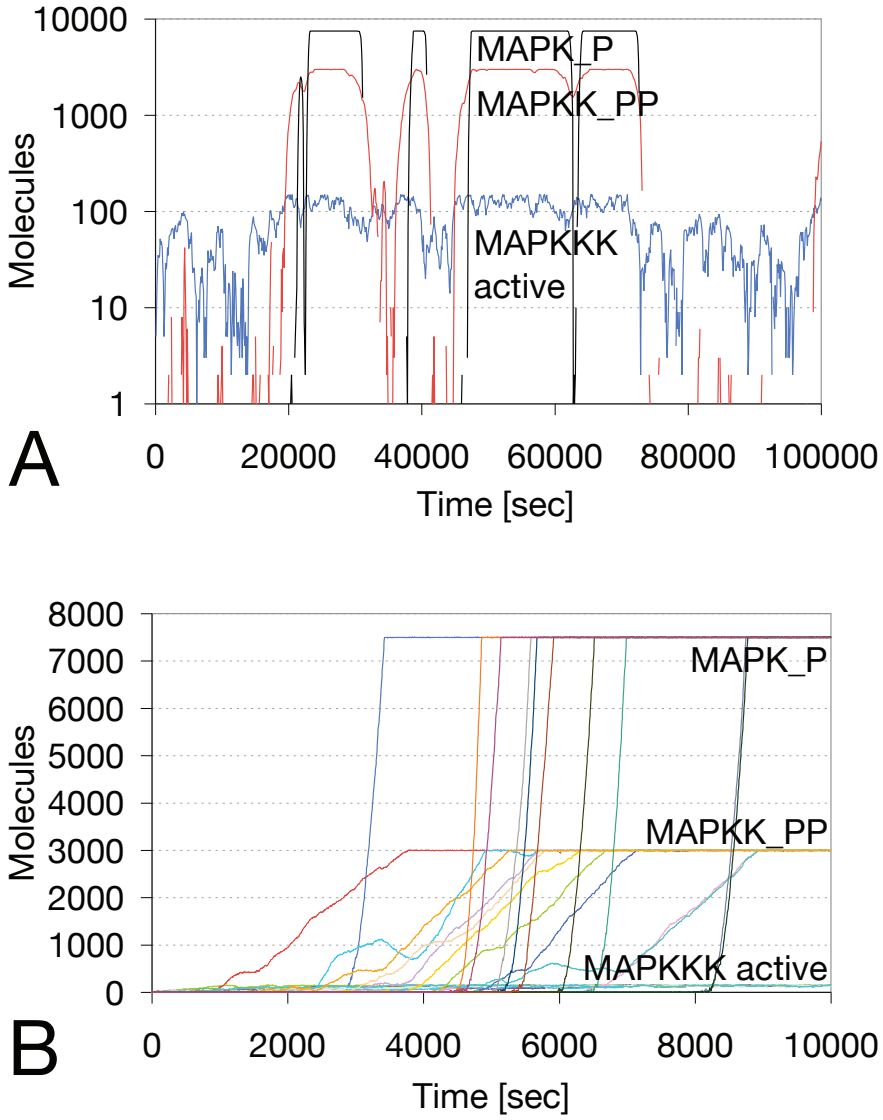


Fig. 3. Noise in the MAPK cascade. (A) One stochastic simulation trace, where E1 = E2 = 20 makes the cascade switch between the "on" and "off"state, as activating and deactivating reactions at the first level are equally strong. (B) Ten stochastic simulation traces, where E1 = 21, E2 = 20. If activating reactions are only slightly stronger than deactivating reactions at the first level of the cascade, the cascade can be expected to switch "on". However the time until the "on" state is reached is subject to considerable stochasticity, which is mostly determined by stochasticity during the initial stages of the switching processes.

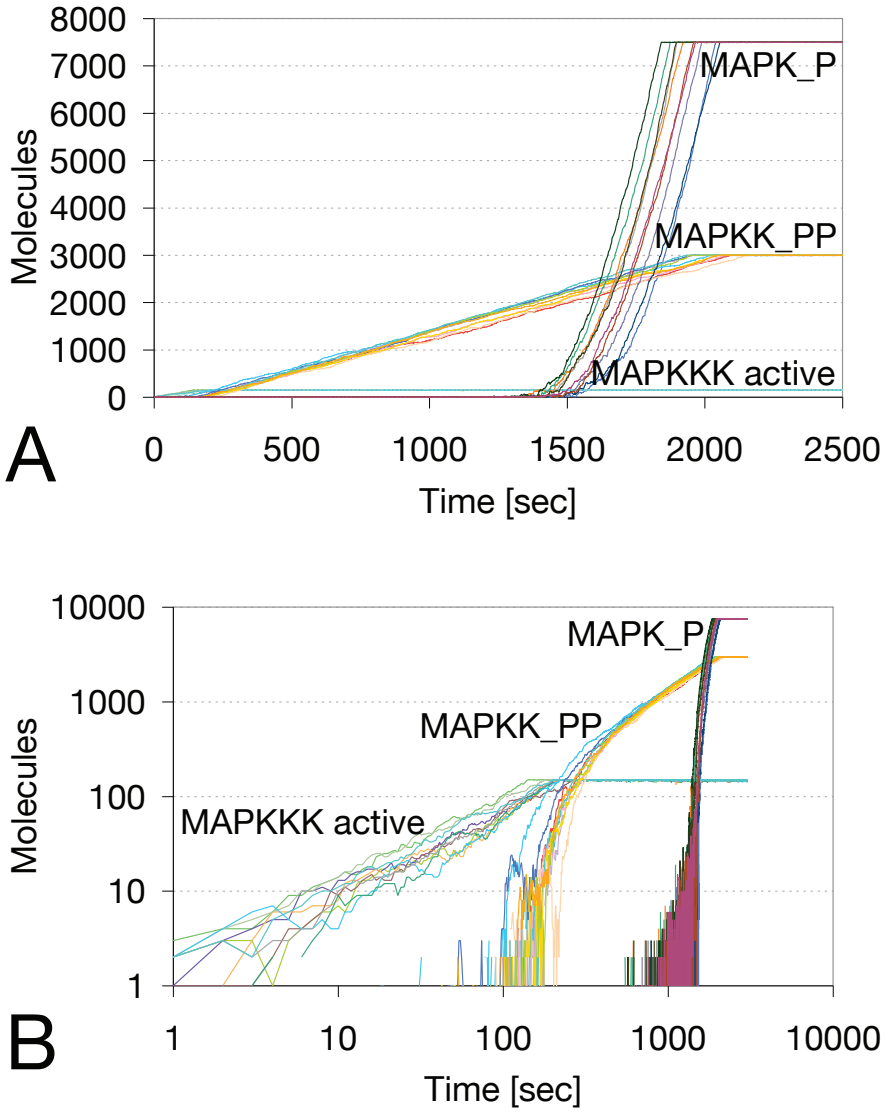


Fig. 4. Increasing the number of E1 molecules to 40 reduces the stochasticity of the time to the completion of the switch (see linear plot A). The stochasticity in the initial stages is considerable in both cases for this system (see log plot B). Ten stochastic simulation traces are shown. E2 = 20; for other parameters, see text.

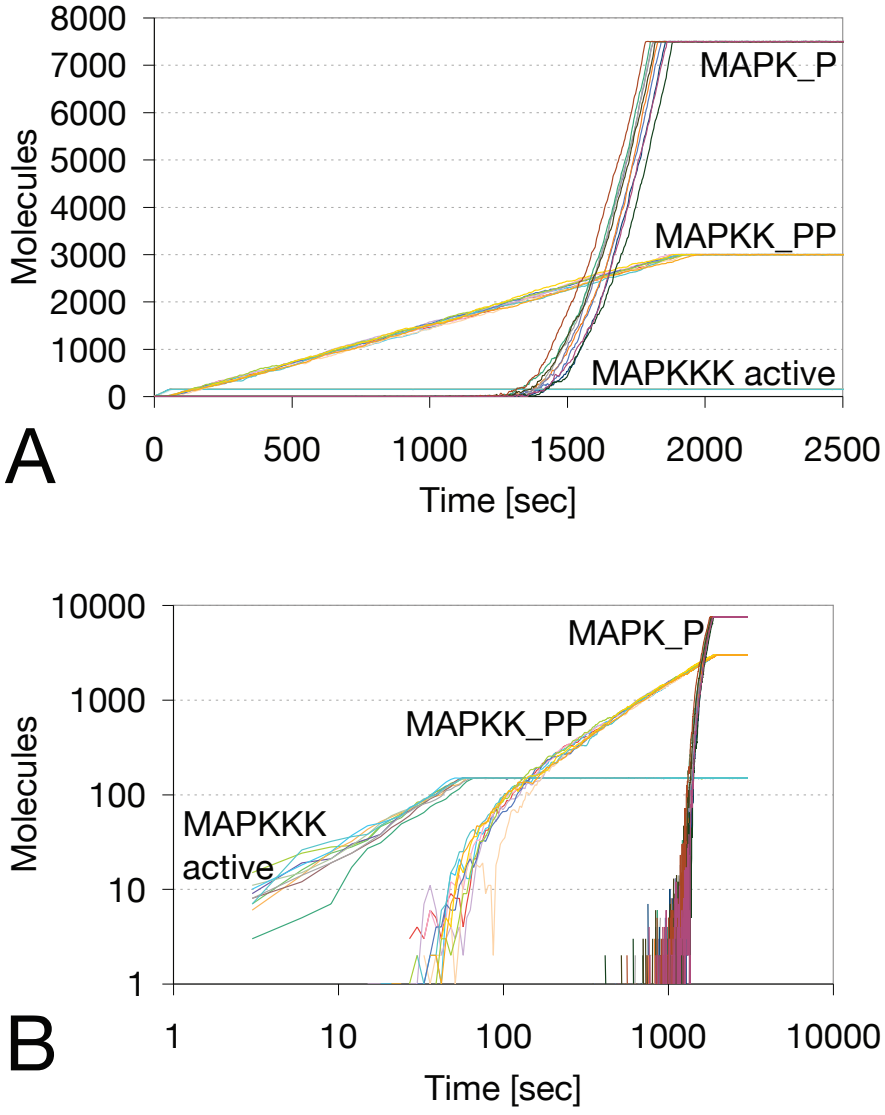


Fig. 5. Increasing the number of E1 molecules up to 100 leads to further reductions in the stochasticity of the time to the completion of the switch (see linear plots A). The stochasticity in the initial stages is considerable in both cases for this system (see log plot B). Ten stochastic simulation traces are shown. E2 = 20; for other parameters, see text.

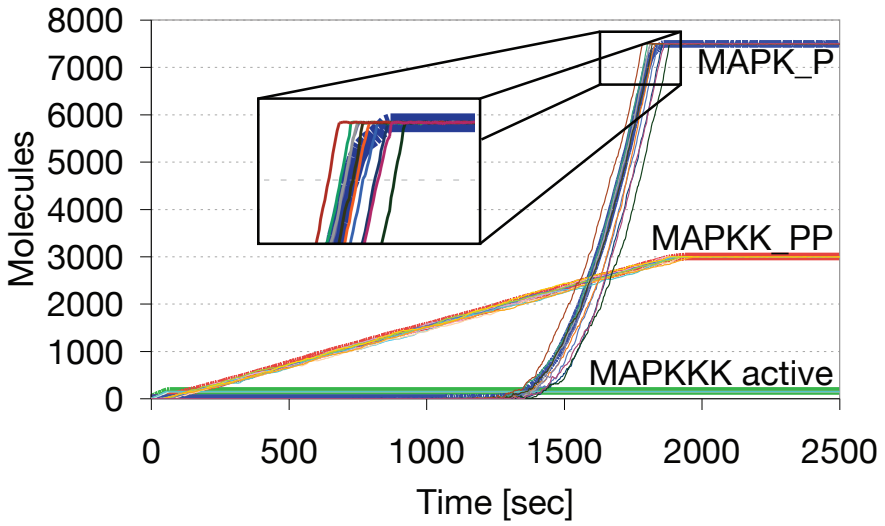


Fig. 6. Averaging 10 stochastic simulations results to obtain a mean timecourse can be misleading. Compare the behavior shortly before the switch is completely "on". The stochastic runs show a variability in the precise timing of when "on" is reached but all agree that "on" is reached very quickly and without a slowdown. This contrasts with the computed mean that suggests a small slowdown at the very end. Parameters are identical to runs in Figure 5.

the Gibson-Bruck algorithm [29, 19]. Each reaction is assumed to follow Michaelis-Menten kinetics, resulting in the following propensity function:

$$\frac{V_{max}[count/sec] * Enzyme[count] * Substrate[count]}{K_M[count] + Substrate[count]}$$

where [count] indicates the absolute number of molecules of this entity within the cell, V_{max} denotes the number of substrate molecules that can be processed by one enzyme at maximal speed, K_M denotes the Michaelis-Menten constant that is an inverse measure of the affinity between enzyme and substrate. We assume $K_M = 300$ and $V_{max} = 10$ for all reactions (see [17, 26] for other possible values). All our simulations start with molecule counts of E2 = 20, MAPKKK = 150, MAPKK = 3000, MAPKK_Ptase = 100, MAPK = 7500,MAPK_Ptase = 2000 and 0 for all other entities, reflecting the equilibrium "off" state. These molecule counts roughly reflect MAPK cascades in yeast, and are about 6 orders of magnitude below corresponding counts in *Xenopus* oocytes [17]. We tested various E1 input signals and found the following results.

1. The cascade remains "off", if E1<E2 (not shown).
2. The cascade switches beween "on" and "off", if E1 = E2 as shown in Figure 3A.

3. The cascade switches reliably to "on" if E1>E2, but at E1 = 21 there is considerable noise in the "signalling time" (from switching E1 "on" until all output MAPK_PP is switched "on" as well; see Figure 3B).
4. Further increasing E1 substantially reduces the variability in signalling times and slightly improves speed (Figure 4+5).
5. Additional tests have shown that the overall signalling speed is very strongly influenced by V_{max} (not shown).
6. The shape of the signal is estimated wrongly if many stochastic simulations are used to compute a supposedly more accurate "mean" (Figure 6). This is caused by the stochastic noise at the very beginning of the activation of the signal transduction cascade. Such noise determines the noise in the overall switch time, whereas the last phases of the "on" switch-process are almost completely deterministic in their dynamic. The result is an 'unfair' averaging when the mean is computed, as can be seen in the unnaturally 'round' transition to the "on" phase

ODE analysis. The ODE solvers (Adaptive Dormant Prince, Runge Kutta 5th order, as implemented in Dizzy [1]) had difficulties with the nature of this signal transduction system that keeps a part of molecular species at zero for most kinetic parameter combinations if molecule populations are not extraordinarily large. Zero is unavoidable in a system, which is either "on" (zero for the "off-state") or "off" (zero for the "on-state").

7 Related work

There are various tools that map visual diagrams to quantitative modelling environments (e.g. SPiM [30], BlenX [31], kappa [2], Snoopy [32], EPN-PEPA [33], JDesigner [14], CellDesigner [11]). However the corresponding graphical notations are not as rich as SBGN and are thus not easily applied to the wide range of scenarios that SBGN was designed for. Since SBGN is emerging as a new standard, it is clearly desirable to translate from SBGN to a quantitative environment.

Since the first draft of SBGN-PD has been published in August 2008, a number of tools are being developed to support it, including the Edinburgh Pathway Editor [13] (`http://www.bioinformatics.ed.ac.uk/epe/`), Arcadia for visualisation (`http://arcadiapathways.sourceforge.net/`) and Athena that is linked to the Systems Biology Workbench (`http://www.codeplex.com/athena`). The graphical editor CellDesigner [11] supports SBGN-PD and can translate it into SBML which is supported by many quantitative analysis tools. However the process of adding quantitative information involves cumbersome manual interventions. This motivated work for SBMLsqueezer [34], a CellDesigner plug-in that supports the automatic construction of generalised mass action kinetics equations. While the automated suggestions for the kinetic laws from SBMLsqueezer might be of interest for some problems, the generated reactions contain many parameters that are extraordinarily difficult to estimate. Thus it is preferable to also allow the user to enter arbitrary kinetic laws that may have to be hand-crafted, but whose equations are simpler and require fewer parameter estimates. In SBGNtext2BioPEPA this is combined with mechanisms to reuse the code for such kinetic laws, greatly reducing practical difficulties and the potential for errors.

8 Conclusion and Perspectives

Here we have provided a mechanism for translating SBGN-PD into code that can be used for quantitative analysis, using Bio-PEPA as an example. In order to do this we have created a textual representation of SBGN-PD that focuses on the essence of SBGN-PD content and avoids the clutter that comes from storing graphical details. We have developed our translator SBGNtext2BioPEPA in Java to facilitate its integration in the Systems Biology Software Infrastructure that is currently under development at the Centre for Systems Biology at Edinburgh (http://csbe.bio.ed.ac.uk/). SBGNtext2BioPEPA contains a parser for our SBGNtext format based on a formal ANTLR EBNF grammar and is freely available [18]. Building on the internal representation of entities, processes, arcs and parameters in our code can make it easy to implement translations of SBGNtext to other modelling platforms.

9 Appendix: Further uses of SBGNtext

9.1 How to store graphical details

Having a conceptual textual representation of SBGN is an important part of the work on the way to a representation that also stores all important graphical details. We consider it to be desirable to have a core graphical representation of SBGN that captures the essence of a graphical diagram in a way that is portable between different editors and that leaves room for each graphical editor or user to specify preferences that are specific to that editor. For example, the relative locations of various glyphs on a map are an integral part of the content, whereas the line-width is usually a standard value taken from the preferences for the corresponding glyph. SBGNtext can be easily extended to include the graphical details of maps in a way that scales well on screens and printers of different resolutions by merely adjusting a single variable. Here is how it works.

Independence from device resolution. If we specify the X axis of a map as the number of pixels available under a given screen resolution, then we can specify the Y axis of that map device-independently as a percentage of the pixel-value of the X axis. This keeps the aspects ratio of a map intact. Then we express all other X and Y coordinates as percentages of these two axes, where 0% is in the left lower corner and 100% is in the right upper corner, respectively. If these coordinates are stored as double floats, then enough numerical precision is available to represent even the biggest conceivable maps on the highest-quality output devices. Since the resolution of the output device affects only a single number (the number of pixels on the X-axis) and all other information is specified independently from screen resolution as percentages, it is easy to adjust such a SBGN diagram to a device with a different resolution: just change the number of pixels available on the X-axis and the pixel coordinates of all other glyphs are computed from the corresponding percentages.

Many glyphs merely need to store their centre point (i.e. [X%, Y%]) , others benefit from storing the rectangle within which they live (two points). The most complex objects are arcs and containers like complexes or compartments. These may require arbitrary lines, which could need many points; this list of points could be limited to key points that are essential for the correct wiring of the map, while avoiding storage of

the large number of intermediate points that can easily be interpolated and that make a visual representation appear polished. The list of points stored for these glyphs should be such that a direct straight line connection between them should still result in a valid SBGN-PD diagram.

Focussing on the essentials. The approach described above results in a full representation of all conceptually important visual information that is required for the consistent drawing of SBGN diagrams in different graphical editors. It avoids the confusion that results from tool-specific algorithms that automatically place glyphs across the map by computing some (tool-specific) "optimal" position from the structure of the network. Explicitly storing the essence of glyph coordinates determined by users not only reduces the burden of implementing (reading in values is easier than computing them), but also allows different users to easily view a new map with the visual preferences they are used to (e.g. consumption arcs always 2 points thick and red).

Why it should not be compulsory to store everything. The approach presented here allows different tools to develop different optical flavours without losing the conceptually important work in generating maps, namely where to put what on the map. It is always possible to store more graphical details to preserve the visual appearance in more detail (e.g. line widths, colours, more points to indicate the shapes of arcs and containers more precisely). However, focussing on the essential core can greatly simplify the implementation of SBGN editing capabilities.

Future implementation. It is beyond the scope of this report to develop full sets of preferences and lists of coordinates for each SBGN-PD glyph. Such work is easily done when implementing the facility to write SBGNtext for a graphical SBGN editor. Once this has been done for the first time, a simple adjustment of the ANTLR grammar described in this work will make it easy to read in the corresponding data for all subsequent users of SBGNtext.

9.2 Automated error checks

The automated parsing of the SBGNtext format opens up the possibility to automatically check for various errors and inconsistencies in a SBGNtext representation of a biochemical network that might otherwise be hard to detect. Low-level parsing errors are detected by ANTLRs excellent error-reporting facilities, so there is not need for extra code to check for these. However, at a higher level inconsistencies can be introduced, because SBGNtext allows for the expression of a few constructs that are not allowed in SBGN (e.g. once an EPN has a given set of EPNState variables, all other EPNs that are in other states have to have the same set of variables, even if they all have to be set to "Not Set"). While it is natural to expect from graphical SBGN editors to write out correct SBGNtext code, one might potentially introduce such automated error checks to facilitate the implementation of correct SBGNtext-output-modules (e.g. by automated checking of the consistency of the various automatically generated IDs).

Acknowledgements. The Centre for Systems Biology Edinburgh is a Centre for Integrative Systems Biology (CISB) funded by BBSRC and EPSRC, reference BB/D019621/1.

References

1. Ramsey S., Orrell D., Bolouri H.: Dizzy: stochastic simulation of large-scale genetic regulatory networks. J Bioinform Comput Biol. Apr;3(2):415-36, (2005).
2. Danos V., Feret J., Fontana W., Harmer R., Krivine J.: Rule-based modelling of cellular signalling. Lecture Notes in Computer Science, vol. 4703, pp.17-41, (2007).
3. Hlavacek W.S., Faeder J.R., Blinov M.L., Posner R.G., Hucka M., Fontana W.: Rules for modeling signal-transduction systems. Sci STKE, vol. 2006, issue 344, page re6, (2006).
4. Hucka M., Hoops S., Keating S., Le Novère N., Sahle S., Wilkinson D.: Systems Biology Markup Language (SBML) Level 2 Version 4 Release 1. Nature Precedings `http://dx.doi.org/10.1038/npre.2008.2715.1` and `http://sbml.org/Documents/Specifications` (2008)
5. Kohn K.W., Aladjem M.I., Kim S., Weinstein J.N., Pommier Y.: Depicting combinatorial complexity with the molecular interaction map notation. Mol Syst Biol, 2:51, (2006)
6. Raza S., Robertson K.A., Lacaze P.A., Page D., Enright A.J., Ghazal P., Freeman T.C.: A logic-based diagram of signalling pathways central to macrophage activation. BMC Syst Biol, 2:36, (2008)
7. Kitano H., Funahashi A., Matsuoka, Y., Oda K.: Using process diagrams for the graphical representation of biological networks. Nature Biotechnology 23:961-966 (2005)
8. Moodie S.L., Sorokin A.A., Goryanin I., Ghazal P.: A Graphical Notation to Describe the Logical Interactions of Biological Pathways. J. Integr. Bioinformatics 3:36 (2006)
9. Demir E., Babur O., Dogrusoz U., Gursoy A., Nisanci G., Cetin-Atalay R., Ozturk M.: PATIKA: an integrated visual environment for collaborative construction and analysis of cellular pathways. Bioinformatics, 18:996-1003 (2002)
10. Le Novère, N., Moodie, S., Sorokin, A., Hucka, M., Schreiber, F., Demir, E., Mi, H., Matsuoka, Y., Wegner, K., Kitano, H.: Systems Biology Graphical Notation: Process Diagram Lev.1. Nature Preceedings `http://hdl.handle.net/10101/npre.2008.2320.1` (2008)
11. Funahashi, A.; Matsuoka, Y.; Jouraku, A.; Morohashi, M.; Kikuchi, N.; Kitano, H.: CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks, Proceedings of the IEEE vol. 96, issue 8, pp. 1254-1265, `http://www.celldesigner.org/` (2008)
12. Cytoscape Consortium: Cytoscape Home page. `http://cytoscape.org/` (2009)
13. Sorokin A, Paliy K, Selkov A, Demin O, Dronov S, Ghazal P, Goryanin I: The Pathway Editor: A tool for managing complex biological networks-References. IBM J. Res. Dev., vol. 50, pp. 561-573, (2006) `http://www.bioinformatics.ed.ac.uk/epe/`
14. Sauro H.M., Hucka M., Finney A., Wellock C., Bolouri H., Doyle J. and Kitano H.: Next generation simulation tools: the Systems Biology Workbench and BioSPICE integration. OMICS. 2003 7(4):355-72, (2003). For the graphical front end `JDesigner` see: `http://www.sys-bio.org/software/jdesigner.htm`
15. Ciocchetta F. and Hillston J.: A framework for the modelling and analysis of biological systems. Theoretical Computer Science, doi:10.1016/j.tcs.2009.02.037 (2009)
16. Bio-PEPA homepage `http://www.biopepa.org/`. To install the Bio-PEPA Eclipse Plug-in version 0.1 by Adam Duguid follow the links from `http://homepages.inf.ed.ac.uk/jeh/Bio-PEPA/Tools.html` (2009)
17. Huang C.Y., Ferrell J.E., Jr.: Ultrasensitivity in the mitogen-activated protein kinase cascade. Proc Natl Acad Sci USA, 93:10078-10083, (1996)
18. Loewe L.: Homepage for SBGNtext2BioPEPA. `http://csbe.bio.ed.ac.uk/SBGNtext2BioPEPA/index.php` (2009)
19. Gillespie, D. T.: Stochastic simulation of chemical kinetics. Annu. Rev. Phys. Chem. 58:35-55 (2007)

20. Hillston, J.: *A Compositional Approach to Performance Modelling*, Cambridge University Press (1996)
21. Calder, M. and Hillston, J.: Process algebra modelling styles for biomolecular processes. *Transactions on Computational Systems Biology*, in press, (2009)
22. Priami C.: Stochastic π-calculus. *The Computer Journal*, 38(6), pp. 578–589, (1995)
23. Calder M. , Duguid A., Gilmore S. and Hillston J.: Stronger computational modelling of signalling pathways using both continuous and discrete-space methods. Proc. of *CMSB'06*, LNCS vol. 4210, pp. 63–77 (2006)
24. Ciocchetta F., Gilmore S., Guerriero M.-L. and Hillston J.: Stochastic Simulation and Probabilistic Model-Checking for the Analysis of Biochemical Systems. To appear in ENTCS (Proc. Internatl. Workshop Pract. Applic. Stochastic Modelling 2008).
25. Parr, T.: The Definitive ANTLR Reference: Building Domain-Specific Languages. The Pragmatic Bookshelf, Raleigh, NC. `http://www.antlr.org/` (2007)
26. Kholodenko, B.N.: Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. Eur J Biochem vol. 267 (6) pp. 1583-8 (2000)
27. Kwiatkowska M., Norman G. and Parker D.: Using probabilistic model checking in systems biology. ACM SIGMETRICS Performance Evaluation Review vol. 35 (4) 14-21 (2008)
28. Wallace R.A., Misulovin Z.: Long-term growth and differentiation of Xenopus oocytes in a defined medium. P Natl Acad Sci USA vol. 75 (11) pp. 5534-8 (1978)
29. Gibson M.A., Bruck J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. Journal of Physical Chemistry A, 104:1876-1889 (2000)
30. Phillips A.: A Visual Process Calculus for Biology. Jones and Bartlett Publishers (2009) `http://research.microsoft.com/en-us/projects/spim/`
31. Dematté L., Priami C., Romanel A.: The BlenX Language: A Tutorial. SFM 2008, LNCS 5016:313-365, Springer (2008)
32. Heiner, M., Richter R., Schwarick M., Rohr C.: Snoopy - A tool to design and execute graph-based formalisms. Petri Net Newsletter 74 (April) ISSN 0931-1084, pp. 8-22, `http://www-dssz.informatik.tu-cottbus.de/software/snoopy.html` (2008)
33. Shukla, A.: Mapping the Edinburgh Pathway Notation to the Performance Evaluation Process Algebra. MSc. Thesis, University of Trento (2007)
34. Draerger A., Hassis N., Supper J., Schröder A., and Zell A.: SBMLsqueezer: A CellDesigner plug-in to generate kinetic rate equations for biochemical networks. BMC Systems Biology, 2:39 (2008)
35. Systems Biology Ontology, `http://www.ebi.ac.uk/sbo/` (2009)
36. Calder M., Gilmore S., and Hillston J.: Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. In Ingolfsdottir, A. and Nielson, H.R., editors, Proceedings of the BioConcur Workshop on Concurrent Models in Molecular Biology, London, England, August 2004.