Modeling biological complexity using Biology System Description Language (BiSDL)

Modeling biological complexity using Biology System Description Language (BiSDL)

A. Benso, R.Bardini, S. Di Carlo, G. Politano, F. Muggianu
Control and Computer Eng. Dep., Politecnico di Torino
Torino, Italy. Contact: alfredo.benso@polito.it

Eileen Hu
Massachusetts Institute of Technology
Boston, Massachusetts

***Abstract*—The Nets-within-Nets formalism (NWN) allows to model complex biological systems expressing hierarchy, encapsulation, selective communication, spatiality, quantitative mechanisms, and stochasticity. To make NWN usable by life science researchers as well as systems biologists, we introduce a new human-readable description language able to express these same NWN model properties, at different levels of abstraction. BiSDL (Biology Systems Description Language) is derived from the VHDL specification, a standard description language for hardware systems. In this paper we chose a simple signaling pathway example to show how BiSDL enables modeling complex biological systems by separating the behavioral model from the architectural details.**

***Index Terms*—systems biology, modeling language, ontogeny.**

I. Introduction and related work

Biological research is often compartmentalized in very specific sub-domains that, individually, can only describe portions of the overall system. Modeling biological systems requires to understand the relationships between these sub-domains in order to make them work together in the same model. The general lack in understanding the mechanisms at the edges of the different sub-domains is reflected in the existence of different languages proposed in the last years to create biological models.

The *COmputational Modelling in BIology NEtwork* (COMBINE) is an initiative created to coordinate the development of interoperable and non-overlapping standards covering all aspects of modeling in biology [1]. This initiative has identified and classified various languages developed to describe and exchange information about biological models in specific sub-domains of biology.

Table I summarizes the most important characteristics of the considered languages.

TABLE I
Available modeling languages in systems biology

Lang	Main Focus	Mobility	Spatiality	Hierarchy	Ref
SBML	Reaction	Yes	Yes	Yes	[2]
CellML	Cell	Partial	Partial	1-level	[3]
LEMS	Hierarchical	Yes	Yes	Yes	[4]
NeuroML	Neuronal model		Yes	Yes	[5]
BioPAX	Pathways		Partial	Yes	[6]
SBOL	Sequence		Partial	Yes	[7], [8]

All analyzed languages share to some extent the ability to describe entities, processes, and communications between entities or between entities and processes. Nevertheless, none of them has all these characteristics, which are required to describe models of complex and generic biological processes, integrated into a single language.

Our goal is to describe models of complex biological systems including ontogenetic processes, heterogeneous multicellular interactions, spatiality, gene regulation, environment-dependent stimuli and variables, and hierarchical structures of models within models. In this work we present the *Biology System Description Language (BiSDL)*, a new language designed to overcome some of the weaknesses of existing languages.

BiSDL was designed to serve two different purposes:

1) *to be biologist-friendly*: this includes being human-readable, able to model both the behavior and the structure of a biological system, able to support several biological sub-domains, and flexible and modular enough to be used at different levels of abstraction;
2) *to be computation-ready*: differently from other languages, one key BiSDL goal is the simulation of the modeled system. For this reason, the language is designed around the Nets-within-Nets (NWN) formalism, an extension of Petri Nets [9], which allows modeling hierarchical structures and is also suitable for distributed simulations [10]–[12]. Our goal is to create a language that can be automatically translated into a fully executable NWN model.

Although in the development of the language we mainly focused on ontogenetic processes, this choice does not impact the flexibility of the the language. In fact, in its final form, the BiSDL can be used to model a wide spectrum of biological systems. To define the syntax and the structure of the language, we were inspired by the VHSIC Hardware Description Language (VHDL), a very well-known language used to model complex digital circuits and systems [13]. The BiSDL is currently under development and therefore many features are still in a prototype form and may change at any time.

II. Methods

As discussed in the introduction, BiSDL needs to be biologist-friendly and computation-ready. On one hand, this means that some features of the language must be linked to the NWN formalism used to actually simulate the described system. On the other hand, to be biologist-friendly, its syntax

must be able to hide the technicalities of the final implementation, instead allowing focus on the description of the actual biological system.

To illustrate the main features of the language, first we need to briefly describe the NWN formalism, then show how the language is organized to support the description of the structure and the behavior of a real biological system.

A. Nets-within-Nets (NWN)

High-level Petri Nets formalism extends the basic Petri Nets notation by allowing tokens of different types: integers, floats, strings, colored [9]–[11]. The NWN formalism is a high-level Petri Nets formalism introducing an additional type of token named *Net Token*. A Net Token is a token that embeds another Petri Net, and communication between nets at different hierarchical levels is performed through synchronous channels. This implies that Petri Nets can be hierarchically organized, and each layer can be described resorting to the same formalism. This characteristic is extremely important for modeling complex biological systems.

B. BiSDL

In general, in BiSDL, all modules that compose the model of the system are described using the same general template reported in Figure 1. Each module includes: (i) a set of parameters describing the interface of the module, (ii) a set of entities that compose the internal structure of the module, (iii) a set of processes that describe the behavior of the module, and (iv) a set of biological references that link entities and processes to known ontologies.

```
PACKAGE <package1>
...
PACKAGE <packageN>

ONTOLOGY <ONTOLOGY_NAME>=<url>
...
ONTOLOGY <ONTOLOGY_NAME>=<url>

MODULE <name> (<type> <nameparam>,.....)

 BIOLOGICAL REFERENCE
   <ONTOLOGY_NAME>.<ID_WITHIN_THE_ONTOLOGY>

 ENTITIES
   PLACE <name_place>,....
   ENTITY <name_entity>,....
   CHANNEL <name_channel>,....

 INIT
   <place_name>.attribute(<value>)
   <entity_name>.attribute(<value>)
   <entity_name> = <module_entity_name>(<params>,..)

 PROCESSES
   process(  keyword:entities_declaration,
        keyword:entities_declaration,
        ...,
        {transition_function},
        delay(N)
   )

   <module_process_name>(<param>,....)
END
```

Fig. 1. BiSDL template

Since the general idea is to have a language that allows description of a biological system in a way that is both human-readable and ready to be translated into a NWN model, BiSDL syntax has been designed to allow the creation of different types of models: (i) low-level models focusing on NWN structures and (ii) high-level models strictly belonging to the biological domain. The first type of models is mainly used to create building blocks to be included in a set of *language libraries*.

Figure 1 shows a typical BiSDL template. It begins with the `PACKAGE` declarations, which specify the libraries that must be included. Libraries store the domain specific pre-coded modules that allow an easy and straightforward description of biological entities.

The `ONTOLOGY` directives are next. They are used to unambiguously associate the module, and its entities, with known biological entities. To do this, BiSDL requires specification of a set of urls to recognized sources of knowledge (e.g., Gene Ontology [14], Pathway Ontology [15]).

Following is the `MODULE` section, which contains the actual description of the module. A module name must be unique inside the package it belongs to. If necessary, the module declaration is followed by a list of parameters. The value of these parameters must be later specified when the module is instantiated in a model in order to characterize the specific instance. From a biological point of view, a module could correspond to any biological entity or process at any hierarchical level. For example, a module can describe a simple protein, a gene, a complete pathway, a metabolic subnetwork, an entire cell, or a biological process (e.g., gene transcription, functional activation of a protein or its degradation).

Before describing the core structure and behavior of the module, BiSDL requires specification of its `BIOLOGICAL REFERENCE`, i.e., the unique link to an element of one of the ontologies declared in the `ONTOLOGY` section. This is performed according to the following syntax:

```
<ONTOLOGY_NAME>.<ID_WITHIN_THE_ONTOLOGY>
```

Inside each `MODULE`, BiSDL allows description of:

- the *behavioral domain*, which is used to describe the algorithmic behavior of the module. Depending on the abstraction level of the module, it can describe Petri Nets dynamics, or high-level biological functionalities;
- the *structural domain*, which is composed of an interconnection of other modules. This domain allows their instantiation and declaration of how they are interconnected. Again, depending on the abstraction level, this domain can be used to describe a wide range of structures: from interactions between biological sub-systems to network motifs implemented as Petri Nets, without any biological reference.
- the *spatial domain*: BiSDL allows placement of each module of the system in a 3D grid. This feature can be used to model biological districts, to model movement, speed, or to trigger location-based biological constraints.

Following the biological identification of the module, the `ENTITIES` sub-section instantiates all entities that will be used inside the module. `ENTITIES` can be considered the module's components. Their interaction is what defines the behavior of the module. Each entity will be linked to an actual

implementation (MODULE) in the INIT section. ENTITIES can be of three types:

- ENTITY: a module implemented in one of the libraries declared in the PACKAGE section;
- PLACE: a Petri Nets place;
- CHANNEL: a communication component that allows the synchronous communication between two different Petri Nets.

Thanks to these three types of elements, the internal architecture of a module can hierarchically use both the Petri Nets formalism elements and/or other modules. Petri Nets transitions, which are the mechanism that enables the dynamics of Petri Nets, will be introduced later in the PROCESSES section. In an ideal hierarchical model, only the lowest level modules will explicitly reflect the Petri Nets formalism.

After being instantiated, each entity must be initialized in the INIT section. In particular, for each ENTITY it is necessary to declare its actual origin within one of the included libraries, and to set, if present, the required parameters. In the case of a PLACE, the INIT section is used to set a name, the initial token marking (the type of token(s), their value, and their quantity), and the ontology id. Tokens can be of type int, float, double, string, black token (no type), or a complete module (user-defined or taken from one of the included libraries).

Another attribute that can be set in the INIT section for entities of type ENTITY is the relative speed at which they must be simulated. The speed can be a fraction or a multiple of the time unit, which will be used during simulation. This unit will be chosen as the time cycle of the faster module in the model.

For both ENTITY and PLACE modules it is also possible to specify a location within the BiSDL 3D grid (the grid is always present, and when the user doesn't provide coordinates then default coordinates are assigned to entities).

CHANNEL entities, which are used to create communication channels between different networks (even at different hierarchical levels) are not included in this section because they are used as arguments for PROCESSES implementations.

After instantiating and initializing all entities of the model, the PROCESSES section is used to create the behavioral domain of the model. In particular, this section defines the dynamic relationships and functions between entities, the transformations that can be applied to each entity, and the rules by which the behavioral model works. A typical behavioral rule is a Petri Nets transition, which can be defined as follows:

```
PROCESS(keyword:entities_declaration,
                keyword:entities_declaration,...,
                {transition function}, delay(N))
```

The keyword:entities_declaration pairs are used to specify the input/output properties of the transition: the keyword, which can be IN, OUT, or BI, specifying the direction of the arcs from/to the entities listed in the entities_declaration as:

```
{entity name[arcs_number],...,
entity name[arcs_number]}
```

Incoming (IN) arcs are responsible for consuming tokens from places, whereas outgoing arcs (OUT) are responsible for producing tokens into a place; bidirectional arcs (BI) are used to perform a check on tokens presence inside a place.

Transitions, according to the Petri Nets formalism, have both enabling and activation functions, which can be specified in the transition function parameter. The time that needs to elapse from the enabling to the activation of the transition is set by the delay(N) attribute as a fraction of the time unit defined in the INIT section.

CHANNEL entities use a particular transition firing function, which must be associated with a different transition in another network. The channel has two main properties: the direction of the activation and the direction of the token exchange. To set the direction of the activation it is necessary to specify which is the transition that starts the communication (the down-link) and which is the transition that receives the communication in a synchronous way (the up-link). If no specification about value or type of tokens is made, random tokens are exchanged through the channel.

The PROCESS block described above is a simple construct, directly related to the Petri Nets formalism, mostly used to build library modules. Obviously, the PROCESSES section is also used to define higher-level complex relations between modules available in the libraries and defined in the PACKAGE section. In this case the syntax is simply the process function name followed by the required arguments. Typically, these arguments are the entities that need to be put in relation. All these functions usually involve, at the low level, manipulation of tokens.

III. Results

In this section we will present an example of a BiSDL modeling of the *RAS/RAF/MAPK* signaling pathway shown in Figure 2. The pathway is part of a much bigger model but we believe it is enough, in this paper, to show the modeling potential of the BiSDL language.

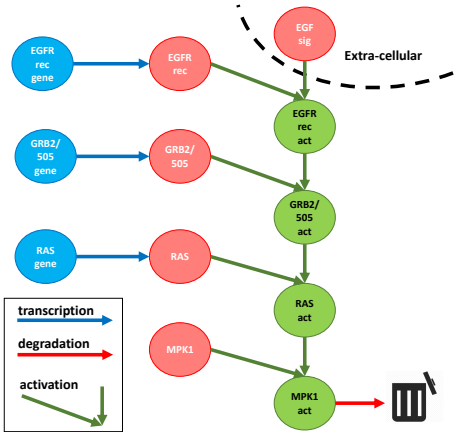

Fig. 2. *RAS/RAF/MAPK* signalling pathway

In summary, after a regulatory cascade, the pathway is responsible for the generation of the *Map Kinase Kinase* active

protein. The pathway is activated when an extracellular *EGF-like* signal binds with its receptor (transcribed from the *EGFR-like* receptor gene). The binding activates the *EGFR-like* receptor, which then binds and activates the *GRB2/505* protein. This protein reacts with and activates the *RAS* protein, which finally binds and activates the *MPK1* protein generating the final *MPK1* activated protein. Physiologic protein degradation is specified for *MPK1* activated, only.

In the BiSDL description of the regulatory cascade of the pathway we want to include the transcription, activation, and degradation mechanisms. Figure 3 shows the code of the top-level description of the pathway model. First of all, the code includes the `PathwayRegulations` library (which includes the basic blocks discussed later in his section), and the ontologies that will be used to uniquely identify the biological entities in the model.

In the `MODULE` declaration the code defines the model interface, i.e., the set of entities that constitute the input/output interface of the module. In this case the interface is composed of four entities: an *EGF-like* signal, the *MPK1* protein and its activated form, and an *EGFR-like* receptor. Then, after linking the model to a unique entry in one of the included ontologies, the code lists, in the `ENTITIES` section, all the entities required to model the pathway: the *RAS* and *GRB-2/505* proteins (with their activated forms), and an *EGFR-like* activated receptor protein.

The `INIT` section is then used to link each of these entities to an actual model. In this case each entity is linked to a `simple_protein model`. As shown in Figure 4, the `simple_protein` is modeled as a simple place in a Petri Net. The `simple_protein` model requires, as parameters, the name of the protein, an ID from one of the included ontologies, and a token type. In this example, each protein is initialized with three black tokens (the standard Petri Nets tokens). Therefore, the `INIT` section creates five separate places (one for each protein) containing three tokens each. The reason for initializing the places with three tokens is simple. As shown later in this section, the activation of a protein takes place when at least four tokens are present in each input place (four is a arbitrarily chosen number, it does not have any biological correlation). The last token in the "protein place" is generated by the transcription module, which models the transcription of a gene into the corresponding protein. In this way are able to simulate all mechanisms of the pathway, from the transcription of genes, through the activation of their proteins, to the degradation of the final product.

The relationships among the five entities are modeled in the `PROCESSES` section. The three transcription processes, model the fact that the *RAS*, *GRB-2/505*, and *EGFR-likeRec* proteins are actually transcribed by their respective genes. As shown in Figure 4, the transcription module creates a new place modeled by the `simple_gene` module. Therefore, with the three `transcription` processes, the model includes three additional places, each initialized with one black token, modeling the genes that transcribe the required proteins. The `PROCESSES` section of the transcription model contains only

```
PACKAGE PathwaysRegulations
ONTOLOGY Proteins =
"https://research.bioinformatics.udel.edu/pro/entry/PR%3A"
ONTOLOGY Gene     =
"https://www.ncbi.nlm.nih.gov/gene?cmd=Retrieve&dopt=full_
report&list_uids="
ONTOLOGY PathwayOntology    =
"http://bioportal.bioontology.org/ontologies/PTS/?p=classes
&conceptid=http%3A%2F%2Fscai.fraunhofer.de%2FPWDICT%23"

MODULE RAS_RAF_MAPK_sig_path (ENTITY EGF-like_signal,
                              ENTITY mpk_1_act,
                              ENTITY mpk_1,
                              ENTITY EGFR-likeRec)

  BIOLOGICAL REFERENCE
    PathwayOntology.ID0176

  ENTITIES
    ENTITY RAS, RAS_act, EGFR-likeRec_act
    ENTITY GRB-2/505, GRB-2/505_act

  INIT
    GRB-2/505 = simple_protein("GRB2/505",
                               Proteins.000008220,
                               black_token()*3)
    RAS = simple_protein("RAS",Proteins.000013743,
                         black_token()*3)
    GRB-2/505_act = simple_protein("GRB-2/505_act",
                    Proteins.000008220,black_token()*3)
    RAS_act = simple_protein("RAS_act",
                  Proteins.000013743,black_token()*3)
    EGFR-likeRec_act = simple_protein("EGFR-likeRec_act",
                  Proteins.000006933,black_token()*3)

  PROCESSES
    transcription(RAS, Gene.3845)
    transcription(EGFR-likeRec, Gene.1956)
    transcription(GRB-2/505, Gene.2885)
    activation(EGFR-likeRec, EGF-like_signal,
               EGFR-likeRec_act)
    activation(GRB-2/505, EGFR-likeRec_act,
               GRB-2/505_act )
    activation(RAS, GRB-2/505_act, RAS_act)
    activation(mpk_1, RAS_act, mpk_1_act)
    degradation(mpk_1_act, 100)
END
```

Fig. 3. *RAS/RAF/MAPK* signalling pathway BiSDL model

one process that models a transition that can be read as: *"if a token is present in the 'gene' place, then a new token is generated in the 'protein' place"*.

After creating the transcription model, the main signaling module creates the `activation` processes. An activation process models the transformation of a protein into its active form. As shown again in Figure 4, the `activation` module requires three entities to participate in the process: the protein in its inactive form (`protein_to_activate`), an activated protein (`protein_active`) that enables the activation mechanism, and the final output protein in its active form (`protein_act`). The only process in the module is a transition, which creates a token in the `protein_act` place when at least four tokens are present in the `protein_to_activate` and `protein_active` places.

The last process included in the signaling pathway model is the degradation process, which is used to model the `mpk_1_act` degradation. This process consumes one token from the "input" protein place every 100 simulation cycles.

A prototype version of a BiSDL compiler is currently able to translate BiSDL models into executable Python code based on the SNAKES[1] library [16].

The BiSDL compiler is able to translate each BiSDL module into a Python class by extending one of the basic classes of the SNAKES library. Figure 5 shows the automatically generated

[1]an efficient library for the design and simulation of Petri-Nets

```
MODULE simple_protein(STRING name, STRING ID, TOKEN token)
  BIOLOGICAL REFERENCE
    ID
  ENTITIES
    PLACE protein
  INIT
    protein.name = name
    protein.token(token)
END
```

```
MODULE simple_gene (STRING name, TOKEN token, STRING ID)
  BIOLOGICAL REFERENCE
    ID
  ENTITIES
    PLACE gene
  INIT
    gene.name = name
    gene.token( token )
END
```

```
MODULE transcription (ENTITY protein, STRING ID)
  ENTITIES
    PLACE gene
  INIT
    gene = simple gene(protein.name + "_wt",ID,
                       black_token())
  PROCESSES
    process(BI:{gene}, OUT:{protein} )
END
```

```
MODULE degradation (ENTITY protein, INT n)
  PROCESSES
    process(IN:{protein}, delay(n) )
END
```

```
MODULE activation (ENTITY protein_to_activate,
                   ENTITY protein_active,
                   ENTITY protein_act)
  PROCESSES
    process(BI:{protein_active[4],protein_to_activate[4]},
            IN:{protein_to_activate}, OUT:{protein_act} )
END
```

Fig. 4. BiSDL library elements

Python code of the `transcription` process.

```
class transcription:
    def __init__(self ,protein,ID,net):
                gene = simple_gene(protein.module_place.name + "_wt",
                    ID, [BlackToken()], net)

        process_func0=ProcessFunction()

        net.process("transcription", process_func0,
          output_places = [protein], bidir_places=[gene])
```

Fig. 5. Python code of the `transcription` processa

The compiled Python model of the BiSDL system can then be executed to simulate the dynamics of the system. This in turns allow biologists to run "what if" simulations, where, "faults" (e.g., mutations, or errors in some biological mechanisms) can be introduced in the code to understand their impact on the system's dynamics.

IV. Conclusion

BiSDL is an ongoing effort to create a new language able to model complex biological use cases. BiSDL is intended to overcome several weaknesses of existent languages and to group their strengths. Whereas other languages are based on the XML syntax, we chose a much more human-readable format to make the language biologist-friendly. BiSDL enables description of geometry and mobility of entities involved in the model, and it is based on a hierarchical organization.

References

[1] COMBINE, "The COMBINE standards," [Online] https://co.mbine.org/standards, Aug. 2018.

[2] C. Chaouiya, D. Bérenguier, S. M. Keating, A. Naldi, M. P. Van Iersel, N. Rodriguez, A. Dräger, F. Büchel, T. Cokelaer, B. Kowal *et al.*, "Sbml qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools," *BMC systems biology*, vol. 7, no. 1, p. 135, 2013.

[3] C. M. Lloyd, M. D. Halstead, and P. F. Nielsen, "Cellml: its future, present and past," *Progress in Biophysics and Molecular Biology*, vol. 85, no. 2, pp. 433 – 450, 2004, modelling Cellular and Tissue Function. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S007961070400015X

[4] R. C. Cannon, P. Gleeson, S. Crook, G. Ganapathy, B. Marin, E. Piasini, and R. A. Silver, "Lems: a language for expressing complex biological models in concise and hierarchical form and its use in underpinning neuroml 2," *Frontiers in neuroinformatics*, vol. 8, p. 79, 2014.

[5] P. Gleeson, S. Crook, R. C. Cannon, M. L. Hines, G. O. Billings, M. Farinella, T. M. Morse, A. P. Davison, S. Ray, U. S. Bhalla *et al.*, "Neuroml: a language for describing data driven models of neurons and networks with a high degree of biological detail," *PLoS computational biology*, vol. 6, no. 6, p. e1000815, 2010.

[6] R. N. Goldberg, M. Cary, and E. Demir, "Biopax a community standard for pathway data sharing," *Nature Biotechnology*, vol. 28, no. Nature Biotechnology, 2010.

[7] M. Galdzicki, K. P. Clancy, E. Oberortner, M. Pocock, J. Y. Quinn, C. A. Rodriguez, N. Roehner, M. L. Wilson, L. Adam, J. C. Anderson *et al.*, "The synthetic biology open language (sbol) provides a community standard for communicating designs in synthetic biology," *Nature biotechnology*, vol. 32, no. 6, p. 545, 2014.

[8] R. S. Cox, C. Madsen, J. A. McLaughlin, T. Nguyen, N. Roehner, B. Bartley, J. Beal, M. Bissell, K. Choi, K. Clancy *et al.*, "Synthetic biology open language (sbol) version 2.2. 0," *Journal of integrative bioinformatics*, vol. 15, no. 1, 2018.

[9] R. Valk, "Object petri nets: Using the nets-within-nets paradigm, advanced course on petri nets 2003 (j. desel, w. reisig, g. rozenberg, eds.), 3098," *Appendix A: Proof of Theorem*, vol. 3, 2003.

[10] R. Bardini, G. Politano, A. Benso, and S. D. Carlo, "Using multi-level petri nets models to simulate microbiota resistance to antibiotics," in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Nov 2017, pp. 128–133.

[11] R. Bardini, A. Benso, S. Di Carlo, G. Politano, and A. Savino, "Using nets-within-nets for modeling differentiating cells in the epigenetic landscape," in *International Conference on Bioinformatics and Biomedical Engineering*. Springer, 2016, pp. 315–321.

[12] R. Bardini, G. Politano, A. Benso, and S. Di Carlo, "Multi-level and hybrid modelling approaches for systems biology," *Computational and Structural Biotechnology Journal*, 2017.

[13] D. L. Perry, *VHDL (2Nd Ed.)*. New York, NY, USA: McGraw-Hill, Inc., 1993.

[14] Gene Ontology Consortium, "Gene ontology," [Online] http://www.geneontology.org/, Aug. 2018.

[15] BioPortal, "Pathway onthology," [Online] http://bioportal.bioontology.org/ontologies/PW?p=classes&conceptid=root, Aug. 2018.

[16] F. Pommereau, "Snakes: a flexible high-level petri nets library (tool paper)," in *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 2015, pp. 254–265.