Technical Report CoSBi 12/2007

An Automated Translation from a Narrative Language for Biological Modelling into Process Algebra

Maria Luisa Guerriero
DISI, University of Trento
guerrier@dit.unitn.it

John K. Heath
School of Biosciences, University of Birmingham
j.k.heath@bham.ac.uk

Corrado Priami
CoSBi
and
DISI, University of Trento
priami@cosbi.eu

This is the preliminary version of a paper that will appear in Proceedings of CMSB07, LNBI 4695, 136-151, 2007
available at http://www.springerlink.com/content/vt23126776012835/

An Automated Translation from a Narrative Language for Biological Modelling into Process Algebra

Maria Luisa Guerriero[1], John K. Heath[2], and Corrado Priami[1,3]

[1] Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Italy
[2] CRUK Growth Factor Group, School of Biosciences, University of Birmingham, UK
[3] The Microsoft Research - University of Trento
Centre for Computational and Systems Biology, Italy
{guerrier,priami}@dit.unitn.it, j.k.heath@bham.ac.uk

Abstract. The aim of this work is twofold. First, we propose an high level textual modelling language, which is meant to be biologically intuitive and hence easily usable by life scientists in modelling intra-cellular systems. Secondly, we provide an automatic translation of the proposed language into Beta-binders, a bio-inspired process calculus, which allows life scientists to formally analyse and simulate their models. We use the Gp130 signalling pathway as a case study.

1 Introduction and Motivations

Process calculi, originally developed for modelling mobile communicating systems [10], have proved to be both useful and powerful tools for modelling biological signalling pathways [14, 12, 13, 2, 9]. The need for such modelling approaches has arisen from the complexity of these processes, which is not easily analysed by biological intuition [6]. A key challenge in the application of process calculi to biological problems is specification of models. The issue poses several practical and computational problems. First, it is unlikely that most practising biologists would be motivated to formulate their ideas in a non-intuitive language devised for computational execution. Second, the process of specifying the model can identify areas of ignorance or uncertainty in biological understanding which may not be apparent to a computer scientist taking models from the literature. The biological literature frequently articulates biological models in the form of informal diagrams or employs various formal graphical notations (e.g. Kohn molecular interaction maps [8] and Kitano diagrams [7]), but these can be confusing or ambiguous and particularly may conceal or abstract biological knowledge that may be useful for the computational modeller. It is also clear that a two-dimensional static representation may not adequately represent all pertinent features of the dynamic evolution of a temporal and spatially directed process. Indeed these ways of describing a biological process in reality involve the biologist translating a narrative of events into a graphical format.

Prompted by these considerations we here present an approach to biological model specification which is based on a narrative style language. In developing

this language there were three desiderata. First, the language should be biologically intuitive using formalisms and syntax familiar to biologists. Second, the language should exploit the specific advantages of a process calculus based approach to modelling highlighting, for example, the roles of concurrency, dependency and spatial confinement. Thirdly, the language should encourage the biologists to critically examine their understanding in the process of model specification highlighting points of uncertainty.

In the proposed framework, biologists can specify their model by providing a textual description of the system, containing the list of compartments, the list of entities (i.e. proteins) composing the model, the list of reactions with their rate parameters, and a narrative describing the evolution of the system.

A translation into process calculus (namely, Beta-binders [13]) has been developed and a few models have been specified into the proposed language and translated into Beta-binders. This allowed us to run the models by using the BetaWB simulator [15].

The front-end of the translation is designed to be independent of the language, so that when implementing a translation to other modelling languages it could be reused. The main aim of the current work, in fact, is to define an high level language which hides the implementation details to the modeller, and is generic so that it could be translated into other kinds of formalisms different from Beta-binders (e.g. other process algebras, SBML [5], and possibly differential equations). This would allow us to have a common ground in which models could be specified, but also allow modellers to choose the output language that better fits their aims. In fact it is difficult to find one formal language which is appropriate to describe all kinds of systems, or that could be used to study all aspects of the systems. In addition, starting with the same input model, would make the comparison of different formalisms much easier.

We use as a case study the Gp130 signalling pathway [11]. This was chosen as an example of signalling concurrency which has yet to be explicitly articulated in a process calculus model.

In Sect. 2 the modelling language is described, using the Gp130 pathway as a modelling example. In Sect. 3 the translation algorithm is described, and the generated Gp130 pathway Beta-binders model is shown. Finally, we draw some conclusions on the pros of our approach and on issues that need to be tackled.

2 The Narrative Language

In this section we present the modelling language. The structure of models in the proposed language is inspired by SBML [5], one of the most well known description languages for biology. The main differences between the two languages are the following ones. First, SBML species have only one possible state, while in our language they can have multiple states (this choice was made bearing in mind that our target language, process algebra, is a multistate one. Secondly, SBML is very abstract and it does not represent details about species and reactions, while the proposed language can easily represent them. Finally, the description

of system evolution in SBML is given in terms of reactions with reactants and products, while in our language it is given in terms of a narrative of events.

A model in our language is composed of four sections:

- the description of the biological compartments in which the involved entities can be located during the evolution of the system;
- the description of the entities composing the system;
- the description of the reactions occurring;
- the narrative description of the evolution of the system, i.e. the list of the events occurring.

A *compartment* is identified by an integer number; moreover, its name, size, and number of spacial dimensions can be specified. Compartments could represent cellular or sub-cellular compartments, but also abstract locations.

A *component* (i.e. a protein) is identified by its name, and it can be seen as a list of interaction sites. Each site is defined by a name and a state (e.g. phosphorylated, bound, active, etc.). This choice reflects the fact that each basic event in the evolution of intra-cellular systems is the modification of one interaction site for each protein involved in the reaction. However, since interaction sites are not always known, states can also be associated to the protein itself, to represent modifications involving generic interaction sites. If the position of the protein is relevant, the compartments in which it can be located during the system evolution can be specified. Finally, the initial quantity/concentration of the component should be set.

A *reaction* is identified by an integer number; its type (e.g. phosphorylation, binding, etc.) and the reaction rate parameter should also be specified.

A reliability value can be associated to each numerical value (e.g. rate parameters and initial quantities); it is a percentage value that can be used to distinguish between values that are certain because obtained from wet experiments, and others which are the result of not verified assumptions. Modellers can take this information into account during the important step of model refining by means of parameter space search and sensitivity analysis.

Finally, the evolution of the system is described by means of a *narrative of events*. This narrative is a sequence of basic events, each of which is a textual description of a reaction involving at most two components. Events can be grouped into processes.

An *event* is identified by an integer number; in addition to the textual semi-formal description of the event, the identifier of the reaction associated with the event should be specified. The description of the event is a string of the form ***if*** *condition* ***then*** *event_descr*, with the conditional part being optional. A *condition* is a string of the form *component* ***is*** *state*, *component.site* ***is*** *state*, *component* ***is in*** *compartment*, etc. Multiple conditions can be specified by separating them with the keyword ***and***. *Event_descr* is a string of the form *component reaction* for monomolecular events, or *component reaction component* for bimolecular events, where *reaction* can be for example **phosphorylates**, **dephosphorylates**, **binds**, **activates**. As mentioned before, we assume that each

event involves an interaction or modification of one site for each involved protein. If the exact site is known, it can be specified with a clause of the form ***on*** *sites*; otherwise, it is assumed that one of the component's defined states is involved. Hence, conceptually, each step involves a single site; however, a list of sites (separated by semicolons) can be specified as a shortcut for simultaneous steps involving different sites (e.g. simultaneous phosphorylation of two sites).

In real life, two events occurring in a system of interacting entities can be either concurrent (independent events, e.g. events involving different proteins), or sequential (one event can occur only after the other one has occurred, e.g. a phosphorylation of a site of a protein allowed only after it is bound to another protein), or alternative (if one event occurs, the other one cannot occur, e.g. binding of competing ligands to a receptor).

Consequently, in our language it is necessary to distinguish between concurrent, sequential, and alternative events. If a reaction event is alternative to another one, the identifier of the alternative event should be specified. Conditions should be used to enforce the ordering of sequential events. Events that are not explicitly declared either alternative or sequential, are considered independent and are treated as concurrent events.

The grammar of the language follows. Some attributes are left as optional and are not actually used in the translation to Beta-binders, because they refer to details that cannot be easily handled in Beta-binders, or about which we are not interested at the moment; they have been added to allow them to be used in an hypothetical translation to other languages that might handle those details.

$$
\begin{array}{lll}
\langle model \rangle & ::= & \langle comparts\_decl \rangle \langle compons\_decl \rangle \langle reacts\_decl \rangle \langle procs\_decl \rangle \\
\langle comparts\_decl \rangle & ::= & \text{Compartments } \langle comparts\_list \rangle \\
\langle compons\_decl \rangle & ::= & \text{Components } \langle compons\_list \rangle \\
\langle reacts\_decl \rangle & ::= & \text{Reactions } \langle reacts\_list \rangle \\
\langle procs\_decl \rangle & ::= & \text{Narrative } \langle procs\_list \rangle \\
 & & \\
\langle comparts\_list \rangle & ::= & \langle compartment \rangle \\
 & | & \langle compartment \rangle \langle comparts\_list \rangle \\
\langle compons\_list \rangle & ::= & \langle component \rangle \\
 & | & \langle component \rangle \langle compons\_list \rangle \\
\langle reacts\_list \rangle & ::= & \langle reaction \rangle \\
 & | & \langle reaction \rangle \langle reacts\_list \rangle \\
\langle procs\_list \rangle & ::= & \langle proc \rangle \\
 & | & \langle proc \rangle \langle procs\_list \rangle \\
 & & \\
\langle compartment \rangle & ::= & (\langle id \rangle, \langle compart\_name \rangle, \langle opt\_size \rangle, \langle opt\_unit \rangle, \langle opt\_dim \rangle) \\
\langle component \rangle & ::= & (\langle name \rangle, \langle opt\_inform\_descr \rangle, \langle opt\_sites\_def \rangle, \\
 & & \langle opt\_states\_def \rangle, \langle opt\_comparts\_def \rangle, \langle initial\_quantity \rangle) \\
\langle reaction \rangle & ::= & (\langle id \rangle, \langle react\_type \rangle, \langle rate\_const \rangle) \\
\langle proc \rangle & ::= & \text{Process } \langle opt\_inform\_descr \rangle \langle events\_list \rangle \\
\langle events\_list \rangle & ::= & \langle event \rangle \\
 & | & \langle event \rangle \langle events\_list \rangle \\
\langle event \rangle & ::= & (\langle id \rangle, \langle form\_descr \rangle, \langle react\_id \rangle, \langle opt\_altern\_event \rangle)
\end{array}
$$

⟨*opt_sites_def*⟩ ::=
| ⟨*sites_def*⟩

⟨*sites_def*⟩ ::= ⟨*site_def*⟩
| ⟨*site_def*⟩; ⟨*sites_def*⟩

⟨*site_def*⟩ ::= ⟨*name*⟩ : ⟨*state_name*⟩ : ⟨*is_active*⟩

⟨*opt_states_def*⟩ ::=
| ⟨*states_def*⟩

⟨*states_def*⟩ ::= ⟨*state_def*⟩
| ⟨*state_def*⟩; ⟨*states_def*⟩

⟨*state_def*⟩ ::= ⟨*state_name*⟩ : ⟨*is_active*⟩

⟨*opt_comparts_def*⟩ ::=
| ⟨*comparts_def*⟩

⟨*comparts_def*⟩ ::= ⟨*compart_def*⟩
| ⟨*compart_def*⟩; ⟨*comparts_def*⟩

⟨*compart_def*⟩ ::= ⟨*id*⟩ : ⟨*is_active*⟩

⟨*initial_quantity*⟩ ::= (⟨*quantity*⟩, ⟨*opt_reliability*⟩)

⟨*rate_const*⟩ ::= (⟨*rate*⟩, ⟨*opt_unit*⟩, ⟨*opt_reliability*⟩)

⟨*form_descr*⟩ ::= ⟨*event_descr*⟩
| if ⟨*conds*⟩ then ⟨*event_descr*⟩

⟨*conds*⟩ ::= ⟨*cond*⟩
| ⟨*cond*⟩ and ⟨*conds*⟩

⟨*cond*⟩ ::= ⟨*names*⟩ is ⟨*state_name*⟩
| ⟨*names*⟩ is not ⟨*state_name*⟩
| ⟨*names*⟩ is in ⟨*id*⟩
| ⟨*names*⟩ is not in ⟨*id*⟩

⟨*names*⟩ ::= ⟨*name*⟩
| ⟨*name*⟩.⟨*name*⟩
| ⟨*name*⟩; ⟨*names*⟩
| ⟨*name*⟩.⟨*name*⟩; ⟨*names*⟩

⟨*sites*⟩ ::= ⟨*name*⟩
| ⟨*name*⟩; ⟨*sites*⟩

⟨*event_descr*⟩ ::= ⟨*complex_name*⟩⟨*bimol_react*⟩⟨*complex_name*⟩ on ⟨*sites*⟩
| ⟨*complex_name*⟩⟨*bimol_react*⟩⟨*complex_name*⟩
| ⟨*complex_name*⟩⟨*monomol_react*⟩ on ⟨*sites*⟩
| ⟨*complex_name*⟩⟨*monomol_react*⟩
| ⟨*complex_name*⟩ relocates from ⟨*id*⟩ to ⟨*id*⟩
| ⟨*complex_name*⟩ degrades
| ⟨*complex_name*⟩ degrades ⟨*complex_name*⟩
| ⟨*complex_name*⟩ synthesises ⟨*complex_name*⟩
| ⟨*complex_name*⟩ homodimerises
| ⟨*complex_name*⟩ dehomodimerises
| ⟨*complex_name*⟩ dimerises with ⟨*complex_name*⟩
| ⟨*complex_name*⟩ dedimerises from ⟨*complex_name*⟩

⟨*complex_name*⟩ ::= ⟨*name*⟩
| ⟨*name*⟩ : ⟨*complex_name*⟩

$\langle id \rangle$ ::= Int
$\langle opt\_size \rangle$ ::=
| Int
$\langle opt\_unit \rangle$ ::=
| Str

$\langle opt\_dim \rangle$ ::=
| Int
$\langle name \rangle$ ::= Ide
$\langle opt\_inform\_descr \rangle$ ::=
| Str
$\langle quantity \rangle$ ::= Int | $Real$
$\langle opt\_reliability \rangle$ ::=
| Int
$\langle rate \rangle$ ::= Int | $Real$ | inf
$\langle react\_id \rangle$ ::= Int
$\langle opt\_altern\_event \rangle$ ::=
| alternative to $\langle id \rangle$
$\langle is\_active \rangle$ ::= $Bool$

$\langle compart\_name \rangle$ ::= nucleus | cytosol | exosol
| cellMembrane | nucleusMembrane | Ide

$\langle react\_type \rangle$ ::= phosphorylation | dephosphorylation
| binding | unbinding
| homodimerization | dehomodimerization
| dimerization | dedimerization
| activation | deactivation
| hydrolysis | dehydrolysis
| degradation | synthesis | relocation

$\langle state\_name \rangle$ ::= phosphorylated | bound | active | hydrolysed | dimer

$\langle bimol\_react \rangle$ ::= phosphorylates | dephosphorylates | binds | unbinds
| activates | deactivates | hydrolyses | dehydrolyses

$\langle monomol\_react \rangle$ ::= phosphorylates | dephosphorylates | hydrolyses | dehydrolyses

2.1 Case Study: a Narrative Model of the Gp130 Signalling Pathway

In this section we describe a model of the Gp130 signalling pathway in the proposed language.

The Gp130 pathway is the subject of significant clinical and biological interest not least due to the key role it plays in human fertility, neuronal repair and haematological development [11]. Robust experimental platforms are available and there is much information on pathway components and behaviour. Various features of this pathway make it an attractive case study for the modelling approach. First, the Gp130 signalling pathway involves a family of private and

public receptors where biological outcomes are dictated by the relative occupation of different receptor combinations. Thus, it exhibits signalling concurrency, which has not been explicitly tackled by computational modelling approaches yet. Moreover, a key feature of the Gp130 system is nuclear/cytoplasmic shuttling of key signalling components whose dynamics have been investigated using imaging techniques; hence, this is an excellent test case for developing spatially confined compartment models which can be tested against high quality datasets.

The model is made of six entities: two ligands (LIF and OSM), three membrane-bound receptors (gp130, LIFR and OSMR) and one effector (STAT3).

Four compartments are involved in the system: the exosol (the extracellular space, where the two ligands are located), the cell membrane (location of the receptors), and the cytosol and the nucleus (compartments between which the effector shuttles). Table 1 lists the compartments with their attributes.

Table 1. List of compartments (The volumes are calculated based on the average cell radius and ratio between intra-cellular compartments volumes stated in [1]).

id	name	size	unit of measure	dimensions
1	exosol	$9.95 \cdot 10^{-12}$	l	3
2	cellMembrane	$12.57 \cdot 10^{-8}$	dm^2	2
3	cytosol	$2.10 \cdot 10^{-12}$	l	3
4	nucleus	$0.25 \cdot 10^{-12}$	l	3

The components representing the involved proteins are listed in Table 2. The definition of each component contains its name, an informal description, the list of sites (each defined by its name, state, and a boolean flag specifying whether it is or not in an active state at system initialisation), the list of protein states (each defined by its name and the active/inactive flag), the list of compartments in which the protein could be located (each defined by its identifier referring to the definition in Table 1, and the active/inactive flag), and finally its initial quantity and the reliability of this numerical value.

Table 3 contains the list of reactions. Each reaction definition contains an identifier, its type, and a rate parameter with its unit of measure and reliability. The rate parameter is the reaction kinetic constant (K_a, K_{off}, etc.). In order to be used in stochastic models (such as Beta-binders), the given kinetic constants need to be translated into stochastic reaction rates. As described in [3], for first order reactions (unbinding, phosphorylation and relocation events) the stochastic rate r is equal to the kinetic rate k; for second order heterologous reactions (binding events) $r = \frac{k}{V \cdot N_A}$, where V is the reaction volume and N_A is Avogadro's number, while for second order homologous reactions (dimerization events) $r = \frac{2 \cdot k}{V \cdot N_A}$.

Finally, the narrative of events is shown in Table 4. The events are grouped into processes, relative to the binding/unbinding of ligand/receptor pairs, the downstream LIF and OSM pathways, and the downstream STAT3 pathway, which starts with the binding of STAT3 to one of the receptors and ends in its translocation into the nucleus. Each event is described by an identifier, the semiformal description, the identifier of the reaction referring to the definition in Table 3 and the optional identifier of the alternative event.

Table 2. List of components (The initial quantities values for the ligands LIF and OSM are calculated based on the known extracellular concentration values, 500pM).

name	descr	site	site_state	site_act	state	state_act	compart	compart_act	initial_quant	reliab
LIF	ligand				bound	false	1	true	3000	100%
OSM	ligand				bound	false	1	true	3000	100%
gp130	receptor	LIF	bound	false	dimer	false	2	true	1000	50%
		OSM	bound	false						
		Y767	phospho	false						
		Y814	phospho	false						
		Y905	phospho	false						
		Y915	phospho	false						
LIFR	receptor	LIF	bound	false	bound	false	2	true	1000	50%
		OSM	bound	false	dimer	false				
		Y981	phospho	false						
		Y1001	phospho	false						
		Y1028	phospho	false						
OSMR	receptor	OSM	bound	false	bound	false	2	true	1000	50%
		Y917	phospho	false	dimer	false				
		Y945	phospho	false						
STAT3	effector	Y705	phospho	false	bound	false	3	true	5000	0%
		gp130	bound	false	dimer	false	4	false		
		LIFR	bound	false						
		OSMR	bound	false						

Table 3. List of reactions.

id	type	rate	unit of measure	reliability
1	binding	$8 \cdot 10^5$	$M^{-1}s^{-1}$ (k_a)	50%
2	unbinding	$6 \cdot 10^{-4}$	s^{-1} (k_{off})	50%
3	binding	$8 \cdot 10^5$	$M^{-1}s^{-1}$ (k_a)	50%
4	unbinding	$6 \cdot 10^{-3}$	s^{-1} (k_{off})	50%
5	binding	$8 \cdot 10^5$	$M^{-1}s^{-1}$ (k_a)	50%
6	unbinding	$6 \cdot 10^{-3}$	s^{-1} (k_{off})	50%
7	binding	$8 \cdot 10^5$	$M^{-1}s^{-1}$ (k_a)	50%
8	unbinding	$6 \cdot 10^{-4}$	s^{-1} (k_{off})	50%
9	binding	$8 \cdot 10^5$	$M^{-1}s^{-1}$ (k_a)	50%
10	unbinding	$6 \cdot 10^{-4}$	s^{-1} (k_{off})	50%
11	dimerization	inf	$M^{-1}s^{-1}$	50%
12	phosphorylation	0.2	s^{-1} (k_{cat})	50%
13	binding	10^4	$M^{-1}s^{-1}$ (k_a)	50%
14	dimerization	inf	$M^{-1}s^{-1}$ (k_a)	50%
15	phosphorylation	0.2	s^{-1} (k_{cat})	50%
16	unbinding	10^{-3}	s^{-1} (k_{off})	50%
17	homodimerization	inf	s^{-1}	50%
18	relocation	10	min ($t_{1/2}$)	50%
19	relocation	100	min ($t_{1/2}$)	50%

3 The Translation into Beta-binders

3.1 Beta-binders

Beta-binders [13] is a language, which belongs to the family of the bio-inspired process calculi, strongly inspired by pi-calculus. The main advantage of process

Table 4. List of events.

id	description	react	alt
	LIF-LIFR binding		
1	if LIFR.LIF is not bound and LIF is not bound then LIF binds LIFR on LIF	1	
2	if LIFR.LIF is bound and LIF is bound then LIF unbinds LIFR on LIF	2	
	LIF-gp130 binding		
3	if gp130.LIF is not bound and LIF is not bound then LIF binds gp130 on LIF	3	
4	if gp130.LIF is bound and LIF is bound then LIF unbinds gp130 on LIF	4	
	OSM-LIFR binding		
5	if LIFR.OSM is not bound and OSM is not bound then OSM binds LIFR on OSM	5	1
6	if LIFR.OSM is bound and OSM is bound then OSM unbinds LIFR on OSM	6	
	OSM-OSMR binding		
7	if OSMR.OSM is not bound and OSM is not bound then OSM binds OSMR on OSM	7	
8	if OSMR.OSM is bound and OSM is bound then OSM unbinds OSMR on OSM	8	
	OSM-gp130 binding		
9	if gp130.OSM is not bound and OSM is not bound then OSM binds gp130 on OSM	9	3
10	if gp130.OSM is bound and OSM is bound then OSM unbinds gp130 on OSM	10	
	LIF pathway		
11	if LIFR.LIF is bound then LIFR dimerises with gp130	11	2
12	if gp130 is dimer then gp130 phosphorylates on Y767;Y814;Y905;Y915	12	
13	if LIFR is dimer then LIFR phosphorylates on Y981;Y1001;Y1028	12	
	OSM pathway		
14	if LIFR.OSM is bound then LIFR dimerises with gp130	11	6
15	if OSMR.OSM is bound then OSMR dimerises with gp130	14	8
16	if gp130 is dimer then gp130 phosphorylates on Y767;Y814;Y905;Y915	12	
17	if OSMR is dimer then OSMR phosphorylates on Y917;Y945	12	
	STAT3 pathway		
18	if gp130.Y767 is phospho and STAT3 is in 3 then gp130 binds STAT3 on gp130	13	
19	if LIFR.Y981 is phospho and STAT3 is in 3 then LIFR binds STAT3 on LIFR	13	
20	if OSMR.Y917 is phospho and STAT3 is in 3 then OSMR binds STAT3 on OSMR	13	
21	if STAT3.gp130 is bound then STAT3 phosphorylates on Y705	15	
22	if STAT3.LIFR is bound then STAT3 phosphorylates on Y705	15	
23	if STAT3.OSMR is bound then STAT3 phosphorylates on Y705	15	
24	if STAT3.gp130 is bound and STAT3.Y705 is phospho then gp130 unbinds STAT3	16	
25	if STAT3.LIFR is bound and STAT3.Y705 is phospho then LIFR unbinds STAT3	16	
26	if STAT3.OSMR is bound and STAT3.Y705 is phospho then OSMR unbinds STAT3	16	
27	if STAT3.Y705 is phospho and STAT3 is not bound then STAT3 homodimerises	17	
28	if STAT3 is dimer then STAT3 relocates to 4	18	
29	if STAT3 is in 4 then STAT3 relocates to 3	19	

calculi is that, in addition to simulation, they allow for analysing statically the models (e.g. causality, locality, equivalence and reachability analysis). They also allow the modeller to easily execute so called in silico genetics experiments, i.e. to modify some components of the system, and execute simulations to verify the modified system behaviour.

Beta-binders language is quite new, but much work has been done with it in the past few years, and a simulator has also been recently developed [15].

Beta-binders was developed to better adhere to the structure and dynamics of biological systems. By introducing the concept of *affinity*, the calculus relaxes the *key-lock* model of interaction, commonly assumed in classical process calculi, and hence it permits us to model more correctly domains and interactions between enzymes and small molecules based on their types and affinities. In Beta-binders, *pi-processes* are encapsulated into *boxes* (also called *bio-processes*) with interaction capabilities, represented by specialised binders (called *beta binders*).

Beta binders have the form $\beta(x : \Gamma)$ (active) or $\beta^{\mathrm{h}}(x : \Gamma)$ (hidden) where the name x is the subject of the beta binder and Γ represents the type of x. The actions that bio-processes can execute are communications ($x(y)$ for input and $\overline{x}\langle y\rangle$ for output) and operations to manipulate the interaction sites of the boxes ($\mathsf{expose}(x, \Gamma)$, $\mathsf{hide}(x)$ and $\mathsf{unhide}(x)$). The system is a parallel composition of bio-processes that can be either the deadlock bio-process Nil or the elementary bio-process $\mathbf{B}[\, P \,]$.

The reader is referred to [15] for a detailed description of the syntax and semantics of the language.

3.2 The Translation Algorithm

Beta-binders bio-processes are an intuitive representation of proteins, and hence in the translation into Beta-binders we choose to have one bio-process for each component. Component interaction sites, states and locations are translated into beta binders on the interface of the respective bio-processes. Two beta binders are defined for each state and site, to represent their active/inactive state, and only one of them is set to be initially visible depending on the active/inactive flag value.

The translation is subdivided in two main steps. First, one bio-process is created for each component (Algorithm 1). Then, the pi-processes representing the translation of events are added to the bio-processes (Algorithm 7).

As Algorithm 1 describes, the bio-processes are initially empty, and then beta binders are added on their interface.

Algorithm 1 ComponentsToBioprocesses

```
1: for all component ∈ Components do          ▷ each component is a bio-process
2:     component.bioprocess ← new empty bio-process
3:     CompartsToBetaBinders (component)
4:     StatesToBetaBinders (component)
5:     SitesToBetaBinders (component)
6: end for
```

The names of the beta binders representing possible compartments, states and sites are constructed based on their names. As Algorithm 2 shows, the subject of the beta binder representing a compartment is the compartment name, while its type is the compartment name concatenated with the compartment identifier. In the example, STAT3 is located in compartment 3 (the cytosol), hence a visible binder $\beta(cytosol : cytosol\_3)$ is added to $STAT3$ bio-process.

As Algorithm 3 shows, two beta binders are created to represent a state in its active/inactive forms, and the subjects of the beta binders are the state names, while their types are the state names concatenated with the component name. In the example, LIFR can be a dimer (but it is a monomer at system initialisation), hence a visible binder $\beta(monomer : monomer\_LIFR)$ and an hidden binder $\beta(dimer : dimer\_LIFR)$ are added to $LIFR$ bio-process.

As Algorithm 4 shows, two beta binders are created to represent a site in its active/inactive forms, and the subjects of the beta binders are the site name

Algorithm 2 CompartsToBetaBinders (c)

for all $compart \in c.compartments$ **do** ▷ each location is a beta binder
  $binder \leftarrow$ new beta binder in $c.bioprocess$
  $binder.name \leftarrow compart.name^{\wedge} compart.id$ ▷ $\beta(cytosol : cytosol\_3)$
  $binder.type \leftarrow compart.name^{\wedge} compart.id$
  if c is in $compart$ at initial state **then**
    $binder.is\_visible$
  else
    $binder.is\_hidden$
  end if
end for

Algorithm 3 StatesToBetaBinders (c)

for all $state \in c.states$ **do** ▷ each state is a pair of beta binders
  $binder1, binder2 \leftarrow$ new beta binder in $c.bioprocess$
  $binder1.name \leftarrow state.name$ ▷ $\beta(dimer : dimer\_LIFR)$
  $binder2.name \leftarrow state.opposite\_name$
  $binder1.type \leftarrow state.name^{\wedge} c.name$ ▷ $\beta(monomer : monomer\_LIFR)$
  $binder2.type \leftarrow state.opposite\_name^{\wedge} c.name$
  if c is in $state$ at initial state **then**
    $binder1.is\_visible$ and $binder2.is\_hidden$
  else
    $binder1.is\_hidden$ and $binder2.is\_visible$
  end if
end for

concatenated with the states names, while their types are the state names concatenated with the component name and the site name. In the example, site Y981 of receptor LIFR can be phosphorylated (but it is dephosphorylated at system initialisation), hence a visible binder $\beta(Y981\_depho : depho\_LIFR\_Y981)$ and an hidden binder $\beta(Y981\_pho : pho\_LIFR\_Y981)$ are added to $LIFR$ bio-process.

Algorithm 4 SitesToBetaBinders (c)

for all $site \in c.sites$ **do** ▷ each site is a pair of beta binders
  $b1, b2 \leftarrow$ new beta binder in $c.bioprocess$
  $b1.name \leftarrow site.name^{\wedge} site.state.name$ ▷ $\beta(Y981\_pho : pho\_LIFR\_Y981)$
  $b2.name \leftarrow site.name^{\wedge} site.state.opposite\_name$
    ▷ $\beta(Y981\_depho : depho\_LIFR\_Y981)$
  $b1.type \leftarrow site.state.name^{\wedge} c.name^{\wedge} site.name$
  $b2.type \leftarrow site.state.opposite\_name^{\wedge} c.name^{\wedge} site.name$
  if $site$ is in $state$ at initial state **then**
    $binder1.is\_visible$ and $binder2.is\_hidden$
  else
    $binder1.is\_hidden$ and $binder2.is\_visible$
  end if
end for

As Algorithms 5, 6 and 7 describe, each monomolecular event step is translated into one sequential pi-process which is placed into the bio-process representing the involved component, while each bimolecular event step is translated into two sequential pi-processes which are placed into the bio-processes representing the involved components.

The constructed pi-processes consist in sequences of communications, hide and unhide operations (Algorithms 5 and 6). The names of bio-processes, pi-processes, beta binders and types follow a template, so that they are standardised, it is possible to refer to the previously defined beta binders, and no name clash occurs. Reaction rates and types affinities are assigned based on the input definitions. The constructed sequence of events represents a single event step, so the reaction rate is assigned to the first action, while the others are assigned infinite rates (so that after the first one occurs, then the others are immediately executed). The actual sequence of actions depends on the reaction type. In the example, event 17 involves the phosphorylation of sites Y917 and Y945 on OSMR, translated into a sequence of two pairs of hide/unhide actions. Moreover, the phosphorylations can occur only if OSMR is a dimer, hence the sequence is prefixed by an additional hide/unhide pair on the beta binder representing the dimer state. The final pi-process is, therefore, $Pho\_17 = \mathsf{hide}(0.2, dimer)\,.\,\mathsf{unhide}(inf, dimer)\,.\,\mathsf{hide}(inf, Y917\_depho)\,.\,\mathsf{unhide}(inf, Y917\_pho)\,.\,\mathsf{hide}(inf, Y945\_depho)\,.\,\mathsf{unhide}(inf, Y945\_pho)\,.\,Pho\_17$, and it is added to $OSMR$ bio-process.

Algorithm 5 EventToPiprocess (*event*)

if *event.reaction* is relocation **then**
  $piproc \leftarrow \mathsf{hide}(binder\_from)\,.\,\mathsf{unhide}(binder\_to)$
else if *event.reaction* is phosphorylation **then**
  $piproc \leftarrow \mathsf{hide}(binder\_site\_depho)\,.\,\mathsf{unhide}(binder\_site\_pho)$
else if *event.reaction* is dephosphorylation **then**
  $piproc \leftarrow \mathsf{hide}(binder\_site\_pho)\,.\,\mathsf{unhide}(binder\_site\_depho)$
else if ... **then**
end if
if *event.condition* is specified **then**
  $piproc \leftarrow \mathsf{hide}(binder\_cond)\,.\,\mathsf{unhide}(binder\_cond)\,.piproc$
end if

The ordering of the events is given in the following way (Algorithm 7). If two reactions are concurrent, they are translated into processes placed in parallel composition. If, instead, they have to be executed one after the other, the second one is prefixed by one operation on a binder which is unblocked at the end of the first one. If, finally, they are mutually exclusive, both events are prefixed by one operation on a binder which is blocked at the end of the other one.

3.3 Case Study: the Translation of the Gp130 Signalling Pathway Model

A prototype of the tool has been developed and it is currently under integration into BetaWB. The model described in Sect. 2.1 was translated into Beta-binders

Algorithm 6 EventToPiprocesses (*event*)

if *event.reaction* is phosphorylation **then**
  $piproc1 \leftarrow \overline{binder\_event.id}\langle\rangle$
  $piproc2 \leftarrow binder\_site\_depho() \,.\, \mathsf{hide}(binder\_site\_depho) \,.\, \mathsf{unhide}(binder\_site\_pho)$
  AddAffinity $(binder\_event.id, binder\_site\_depho)$
else if ... **then**
end if
if *event.condition* is specified on *event.component*1 **then**
  $piproc1 \leftarrow \mathsf{hide}(binder\_cond) \,.\, \mathsf{unhide}(binder\_cond) \,.piproc1$
end if
if *event.condition* is specified on *event.component*2 **then**
  $piproc2 \leftarrow \mathsf{hide}(binder\_cond) \,.\, \mathsf{unhide}(binder\_cond) \,.piproc2$
end if

Algorithm 7 NarrativeToPiprocesses

for all $ev \in Events$ **do** ▷ each event is one or two sequential pi-processes
  if ev is monomolecular **then** ▷ one pi-process is inserted into the bio-process of the involved component
    $pproc \leftarrow$ EventToPiProcess (ev)
    if ev is alternative to $prev\_ev$ **then**
      $pproc \leftarrow \mathsf{hide}(binder\_prev\_ev\_blocked) \,.\, \mathsf{unhide}(binder\_prev\_ev\_blocked) \,.pproc$
      $prev\_ev.pproc \leftarrow \mathsf{hide}(binder\_ev\_blocked) \,.\, \mathsf{unhide}(binder\_ev\_blocked) \,.prev\_ev.pproc$
    end if
    AddPiprocessInParallel $(ev.component.bioproc, pproc)$
  else if ev is bimolecular **then** ▷ one pi-processes is inserted into the bio-process of each involved component
    $\langle\, pproc1, pproc2 \,\rangle \leftarrow$ EventToPiProcesses (ev)
    if ev is alternative to $prev\_ev$ **then**
      if both ev and $prev\_ev$ involve $ev.component1$ **then**
        $pproc1 \leftarrow \mathsf{hide}(binder\_prev\_ev\_blocked) \,.\, \mathsf{unhide}(binder\_prev\_ev\_blocked) \,.pproc1$
        $prev\_ev\_pproc \leftarrow \mathsf{hide}(binder\_ev\_blocked) \,.\, \mathsf{unhide}(binder\_ev\_blocked) \,.prev\_ev\_pproc$
      end if
      if both ev and $prev\_ev$ involve $ev.component2$ **then**
        $pproc2 \leftarrow \mathsf{hide}(binder\_prev\_ev\_blocked) \,.\, \mathsf{unhide}(binder\_prev\_ev\_blocked) \,.pproc2$
        $prev\_ev\_pproc \leftarrow \mathsf{hide}(binder\_ev\_blocked) \,.\, \mathsf{unhide}(binder\_ev\_blocked) \,.prev\_ev\_pproc$
      end if
    end if
    AddPiprocessInParallel $(ev.component1.bioproc, pproc1)$
    AddPiprocessInParallel $(ev.component2.bioproc, pproc2)$
  end if
end for

by using this prototype, and then simulated by using BetaWB. Figure 1 is a BetaDesigner screenshot showing the graphical visualisation and part of the imported Beta-binders code which has been obtained from the translation of the Gp130 pathway model.

We do not present in this work any biologically relevant result: in order to achieve this final goal, more details on the pathway dynamics should be taken into account and precise information on reaction rates should be acquired. Moreover, some aspects of the translation should be improved for the tool to be practically used for translation of complex models. This work was primarily meant to be a sort of feasibility study. We believe that the proposed language, together with the automatic translation into Beta-binders and the existing simulator, allows the modeller to describe biological systems in simple words, simulate the

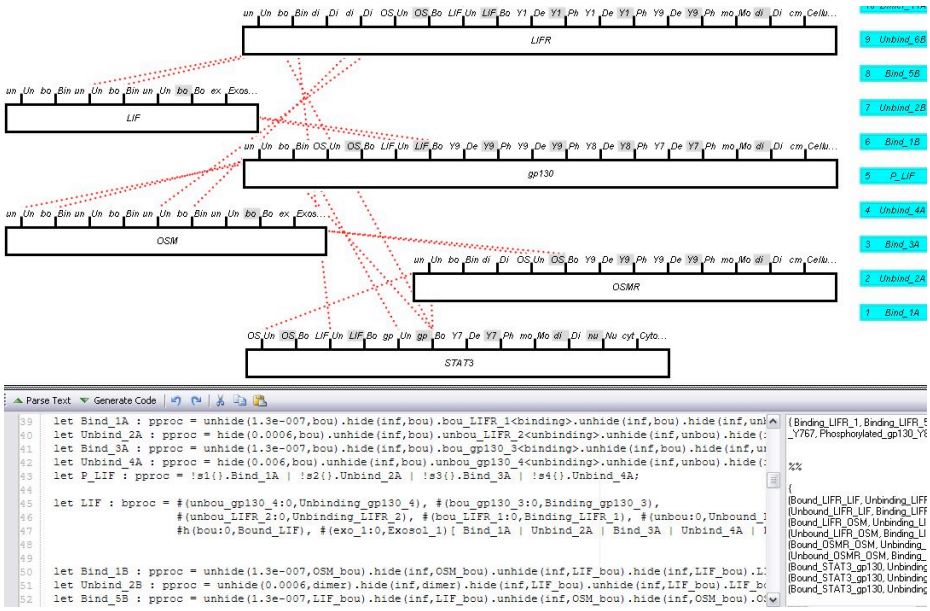

Fig. 1. The Gp130 pathway Beta-binders model imported in BetaDesigner.

model, and obtain sensible results. The Gp130 model we have described in this work was developed by a biologist who had no previous experience in modelling. This gives us some confidence that our main goal, which is to have a user-friendly language which biologists feel comfortable with, was achieved.

4 Conclusions and Further Work

The proposed modelling language and the automatic translation into a formal language allow us to hide the formal details from life scientists. Therefore, we believe that life scientists could easily use the textual language to describe systems, and automatically obtain simulation and analysis results.

The choice of which primitives had to be included in our language has been done to have a simple and basic set of events which could be described. The language and the tool can be extended with new constructs and some improvements on the already present ones can be done. For example, the fact that two binders are used to model the active/inactive states of sites and proteins is not optimal, in terms of code readability, of simulation efficiency and biological correctness. The usage of events [15] and biological transactions [4] to tackle this problem is under investigation. We are also considering the suitability of extending Beta-binders with constructs representing conditions on the state of beta binders, which would greatly reduce the number of actions needed to translate conditions for sequential and alternative events.

A formal comparison with other description languages will be done, and the interchangeability with graphical notations should be taken into account.

Finally, we believe that in order to fully benefit of this approach an integration with some of the many biological databases (i.e. the automatic extraction of data and parameters) would be much useful. In addition to this, another interesting aspect is the automatic extraction of information from the simulation output to be used again to refine the input model.

Acknowledgments. The authors wish to thank Nicholas Underhill-Day (Cancer Research UK).

References

1. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular biology of the cell*. Garland Science, 2002.
2. Muffy Calder, Stephen Gilmore, and Jane Hillston. Modelling the Influence of RKIP on the ERK Signalling Pathway Using the Stochastic Process Algebra PEPA. *T. Comp. Sys. Biology*, 7:1–23, 2006.
3. Luca Cardelli. On Process Rate Semantics. Available at http://lucacardelli.name/Papers/On%20Process%20Rate%20Semantics.pdf, 2007.
4. F. Ciocchetta and C. Priami. Beta-binders with Biological Transactions. Technical Report TR-10-2006. Technical report, The Microsoft Research - University of Trento Centre for Computational and Systems Biology, 2006.
5. M. Hucka, A. Finney, H.M. Sauro, H. Bolouri, J.C. Doyle, and H. Kitano. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19:524–531, 2003.
6. Boris N. Kholodenko. Cell signalling dynamics in time and space. *Nature Reviews Molecular Cell Biology*, 7(3):165–176, 2006.
7. Hiroaki Kitano. A graphical notation for biochemical networks. *Biosilico*, 1(5):169–176, 2003.
8. K. W. Kohn. Molecular interaction map of the mammalian cell cycle control and dna repair systems. *Molecular Biology of the Cell*, 10:2703–34, 1999.
9. M. Kwiatkowska, G. Norman, D. Parker, O. Tymchyshyn, J. Heath, and E. Gaffney. Simulation and verification for computational modelling of signalling pathways. In *Proc. Winter Simulation Conference*, pages 1666–1675. Omnipress, 2006.
10. R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice hall, 1989.
11. Underhill-Day N. and Heath J. K. Oncostatin m (osm) cytostasis of breast tumor cells: Characterization of an osm receptor β-specific kernel. *Cancer Research*, 66(22):10891–10901, 2006.
12. A. Phillips and L. Cardelli. A Correct Abstract Machine for the Stochastic Pi-calculus. In *BioConcur '04, Workshop on Concurrent Models in Molecular Biology*, 2004.
13. C. Priami and P. Quaglia. Operational patterns in Beta-binders. *Transactions on Computational Systems Biology*, 1:50–65, 2005.
14. Corrado Priami, Aviv Regev, William Silverman, and Ehud Shapiro. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25–31, 2001.

15. A. Romanel, L. Dematté, and C. Priami. The Beta Workbench. Technical Report TR-03-2007. Technical report, The Microsoft Research - University of Trento Centre for Computational and Systems Biology, 2007.