Master Thesis

BiSDL

A new Biological Systems Description Language

Flavia Muggianu

Supervised by
Alfredo Benso
Stefano Di Carlo

Final Project Report for the
Master in Computer Engineering

Computer Engineering
Polytechnic of Turin
Italy, Turin
July 2018

Acknowledgments

Questa tesi segna la fine di un percorso, possibile grazie a tante persone. Un enorme ringraziamento a:
I miei genitori, che mi hanno dato ogni possibilità.
Giulia e Claudio, che mi hanno sempre saputo consigliare la cosa giusta.
Rosalba, con la quale ho superato tutte le avventure e sventure Torinesi.
Lo Sciame, che anche da lontano è sempre stato vicino.
Alessandro, Francesco, Luca, Nicolò e Stefano, che hanno reso il Politecnico meno faticoso.
Roberta, coach e persona indispensabile in quest'ultimo anno.
Alberto, Alessandro e Andrea, che hanno reso il Lab6 un luogo di lavoro davvero piacevole.
I miei relatori, che mi hanno dato la possibilità di lavorare a un bellissimo progetto.
I ragazzi di HKN, con cui é stato possibile creare e partecipare a progetti unici.
E ultimo ma non per questo meno importante: Roberto, che mi ha supportato e sopportato in quest'ultimo periodo.

Contents

List of Figures

Chapter 1

Introduction, Motivations and Goals

1.1 Ontogeny Overview

Biology can be splitted in various sub field, among which we can find the Ontogeny field. The Ontogeny is a developmental phase that start from the embryonic stage. It is a complex field because want to study how from the embryo, that is a simple biological system, cells populations change themselves in their structure and in their functionalities. These developments involve various mechanisms: cell differentiation, cells mobility and different types of cell signaling, signals also depending from the spatial organizations of the cells.

All these mechanisms and their cooperation are hard to model. The model has to take into account both the inner cell state, its position in the space in respect to other cells, that can be of the same type or different, and which intercellular communication can be used. All these mechanisms are the product of underlying phenomena, that happen in the inner structure of the cells: molecular pathways and cascades, protein expressions, gene regulations; in other words all the interactions between proteins and genes that can change the state of the cell, defining its fate, its ability to move and its external communication capabilities.

1.2 Net Within Net

From the above overview become clear that these systems are highly hierarchically structured, this lead us to find a formalism that can well describe this characteristic: the Net Within Net formalism. NWN is an extension of the Petri Net formalism.

The PN formalism is one of the mathematical modeling languages that can represent distributed systems. It's a bipartite graph in which nodes are of two types: Transition and Place. Place and transitions nodes are connected with directional arcs and their directions define which are the pre-conditional and post-conditional places for transitions. The conditions are specific for the transitions, in particular the pre-conditions regulate the enabling of the transition and the post-conditions define the activation functions of transitions. When a Transition can be enabled then

it can fire and this means that that the activation function can start. Conditions regulates the exchange of elements, named Tokens, between places. The Tokens can be of different types: integers, floats, strings, colored or they can have no type. If they they have no type they are named Black token.

The NWN formalism extends the PN formalism with the addition of another type of tokens: Net Token, this means that a Petri Net can be contained in a Place as a Token and can be exchanged through the net. This implies that Nets can be hierarchically organized and each layer can be specified at the same level of detail. This is perfectly compliant with the requirements of modeling ontogenic processes in biological systems.

Another particular feature of Petri Net that we want to use are Channels. A channel is a synchronous communication mechanism between different nets. It is a property of a Transition, in particular the channel is a link between two Transitions, one named upper-link and the other named down-link. The down-link is the transition that belong to the net owning the channel, the upper-link is the transition belonging to a different net and is responsible for the activation of the Channel. So a Channel is activated when the upper-link fires, this lead to the firing of the down-link transition. Channels are also able to exchange tokens between the two transitions.

1.3 Issues in modeling complex biological systems

In general modeling a biological system it's a complex activity in respect to the modeling of a single part of the system [**1**]. This complexity arise mainly because the biological sub field have studied separately, so the knowledge coming from these various sub field is hard to put together and furthermore understand phenomena at the boundaries of biology sub domains is not easy.

Understanding interconnections between sub fields of knowledge requires experts in both sub fields. These efforts in modeling complex biological systems are forwarded in the description of models, different notations, different ways to structure entities involved. Yet those different mechanisms should be put together in one model description. Furthermore all biological phenomena are the combination and result of mechanisms that work at different system levels. These system in fact are organized hierarchically, ranging from the intracellular architectures and mechanisms to interactions between different types of cellular populations that compose organs.

Generally speaking, modeling complex biological systems has different purposes:

- Breaking the boundaries among different disciplines. Indeed to understand all the mechanisms involved in the whole system is necessary to pick concepts coming from different sub domains and look for their relations, avoiding to have a short-sighted vision on biological mechanisms.
- To describe complex phenomena in a structured way to share knowledge among experts that can improve their research works.

- To store knowledge in a unambiguous way, to be recognized among all the scientific community. This knowledge need to be stored electronically to be better

  exchanged, exploited and analyzed.

- To simulate or analyze model of complex systems to understand their structure or dynamics. This can lead to explore some mechanisms not

  well understood yet new knowledge.

Modeling complex biological systems means to face with different challenges:

- Which formalism should be used to unambiguously describe all the mechanisms involved.

- To find a way to reuse the already structured knowledge, or other models, to integrate them to gain new insight and generate more knowledge.

1.4 Our perspective

After all these considerations it becomes clear that there are various different variables to cope with when modeling biological systems. Furthermore to simulate these models it is necessary to translate the biological knowledge in a way understandable by the computer. To do that the need to choose a language that can describe the model and act as interface between the modeler and the computer. The language should be clear, understandable by all experts involved in the model construction, with a simple structure to not increase the complexity of a system already complex by its nature.

This last point introduce another expert among these coming from different biological subdomains: the computer engineer or at least the computational biologist. All the people involved should cope with different issues, but there is one element that put in connection all of them: the language used to describe the model of the system. The language should be used by biologists to describe the model and to discover

research results produced by other, in other words the language can function as the mean to exchange knowledge. On the other hand, computer experts can contribute to the language intending it as an interface for gathering all different pieces of information, which software tools can then elaborate on, to extract new knowledge For these reasons the language has to cover the needs of all these figures.

The language also should give to the experts all the options to choose the better configuration of the model. Indeed the model doesn't depends only on the existent knowledge about a specific phenomena. The model at the matter of fact is not a perfect description of the real system, but it is the product of:

- the lack of knowledge in some biological mechanisms

- the choices of the modeler, that maybe want to focus on some aspects of the systems and model other aspects in a more approximated way.

- the computational limits. Indeed the complexity of the system can go beyond the computational power, for this reason entities and mechanisms involved need to be downsized.

Chapter 2

State of the Art

2.1 Introduction

A great effort was made in computational biology to exchange resources about modeling biological systems, various tools was born to store, exchange and run models. Although there is no common standard, a lot of attempts to define one were made: exist various XML (eXtensible Markup Language) based computer-readable model definition that enable models to be exchanged between software tools. For this purpose a community of experts from all over the world was established:

> The 'COmputational Modeling in BIology' NEtwork (COMBINE) is an initiative to coordinate the development of the various community standards and formats for computational models. By doing so, it is expected that the federated projects will develop a set of interoperable and non-overlapping standards covering all aspects of modeling in biology. [**1**]

This initiative has identified and classified various languages born to describe and exchange informations about biological models in specific subfield of biology and with different objectives. In Figure 2.1 [**2**] all languages identified are shown.

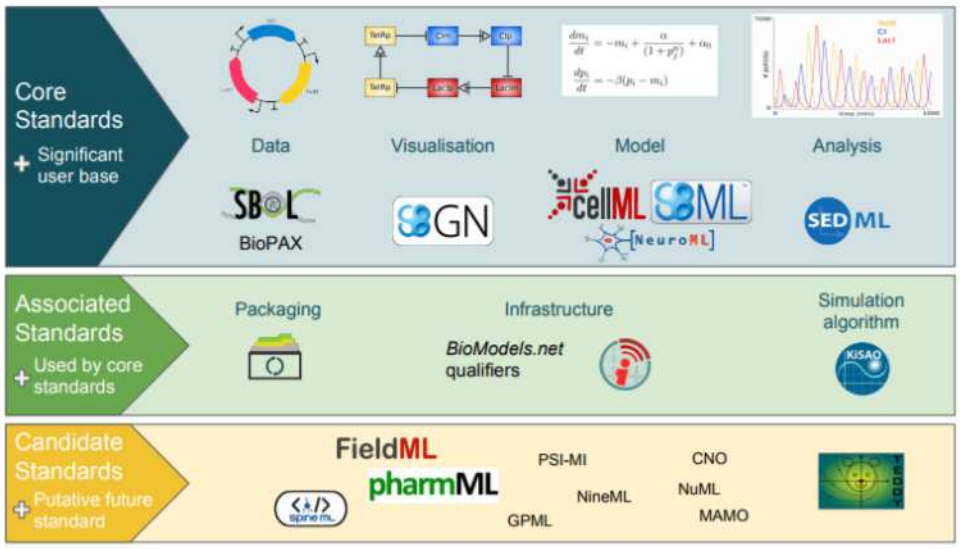

Figure 2.1: COMBINE standards and associated standardization efforts

The community has detected some core standards to accomplish different activities related to the modeling of biological systems. We describe them starting from the one on the left.

2.2 Formats for DATA exchange

The BIOPAX[**3**] and SBOL[**4**] languages are focused on data, that means that these languages annotate experimental data and literature to organize and categorize them.

2.2.1 BIOPAX

Data about biological pathways increased in their quantity in the past years, they have no common representation and they are spread among 300 different internet accessible databases. All the databases have different notations. To solve the issue of exchanging data between different DBs having incompatible notations a research group developed the BioPAX Data Exchange format, to make pathway data substantially easier to collect, index, interpret and share. The group collaborates with several other efforts and databases. Examples of collaborators are Chemical Markup Language (Murray-Rust and Rzepa, 2002), SBML and CellML (Lloyd et al., 2004), BioCYC, BIND, Reactome and WIT. Biological pathway can be described a different level of details, for this requirement the BioPAX language provides the possibility to represent different types like: metabolic pathways, molecular binding interactions, hierarchical pathways, signal transduction pathways, gene regulatory networks and genetic interactions. All these level of detail are not developed at the first moment, indeed there are several version of the language, called levels. The first level included only metabolic pathways, the second one added the capability to describe molecular binding interactions and hierarchical pathways, finally the last level added the possibility to represent signal transduction pathways, gene regulatory networks and genetic interactions. Due to these huge changes between levels there are some backward incompatibilities. The BioPAX language format follow the Web Ontology Language (OWL). Now we want to describe the structure of classes that are used in BioPAXand that are summarized in the Figure 2.2:

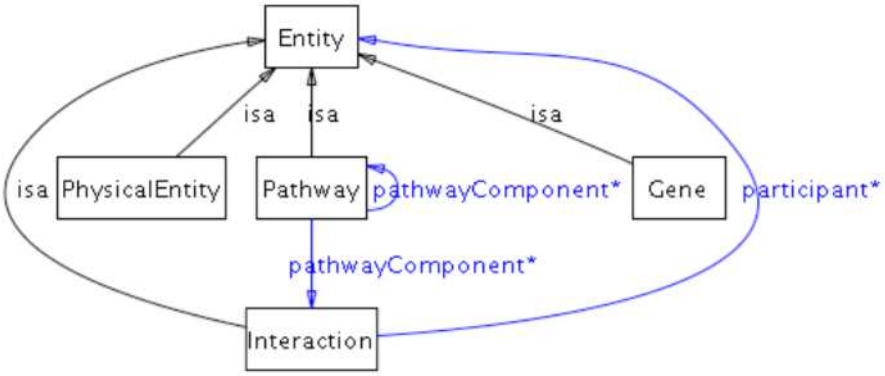

Figure 2.2: BioPAX structure of main classes

Classes are organized in a tree, the root of the tree is the Entity class, the root

has three children PhysicalEntity, Pathway, Gene and Interaction. From the image we can see that in the Interaction are involved entities and that Pathway is composed by both Interactions and Pathways. Each entity should has a precise referentation to a DB or to a literature source. Each subclass of Entity has in turn subclasses, to better specify the different biological cases. The interaction has as children: Control, Conversion, GeneticInteraction, MolecularInteraction and TemplateReaction. Some of these classes has in turn other subclasses, that are listed in the Figure **??**.

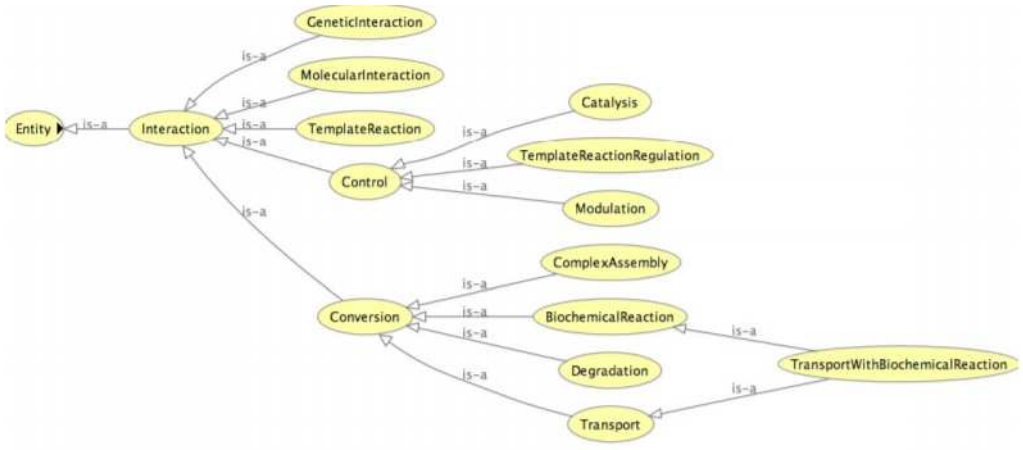

Figure 2.3: BioPAX Interaction subclasses

The Control class define an interaction in which an entity control in some way another entitie, that in biology could mean that the entity regulates, modifies, activates or inhibits another one, in other words it has an influence. The class Conversion is an interaction in which one or more entities are transformed in something else. The TemplateInteraction instead happens when a macromolecule is polymerized from a template macromolecule. The Physical children are: DNA, RNA, DNARegion, RNARegion, SmallMolecule, Complex and Protein. They represent the type of entities that can interact.

Some classes that are important not for the structure of data but for the specification of some characteristics that cannot be described by the use of entities' properties are the Utility classes. These classes provide the possibility to set some complex property like stoichiometry, molecule structure, reference in DataBases and also cross references to be unambiguously identified among different DataBases. To give an unambiguous identity an important class is the ControlledVocabulery that provide the possibility to one of the various CVs, like the various biological Ontology that can be found through the internet.

2.2.2 SBOL

In the development of Synthetic Biology projects the exchange of knowledge is very important to boost the design process. To help the design process a common language to describe and exchange information was needed. Exchanged information needs to be accurate enough to allow the reproduction of such designs in different wet labs. Synthetic Systems need a structure of annotations more complex than that

used by the most recognized formats usually exploited for natural system description, more simple and flat. The encoding FASTA format can describe nucleotide sequences, but not model design. SBML is able to describe a biological process at high level without specifying anything about nucleotide sequences. To cover these lacks and to describe biological entities from the

engineering point of view a new format was specified. Synthetic Biology Open Language (SBOL) introduces a standardized format for the electronic exchange of information on the structural and functional aspects of biological designs. To accomplish the description of these two aspects SBOL makes use of different classes that specify in a separate way the structure and the functionalities and put them in relation. Figure 2.4 can visually show the relation between classes:

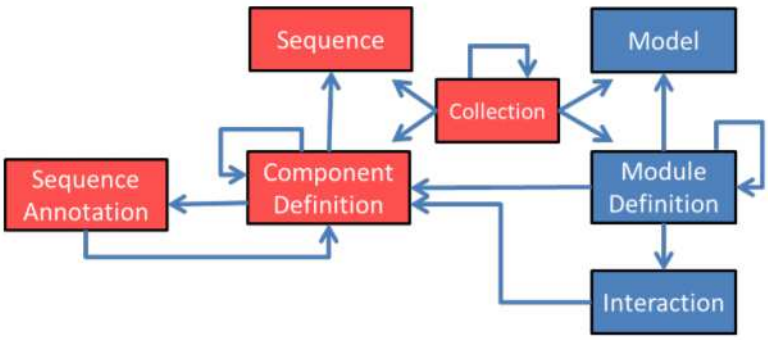

Figure 2.4: SBOL main structure

The module class contains all the Component Definitions that compose the module. It is possible to specify Sequence and SequenceAnnotation, identified with the homonym classes, for the definitions of the class Component. Component can be put in hierarchical relations adding references to subcomponents. The Interaction class specifies the qualitative relations between components. If a reference to a complete model exists can be referenced through the class Model. The Model can be written in other languages such as SBML and CellML. This is only an overview of the SBOL model, the structure of classes is more detailed and is shown in Figure 2.5:

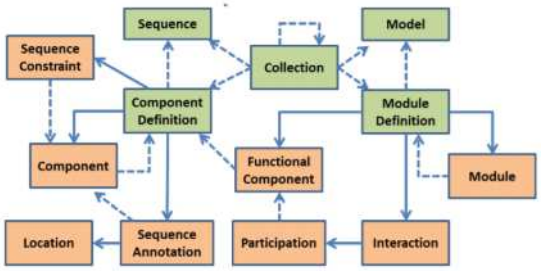

Figure 2.5: SBOL structure

Dotted arrows depict reference-based relations between objects: parent objects don't hold children objects within themselves, but rather references to them, so to avoid redundancies. Beyond the definition of those classes SBOL also makes use of existing Semantic Web practices and resources, such as Uniform Resource Identifiers (URIs) and ontologies, to unambiguously identify and define genetic design elements. This format has been designed to support the explicit and unambiguous description of biological designs by means of a well defined data model.

2.3 Format for visualization: SBGN

In the same set from Figure 2.1 we encounter the System Biology Graphic Notation (SBGN) [**5**], the only format identified to accomplish the visualization of models. Graphical visualization is important in

biology, especially for users that are not confident with the computational biology.

The Systems Biology Graphical Notation (SBGN) project aims to describe, in a unambiguous and visual way, signaling pathways, metabolic networks, and gene regulatory networks, to share knowledge between biologists and researchers in life sciences. The requirements of this project are strictly related to the exchange- ability of knowledge:

- Notation has to be unambiguous and able to represent the main biological processes, interactions and entities.
- The language needs to be modular to increase the reusability and exchangeability and to decrement redundancy.
- Open access, to be easily spread among the scientific community.
- Easy to be written in an automatic way by a software, starting from a mathematical description.

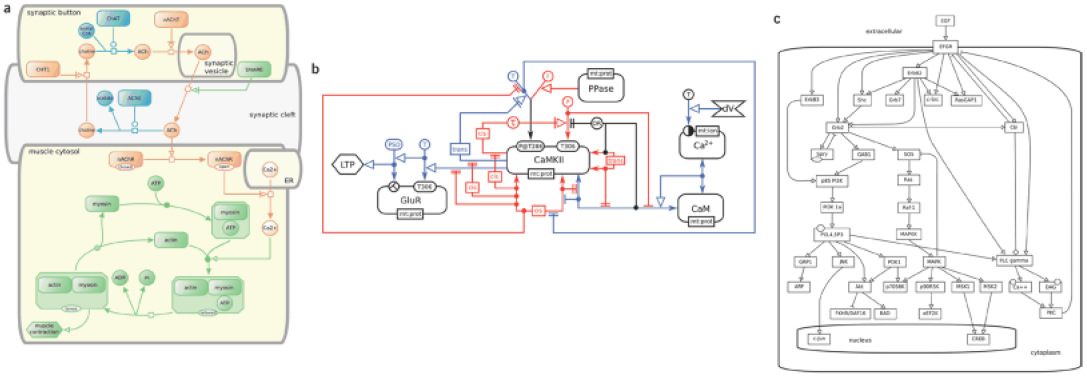

Figure 2.6: SBGN example showing the three different perspective of the same biolgical case. a) Process Description b) Activity Flow c) Entity Relationship [**5**]

The representation of biological objects, their properties and their interactions is articulated in different point of views: the processes description, the structure description and the flow of the activities. To accomplish the requirement of explicitly represent the biological objects with these three different views were created three orthogonal languages:cesses description, the structure description and the flow of the activities. To accomplish this requirement were created three orthogonal languages:

- Process Description (PD), describing the temporal development of biochemical interactions, implying the presence of the same molecule in different states. This means that the language describes the path of the molecule with all its changes in state. This path is included in a bigger biochemical network.

- Entity Relationship (ER). In contrast to the Process Description language, which focused on description of the interactions functioning and their temporal development, this one focuses on how entities influence each other, specifying in which interaction or process they are involved. These interactions can represent the experimental outcomes, not all the interactions involved in the whole process. It is very useful to see the cause-effect relations between multiple entities.

- Activity Flow (AF). This language wants to give a less detailed description of process and entity relations, to describe biological cases for which no or indirect knowledge is available. In this way generic interactions and effects of perturbations can be described. Also information about entities of states is omitted. The Figure 2.6 shows an example of these threes languages.

2.4 Formats for model exchange

Model description is one of the most sensitive tasks in the perspective of exchanging models. In fact, while exchanging data or visual representations is natural to standardization, model architectures include more and diverse complexities. For example, they may rely on different formalisms, or include diverse references to DBs or ontologies, and carry along information on how to handle simulations. The initiative has detected three languages: SBML, CellML and NeuroML.

2.4.1 SBML

Simply put, System Biology MArkup Language (SBML) [**6**] is a machine-readable format for representing models. It's oriented towards describing systems where biological entities are involved in, and modified by, processes that occur over time. An example of this is a network of biochemical reactions. SBML's framework is suitable for representing models commonly found in research on a number of topics, including cell signaling pathways, metabolic pathways, biochemical reactions, gene regulation, and many others. SBML does not represent an attempt to define a universal language for representing quantitative models. A common intermediate format—a *lingua franca* —enabling communication of the most essential aspects of models. Also SBML is a XML format that is composed by a set of lists:

- List of Function definitions: contains mathematical functions with an assigned name to be then referenced throughout the rest of the model.

- List of Unit definitions: a set of named units of measure definitions to be referenced when quantities are used in the model.

- List of Compartments: a set of containers of finite size, that are declared to contain molecular species. They may or may not represent actual physical structures.

- List of Species: a set of pool of molecular entities of the same kind, located in a compartment and participating to reactions.

- List of Parameters: a pool of quantities with symbolic names, parameters can be constants or variables. They may be global to a model or local in a single reaction.

- List of Initial assignments: these are used to determine the initial conditions of the model, is a mathematical expression whose values of symbols can be derived from others. These expressions can be also used to set values of symbols at the start of simulation.

- List of Rules: mathematical expressions added to the reactions-related expressions. They can define how a symbol value changes with respect to other symbols or to define the change rate of a symbol. They can be also used together with the reaction rate equations to determine the behaviour of the model with respect to time. Rules constrain the model for the entire duration of simulated time.

- List of constraints: means for detecting out-of-bounds conditions during a dynamical simulation and optionally issuing diagnostic messages. Constraints are defined by an arbitrary mathematical expression computing a true/false value from model symbols. An SBML constraint applies at all instants of simulated time; however, the set of constraints in a model should not be used to determine the behavior of the model with respect to time.

- List of Reactions: they describe processes like transformation, transport or binding that change the amount of some species. To describe how quickly they take place they have an associated kinetic rate expression.

- List of Events: they describe a change in one or more symbols (species, compartments, parameters) that happens instantly and discontinuously after a triggering condition is satisfied.

From these lists we can see that this standard is born to describe biochemical network models. Other optional elements of the SBML format are the sboTerms a Annotations, which are useful to make references with databases or ontologies, identifying unambiguously biological entities. When users doesn't use this option become difficult to extract all the entities involved because no unambiguous reference to them is provided. In addition, often users adopt names understandable by them, this make the whole model not useful or hard to exploit. To use SBML better developers built a set of software tools to use models, indeed the format was born as a machine-readable language and for humans is not easily understood without software interfaces.

2.4.2 CellML

The CellML language [**7**] is an open standard based on the XML markup language. CellML originated at the Auckland Bioengineering Institute at the University of Auckland, and its development is now led by the CellML Editorial Board.

Although CellML was originally intended for the description of biological models, it has a broader application. CellML includes information about model structure

(how the parts of a model are organizationally related to one another), mathematics (equations describing the underlying processes) and metadata (additional information about the model that allows scientists to search for specific models or model components in a database or other repositories). CellML includes mathematics and metadata by leveraging existing languages, including MathML and Resource Description Framework (RDF). In the future, CellML may also use other existing languages to specify data, define simulations and render information.

The CellML project is closely affiliated with another XML-based language project currently underway at the University of Auckland: FieldML. The two combined languages will provide a complete vocabulary for describing biological information at a range of resolutions, from the subcellular to the organismal level.

CellML is composed of different elements we list below:

- Component: it is a functional unit, and it can correspond to a physical entity, an event, a species or just an abstract convention. It has an attribute that is the name. It can contain variables and mathematical functions that put in relations variables. The variables can have three attributes: name, unit of measure and public interface.

- Equations: the said equations can be described using MathML an XML format. Each equation can have a numerical ID to have the possibility to reference it throughout the model.

- Grouping: the CellML compartments can be grouped together in two different ways: encapsulation and containment.

- Containment is used to describe the physical or geometric organisation of a model, such as biological structure. This type of grouping specifies that components are physically nested within their parent component, for example an ion channel may be physically embedded within a membrane.

- Encapsulation allows the modeller to hide a complex network of components from the rest of the model and provides a single component as an interface to the hidden network. Encapsulation effectively divides the network into layers, where connections between the layers must only be made through the interface components. For example, electrophysiological models often have activation and inactivation gates encapsulated within ion channels. It is useful to use encapsulation in this instance because gate properties are specific to individual channels.

- Connection: two compartments can be connected through this element, that substantially is a mapping of variables, associating a variable of one component to a variable of the other component. Two compartments can have in common only a connection. When a connection is enabled then a value of a variable is transferred to the variable of the other compartment. The direction of the value transport depends on the value of the public interface and private interface attributes of the variable.

- Units: the quantities in the model have an associated unit of measure that need to be declared in the unit elements. The majority of these are based on

the International System of Units (SI) although some non-SI units that are particularly common in biological systems are also provided. Additional units can be defined as complexes and variations of SI units.

- Reaction: CellML contains a Reaction element which is used to describe individual reaction steps in a pathway. This includes a description of the reaction kinetics, the reactants, products and any enzyme catalysts or inhibitors. Often in models the reaction element implementation had to be rewritten, this breaks the modelmaking it not reusable anymore. For this reason its usage is discouraged and maybe in the next version it will no longer exist.

- Imports: from the

  latest version it is possible to reference in a model file compartments from other files. This feature promotes reusability of models and components and allows CellML models to be incorporated into hierarchical frameworks.

- Metadata: these are optional information that may be included, they do not have necessarily be included to make the model valid but it is strongly recommended to insert them. These information are related to the model, but they don't belong to the model: date, authors names, annotation about biological roles of variables or compartments, annotations to suggest some optimization of parameters during the simulation.

Therefore the CellML language is simple, not necessarily related to biology and provide modularization, reusability and a strong robustness from the physical point of view, due to the strongly required use of units of measure.

2.4.3 NeuroML

NeuroML [**8**] is an international, collaborative initiative to develop a language for describing detailed models of neural systems. The NeuroML project focuses on the development of an XML-based description language that provides a common data format for defining and exchanging descriptions of neuronal cell and network models. The current approach in the project uses XML schemas to define the model specifications. Experimental neuroscience data is measured at different levels of detail. NeuroML is able to describe the models that are created upon these data. For each level of detail NeuroML provides a suitable abstraction level to include them in the model. From the scheme in Figure 2.1 we can see that the models can be easily imported in various simulators, due the machine-readable format of XML. Then the output of simulations can be compared to experimental data and maybe suggest new type of experiments. The cycle of these activities is shown in Figure 2.7[**8**]

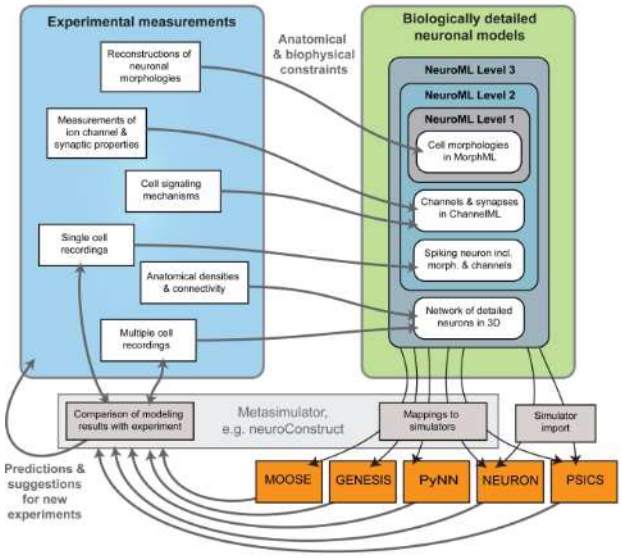

Figure 2.7: NeuroML cycle of usage and relations with external environment

The XML is also the format used by languages described above, but the XML elements used in those language are more simpler and generic, NeuroML instead provides spe-

cific and complex elements, specifically related to the Neuroscience domain. This complexity leads to the compartmentalization of NeuroML in different levels, each one corresponding to a layer of the neural system hierarchy. NeuroML is structured in three levels to partitionate the model description into the anatomical structure and the various physiological mechanisms that underlie the electrical behavior of neurons and networks and reflects the manner in which they are commonly implemented in neuronal simulators. The three levels are associated with a different description language:

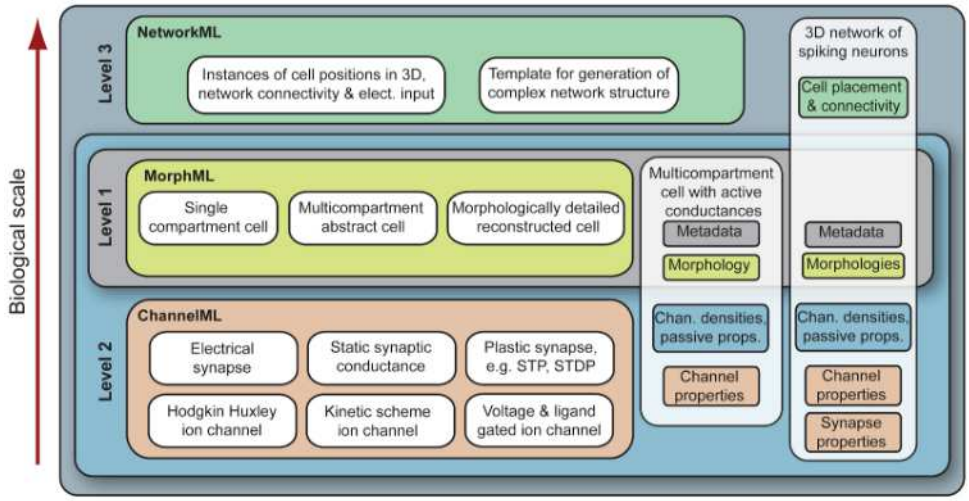

Figure 2.8: NeuroML levels structure

The Figure 2.8 [**8**] shows these levels in a schematic way.

- Level 1 - MorphML: describe the neuronal morphology and add relevant back-

ground data (metadata about authors, provenience, citations) to the model covering different degrees of complexity in describing structural information, from the notion of a simple cell to the specification of its internal substructures.

- Level 2 - ChannelML: describe conductance like voltage-gated membranes, static and plastic synaptic processes.

- Level 3 - NetworkML: specifies 3D locations of neurons, connections between populations and external electric inputs. This compartmentalization of the language in three layers intends to make the language modular and reusable.

This compartmentalization of the language in three layers intends to make the language modular and reusable. The machine-readable format used for NeuroML needs some additional tools to make the model more understandable by the users. Import and export functions currently supported by simulators of NeuroML at the moment work in a lossy way. This is because they don't make use of some features of NeuroML, specifically for the ChannelML level. For these reasons, NeuroML should currently be considered less as a format for creating a new cell model from scratch and more as one format for the storage of stable models and components that are being made available for wider usage. NeuroML was created having in mind the priority to the description of existing models, and so it has been developed to have a backward compatibility. For this reason some complex features of neural networks can't be described with current specifications of the

schema. However NeuroML has reached a state of maturity where it can be used to specify a wide range of published single neuron and network models.

2.4.4 LEMS

At some point, while using NeuroML to describe neuronal models, the need arose to have new means to describe synapses. From this started the project of creating a new language, flexible enough to describe the synapses, was born. In fact, descriptions of various types of synapses range from highly detailed biochemical models to much more abstract ones. To provide this flexibility developers thinked up for LEMS[**9**]: a Low Entropy Model Specification. LEMS wants to be a compact, minimally redundant, human-readable, human-writable, declarative way of expressing models of biological systems. It differs from other systems such as CellML or SBML in its requirement to be human-writable and in the inclusion of basic physical concepts such as dimensionality and physical nesting as parts of the language. The main goal is to enable model developers to write declarative models in LEMS in much the same way as software developers write software applications in computer languages such as C, Java or Python. Modules written in LEMS are based on user-defined types called ComponentType that contain parameter declarations, reference declarations and specifications of what children an instance of a type can have. Typically they also a Dynamics specification which can contain build-time and run-time declarations. Build-time declarations apply when a simulation is set up, for example to connect cells. Run-time declarations specify the state variables, equations and events that are involved. There are also Run, Show and Record Dynamics for creating type definitions that define simulations and what should be recorded or dis- played from them. The models contain Components which are instances of ComponentType,

they have the same relation of objects and classes used in Java. There is also the concept of inheritance. A ComponentType is able to extend another one, adding parameter to the parent. A type can contain elements for specifying the following aspects of the structure and parameters of a model component:

- Parameter: dimensional quantities that remain fixed within a model
- Child: a required single sub-component of a given type
- Children: variable number of sub-components of the given type
- ComponentRef: a reference to a top-level component definition.
- Link: a reference to a component definition relative to the referrer
- Attachments: for build-time connections
- EventPort: for run-time discrete event communication
- Exposure: quantities that can be accessed from other components
- Requirement: quantities that must be accessible to the component for it to make sense
- DerivedParameter: like parameters, but derived from some other quantity in the model

The EventPort and Attachments declarations don't have any corresponding elements in their model component specification. They only affect how the component can be used when a model is instantiated. EventPorts specify that a model can send or receive events, and should match up with declarations in its Dynamics specification. An "Attachments" declaration specifies that a run-time instance can have dynamically generated attachments as, for example, when a new synapse run-time instance is added to a cell for each incoming connection. The Low Entropy concept included in the name represent the vision around this language. It's a loose analogy with the physical concept. The model that came out from the description of the description of a domain expert is concise, the components are highly structure and are used physical quantities. When the model then is converted to be run in a computer the hierarchy and the architecture is deconstructed, quantities are divided in units and in dimensionless quantities, domain specific mechanisms are translated into differential equations or others math formalisms. That means that the entropy, during this process of transformation of the domain specific model into a runnable one, increases significantly. The models only in the executable form are them with the highest entropy. The aim of LEMS is to mantain a low entropy in the description of the model, not only to be have a format better readable, but also to be able to translate it in different formats at high entropy. The contrary, transform in high entropy description in a low entropy one, is not always possible.

2.5 Format for simulation analysis exchange: SED-ML

Reproducibility of results is a basic requirement for all scientific endeavors. This is not only true for experiments in the wet lab, but also for simulations of computational biology models. The Minimum Information About a Simulation Experiment (MIASE) is a reporting guideline describing the minimal set of information that must be provided to make the description of a simulation experiment available to others. SED-ML [**10**] encodes the description of simulation experiments in XML, in an exchangeable, reusable manner. SED-ML covers the description of the most frequent type of simulation experiments in the area, namely time course simulations. SED-ML documents specify:

- which models to use in an experiment,
- modifications to apply on the models before using them,
- which simulation procedures to run on each model,
- what analysis results to output,
- and how the results should be presented.

These descriptions are independent of the underlying model implementation. SED-ML is a software-independent format for encoding the description of simulation experiments; it is not specific to particular simulation tools.

2.6 COMBINE perspective

This analysis of languages used in system biology made clear that none of the languages have the characteristics to model ontogenetic processes. A first reason is the lack of spatiality and mobility descriptions. Except for CellML that can describe spatiality relying on FieldML, and NeuroML that can describe the spatiality in the neurological field, the other languages don't give this support. Another intrinsic characteristic of the ontogenetic processes is the hierarchical arrangement of involved components and their differents types of communication, others features that the existing language are not able to completely handle. This is a common thought among the scientific community, in fact also the COMBINE community, that is a network formed by the communities developing standards and formats to share computational models, agreed on the need of a new language that permits to build and exchange models of multicellular environments.

For this purpose they created a group to develop MulticellularML and they discuss about it in this forum [**11**]. According to them, the language must have features so to support the description of: e:

- movement
- binding
- growth

- birth
- death

In addition to these, from the analysis of models for ontogenetic processes, the need emerges to suitably describe:

- differentiation
- position
- signal communication

The contributors to the COMBINE community also state use cases that the language should be compliant with:

- Intestinal crypt
- immune synapse
- gap junction
- cell sorting
- monolayer and MCS growth
- apical construction
- invasion

Also, they pose questions about the language design:

- Which is the heterogeneity level of cells to support?
- Is possible to extract from the model the list of the molecules involved within? (SBML limit)Does the definition Multicellular model refer to biological tissues, or rather to entire organs?
- Does the definition Multicellular model refer to biological tissues, or rather to entire organs?
- Does the model cover non-compartimental examples such as 3D spatial continuous models?
- Should the Ontology be optional or not? If yes then the model is easy to build, if not then models are more reusable and understandable by others
- Are Place elements included in a model? Sometimes it can be hard to manage them, for example center-based agent models have locations and volumes but not morphology.

These questions helped us to design our idea and our project, we take them as inspiration to build a tool that is compliant to the needs of the community. We will present our approach to these issues in the following chapters. Some of contributors thinks that use cases are not only a scope but also a starting point to build the language together with the computational models on which the simulations are based on. Someone instead disagrees, because if the language should become a standard it has to be independent from any computational model and not strictly tied to any use case. Some of the contributors thinks that the language should contain a hierarchy like this: general and geometric information global variables (a function of time) local variables (a function of time and space). They point out MorpheusML, as an example to follow, that can be generalized to cover the MulticellularML goals. The property of Morpheus MDL are:

- Encapsulation: some details of simulations can be attached within the description of the model , the initial conditions are also included. Encapsulating all these informations in one file makes the process of archiving and restoring the model simulation much simpler.
- Architecture: it is two-tied, on one hand there is the XML describing the model, on the other hand symbolic links put in relation component and process, in this way the modularization property is increased.
- Elements:
  - Space and Time : spatio-temporal aspects
  - CellPopulations : initial conditions and simulation state
  - Analysis : configuration of data output and visualization
  - Description : title and annotations about model
  - CellTypes : behaviour, dynamics and intracellular dynamics of cells
  - CPM : parameters of cell-potts models
  - PDE : reaction-diffusion models

XML represents this information in a hierarchical tree-like structure that reflects the structure of the modeled biological system. Model components can be linked using symbolic identifiers. Symbolic identifiers and references establish interactions

and feedbacks between (sub)models to represent the network-like complexity in biological processes. MorpheusML has the advantage of including to include time and space specifications, but it is also built to make simulations with specific mathematical formalisms, with a specific spatiality and geometry that is the cellular potts architecture. Also, MorpheusML focus on a specific subdomain of biology that is that one of the cellular populations dynamics.

2.7 Comparison between languages

All these languages are the consequence of the need to make the computational models more Reproducible, Accessible, Portable, and Transparent (RAPT). The

RAPT properties pose the intrinsic requirement that these languages need to be simulator-independent.

All these languages used to describe models can be classified in two approaches: domain-specific and general.

- DOMAIN-SPECIFIC:
  - SBML: The SBML born to describe Biological Systems, but at the matter of fact it is used to describe biochemical models, this means that also this language is specialized in a biological subdomain. Indeed also its syntax is too specific for this subdomain cause the main species involved in the model are Reactants and Products. Furthermore is too tied to the use of mathematical formalisms, such as differential equations. SBML make use of SBO (System Biology Ontology): a set of controlled, relational vocabularies of terms commonly used in Systems Biology, and in particular in computational modelling. It consists of seven orthogonal vocabularies defining: reaction participants roles (e.g. "substrate"), quantitative parameters (e.g. "Michaelis constant"), classification of mathematical expressions describing the system (e.g. "mass action rate law"), modelling framework used (e.g. "logical framework"), the nature of the entity (e.g. "macromolecule"), the type of interaction (e.g. "process"), as well as a branch to define the different types of metadata that may be present within a model
  - NeuroML: is well structured, reusable, but it is also the most subdomain specific language between all the others. The domain specificity characteristic of this language lead to structural changes every time a new concept need to be described. That's happened when the LEMS project started. LEMS want to be a human readable and low entropy language. But it is also necessary to consider it was born for supporting the NeuroML language, and this shaped its structure. Besides that, LEMS is too generic to be considered as a domain-specific language: the user can specify model elements freely, without necessary references to biological concepts or structures. In NeuroML the data and logic required to fully describe and execute the model is spread across the model scripts, the documentation of the model description language and the simulation engine. This hampers the exchange of models between software tools and their transformation into human readable formats, limiting the RAPT of models defined in such formats.
- GENERAL.
  - CellML: want to be a tool to create general models, not only related to a specific domain. This approach give the possibility to describe a wide range of systems, and also give considerable flexibility for implementing new mechanisms as they are discovered, without the need to alter the inner structure of the language. but it also loose the possibility to unequivocally identifying domain specific features, this lead to a harder reusability of models. This general language tool lead also to multiple ways to describe a unique solution, that means an inconsistency in the identification of biological entities and processes.

All these languages are based on the XML standard, this aspect makes models exchangeable by softwares but not understandable by human without a tool that work as interface. Moreover they are all oriented to mathematical models, often based on Ordinary Differential Equations.

From this analysis emerge the need to have a language that is domain specific for different subdomains of the biology, without be too general. Obviously it is hard to include all the biological subdomain in the syntax of a language, because this leads to an exponential growth of the language lexic, that make it harder to learn. This new language need to be enough general to give the chance to add concepts without making structural changes, but it also should has a strict bond to the specific ontologies, to give unambiguous identities to the models and their components. In other words it is necessary to find a trade-off between these two approaches, stating a new semi-general approach. This semi-general approach gives also the opportunity to set hierarchical architectures to models. The other important aspect of this approach is how the reusability that can be increased. Reusability can be allowed through: a modularization of models description and the attachment of informations about simulation, like the initial conditions of the models. The modularization of the description of models decrease also the redundancy, because references to other modules can be made. From the analysis come out that the concepts that all languages have in common is the presence of entities, processes, communications or relations between entities or between entities-processes and for all these concept a spatial specification is needed. We have compared only the languages used to describe the model itself, the other languages including collateral informations need to be added not in the language structure but they need to be easily linked and wrapped in the language.

Chapter 3

How To Build a Language

3.1 Aspects to consider when setting requirements

In the process of language creation the mantra should be "Keep it simple, less is more". Only the necessary concepts of the domain need to be included, and generalization for future improvement should be avoided because the language could become too complex. Few elements for the language is the right choice, this reduces the learning curve. These good practices help to avoid redundancy of concepts in the language, which might be confusing for users when they make use of the language.

The language will be adopted by end-users, then it is necessary to implement their own notation, but to make the language understandable by more people than end-users it is better to choose a descriptive notation, and when this is not possible it is good practice to give the chance to add comments. To avoid confusion and reduce the learning curve choose a syntax that make a great differentiations between different terms and concepts. To avoid redundancy, to make the language understandable at a glance and to increase compactness it is better to make it modular. Furthermore it is better to have the same style among modules. If some elements of the language are complex and not very easy to use, or there is some implicit feature, it is better to provide good practices and usage conventions.

The syntax used by the end-user should reflect his way of thinking but at the same time needs to be compliant with the abstract syntax tree (AST), to make the translation from the language to the underneath architecture of the compiler easier. A way to improve this aspect is the modularization of the language and the introduction of interfaces.

The purpose is to use this language to describe a precise task, for this reason it is necessary to identify some primary tasks and to ry to find a language that can describe all of them. Finding out the right tasks is not enough, they need to be explained by the end-users to import their own lexicon and way of thinking to the language syntax. If they prefer to explain their domain with drawings maybe it is better to create a visual language in contrast to a textual one. To build the new DSL other languages definitions, type systems and features can be exploited.

3.1.1 Conceptual Level

In this section we want to address the questions: how will the language be used? Which work process has it to support?

To give an answer to these questions it is important to find the tasks users will need to accomplish. One of the more frequent error made by developers, when they are designing a language, is their will to cover every detail in the user way of expression. This can lead to a useless enrichment of the language. Other than to be useless, it can be also an issue in the usage of the language and in the learning phase. Indeed users most of time change their way of express themselves to be able to use the language.

We said that is better to follow the approach "less is more"; so the language should be:

- Easy to learn
- Easy to build
- Able to receive more tool support
- Be able to be analyzed more easily

To avoid to make this error it is important not to try to cover all possible usages from the beginning. When designing a language we should think about the usage of this language. Often we think that the major activity within which the language is involved is its writing, but actually people most of time read the language, to check their own work or to discover something when reading works of someone else. In our case the activity of reading models occurs very often, indeed models of biological systems are built with the aim to exchange them with all the research community. So the language design has to take into account these two activities at the aim to facilitate both. Models need to be exchanged also to be extended and modified, for this reason changing the model description needs to be easy and not too invasive for the whole model. The language structure should allow for modification and extension of models without changing all the description but only the pieces of interest. Our language wants to describe models that will be then simulated, for this reason the language has to support this activity. A well organized documentation is very important to promote the usage of the language by many people.

3.1.2 Human Level

In this section we should cope with this question: Which are the users willing to use it?

Building the language is only a part of the work, the other part consists in trying to find a connection with the users. Indeed, who cares we built the perfect language, if nobody is going to use it? Often people are not open to new tools, and this happens basically for two reasons: resistance to change and the effort needed for learning to use them. To overcome these problems an efficient communication of our project is necessary, to specify which actual issues the language overcomes and

how easy it is to learn its usage. To achieve these objectives it is also important to connect with users, talk with them trying to understand what make their life difficult. To positively achieve this connection it is better to find someone that want to make a change trying

to use something better. We should communicate and connect with users without presenting us as the best experts in their domain in their domain, but rather as explorer of domains to make it understandable by machines.

3.1.3 Technical Level

Which are the technologies involved?

The technologies involved in the use of the language are various. A compiler is needed to translate the language in other languages or in the bytecode. An Integrated Development Environment is useful to work with plugins, test, debug and to easily write the code. A specialized editor can help in writing the code, with suggestions about syntax errors. To write a compiler is necessary build the Abstract Syntax Tree, in other words the code get divided in the fundamental pieces and then structured in a tree architecture. To build the AST are needed the Lexer and the Parser. The Lexer should be able to recognize the syntax of the the language and split the code in tokens corresponding to syntax elements. The Parser has to work with the tokens produced by the Lexer, it has to define the grammar to build the AST. Then the AST is used to generate the code in other languages or the bytecode, as said before. To develop all these tools there are various options:

- JetBrains MPS: a tool that help to build different families of languages, using different notations (textual, graphical, tabular). It overcomes the development of Lexer and Parser and help you to directly build the AST. One of the best advantage of this tool is the fact that help you to build a customized editor for your language, in a very easy way.

- XTETX: a tool to build a single text language, similar to MPS because is not like the others parser generator but can also build a class model for the AST and a customized editor to be used inside eclipse, a browser or any any editor that supports the Language Server Protocol.

- Custom approach: You have to write the Lexer and the Parser on your own and generate the code you desire. This is a more flexible approach, that give you a total control, in contrast is more costly than use the others two precedent options.

3.2 Our requirements

Strategies to build the language. The design has to answer these questions:

- Which are the users will use it? Our purpose is to build a language that can be used by biologists without or with few knowledge about computer science. To accomplish our purpose we need to create a language that is as much as possible near to the current biologist way of speak .

- How will be used? Which work process has to support? The major aim of the language is to be able to describe models of ontogenetic processes inside biological system. So the most important requirements of the language are: capability to describe different layers of hierarchical systems, capability to describe the spatiality and geometry of the system needed to support the developmental and motility processes.

- Which are the technologies involved? Models need to be described not only for an explicative and an expositive purpose of biological process. The aim of these models is to explore systems to bring out new knowledge or to bring out some new perspectives and ideas in laboratory experiments. In other word they can be a guide to explore the biology field. To accomplish these purposes the models need to be simulated. The language should describe a model intended to be simulated, the languages should interfaces with a software simulator. In our case the simulator is based on a specific Petri Net formalism, for this reason the language has to be able to describe and manage the elements of this formalism.

3.3 Lexical research

Building a new language requires an analysis on the language domain and on the ways people belonging to this domain use it. In the biology domain and specifically in the bioinformatic domain a standardized way to communicate is represented by ontologies that try to define a strict way to share knowledge in a context that is too various and full of ambiguities by its nature.

The strength of a Domain Specific Language is the closeness to the user way of thinking. This motivation leads us to base the syntax on the existing ontologies such as the GeneOntology or the Cell Behaviour Ontology. From these ontologies we wanted to take the more frequent terms and the more frequent relation between terms. This research brought us to the construction of a syntax closer to the core of the language used in the biology field. In this way we can give to the biologist a tool that speak his own language, and we avoid the skepticism that arise when people should use a new tool, last but not least can also help shorten the learning curve.

For this reason our purpose is to analyze words correlations in ontologies to use these correlations as foundation of our language. To better analyze these correlations a network of co-occurrences of words is built, the network is the graphical visualization of the co-occurrences matrix that contains the values of co-occurrences of words, these values are represented by thedges in the network, meanwhile each vertex represent a word and its size is proportional to the occurrences of the single word.

In general the correlation between words depends on various parameters if we are in a complete text because there is a correlation also between periods and paragraphs and so distance between words has to be taken in account. Ontologies can be thought as a set of sentences not correlated, for this reason co-occurrences between words should be computed only in the sentence context. In this case the statistical

significance depends no more in the distance between words (in fact sentences are little unlike the periods of an article) but in the length of the sentences.

The problem in find a threshold of significance building a network of co-occurrences is an open issue, some studies have used random permutations of the abundance matrix to compute a null distribution of similarity scores, and then selected as a threshold the similarity value corresponding to a conventional p-value choice of 0.01 or 0.05. Other studies have used arbitrarily chosen thresholds. Sometime custom strategies are implemented as is explained in [**12**] : they use a repeated element-wise random permutation of the noise-added abundance matrix to first compute a null distribution of similarity scores. They then compute the size of the largest component—the largest set of nodes for which any pair is connected by some sequence of edges—in the induced network for a wide range of threshold values.

In [**13**] the authors explain that to decide if a value of co-occurrence is valid or not we can compute the p-value of the probability distribution that better represent this case. In the general case the number of co-occurrences of two words follows the hypergeometric distribution, but this is a too crude approximation, because the difference in sentences length is not taken in account. The expected value for the co-occurrences that take in account the length is:

$$E\big(X\big) = \frac{n_1 n_2}{M\big(M-1\big)} \sum_{k}^{N} L_k\big(L_k - 1\big) \tag{3.1}$$

The so-computed relatedness is used to graphically display the relations among words, building up clusters around concepts. The graph was built using Gephy, that taking the score of relatedness of words compute clusters between entities involved. In such graph, vertices are words, their sizes are proportional to the frequency of words, arcs are relations between words, arc thickness is equivalent to the relation strength, and the colors represents the concept clusters.

This evaluation of words and relations from ontologies is the means through which we select the building blocks of language, in particular the syntax and part of the semantic.

To compute the relatedness of words and have a textual representation of the words scores representing:

- Node degree: the sum of connections of a node.
- Strength of node: the node degree is weighted with the connections weights
- Betweennes centrality: the number of shortest paths between pairs of vertices that pass through a given vertex

we used the Python library NetworkX that create a network, based on the co-occurrences matrix,in which every node is a word and every arc is the relation between words. We've computed these scores only for the Gene Ontology and in the 3.3 Figure we show the results for the Cellular Component, a Gene Ontology branch.

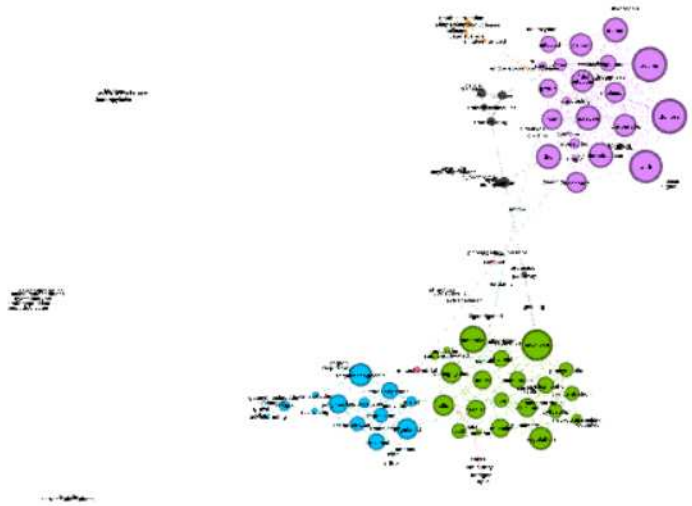

Figure 3.1: Network computed from a GeneOntology branch: Molecular function

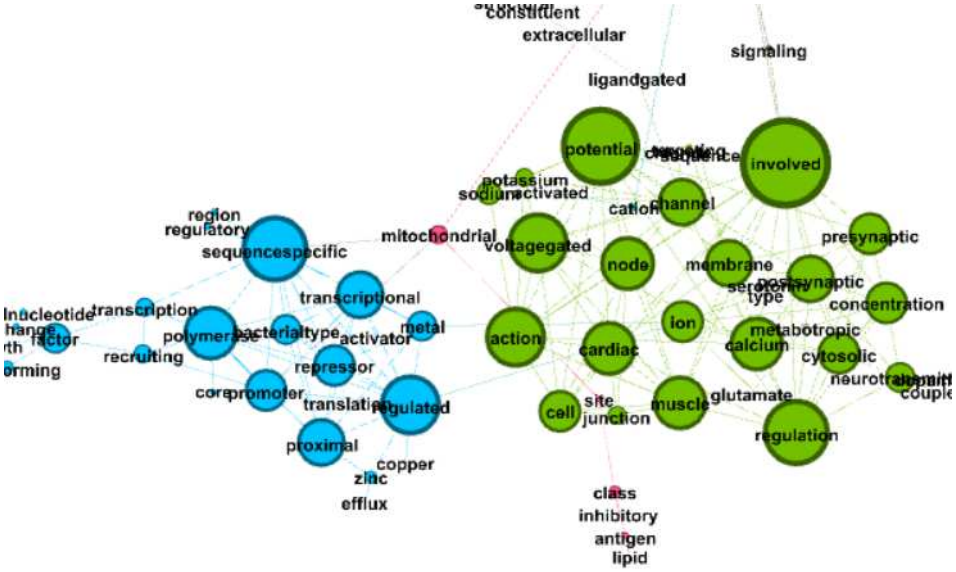

Figure 3.2: Zoom on the precedent image

This ontologies analysis leads us to build a syntax very constrained to some concepts and with a very large vocabulary. On the overall, two main sources guided us in designing the language: Cell Behaviour Ontology instructed the syntax structure, while Gene Ontology provided the concept organization to embed in it. The structure of the language devises:

- METADATA about model: date, author, name of the model
- MAIN part:
  - *Environment*, which is the most external component of the model de-

CELLULAR COMPONENT		
degree	**strength**	**betweenness**
('synthase', 27)	('collagen', 96)	('junction', 0.013301597879795254)
('junction', 24)	('viral', 86)	('base', 0.013210823388632838)
('viral', 23)	('factor', 81)	('layer', 0.012044139147676285)
('region', 19)	('reticulum', 80)	('particle', 0.011660716552119437)
('reductase', 19)	('type', 77)	('growth', 0.01144129944795251)
('dehydrogenase', 18)	('endoplasmic', 77)	('collagen', 0.0113974228185688)
('granule', 18)	('trimer', 68)	('region', 0.011147029500466945)
COMPARING		
Name: junction | Betweenness Centrality: 0.013301597879795254 | Degree: 24 | Strength: 53		
Name: base | Betweenness Centrality: 0.013210823388632838 | Degree: 11 | Strength: 20		
Name: layer | Betweenness Centrality: 0.012044139147676285 | Degree: 12 | Strength: 30		
Name: particle | Betweenness Centrality: 0.011660716552119437 | Degree: 17 | Strength: 64		
Name: growth | Betweenness Centrality: 0.01144129944795251 | Degree: 18 | Strength: 48		
Name: collagen | Betweenness Centrality: 0.0113974228185688 | Degree: 15 | Strength: 96		
Name: region | Betweenness Centrality: 0.011147029500466945 | Degree: 19 | Strength: 25		
Name: cytosolic | Betweenness Centrality: 0.010245011079682355 | Degree: 15 | Strength: 22		

Figure 3.3: Example of data analysis computed using NetworkX

scription, and has as properties a name, a type (chosen from a pool of specific environment defined by the language) and spatial information about shape and dimensions. This component includes entities and processes.

- *Entity*, which is one subcomponent of the Environment, that can have a list of entities that compose the structural aspect of the system. The Entity has a name, that identifies it unequivocally among other entities, and a type chosen among those proposed by the language. For each type of entity it is necessary to give some specification, also these are specified by the language. All types of entities have in common the spatial properties: shape, dimension and position related to the environment.
- *Process*, is the other subcomponent of the environment and it describes the behavioural aspect of the biological system. The processes can describe a single functionality of an Entity or a relation between entities. As the other component of the language also processes have a name (to be unambiguously identified within the system), a type to be chosen from those provided by the language and for each type give the necessary specifications.

- HIERARCHY of the system:

  Each entity can define its internal architecture, becoming an environment itself. So, for every entity the same specification already describe above in the Environment section are needed.

Also the processes can by specified at an higher level of detail, pointing out the subprocesses that can make arise the global process. The subprocesses are specified like the parent processes.

Due to this complex structure, and for the large vocabulary included in the language, the need to build an editor arises. To guide the user in the use of the language for describing his model the editor makes use of suggestions and tips to better describe each concept included in the language.

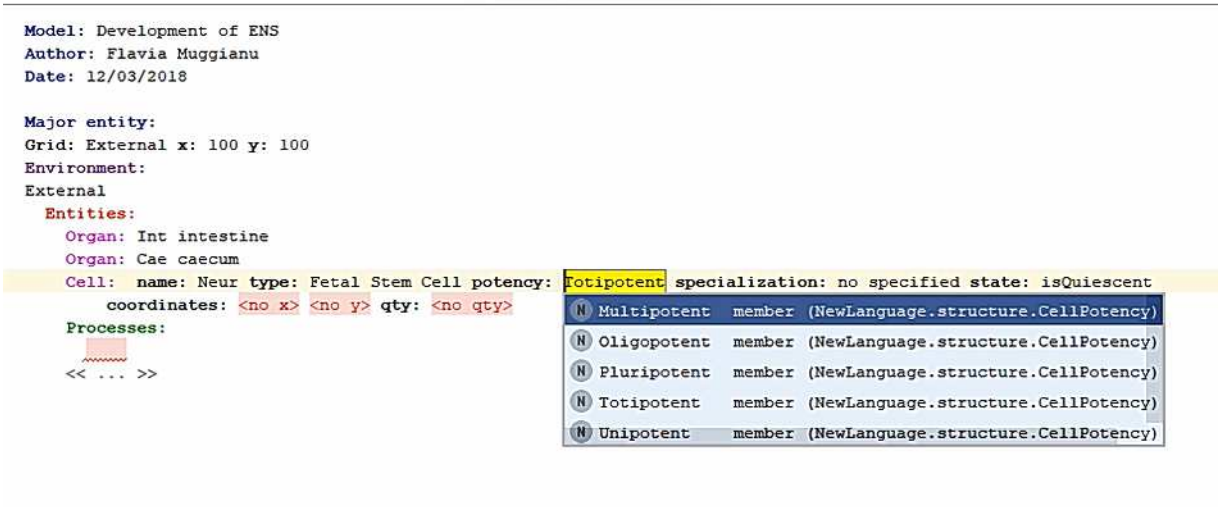

Figure 3.4: Example of editor suggestions

We shown our prototype tool to some professors teaching at the University of Turin, in the department of biology. The aspects they appreciated the most were the easy to use editor and the underlying formalism. Indeed the simple structure of the language, enriched by the editor suggestions, reduces the effort to learn tool usage. Despite the large vocabulary implemented in the language, the users didn't feel any effort in using the language. Indeed the Editor, in addition to suggestions during the set of specifications for entities and processes, guides the user in structuring correctly the model parts, to make the right references and to not forget any fundamental information. The quantity of effort in learning a new tool usage is one of the bigger impediments in the tool diffusion among potential users, for this reason is fundamental to build all the tools that help users when adopt your language.

Despite the good feedback received from users, we decided to make a structural change from the first version of the language. This decision was taken observing that the complexity of this language and its huge vocabulary make clear that this solution is not easily scalable. Each time a new concept needs to be included a modification to the syntax of the language is performed, moreover the language compiler makes a great effort translating the concept in the relative implementation. Furthermore if a more experienced user wants to be able to go in deep in the domain concepts implementations and to change the model of components at the lower level of abstraction, she can't. We want to give to expert users the possibility to manage the underlying formalism, that is, by the way, based on Petri Nets. For these reasons we decided to change the structure of the language to make it simpler, lighter and

usable by a wider target of users. This means that the layer has to cover different layers of usage, from the biologists use mode, to the computational biologist mode. Some of the new language requirements change in respect to the first version:

- Which are the users will use it? Our purpose is to build a language that can be flexible enough to be used by biologists without or with scarce knowledge about computer science, but also by experts in computational biology. To accomplish our purpose we need to create a language that can range from low level implementation of models, very tied to the underlying formalism, to high level implementations, more tied to biological definitions of elements.
- How will be used? Which work process has to support? In this new version we decided to have two different type of users, the biologist and the computational biologist. The first one works only on the model description. The second one has to work both in the model description and in the implementation of lower level functional blocks, to create libraries and to give to biologists tools to work on their process.
- Which are the technologies involved? The answer is still the same described before.

In the next chapter, further analyses about our languages and how it changed from its first version are provided.

Chapter 4

Development of the Language

4.1 Introduction

In this section we present further analysis that had guide us to the full description of our language.

In the 2 Chapter we analyzed which are the weaknesses of existing languages with respect to the ontogenetic field and we stated the need of a semi-general language. In other words a language that has as basic concepts and structures the elements that are in common between the biology subdomains. At the same time it has to be able to define unambiguously other concepts and structures from the biology that the user want to introduce in his model. The fundamental characteristics that we need to introduce are the spatiality properties and the motility processes. The other big issue of existing languages is their not-human readability, all of them are based on XML syntax and to be more understandable they need a graphical interface, especially for biologist users

In the 3 Chapter is described how to structure a language and our analysis on the specific domain that we want to cover with our language. From these analysis are arised some needs like the ability of the language to cover a lot of domain concepts, without have a huge vocabulary and a complex syntax. The structure need to be simple and easy to understand, the notation need to be descriptive and unambiguous, this to cover a large target of users.

As suggested in the 3 Chapter is better to use some other languages concepts and structures if they can fit our specific domain. The language that we have to develop has the main purpose of describing model of complex systems, for this reason we decide to get inspired by the experience gained in an other field in the description of complex systems like the VHSIC (very high-speed computer chip) Hardware systems are. In that field the experts make use of the Gajski-Kuhn Y-chart to guide the design phase of the systems. We get inspiration from this diagram to be guided in the specification of language features. We choose to follow this diagram because we have found various points in common, that we analyze in the section 4.3

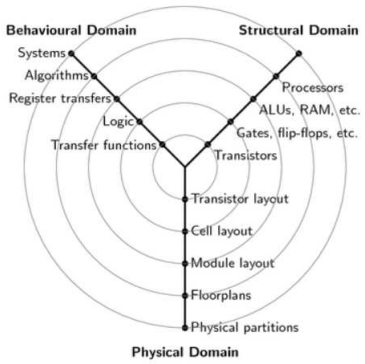

Figure 4.1: Gajski-Kuhn Y-chart

4.2 Gajski-Kuhn Y-chart

The Gajski-Kuhn chart (or Y diagram, Figure **??**) [**14**] depicts the different perspectives in VLSI hardware design. Very-large-scale integration (VLSI) is the process of creating an integrated circuit (IC) by combining hundreds of thousands of transistors or devices into a single chip. VLSI began in the 1970s when complex

semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. Before the introduction of VLSI technology most ICs had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets IC designers add all of these into one chip. A good practice in any system level HW+SW design is to separate/isolate the "application/usage model" from "architecture" and "implementation" details and find the right mapping between them. This is what the chart summarizes.

The Gajski-Kuhn Y-chart is a model which captures the considerations in designing semiconductor devices. The three domains of the Gajski-Kuhn Y-chart are on radial axes. Each of the domains can be divided into levels of abstraction, using concentric rings. At the top level (outer ring), we consider the architecture of the chip; at the lower levels (inner rings), we successively refine the design into finer detailed implementation. The three domains correspond to three different perspectives described below:

- Behavior. This domain describes the temporal and functional behavior of a system.
- Structure. A system is assembled from subsystems. Here the different subsystems and their interconnection to each other is contemplated for each level of abstraction.
- Physical. This domain takes care of the geometric properties of the system and its subsystems. So there is information about the size, the shape and the physical placement. Here are the restrictions about what can be implemented e. g. in respect of the length of connections."

With these three domains the most important properties of a system can be well specified. The domain axes intersect with the circles that show the abstraction levels. The physical and structural domains specify different layout and components respectively that are explicit and don't need more explanations. The behavioural domain describe different level of abstraction of the system functionalities that need some clarification, below we explain in more detail the characteristics of the five circles from highest to lowest level (outer to inner circles):

- Architectural. A system's requirements and its basic concepts for meeting the requirements are specified here.
- Algorithmic. The "how" aspect of a solution is refined. Functional descriptions about how the different subsystems interact, etc. are included.
- Functional block or register-transfer. Detailed descriptions of what is going on, from what register over which line to where a data is transferred, is the contents of this level.
- Logic. The single logic cell is in the focus here, but not limited to AND, OR gates, also Flip-Flops and the interconnections are specified.
- Circuit. This is the actual hardware level. The transistor with its electric characteristics is used to describe the system. Information from this level printed on silicon results in the chip.

4.3 BiSDL Y-Chart

We have seen that the Gajski-Kuhn chart guides the design in every VLSI detail. We want to build a similar chart to help us in the design of the language. The main difference is that the Gajski-Kuhn chart wants to depicts the real system in every part, instead when we talk about system biology models we are not describing in every detail the real system, but in general approximations and omissions are done, this is mainly due for the lack of informations about real biological systems. The common features that we found in these two different domains are the three perspective of the Y-chart, that are the same perspective that we need to take in account when design a model of a biological system. Also the different levels of detail described by the chart are present in our case. Ideed using a similar approach, the three domains can be partially adapted:

- **Structure** domain is the key point of system biology, relations and communications between subsystems are crucial to the whole system evolution
- **Spatiality** domain it's a key point in the ontogeny field, shape, orientation and physical position are fundamental in developmental processes
- **Behaviour** domain can describe specific processes of the system.

The physical domain become the spatiality domain, this because not only shapes and dimensions are important in this case, but also positions, relative positions and directional movement. However each domain is organized hierarchically, from the higher level of detail to the lower one, but in this case are present only four circles.

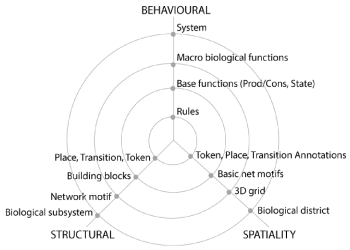

Figure 4.2: Biological Y-chart

In the follow paragraph we describe in detail the Figure **??**

- First circle: Here we have place,transition, token, annotations and channel that are basic PN formalism elements, used to specify the structure and the spatiality of the system in a very low level of detail. The behaviour at this level of detail is expressed through the transitions rules.

- Second circle: At this level the PN elements are composed to create more complex network motifs that can represent architectural, spatial or behavioural modules but with no specification on the domain.

- Third circle: The network motifs, specified in the above described layer, become at this level modules belonging to the biology domain. Each module not only belong to the biology domain but also describe the biology entity from one of the three perspective: behavioural, structural and spatial.

- Fourth circle: This is the higher level of detail, the PN formalism is invisible and modules are purely biological modules that represent: the structure of the system through the biological subsystem implementations, the various processes and functionalities representing the whole system behaviour and the distinct biological district that identify the portions of the system with a specific shape and orientation in the space

The two outer circles deal with modules bio-domain specific, instead the most internal two deal with general Petri-Net constructs and basic elements. This architecture is reflected in the language and in its usage. Indeed the language offers basic instruments to manage PN formalism to build no domain specific nets, in this ways implementations refer to the two inner circles of the diagram. Beyond that, the language offers tools to give to modules domain specific meaning and to use modules as black boxed biological entities, we are referring in this case to the external two circles. These latters tools include the ability to reference ontologies, to univocally identify as a biological entities the modules. This made clear the fact that the domain specification is an intrinsic feature of the language but to be completely expressed libraries, that refer to domain specific ontologies, need to be builded up.

We say that also the usage of the language foolow in some way the Y chart architecture. Indded the expert user can manage at very low level (petri net formalism) the description of the model. Instead the non-expert user can use only bio-domain specific modules, contained in libraries, to describe its model.

In the next paragraph we analyze the chart more in detail:

- **Behavioural domain**: the language must be able to describe the behaviour of the model, this means the processes and functionalities that will be runned during the simulation. Seeing these at different level of detail we can identify: System, it means the functionalities performed from the system as a whole. Macro Biological Functions are simple processes from which arise the system functionalities. Base functions: generic functions like producer/consumer, change of state implemented by basic network motifs. They do not have a domain specific meaning. Rules: basic rules of transitions, that can activate or enable them.

- **Structural domain**: The model is composed by various entities and modules, and the language should describe them. Also in this case various levels of detail are shown in the chart. biological subsystem: differents entities that when connected compose the whole system. network motifs: implemented in the petri-net formalism compose the high modules are described by the composition of and stored in frameworks. building blocks: network motifs describing structures without any biological reference, basic petri net elements like places, transitions, arcs.

- **Spatiality domain**: The language should make explicit the spatiality properties of entities involved in the system. Biological district: macro spatial uniform departments of the model are identified 3D spatial grid: give to the entities of the systems a location in space Basic net motifs connect the grid positions to give directions and express quantitatively their relative distances. Place, transitions, arcs, annotations are used to connote characteristics.

4.4 VHDL

As the Y chart is inspired by the Gajski-Kuhn Y chart the language is inspired by the VHDL.

VHDL (VHSIC Hardware Description Language) [**15**] it's a language to describe the hardware, used in the ICT (Information and Communication Technologies) field. The design phase of hardware development exploit the Gajski-Kuhn Y chart shown above.

VHDL allows the designer to work at various levels of abstraction. Many of the levels are shown pictorially in the Gajski/Kuhn chart. Although VHDL does not support system description from the physical/geometry perspective, many design tools can take behavioral or structural VHDL and generate chip layouts. It is used to document, describe, design, simulate and synthesize. The fundamental unit of VHDL is the Design Entity: which describes components as a black boxes, identifying only the interfaces to the external environment. The Design Entity

can represent components at different levels of the system hierarchy: a logic gate, an integrated circuit, a PCB o an entire system. Then a Design Entity can be associated with different internal architectures, which can follow three approaches. In the body architecture are specified: behaviour, interconnections, input-output relations, components in terms of defined entities. The three approaches are:

- Dataflow: it is based on logic expression that describe the architecture. The possible expressions are: signal assignment, instances of components, blocks declarations, procedures to be used and processes. It can also contain timing informations, that should be respected during simulations.
- Structural: it is made of components interconnected between each others, specifying a system hierarchy. The components included must be present in some reference library. Components declaration can be wrapped in a "package". If more than one implementation of a component exists it is necessary to specify which one is used. There's a PORT MAP section that can specify: a signal, a costeat, the open state or an entity's gate. In general components are connected by the signals.
- Behavioural: it describes algorithms that implement behaviour, indeed it is described following the same way as classic sequential languages (C, Fortran, Pascal, ecc..) . Instructions are executed within a process that is seen as a unique concurrent instruction. This is useful to simulate circuit parts without go down into detail.

Description can have also a mixed approach, this means that more than one architecture types can be specified for a component.

4.5 Biological Systems Description Language BiSDL

4.5.1 Nets Within Nets

Before looking in detail at the language that we have developed, is better to explain well the NWN. formalism that we have introduced in the **??** chapter.

To describe the NWN formalism is necessary first introduce some aspects of the PN formalism [**16**]. This mathematical modeling language make use of a bipartite, directed and weighted graph in which vertex are divided into Place and Transition. Simply put Places are conditions and Transitions are events that use and manage conditions. A Petri Net can have an initial marking, it means that each place or only some of them has an associated integer value, this value represent the quantity of Tokens owned by a Place. The presence of Tokens in a Place can have different meanings, they can simply mean the True value of conditions, or maybe they can identify some resources quantity. Transitions are associated to two functions: the Enabling function and the Firing. The Enabling function is equivalent to the check of the pre-conditions, the firing function is the activation of post-conditions. Enabling and Firing depend also on the weight of the arc that define the number of tokens involved in the condition. In the Figure 4.3[**16**] we show the formal definition of a Petri Net:

A Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$ where:

$P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of places,
$T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of transitions,
$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
$W: F \to \{1, 2, 3, \cdots\}$ is a weight function,
$M_0: P \to \{0, 1, 2, 3, \cdots\}$ is the initial marking,
$P \cap T = \varnothing$ and $P \cup T \neq \varnothing$.

A Petri net structure $N = (P, T, F, W)$ without any specific initial marking is denoted by N.

A Petri net with the given initial marking is denoted by (N, M_0).

Figure 4.3: Formal definition of a Petri Net

The graph arcs connect places to transitions, so we can have Input or Output places. These three entities can have different interpretations depending from the application domain or from the functionalities that need to be described. Some of them are listed in the Figure 4.4[**16**].

Input Places	Transition	Output Places
Preconditions	Event	Postconditions
Input data	Computation step	Output data
Input signals	Signal processor	Output signals
Resources needed	Task or job	Resources released
Conditions	Clause in logic	Conclusion(s)
Buffers	Processor	Buffers

Figure 4.4: Different Interpretations of Petri Net objects

If a transition doesn't have input places than it is called a source transition, this means that it lacks of pre-conditions, so it is always enabled. Instead a transition without output places is named sink transition, it is clear that it will consume tokens without produce anything. In the Figure 4.5[**16**] we provide an example of a simple Petri Net representing a chemical reaction.

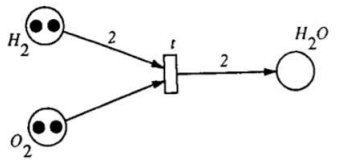

Figure 4.5: Chemical reaction implementation

Petri Nets can model different basic concepts, useful in different fields of knowledge: finite-state machines, parallel activities, dataflow computation, communication protocols, Synchronization Control, producers-consumers paradigm.

After this overview about PN. formalism we can introduce some concepts about the NWN [**17**] formalism. We said that the Tokens of a Petri Net are simply described with an integer denoting the presence of a precise quantity of unidentified resources. Actually there are different type of tokens, they can be Integer, Float, Double, String, Colors. But if there is the need to represent some complex resources or entities with tokens then is necessary a more complex type for tokens. The natural consequence of this requirement is the use of Petri Net as tokens, this increase the expression of models without increase the complexity of the formalism. In the Figure 4.6[**17**]

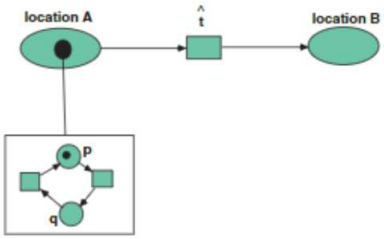

Figure 4.6: General example of NWN

This extension of the formalism naturally describe systems with hierarchical architectures, giving the possibility to describe each layer of the system at the same level of detail. Examples of the implementation of this formalism can be found in [**18**] and [**19**]. This new formalism require an inter-layer communication mechanism, one solution is the concept of Channel. Channel is a synchronous communication mechanisms that is able to fire two transitions, belonging to different nets, at the same time. It can also exchange tokens between the two nets. The transitions involved in this mechanisms have different roles. One is called uplink and the other is named downlink. The uplink is responsible for the activation of the channel.

4.5.2 VHDL common features

As VHDL intends to describe all the aspects of Digital Systems models, and its design follows the relative Gajski-Kuhn Y chart, the BiSDL as well intends to be able to describe all the aspects of the three domains described in the Biological Y chart. VHDL inspired us because it has some features compliant with the description of models of biological systems, like:

- Modularity
- Differentiation between structural architecture and behavioural architecture
- The concept of entity: the module seen as a black box.

Modularity is important because we are dealing with systems that are composed by subsystems, in other words they are modular by their nature. Modularity give also another advantage that is reusability of subcomponents of the system without structural modification, but also give the possibility to do easily modifications on some system parts without affecting the whole system.

For each module different aspects need to be described: the submodules that compose it, the processes in which these submodules are involved, the spatiality relation between them and the setup of the module at the start of simulation. The structure of the first version of the language is mainteined in this way.

Another important aspect of the module description are the arguments that enable the usability of the module as a black box, in other words without knowledge about its internal functioning and implementation. These arguments can be seen as the interfaces or setup parameter of the module, because they permit to connect the module to other modules and to set some internal parameters. To be connected

the modules they need to share some submodule or to use a special entity that is the channel, a synchronous mechanism of communication.

The modules to be unambiguously identified as a biological entities need a reference to some recognized source of knowledge. This feature is useful not only to overcome the limitations of general languages but also to make the language able to include new modules belonging to different biology subdomains. Indeed when a new entity or process need to be included is required only the implementation with low level elements of formalism and the biological reference to be attached.

The modules with this biological meaning can be then stored in libraries, to be then reused mostly by non-expert user, in this way we obtain the first version of the language, in which only biological elements could be used. The internal structure architecture of modules can use both petri net formalism elements and also other modules. This satisfy the requirement of a hierarchical architecture

Each module is described following the same template that we illustrate below. The template cover both the usage types:

- non-expert user mode
- expert user mode

They use the language in different ways, we specify these aspects later.

4.5.3 BiSDL: main concepts and structure

This is the core section of the whole work, because we are going to explain the language.

All the modules that compose the model of the biological system are described using the same syntax and template, exploiting each time different features, according to the level of abstraction of the module implied. The structure of modules is similar to the one used for the first version of the language discussed in the **??** chapter.

Summarizing each module can have:

- Set of parameters that are used as interfaces when it is exploited as a black box.
- Set of entities that compose the internal structure.
- Set of processes in which are involved the entities.
- Biological reference.

Entities and processes are mainly of two types: these belonging to the biological domain and these belonging to the Petri Net formalism. The first type modules are much more complex and they are contained in some library. These modules are implemented using entities and processes typical of the Petri Net formalism and in addition they have biological references. The syntax of each module is similar: name and parameters, both change according to the implementation choices. The second

type of entities and processes are the basic element of NWN formalism: Places, Channels and Transitions. The syntax is fixed in the language.

In the follow paragraphs we explain in detail the template (Figure 4.7) used for modules implementation and the language syntax.

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

ONTOLOGY <ONTOLOGY_NAME>
.
.
ONTOLOGY <ONTOLOGY_NAME>


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

      BIOLOGICAL REFERENCE
            <ONTOLOGY_NAME>.<ID_WITHIN_THE_ONTOLOGY>

      ENTITIES
            PLACE <name_place>,....
            ENTITY <name_entity>,....
            CHANNEL :<name_channel>(),....

      INIT
            <name_place>.attribute(<value>)
            <name_entity>.attribute(<value>)
            <name_entity> = <name_module_entity>(<parameter>,....)

      PROCESSES
            process(<places_declaration>,..., {function_of_transition}, delay(N))

            <name_module_process>(<param>,....)

            foreach( <name_entity_vector> | <name_place_vector> ){
            ...
            }
END
```

Figure 4.7: BiSDL Template. The main structure of the language and its main keywords and constructs

- PACKAGE declarations followed by the name of a domain specific library must be included when some library modules need to be used. Each library correspond to a PACKAGE declaration. Libraries can be organized in trees, so if we want to use a sublibrary, all the parent in the library tree should be declared, separated by a dot. This is a crucial feature because the domain specific entities belong only to libraries. They cover the two outer circles of the biological Y diagram, that are the ones used by the non-expert user. The organization of libraries in a tree can reflect the organization of the biological knowledge, in this way the user is helped in the search of the right modules.

- ONTOLOGY declarations followed by the ontology/database/paper name must be included for each module with biological meaning. Sometimes including this reference means to add complexity to elements of the model that have very simple implementation, but this is necessary to give domain robustness to the libraries. Pose the reference to ontologies as a requirement overcome the limitations of other languages analyzed in the background, without this

requirement users can not include the domain specific reference this lead to a model not reusable or reusable with difficulties.

- The MODULE keyword starts the module implementation, that end with the END keyword. Module has a name that need to be unique inside the package at which it belongs. Then the arguments, described above, has to be declared. Modules with same name can coexist in the same library only if they have different arguments. The arguments are then used inside the module body.
- BIOLOGICAL REFERENCE identify the portion of code that include the ontology definition that matches the module identity. The identity has to refer to sone of the ontology declared in the ONTOLOGY section, the id within the ontology needs to be specified with the follow syntax:

  ontology_name.id

- ENTITIES start one of the most important code section, because here are listed all the submodules that compose our module. They can be of different types:
  - PLACE: Petri Net basic element
  - CHANNEL: this is a special component that put in connection two transition, making able two different nets to synchrounously connect. They have a particular syntax that is:

    : name_channel ()

  - ENTITY: here can be initialized names for modules implemented in one of the libraries declared above with the PACKAGE inscription. For the elements of type ENTITY in this section is necessary only to declare the name, the specific module used to implement them need to be declared in the next INIT section.

  When one of the subcomponent is needed in a quantity bigger than one we need to use a vector, various entities of the same type grouped together. To declare a vector this notation is used:

  name_place * N
  where N is the numerosity of such group.

- The INIT section aims to set some properties of entities:
  - for ENTITY components need to be declared the implementation module picked from some library declared above, setting the required parameters. Sometimes is necessary to access some internal structure of the implementation to better connect modules, each place used in the module structure can be accessed using this syntax:
    name_entity.name_place
    Information about places that can be accessed need to be specified in the module documentation.

– for PLACE components should be set a name, the initial marking tokens and the ontology id. When setting the initial marking of places some specifications should be made: the type of token, the value and the quantity if there is more than one token of same type and value. If there is more than one token but with different type or value they should be listed, separated by a comma. The token type can be int, float, double, string, black_token or can be a module defined from the user or a module belonging to some library included with PACKAGE keyword. The operations that can be performed on tokens change according to their type. For the numerical types all the basic operations are allowed (sum, difference, division, multiplication). For the string type is allowed the concatenation, performed using the symbol '+'. The syntax to assign tokens to a place is:

  name_place .token(token_type * N)
  Where N is the amount of the token.

  The assignment is always an operation of adding tokens to the pool of place tokens. In the PROCESS section description we explain in detail how menage tokens to be moved and deleted.

– for both type of subcomponents the location in the grid needs to be specified when needed a specific location in space. declare the state of submodules at the start of simulation, this means that for ENTITY type entities need to be declared the implementation used, for PLACE entities need to be set the initial marking if there's one. CHANNEL entities are not included in this section because they are used as arguments for ENTITY implementations or for PROCESSES implementations. The syntax to assign tokens to a place is:

– when managing a group of components is required to access each element of the group using its index:

  name_component.index
  Where index is the value of the index

  If there is the necessity to access each entity of a vector to make the same operation on them the foreach construct is used

  foreach(name_component)
  {
  operation on name_componenti
  }

  for operation we intend one of the initialization operations mentioned before.

  if the operation on the group entities is different for some or for all, then the for ccle need to be used, making a check inside the body on the index of the subcomponent of the group:

for(i=0 ; i less_equal group_size ; i++)
{
operation on name_component.i
}

Where i can be freely chosen.

- Some characteristics of PLACES and ENTITIES can be accessed or set as attributes: name, token, unit of measure, ontology_id and last but not least the coordinates within the 3D grid. For ENTITY components in general these attributes are setted as parameters of the module declaration.
- The coordinates are related to the grid shown in the Y chart, that is necessary to bring out the biological properties related to the spatiality. The grid is always present, when the user doesn't provide coordinates then default coordinates are assigned to entities.
- There is one other attribute not mentioned before, the reason is that this attribute is more related to the simulation then to the model itself. This attribute can be set only for entities of type ENTITY and it is the relative speed with which they will be simulated. The speed can be a fraction or a multiplicator of the time unit which will be chosen by the highest module of the systems, so theminimum time cycle of the simulation is equal to the time cycle of the faster module. The speed assigned by the high level module affects in particular the delay of transitions of lower modules.

- The PROCESSES section include all the relations between entities or the functions of single entities. The relations can represent spatial relations that participate in implementing the spatiality of the model, or functional relations intended as those processes implementing the behaviour of the model. For this section the language provides some basic relations between entities. These low level functions belong to the second inner circle of the Y chart and can be described only through one construct with floating parameters.

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

  This construct represent a transition, that is one of the minimal element to describe the functional behaviour of the system. To specify the transition function is necessary first to identify the entities involved and the directions of the arcs that connect them to transitions.
  The entities with the same direction of arcs are listed in **entities_declaration**, the **keyword** instead specify the arcs directions.
  keyword can be: BI, IN, OUT that represent respectively bidirectional, incoming and outgoing arcs.
  The incoming arcs are these responsible for consuming tokens, the outgoing arcs are responsible for producing tokens, the bidirectional arcs perform a

check on the tokens presence. Without specifications in the transition function: random tokens will be consumed or checked and black tokens will be produced. Th arcs will no have any effect only if for their places there is a customized operations specified in the transition function.

entities_declaration is the list of entities involved and has a precise syntax to be declared:

{ **entity_name[arcs_number],...,entity_name[arcs_number]** }

The process construct has as last attribute the **delay(N)**, this define the amount N of time that elapses from the enabling to the activation of the transition. The N value is a fraction of the time unit, as explained for the attribute relative time of the entities described above, in the INIT section.

Transitions, according to the PN formalism, have both enabling and activation functions, which can be specified in the declaration of these basic functions in the { **function_of_transition** } parameter.
The function of transition has the main purpose to manage tokens between places. We know from the PN formalism that transitions have two types of functions: enabling and the firing.

- **Enabling**: If to enable the transition is required only the token presence in the pre-condition places no specification need to be made, instead if some customized checking need to be performed we have to use the IF construct.
  { **if(** ***customized condition*** **)** }

  The customized condition mostly involve the checking of token values, to access them the follow syntax can be used:
  place_a.token.value(value_specification) It search for the presence of a token with the value specified
  place_a.token.type(type_specification) It search for the presence of a token with the type specified

  All these type of checking operations can be put together using the Boolean operands: and/or having respectively these syntax **AND** and **OR**.
- **Firing** If there is the customized enabling function the Firing function is put inside the IF construct body. otherwise is the only function of transition. All the operations performed by this function are listed separated by **;**. Besides the basic operations on token values (addition, difference, division, multiplication, concatenation) and on post-conditional places token assignment, other type of operation can be:
  place_a.token=place_b.token exchange of random token between two place

place_a.token=place_b.token.value(value_specification) exchange of a specific token between two places.

place_a.token=place_b.token.type(type_specification) exchange of a specific token between two places

place_b.token++ adding a black token to the pool.

place_b.token- - deleting a random token

place_b.token.value(value_specification)++ adding a token with specified value to the pool. Can be done also specifying the type

place_b.token.type(type_specification)- - deleting a token with specified type to the pool. Can be done also specifying the value
To repeat the same operation more then one time, it needs to be put inside a for loop.
A special firing function of the transition is the channel, that is an as-

sociation with another transition. The channel has two main properties: the direction of activation and the direction of the token exchange. To set the direction of the activation we have to specify which is the the the transition that can start the communication (named upperlink) and which is the transition that receive the communication in a synchronous way (named downlink). The downlink is the module that declare the channel in the ENTITIES section, the upperlink access the channel with this syntax:
module_downlink:channel()
In the process construct we can specify through the function declaration which tokens are exchanged within the channel and which are the input and output places. the syntax is the follow:
when(module_downlink:channel(tk)) { tk = place_IN.token }
or
when(module_downlink:channel(tk)) { place_OUT.token = tk }

If no specification about value or type of tokens is made then random tokens will be exchanged through the channel.

The **process** construct described above is a simple construct, defined in the Y chat as basic building block, more complex relations are the modules included in libraries announced with the PACKAGE inscription and corresponding in the Y chart to the two outer circles.
Their syntax is simply the name followed by the specifications of arguments that differ from one to another and are documented in the library. Typically the arguments of these relations are entities to be put in relation between each other and functions performed by such relation. All these functions involve manipulation of tokens, these operations differs from the type of tokens involved. Indeed in base of their type differ type of checks and operations can be done.

As anticipated, when introducing the template, the language can be used by different types of users. The non-expert user can rely on modules belonging to domain specific libraries, in this way their have the possibility to deal only with modules with a biological meaning. This usage can be associated to the two more external circles of the Y chart. The expert user instead can use all the language basic tools to build his own modules or to customize the existing ones. Therefore these users can explore all levels of detail of the Y chart.

4.6 Use case: C. Elegans

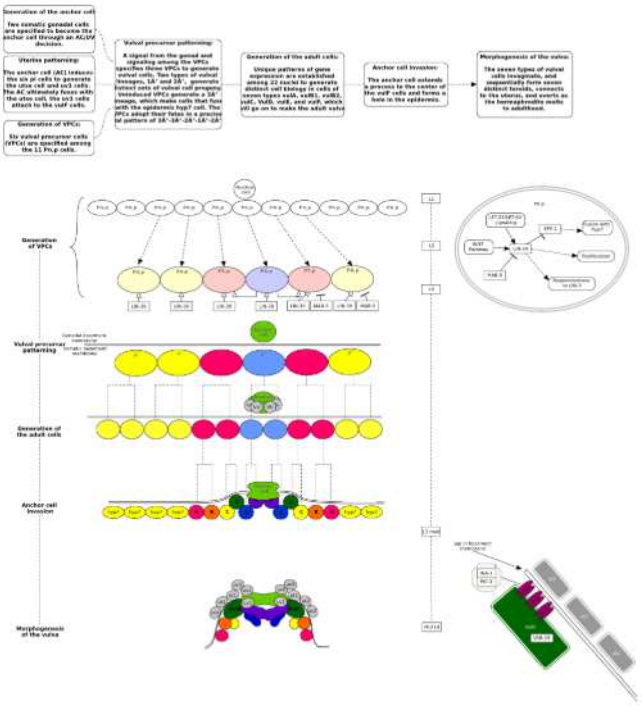

Figure 4.8: Scheme of C. elegans vulval development [**21**]

The Figure 4.8 describe the C. elegans vulval development [**20**] that is one of the most studied use case of animal organogenesis. It has achieved this importance because is a relatively simple system to analyze. What underlies the development of this system is the complex integration of various intercellular signaling, signal transduction, and transcriptional mechanisms . This organ is the connection between the hermaphrodite uterus and the outside of the nematode. The most important actor of this system is the anchor cell that is a single cell of the somatic gonad. It has the role of organize and coordinate the development of the vulva from the epidermal precursors and also the physical connection of the epidermis with the uterus.

The anchor cell to accomplish this goal create biomolecular gradients by releasing signaling molecules that diffuse outwardly. These gradients are critical for cellular identity and cell relocation. The gradient produced by cell influences cellular fate by their temporal and spatial characteristics. In this case the cells affected by this

mechanisms are six differentiating cells that can be differentated in three different types of fates: 1 generates vulE and vulF mature vulval cells, 2 generates vulA, vulB1, vulB2, vulC and vulD cells, 3 generates non-specialized epidermis.

These differentiating cells are disposed in a horizontal way and the anchor cell stay up them in th middle of the length of the space occupied by the si cells. down these cells is located the hypoderm, a population of epidermal cells distributed along the length of the six differentiating cells. The signal received by these cells from the anchor cell differs according to distance of each cell from the anchor cell. The nearest cell has a juxtacrine interaction with the anchor cell, the two cells that are on the left and on the right of the nearest one, have a paracrine interaction with the anchor cell. All of the six cells have and interaction with the hypoderm, and also a bilateral interaction with the nearest differentiating cells that are on the left and on the right.

All these signals affect the inner biochemical processes of the differentiating cells. The signals sended from the anchor cell and the hypoderm are catched by the EGF (LIN-3) receptor that is involved in the regulative pathway of the MPK-1 protein. The bilateral signaling is mediated by the LIN-12 (Notch). EGF promotes the 1 fate while LIN-12 promotes the 2 fate. Both EGF and Notch prevent cells from get the 3 fate. These two are also antagonist: EGF downregulate the Notch-like receptor LIN-12, while the LIN-12 signaling induces negative regulators of EGF-receptor signaling such as MAP kinase phosphatase LIP-1 and the tyrosine kinase ARK-1.

4.7 C. Elegans vulva development model - Analysis

To describe the model with our language is useful to use the Y chart: to be sure to cover all levels of detail, without leaving out any aspect of the model. To read the chart in a systematic way, following a top-down approach in the description of the model, we follow the arrows. The model has a hierarchical architecture composed by two layer, for each one the description is helped by the chart use. The first layer named system net describe the system at higher level of detail, the second layer describe more in detail the developmental cells, while the anchor cell and the hypoderm are not described in detail but they are seen as black boxes.

Starting from the first layer, looking at the behavioural domain, we can identify the differentiation of cells as the major process, considering the system at the higher level of detail. The structural domain wants to identify the major subsystems that in this case can be identified in the anchor cell, six differentiating cells and the hypoderm. These three entities at the same time identify three different biological districts in the spatial domain. At the same level in the second layer the System behaviour is covered by the living cell functions. The biological subsystems are genes, receptors, proteins and pathway proteins, from the spatiality point of view there's not a specific organization of biological districts. This means that the biological districts are not considered in this system, because the focus is only on the proteins interactions, that have not a precise location within the cell. Going

in deep with the level of detail we identify the different types of signalling as the macro biological functions, referring to the behaviour domain of the first layer. The network motifs that describe the structure are those describing the various types of cells and the channels through which signals can pass. The biological districts are mapped into a 3D grid that cover 13 placements: 1 for the anchor cell, 6 for each developing cell and 6 for the hypoderm that is distributed along the width occupied by developing cells. Dealing about the second layer the macro biological functions in this system are the reception of external signals and the functionalities within pathways: transcription, activation and degradation of proteins. In this case the network motifs present are those of pathways components. This model lacks also a 3D grid for the same reasons of the lack of biological districts. At the third circle, talking about the System Net, we can found the base functions of petri nets used to implement Macro biological functions previously named, in particular: signals are implemented through read/write cross layer functions for the lateral signalling, and with a write cross-layer for direct signalling of anchor cell and hypoderm. The petri nets that build the structure cells are simple places, they differ only for the tokens they contain. Relative positions and directions are expressed by the network architecture, positions of places in the grid and the tokens flow. At the low level in the System Net case are not present significative rules in transitions. The structure is specified by tokens: for the anchor cell is only an integer, for the hypoderm is a black token, instead for differentiating cells is a more complex token specified under the PN formalism. Annotations specify directions of lateral signalling and the type of anchor signals, that depend on nearness of developing cells to the anchor cell. Speaking about the net token: the general basic functions used in transcription and degradation are respectively these of continuous production of tokens and of deletion of tokens with the help of a counter. The building blocks are those for activation of proteins. There are not net motifs for the spatiality explicit. The rules are present in the degradation and in the transcription to implement biological conditions. Black tokens are used to represent quantity of resources (genes, miRNAs, proteins). Annotations make a distinctions between signals received from left or right cells.

With this analysis the model has been fragmented to be better described in detail and furthermore to better write a modular code in our language.

We can starting describing the implementation of the first layer of the model.

4.7.1 System net implementation (performed by non-expert user)

There are three main entities: anchor cell, differentiating cells and hypoderm. These entities are involved in communication processes, in other words in the exchange of signals. So, the following model (4.11) describe at high level the whole system of the first layer:

So in the ENTITIES section we have to put the entities listed above. Than in the INIT section is specified the implementation chosen: the simple_cell module that take as arguments the name of the cell, the type of token and the ontology reference. The complexity of the cell increase if the token used is a complex one, indeed anchor

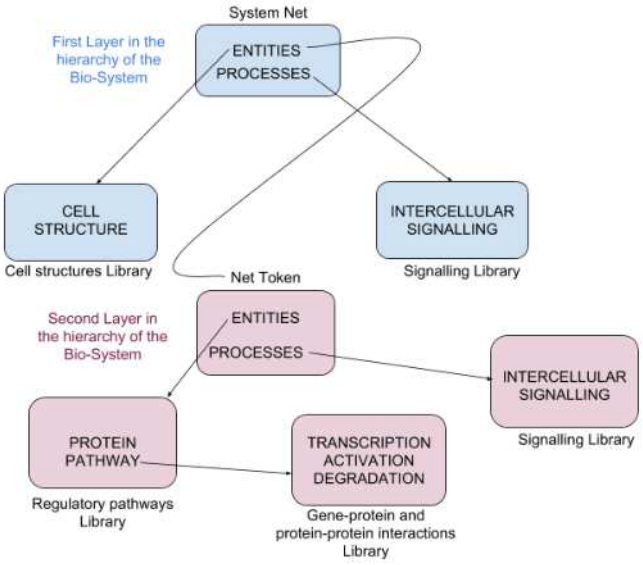

Figure 4.9: Use Case Model architecture.

Figure 4.10: System low level detail view.

cell and hypoderm have a simple token, an int that identify only the existence of the cell, instead differentiating cells have another module as token. This last token with its complexity represent the second layer of the system hierarchy, that we eill see in detail in the next paragraphs. We set another entities is parameter, that is very important: the location within the grid, to give them a spatial notation.

In the PROCESSSES section we list all the intercellular-communication mechanisms. The anchor cell and the hypoderm communicate with differentiating cell with the same hierarchical signalling, the difference between them is the use of different channels belonging to the tokens of the differentiating cells. The differentiating cells communicate with each other through the bilateral module that exploit the DSL channels, this type of implementation represent the lateral signalling.

Now, for those that are confident with Petri Net modeling, we can go in a more deep explanation of the modules implementations used in the System Net. Indeed the use of these modules increase the transparency for the final user, and it ensure the biological identification of entities through the ONTOLOGY reference.

This choice also improves modularity and reusability. For example when a more detailed implementation of “cell” modules is needed another module can be included, changing only the initialization of cells.

```
PACKAGE CellStructures
PACKAGE Signalling

ONTOLOGY CElegansAnatomy = https://www.wormbase.org/species/all/anatomy_term/
ONTOLOGY CElegansProcesses = https://www.wormbase.org/resources/wbprocess/

MODULE system_net
      ENTITIES
            ENTITY anchor_cell, dev_cell*6, hypoderm_7*6
      INIT
            anchor_cell = simple_cell( "anchor_cell",int(1),
            CElegansAnatomy.WBbt:0004522)

            foreach( hypoderm_7 ){
                  hypoderm_7 =
                  simple_cell("hypoderm_7_"+hypoderm_7.index,
                  black_token(),CElegansAnatomy.WBbt:0004200)
            }
            dev_cell = simple_cell( "Differentiating_cell_0",
            net_token() , CElegansAnatomy.WBbt:0008112 )
            dev_cell = simple_cell( "Differentiating_cell_1",
            net_token() , CElegansAnatomy.WBbt:0008117 )
            dev_cell = simple_cell( "Differentiating_cell_2",
            net_token() , CElegansAnatomy.WBbt:0008121 )
            dev_cell = simple_cell( "Differentiating_cell_3",
            net_token() , CElegansAnatomy.WBbt:0008125 )
            dev_cell = simple_cell( "Differentiating_cell_4",
            net_token() , CElegansAnatomy.WBbt:0008129 )
            dev_cell = simple_cell( "Differentiating_cell_5",
            net_token() , CElegansAnatomy.WBbt:0008133 )

            anchor_cell.coordinates(1,5)
            for(i=0; i<6; i++){
                  dev_cell->i.coordinates(4, i+2)
                  hypoderm_7->i.coordinates(7, i+2)
            }
      PROCESSES
            cellular_communication(dev_cell->2, anchor_cell,
            CElegansProcesses.WBbiopr:00000071, dev_cell->2.token.paracrine
            interaction())
            cellular_communication(dev_cell->3, anchor_cell,
            CElegansProcesses.WBbiopr:00000071,
            dev_cell->3.token.juxtacrine_interaction())
            cellular_communication(dev_cell->4, anchor_cell,
            CElegansProcesses.WBbiopr:00000071, dev_cell->4.token.paracrine
            interaction())
            foreach( dev_cell ){
                  cellular_communication(dev_cell, hypoderm_7,
                  CElegansProcesses.WBbiopr:00000071, dev_cell.token.hyp7_interaction() )
                  if( dev_cell.index<5 ){
                        bilateral( dev_cell->dev_cell.index, dev_cell-> dev_cell.index+1,
                        CElegansProcesses.WBbiopr:00000072 )
                  }
            }
END
```

Figure 4.11: System low level detail implementation. Performed by end-user

4.7.2 Simple cell implementation (performed by expert user and included in CellStructures Library)

In the Figure4.13 is shown the implementation of the simple cell contained in the Figure 4.12 library. It is a very simple implementation. The cell is represented

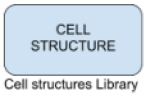

Figure 4.12: Library modules used to describe the cell.

```
MODULE simple_cell (STRING name, TOKEN token, STRING ID)

      BIOLOGICAL REFERENCE
            ID
      ENTITIES
            PLACE cell
      INIT
            cell.name = name
            cell.token( token )
END
```

Figure 4.13: Library module implementation to describe a cell at a high level of detail

only from a place and can be customized through the specification of parameters: the name, the token representing the inner structure of the cell and the ontology reference.

The following library modules specify the processes used in 4.10:

4.7.3 Intercellular signalling implementation (performed by expert user and included in Signalling Library)

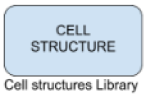

Figure 4.14: Library modules used to describe the cell.

```
MODULE cellular_communication(ENTITY a, ENTITY b, STRING ID, CHANNEL :chan())

      BIOLOGICAL REFERENCE
            ID
      PROCESSES
            process(BI:{a.cell,b.cell}, :chan())
END
```

Figure 4.15

In the Figure 4.15 is shown the implementation of the inter-cellular communication contained in the Figure 4.14 library. Also this module is very simple. The transition is connected through bidirectional arcs with two entities. The enabling function is defined by the arcs structure, there are only bidirectional arcs so if there are tokens in both entities it can be enabled. During the Firing the channel is activated, so another transition will be fired.

```
MODULE bilateral ( ENTITY x, ENTITY y, STRING ID)

      BIOLOGICAL REFERENCE
            ID
      ENTITIES
            PLACE a, b
      INIT

      PROCESSES
            process( BI:{x.cell}, IN:{b], x.cell.token:DSL_right_i(b.value) )
            process( OUT:{b},BI:{y.cell],{y.cell.token:DSL_left_o(value);b.token=value;})
            process( IN:{a}, BI:{x.cell],y.cell.token:DSL_left_i(a.value) )
            process(OUT:{a],BI:{y.cell},{x.cell.token:DSL_right_o(value);a.token=value;})
END
```

Figure 4.16

In the Figure4.16 is shown the implementation of the bilateral communication contained in the Figure 4.14 library. This implementation is a little bit complex, because it has to represent a two-side synchronous communication between two cells, that is performed through the channels. The first two processes are equal to the second two processes, because they perform the same action for each cell. One process is responsible of the lecture of a value from a cell through the channel. The other process is responsible to take this value and writing it into the other cell exploiting its channel. The a and b Places are necessary to store the value between the write and read actions.
The last two implementations of cellular communications can be used only with

Entities that are initialized using the simple_cell implementation, because in both cellular mechanisms is accessed a specific Place named "cell".

4.7.4 Net token implementation (performed by non-expert user)

The initialization of dev_cell places (in the system_net) use as a token a net describing the second layer of the model, the internal structure of the developing cells.

Figure 4.17: Second layer, at a low level of detail.

The Figure 4.18 is the implemetation of the second layer of the system, it represents the internal structures of the differentiating cells. We can see that the entities involved in this subsystem are: the cascade mapkinase, the lin_12 biochemical pathway and various protein both in the active state and the inactive state. The other main entities are all the channels that make the cell able to communicate to the external environment, in this case the external environment is the first layer of the system. All the proteins are defined using the implementation of the module simple_protein, that take into account the name, the ontology reference and a defined quantity of the black_token, because there is the need to only set an amount of resources. The pathways have instead a more customized module implementation.

```
PACKAGE Pathways
PACKAGE Signalling

ONTOLOGY CElegansGenes = "https://www.wormbase.org/species/c_elegans/gene/"
ONTOLOGY CElegansProteins = "https://www.wormbase.org/species/c_elegans/protein/"

MODULE net_token

    ENTITIES
        ENTITY p_mpk, p_lin12, r_chan, r_left, lin_3, mpk_1_act, lin_12_act, right,
        left, lin_12, mpk_1, mpk_1_act, lst, dpy_23
        CHANNEL :left_LS(), :right_LS(), :juxtacrine_interaction(),
        :hyp7_interaction(), :paracrine_interaction(),:DSL_right_i(), :DSL_left_i()

    INIT
        lin_3 = simple_protein("lin_3", CElegansProteins.WP:CE28021, black_token()*3)
        mpk_1 = simple_protein("mpk_1 ", CElegansProteins.WP:CE01583,black_token()*3)
        mpk_1_act = simple_protein("mpk_1_act", CElegansProteins.WP:CE01583, int(0))
        lin_12 = simple_protein("lin_3",CElegansProteins.WP:CE00274, black_token()*3)
        lin_12_act= simple_protein("lin_12_act",CElegansProteins.WP:CE00274, int(0))
        let_23 = simple_protein("let_23",CElegansProteins.WP:CE03840, black_token()*3)
        lst = simple_protein("lst ", CElegansProteins.WP:CE52563,black_token()*3)
        dpy_23 = simple_protein("dpy_23 ", CElegansProteins.WP:CE33814)

        p_mpk = pathway_mpk(lin_3, mpk_1_act, mpk_1, let_23 )
        p_lin12 = pathway_lin12(lin_12_act, mpk_1_act, right, left,:DSL_right_i(),
        :DSL_left_i())

    PROCESSES

        transcription_gene_expression_regulation(
        mpk_1, CElegansGenes.WBGene00003401, lst)
        transcription_gene_expression_regulation(
        lst, CElegansGenes.WBGene00016889, mpk_1_act, lin_12_act)
        transcription_mediated(dpy_23, CElegansGenes.WBGene00001082, lin_12_act,4,4)
        activation(lst, lin_12_act)
        degradation_regulated( let_23, dpy_23, 2, 3)
        degradation( lst, 100)
        juxtacrine_interaction(lin_3, :juxtacrine_interaction())
        paracrine_interaction(lin_3, :paracrine_interaction())
        hyp7_interaction(lin_3, :hyp7_interaction())
        output( mpk_1_act, lin_12_act, :DSL_right_o(), :DSL_left_o())
END
```

Figure 4.18: Implementation of the second layer, at a low level of detail.

The processes that are performed by this system are:

- transcriptions, degradations and activations of proteins, all processes that involve these protein that participate to the pathways.
- all the interactions that exploit channels to communicate to the external environment.

4.7.5 Interactions implementations (performed by expert user and included in Signalling Library)

The following modules implement the processes of the module 4.16:

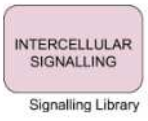

Figure 4.19: Library containing signal processes for the second layer.

The implementations in Figures 4.20, 4.21, 4.22, represent all the interactions with the external environment, in particular they describe the path of the EGF (lin3) that is received from the anchor cell and from the hypoderm to increment

the quantity of the EGF within the cell. All of these interaction exploit a the signal_receiver_throughput module to implement the reception of the signal with a different throughput that represent the power of signals.

```
MODULE juxtacrine_interaction ( ENTITY x, CHANNEL :juxtacrine_interaction())

     PROCESSES
          signal_receiver_throughput( lin_3, :juxtacrine_interaction(), 4)
END
```

Figure 4.20

```
MODULE paracrine_interaction ( ENTITY x, CHANNEL :paracrine_interaction())

     PROCESSES
          signal_receiver_throughput( lin_3, :paracrine_interaction(), 2)

END
```

Figure 4.21

```
MODULE hyp_7_interaction ( ENTITY lin_3, CHANNEL :hyp7_interaction() )

     PROCESSES
          signal_receiver_throughput( lin_3, :hyp7_interaction(), 1)
END
```

Figure 4.22

4.7.6 Protein, pathways implementation (performed by expert user and included in Pathways Library)

The reasons which led at the implementation of the simple_protein are the same of those of simple_cell.

Figure 4.23: Library containing pathways for the second layer.

In the Figure 4.24 is described the simple implementation of the proteins, that is simply an association between the name, the ontology reference and the tokens that represent the amount of protein.

In the Figure 4.25 is shown the custom implementation of the cascade mapkinase. The entities involved are different proteins. The main processes are:

- the transcriptions of the proteins, that take into account the protein and the ontology reference of the gene involved in the transcription

```
MODULE simple_protein (STRING name,STRING ID, TOKEN token)

      BIOLOGICAL REFERENCE
            ID
      ENTITIES
            PLACE protein
      INIT
            protein.name = name
            protein.token(token)
END
```

Figure 4.24

```
PACKAGE PathwaysRegulations
ONTOLOGY CEleganGenes = "https://www.wormbase.org/species/c_elegans/gene/"
ONTOLOGY CEleganProteins = "https://www.wormbase.org/species/c_elegans/protein/"
ONTOLOGY CEleganProcesses = "https://www.wormbase.org/resources/wbprocess/"


MODULE pathway_mpk (ENTITY lin_3, ENTITY mpk_1_act, ENTITY mpk_1, ENTITY let_23)

      BIOLOGICAL REFERENCE
            CEleganProcess.WBbiopr:00000070

      ENTITIES
            ENTITY let_60, let_60_act,let_23_act, sem_5, sem_5_act
      INIT
            sem_5 = simple_protein("sem_5 ",CEleganProteins.WP:CE01784,black_token()*3)
            let_60 = simple_protein("let_60",CEleganProteins.WP:CE03827,black_token()*3)
            sem_5_act=simple_protein("sem_5_act",
            CEleganProteins.WP:CE01784,black_token()*3)
            let_60_act = simple_protein("let_60_act",
            CEleganProteins.WP:CE03827,black_token()*3)
            let_23_act = simple_protein("let_23_act",
            CEleganProteins.WP:CE03840,black_token()*3)

      PROCESSES
            transcription(let_60, CEleganGenes.WBGene00002335)
            transcription(let_23, CEleganGenes.WBGene00002299)
            transcription(sem_5, CEleganGenes.WBGene00004774)
            activation(let_23, lin_3, let_23_act)
            activation(sem_5, let_23_act, sem_5_act )
            activation(let_60, sem_5_act, let_60_act)
            activation(mpk_1, let_60_act, mpk_1_act)
            degradation(mpk_1_act, 100)
            degradation(lst, 100)
END
```

Figure 4.25

- the activations of the proteins, that take into account the protein to activate the protein that is resposible of the activation and the protein the state active
- the degradations of some protein take into account the protein and the amount of time to be performed.

In the Figure4.26 is shown the custom implementation of the lin_12 pathway. The entities involved here are different proteins, in particular we can find: lin_12 in both states active and inactive, two different populations of DSL proteins, one affected by the lateral signalling coming from the left cell and the other affected by the lateral signalling coming from the right cell. There are also two channels, that are the entry point for the lateral signalling coming from the both adjacents cells. The processes involved are:

- the transcription of lin_12, that take into account the lin_12 protein and the ontology reference of the gene involved in the transcription
- the activations of lin_12 performed by the different DSL proteins, and the activation that happens when a certain threshold the amount of lin_12 in the inactive state is reached.
- the degradations of some proteins: lin_12, lin_12 active and DSL.
- the binding of DSL proteins to channels to implement the receiving of bilateral signals.

```
PACKAGE PathwaysRegulations
ONTOLOGY CElegansGenes = "https://www.wormbase.org/species/c_elegans/gene/"
ONTOLOGY CElegansProteins = "https://www.wormbase.org/species/c_elegans/protein/"
ONTOLOGY CElegansProcesses = "https://www.wormbase.org/resources/wbprocess/"


MODULE pathway_lin12 ( ENTITY lin_12_act, CHANNEL :DSL_right_i(t), CHANNEL
:DSL_left_i(t))

     BIOLOGICAL REFERENCE
          CElegansProcess.WBbiopr:00000072

     ENTITIES
          ENTITY lin_12, DSL_input_right, DSL_input_left

     INIT
          lin_12 = simple_protein("lin_12 ", CElegansProteins.WP:CE00274,
black_token()*3 )
          DSL_input = simple_protein("DSL_input", CElegansProteins.WP:CE18343)

     PROCESSES
          degradation(lin_12_act, 100)
          degradation(lin_12, 100)
          degradation(DSL_input_right, 100)
          degradation(DSL_input_left, 100)
          transcription(lin12, CElegansGenes.WBGene00003001)
          activation(lin_12, DSL_input, lin_12_act)
          activation(lin_12, DSL_input, lin_12_act)
          activation_threshold(lin_12, lin_12_act, 6)
          signal_receiver(DSL_input, :DSL_right_i(t))
          signal_receiver(DSL_input, :DSL_left_i(t))

END
```

Figure 4.26

4.7.7 Gene-Protein and Protein-Protein implementations (performed by expert user and included in PathwaysRegulations Library)

In the next Figure 4.28 is described the implementation of the transicription module used b the pathways modules. It involve two entities: the protein to be transcribed and the gene that is the responsible of the transcription. The process itself is

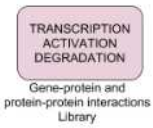

Figure 4.27: Library containing GENE-PROTEIN interactions.

implemented though a transition that is connected with a bidirectional arc to the gene and with an output arc to the protein. That means that as long as the gene is present the transcription can be performed and a protein is produces.

```
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
```

Figure 4.28

The Figure 4.29 shows the implementation of the degradation modules used before. This transition take as input a protein that after the delay disappear as result of degradation.

```
MODULE degradation ( ENTITY protein, INT n)

      PROCESSES
            process( IN:{protein}, delay(n) )
END
```

Figure 4.29

The activation process that involves two different proteins to activate one of them is shown in Figure 4.30. the transition is attached to pproteins through bidirectional arcs, an input arc and an output arc. This means that to be enabled is necessary the presence of a certain amount of the protein to activate and a certain amount of the activator, when transition fires an inactive protein is consumed and an active protein is produced.

```
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
```

Figure 4.30

The Figure 4.31 is the implementation of the mechanism with which the cell send the lateral signalling to adjacents cells. The proteins involved are mpk_1 active, lin_12 active and DSL. The channels are attached to the DSL protein to perform the exit points related to the adjacents cells, in order to send the lateral signals.

The transcription of DSL is mediated by the mpk_1 and in order to be performed a certain amount of mpk_1 is specified, and also the produced amount of DSL is specified. The degradation of the active lin_12 is regulated by the DSl protein, the amounts of both type of proteins that is consumed is specified in the arguments of the module.

```
MODULE output(ENTITY mpk_1_act, ENTITY lin_12_act, CHANNEL :DSL_right_o(),
CHANNEL :DSL_left_o())

     ENTITIES
          ENTITY dsl_output
     INIT
          dsl_output = simple_protein( "dsl_output", CElegansProteins.WP:CE18343)

     PROCESSES
          signal_sender(DSL_input_right, :DSL_right_o(t))
          signal_sender(DSL_input_left, :DSL_left_o(t))
          transcription_mediated(dsl_output, CElegansGenes.WBGene00001103,
           mpk_1_act, 5, 2)
          degradation_regulated( lin_12_act, dsl_output, 2, 2)


END
```

Figure 4.31

The 4.32 Figure shows the mechanism of transcription in which the expression of the gene involved is regulated by another protein. In addition to the transcription mechanism, already saw before, there is a transition that is enabled if a certain amount of the regulator exists and then fires consuming a protein and preventing the transition, responsible of transcription, to fire. In this case there is a competition between two transitions.

```
MODULE transcription_gene_expression_regulation(ENTITY protein,STRING  ID,
ENTITY  regulator)

     ENTITIES
          ENTITY gene
     INIT
          gene = simple gene( protein.name+"_wt",ID, black_token() )
      PROCESSES
          process( BI:{gene}, OUT:{protein} )
          process( BI:{gene, regulator[2]}, OUT:{protein}, IN:{regulator} )
END
```

Figure 4.32

The 4.33 Figure shows the same above process, but involving two regulators and not only one. The difference in this case is also that the amount of proteins is represented by an int value and not by the quantity of tokens. This difference is the consequence of the fact that the regulators involved are mpk_1 and lin_12 both actives, they have an int value because they are subject to numerical analysis from users.

The 4.34 Figure also shows a transcription process, it differs from others because the transcription itself is mediated by an other protein. The transition that represent

```
MODULE transcription_gene_expression_regulation(ENTITY protein,STRING ID,
ENTITY regulator1, ENTITY regulator2)

      ENTITIES
            PLACE gene
      INIT
            gene = simple gene( protein.name+"_wt",ID, black_token() )
       PROCESSES
            process( BI:{gene}, OUT:{protein} )
            process( BI:{gene, regulator1},
OUT:{protein},{if(regulator1.token>0){regulator1.token++}} )
            process( BI:{gene, regulator2},
OUT:{protein},{if(regulator1.token>0){regulator2.token++}} )
END
```

Figure 4.33

the transcription involved, in addition to the gene and the protein there is a protein mediator. The transition can be enabled only if the gene is present and if there is a certain amount of the mediator, when the transition fires a certain amount of the protein is produced.

```
MODULE transcription_mediated(ENTITY protein,STRING ID,ENTITY mediator,INT n,INT m

      ENTITIES
            PLACE gene
      INIT
            gene = simple gene( protein.name+"_wt",ID, black_token() )
       PROCESSES
            process( BI:{gene, mediator}, OUT:{protein[m]},
            {if(mediator.token>=n){mediator.token--}} )

END
```

Figure 4.34

In 4.35 Figure we find the degradation process regulated by another protein. The transiion take as input a certain amount of both proteins involved and when fires the proteins degrade.

```
MODULE degradation_regulated(ENTITY protein,ENTITY regulator,INT n,INT m)

      PROCESSES
            process(IN:{protein[n], regulator[m]})

END
```

Figure 4.35

The next two figures 4.36 and 4.37 describe the implementation of the binding of proteins to become signals receptors. Both the implementations have in common the transiton that bind the channel to the protein. The second one has in addition the transition that modulate the amount of signal received from the receptor. In other word the second implementation is able to change the throughput of the signal.

```
MODULE signal_receiver(ENTITY receptor,CHANNEL :chan(val))

      PROCESSES
           process(OUT:{receptor}, {when(:chan(val)){ receptor.token=val}})

END
```

Figure 4.36

```
      MODULE signal_receiver_throughput(ENTITY receptor,CHANNEL :chan(),IN n)

           ENTITIES
                PLACE receiver
           PROCESSES
                process(OUT:{receiver}, {when(:chan(val)){ receiver.token=val}})
                process(IN:{receiver}, OUT:{receptor[n]})
      END
```

Figure 4.37

The next 4.38 figure is the implementation complementary of the last two, it performs the binding of a channel to a receptor to make it able to receive the signal. The transition links the channel to the receptor.

```
MODULE signal_sender(ENTITY receptor,CHANNEL :chan(val))

      PROCESSES
           process(IN:{receptor}, {when(:chan(val)){ val=receptor.token}})

END
```

Figure 4.38

The Figure 4.39 shows the implementation of a gene, that is similar to those of simple_cell and simple_protein. It associate to a place the name and the ontology ID of the gene. The token is a single black token, it means that the gene is active.

```
MODULE simple_gene (STRING name, TOKEN token, STRING ID)

     BIOLOGICAL REFERENCE
          ID
     ENTITIES
          PLACE gene
     INIT
          gene.name = name
          gene.token( token )
END
```

Figure 4.39

All these implementations written in the BiSDL language need to be compiled and than simulated. We are working on a Petri Net simulator written in Python and based on a library named Snakes. For this reason the first compiler of the BiSDL language will generate Python code, to be compliant with our simulator. In this way we complete the circular pipeline (Figure 4.40) that from the model arrive to the simulation results that can be used to modify the model or to build other models enriched by the knowledge acquired from simulations.

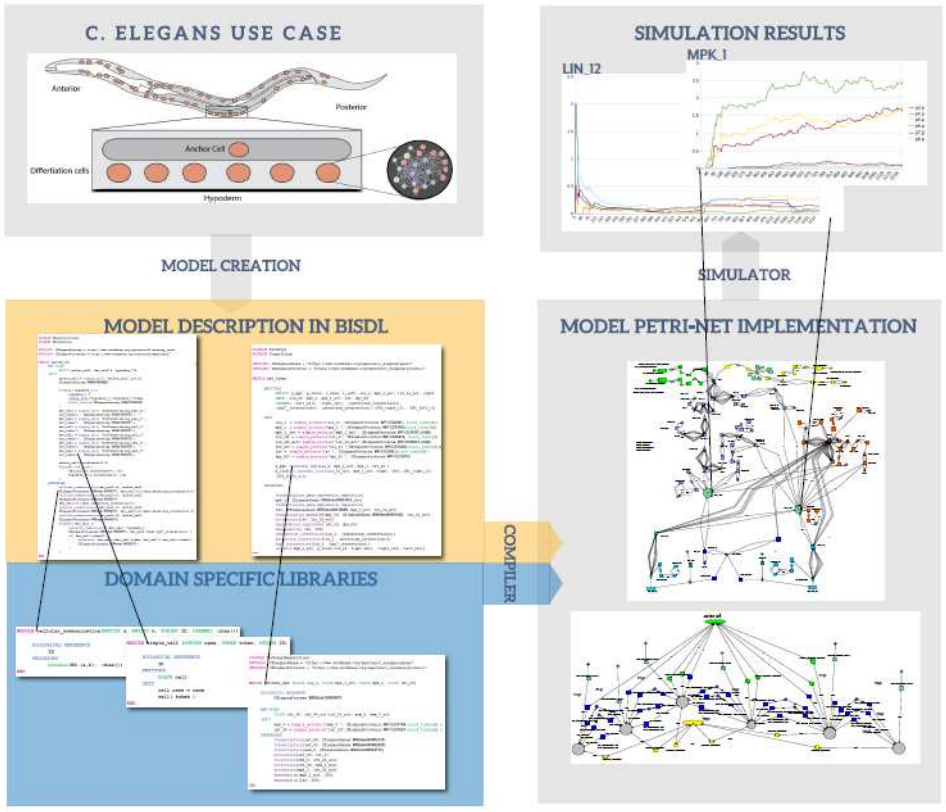

Figure 4.40: Circular pipeline of Biological system model simulation

Chapter 5

Conclusion

The BiSDL differs consistently from from state-of-the-art descriptive languages for biological models, both because it covers a biological subdomain, that of ontogenetic processes, which other solutions don't consider, and because it responds suitably to the need, posed by COMBINE community, of a new language able to model multicellular use cases. In addition to that, BiSDL design is intended to overcome most weaknesses of existent languages and to group their strengths, allowing to handle the criticalities from both the technical and the user interaction sides:

- **Human readability**
  Other languages are all based on the XML syntax, meaning a tool is always necessary to better understand the structure of the model. We choose another approach: provide a much more human-readable language to accommodate the user putting much effort on development of software to machine readability.

- **Spatiality and Motility**
  Thanks to the implementation of an underlying 3D spatial grid upon which different network motifs can describe geometry and mobility of entities involved in the model, providing an explicit representation of spatiality, the language naturally describes these aspects.

- **Hierarchical architecture**
  To reflect the natural architecture of Biological Systems, BiSDL has a hierarchical architecture. This also increase the modularity of the language and its understandability.

- **Modularity** Thanks to the use of modules within modules and to the organization of them in a structured library, the modularity is a predominant characteristic. This is useful for the reusability of models that need few changes in modules to be easily customized and modified.

- **Ontology references**
  We adopt a strict policy in the use of references to biological knowledge bases. In fact we think that is one of the most important requirement for exchangeability of models between different researchers and modelers. This is because if someone describe a model with his own notation, not referencing to an ontology recognized by all, other people will be not able to understand the model, this make the model not acceptable by the community.

- **Easier extensibility** Modularity along with Ontology references make the language easily extensible. Indeed to add a new biological concept is only necessary to build a new module and attaching to it the right ontological reference.

- **Extend the users target** Using modules as black-boxes, along with the human-readability of the language, allows biologists to perform the modeling activity. Indeed have always been computational biologists in charge of modeling biological systems.

BiSDL provides tools to improve the model description under many aspects, both for experimental and computational biologists working on ontogenetic processes, maintaining a systems biology perspective. To reduce the learning curve of the users, to give an easy way to exploit the use of libraries and to easily reference different ontologies, in the next future we want to develop a custom editor,as we did for the first implementation of the language. Furthermore we are developing parser of the language to be able to generate Python code for the very model specifications described, that will be used by a Petri Net simulator that is currently under development as well within our research group.

BiSDL is only the first stage of a long pipeline, as we said at the end of the **??** chapter. With the strengths of BiSDL new models can be described, more complex models than the existing ones. To help users in writing more complex and complete models we we'll provide a rich library and an Editor able to suggest and guide the more inexpert user. These models can be than compiled and simulated with our simulator. The simulator will provide a graphical user interface to help the user change parameters of simulation in an easy way. The parameters can be input perturbations, modulation of the temporal behavior and setting of stochastic variables. The simulation results can be used to infer new knowledge, to better understand the model fails, modifying it, and to guide biologists in new experiments. This whole pipeline can become in the future the preliminary virtual wet-lab that is fundamental to speed-up researchers works.

Bibliography

[1] Bardini, R. and Politano, G. and Benso, A. and Di Carlo, S., Multi-level and hybrid modelling approaches for systems biology, Computational and structural biotechnology journal, Elsevier, Vol. 15, 2017, pp. 396–402,

[2] The COMBINEstandards, https://co.mbine.org/standards

[3] BioPAX Level3 documentation, http://www.biopax.org/release/biopax-level3-documentation.pdf

[4] The SBOL Standard documentation, http://sbolstandard.org/wp-content/uploads/2018/01/BBF-RFC114-SBOL2.2.0.pdf

[5] Nicolas Le Novère, Michael Hucka, Huaiyu Mi, Stuart Moodie, Falk Schreiber, Anatoly Sorokin, Emek Demir et al. "The Systems Biology Graphical Notation", Nature Biotechnology, Vol. 27, No. 8, 01 September 2009, pp. 735-741,

[6] The SBML project specifications, http://sbml.org/Special/specifications/sbml-level-3/version-2/core/release-1/sbml-level-3-version-2-core.pdf

[7] Cuellar, Autumn ; Hedley, Warren ; Nelson, Melanie ; Lloyd, Catherine ; Halstead, Matt ; Bullivant, David ; Nickerson, David ; Hunter, Peter ; Nielsen, Poul

[8] Padraig Gleeson, Sharon Crook, Robert C. Cannon, Michael L. Hines, Guy O. Billings, Matteo Farinella, Thomas M. Morse, Andrew P. Davison, Subhasis Ray, Upinder S. Bhalla, Simon R. Barnes, Yoana D. Dimitrova, R. Angus Silver "NeuroML: A Language for Describing Data Driven Models of Neurons and Networks with a High Degree of Biological Detail ", PLOS, Computational Biology, June 17, 2010 ,

[9] Robert C. Cannon, Padraig Gleeson, Sharon Crook, Gautham Ganapathy, Boris Marin, Eugenio Piasini and R. Angus Silver"LEMS: a language for expressing complex biological models in concise and hierarchical form and its use in underpinning NeuroML 2", Front. Neuroinform., 25 September 2014,

[10] The SED-ML format, https://sed-ml.github.io/documents/sed-ml-L1V3.pdf

[11] MulticellularML forum, https://groups.google.com/forum/#!forum/combine-multicell

[12] Nora Connor , Albert Barberán, Aaron Clauset "Using null models to infer microbial co-occurrence networks", PLOS, May 11, 2017,

[13] Willners, Caroline and Holtsberg, Anders "Statistics for sentential co-occurrence", Working Papers, Lund University, Dept. of Linguistics, Vol. 48, 2001, pp. 135-147,

[14] D. Gajski and R. Kuhn "New VLSI Tools ", Computer, Vol. 16 1983 , pp. 11-14

[15] Zainalabedin Navabi, "VHDL: Analysis and Modeling of Digital Systems", McGraw-Hill, 1997, ISBN: 0070464790

[16] TADAO MURATA"Petri Nets: Properties, Analysis and Applications", PROCEEDINGS OF THE IEE, Vol. 77, No. 4, 4, April, 1989,

[17] Rüdiger Valk, "Object Petri Nets – Using the Nets-within-Nets Paradigm", 4th Advanced Course on Petri Nets, (ACPN), Eichstätt (Germany),September, 2003, pp. 819-848,

[18] Bardini, Roberta and Politano, Gianfranco and Benso, Alfredo and Di Carlo, Stefano,, Using multi-level Petri nets models to simulate microbiota resistance to antibiotics Bioinformatics and Biomedicine (BIBM), 2017 IEEE International Conference, 2017, pp. 128–133

[19] Bardini, Roberta and Benso, Alfredo and Di Carlo, Stefano and Politano, Gianfranco and Savino, Alessandro, Using nets-within-nets for modeling differentiating cells in the epigenetic landscape, International Conference on Bioinformatics and Biomedical Engineering, Springer, 2016, pp. 315–321,

[20] Paul W. Sternberg"Vulval development", WormBook , No. 5, 2005, pp. 1-28,

[21] Wikipathways, `https://www.wikipathways.org/index.php/Pathway:WP1453`