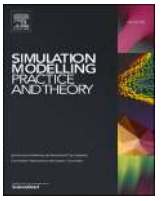
Simulation Modelling Practice and Theory 114 (2022) 102419

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat



Integration and Automation of Modeling of Biological Cell Processes

Cristina Ruiz-Martin [a,*], Gabriel A. Wainer [a], Laouen Belloli [b]

[a] Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive. Ottawa, ON. K1S 5B6. CANADA
[b] Instituto de Ciencias de la Computación (ICC). CONICET-Universidad de Buenos Aires, Buenos Aires, Argentina

ARTICLE INFO

Keywords:
Metabolic Networks
DEVS
SBML
Automatic model generation
Simulation

ABSTRACT

The System Biology Markup Language (SBML) has been used to build numerous models of biological processes. Here we introduce a new method to translate SBML specifications of cellular models into formal specifications for analysis and simulation. To do so we define a generic biological model architecture that can be instantiated with different parameters for different types of cells and at different levels of detail using the information available in SBML models. We discuss said architecture, a prototype implementation and different examples of use of the method with a synthetic model and model of E. Coli.

1. Introduction

Understanding biological processes on cells plays a fundamental role in biology, biochemistry, and medicine. In particular, the study of *Metabolic Networks* is important to understand the metabolism of the cells. Each possible reaction path within the Metabolic Network, called a *Metabolic Pathway,* defines the chained reactions that produce transformations of different metabolites. As we can have an exponential number of Metabolic Pathways, the study of the cell's metabolism becomes overly complex [1]; however, only a small number of pathways take place inside each specific cell.

Observation studies of metabolic pathways are usually expensive and time-consuming. Instead, Modeling and Simulation (M&S) is a good method to study the problem. Building a model allows us to easily conduct a larger number of studies without significantly increasing the cost or duration of the research. Consequently, various M&S approaches have been used to study the biological phenomena. For example, in [2], a discrete-event multi-level model was used to study metabolite channeling (commonly modelled with differential equations at macro-view); and JAMES II was used for integrating various formalisms for different components of the biological system [3]. A comprehensive review of such computational tools can be found in [4]. Regardless of the approach used, biological systems models can be complex to simulate, and it is useful to divide them into smaller pieces that can be tested independently and then combined into larger models. For example, in [5], the authors present a model of *mycoplasma genitalium* composed of 28 independent models of the different parts of the bacterium integrated into a multi-level cell model.

Sharing models of biological process can be improved using standards like the System Biology Markup Language (SBML), which was defined to represent biological systems using a common representation [6]. SBML standardizes data storage, which helps building tools, modeling, visualization, and validation [7]. Although SBML is widely used and there is a vast number of biological data available in SBML, transforming SBML models into simulation models for analysis is still time-consuming. An automated process for M&S of

* Corresponding author.
E-mail addresses: cristinaruizmartin@sce.carleton.ca (C. Ruiz-Martin), gwainer@sce.carleton.ca (G.A. Wainer).

https://doi.org/10.1016/j.simpat.2021.102419
Received 5 July 2021; Received in revised form 30 September 2021; Accepted 4 October 2021
Available online 8 October 2021

models defined using SBML can save time and improve model verification and validation, as well as helping obtaining results that can be replicated [8]. For example, in [9], the authors present a platform for modeling reaction kinetics of biological with SBML integrated with DEVS (Discrete Event Systems Specification), a formal modeling and simulation methodology to build hierarchical and modular models [10]. In [11], SBMLtoODEpy was used to convert SBML models into Python classes to facilitate the integration of biological reactions models into larger biomedical systems modeling projects written in Python. However, these results are usually hard to reuse and to integrate with other models. For example, [9] is specific to reaction kinetics, and [11] converts the SBML model into a Python class, however, it does not provide a full model that can be simulated.

Our research contributions focus on the problems above by providing a method to automate the generation of formal models and their simulations from SBML standard models. We propose a method to build formal models that can be (1) automatically instantiated with any SBML source, and (2) integrated with other models. We define a general modeling architecture for cells that can be instantiated with different parameters for different types of cells, and at different levels of detail. Additionally, as we use DEVS, a hierarchical and modular approach, we also improve model integration. We use SBML models as sources and define a method to manage the SBML files and use their information to instantiate the proposed architecture and generate DEVS models automatically. This process allows us to reuse existing models previously developed in SBML. Additionally, we include the possibility of combining SBML models to develop larger ones. In [12], we introduced a preliminary version of this research in which we explored the ideas of using SMBL to generate simulation models. Here, we extend the research focusing on model architecture and the automatic generation of models. The main contributions of this paper can be summarized as follows:

- We define a flexible structure for modeling biological cells, offering a general framework that can be adapted, allowing to integrate new models when needed. The structure considers general aspect of cells, allowing changes to external components.
- We introduce a formal structure as a container to integrate different models representing the cell's mechanisms. This attempts to modularize the parts of a cell so each new model can be modeled using a different approach without affecting the rest.
- We provide a model structure that directly maps a biological cell into compartments coupled together, resulting in a model that emerges from the interaction between its sub-models (as it happens in real cells). Additionally, we do not assume anything about how each sub-model is defined: we only need their inputs and outputs (according to DEVS semantic). The model structure becomes flexible and new components can be added provided that their interface is consistent with the existing components' interface (or adapting the interface).
- We automate model generation for studying biological processes using SBML and DEVS simulation. SBML describes the model and its parameters which are used to generate a model structure and its Metabolic Pathways ready for running simulations.

The rest of the paper is organized as follows. In section 2, we summarize the related work on the study of biological processes and simulation tools. In section 3, we focus on the model development process, and we explain the DEVS model for a Cell. In section 4, we present different case studies.

2. Background

This research focuses on the study of Metabolic Pathways through M&S. Based on their abstraction level, the models can be classified as micro or macro view models. Macro-view models describe the behavior of the system as one piece, abstracting all the sub-systems and their interactions. On the other hand, micro-view models divide the system into sub-systems that can be modeled independently. The same phenomena can be modeled at both abstraction levels. For example, in [13] a micro-view approach is used to model glycolysis, while in [14] it is modeled using the concentration of metabolites (i.e., substrates and products).

We can also consider *mathematical* or *computational* models. Mathematical models are purely analytical such as in [14], whereas c*omputational* models use computer programs in which the logic describes the phenomena to be modeled, such as in [13].

Models can also be classified on their *time representation*. Some models use a continuous representation of time, while others use discrete time. For instance, the model in [14] uses a continuous time representation and the model in [13] is a discrete-event model.

Different approaches are not independent of each other. We can combine micro and micro view approaches within the same model. In [15][2], the authors explain the advantages of combining macro and micro levels as using various levels allows the models to describe each part at their right level of abstraction. In [2], a discrete event multi-level model is used to study metabolite channeling, a process commonly modeled with differential equations (macro-view). Although this is a continuous process, the number of metabolites in a compartment is discrete.

Biological processes can easily lead to the definition of large models and the need to run complex simulations. To deal with this complexity we can build submodels of each of biological system, and then integrate them into a model of the entire system. In [5] a model is divided into different components modeled independently; in [3], the JAMES II framework is used to mix different formalisms for each component. CytoSolve [16] allows integrating multiple pathway models implemented independently running all simulations in parallel and coordinating their inputs and outputs to generate a final single result from the model interactions.

Another key factor to consider is reusability, a challenge in M&S. In [17], the authors discuss the importance of having modular models about and the key role of standardization to achieve model reusability. Along the same lines, [18] emphasizes the importance of hierarchical modeling and composition in systems biology as well as key challenges in the development of computational models (modularity, standardization, reusability, and complexity).

It is important to count with tools to automate and simplify the M&S process. The System Biology Markup Language (SBML) was defined with this purpose [6]. SBML allows storing and exchanging models, and it is useful for modeling visualization and validation

[7]. The main advantage of SBML is the standardization of biology data storage, which facilitate managing information and obtaining relevant results that can be replicated [8]. For instance, in [9], SBML was integrated with DEVS [10] to develop a platform for modeling reaction kinetics of biological systems. The authors used the LibSBML library [19] to translate the chemical reactions into differential equations and to generate macro-view models that can be easily shared. In [11] the authors developed a SBMLtoODEpy to convert SBML models into Python classes to facilitate the integration of biological reactions models into larger biomedical systems modeling projects written in Python. Similarity, in [20], the authors developed SBML2Modelica to integrate biological reactions into larger Modelica models. In [21] the authors presented a database system designed specifically to store biochemical pathways in SBML. In [22], the authors proposed the *SBML-diff* tool, which allows to compare SBML Models at different levels of detail visually. SBML has also been applied to model the spread of diseases. For example, in [23], the authors present the SBML Array package to represent microsimulation disease models. A complete definition of all the elements in an SBML model can be found in [24], and a summary description is included in Appendix I.

As discussed earlier, we propose a new method to model Metabolic Pathways built as a modular container integrating all the components in the cell subsystems using the DEVS formalism. This allows us reusing the model structure and easily adapt it to each cell. We propose to instantiate the general model for Metabolic Pathways using SBML, automating the modeling process and reusing models. DEVS manages complexity by using a modular and hierarchical approach [10]. The system under study is modeled using (1) basic behavioral models called "atomic", and (2) structural composite models called coupled models (a formal specification is provided in Appendix II). This is useful to model biological systems because it allows using different modeling levels of abstraction for each submodel. Additionally, this improves model reuse, reducing development time and verification reusing existing models. Model definition, simulation, and experimentation are separated. There are numerous DEVS simulators available [25] such as Adevs [26], PyDEVS [27], VLE (Virtual Laboratory Environment) [28], CD++ [29] (which also allows defining Cell-DEVS [30] models), MS4Me [31], and DEVS-Suite [32]. We use Cadmium [33], a DEVS simulator implemented in C++17 using the Boost library standard. At the user level, Cadmium uses two C++ classes for defining atomic and coupled models. Atomic models are defined in a class that provides a structure to define the state of the model and the methods to define behavior. Coupled models are defined instantiating a different class which includes input/output ports, submodels, external input and output couplings, and internal links.

Various researchers used DEVS for modelling biological systems. In [15], Multi-Level-DEVS (ml-DEVS) was proposed to support multi-level models explicitly: the information at macro-levels can be accessed from micro-level components and vice versa. In [2], a discrete event multi-level model is used to study metabolite channeling. Differential equations are usually used to model continuous systems. However, metabolite channeling can be better modeled by using discrete components. Using a multi-level model, the authors address the problem of having a system with continuous and discrete components. In [34], the authors model the evolution of living organisms using DEVS; they propose a micro-view approach where the basic components are individual organisms. Their model of an organism includes the internal organization (structure and functions), and a set of rules for conditional mutation, survival, and self-reproduction. Similarly, in [35], the role of consanguineous marriages in the evolution of congenital disorders in a population was studied. In [36], the authors model glycolysis, modeling each step in the reaction pathway as a DEVS atomic model.

Another research also uses SBML combined with DEVS. The SBML-DEVS platform [37] models reaction kinetics of biological systems and use SBML to save and read the model's data. The authors used the LibSBML library to translate the chemical reactions into differential equations and generate macro-view models that can easily be share.

The above-mentioned research focusses on different subsystems of cells, or in a particular cell. Here we propose a method to build computational models that can be applied to different components within a cell, focusing on how to integrate different models of the cell subsystems. To allow reuse, we propose a general model structure abstracted from cells aspects that can be easily adapted.

The proposed method, and resulting architecture and tool, can be easily integrated on cloud environments, providing M&S as a Service (MSaaS) for biological systems. In [38], we defined a method for distributed simulation of DEVS models using SOAP-based web services. This research was later extended in [39], to improve interoperability via decoupling systems implementations building the first RESTful Interoperability Simulation Environment, an all-purpose WS-based distributed simulation middleware that decouples design and implementation. In [40], we show how to simplify Web APIs for M&S applications as they are rarely used for MSaaS defining an architecture for web services Mashups. Similarly, since 2013, NATO is working on a framework to provide MSaaS [41], and MSaaS is available in recent applications in industry [42], and in recent years advanced cloud services, including microservices [43], edge and cloud computing [44]. The models defined in this research can be executed remotely using these different platforms, enhancing collaboration between stakeholders, and improving model sharing and resource use.

3. A method to model biological cells

Our method follows a 4-step process:

(1) Analysis of the structure and functions of cells to find common components, behaviors, and structures. We model aspects that are common for all the cells, and we parameterize the model to include variable components and behaviors.
(2) The generalized structure and functions of the cell are defined using DEVS. The functions of the cells are performed by the different components, and the behaviors are defined using atomic models. We parameterize models and use a generic model for different types of cells.
(3) We formally define how all components are interconnected.
(4) We define a Parsing and Model Generation of Metabolic Pathways application (PMGMP), which can read an SBML file and generate an implementation of the proposed architecture in DEVS.

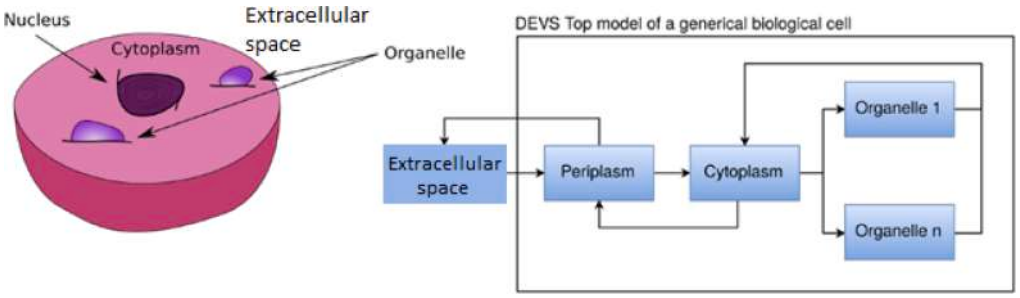


Fig. 1. Defining a general cell and into a general DEVS model.

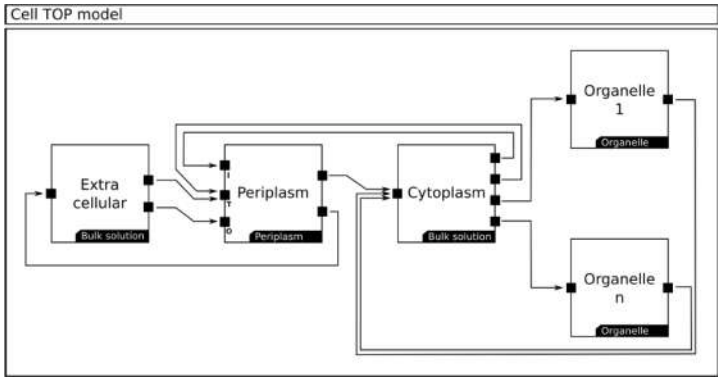


Fig. 2. DEVS-based structure of a cell

In the following sub-sections, we discuss each of these steps in detail.

3.1. Structure analysis

Each cell is a composite of organelles, cytoplasm, periplasm, and extracellular space. What varies from one cell to another is the quantity and behavior of organelles (and the different reactions occurring in the different organelles), the metabolites each cell uses, and the metabolites interchanged and used for reactions. Thus, our model architecture is adapted to include the aspects that varies among cells. We model the common aspects, and we parameterize the behavior and the relationship structures that change. For example, all cells have an extracellular space, and we represent it as a DEVS model. However, the reactions that occur on the extracellular space and the metabolites available change from one cell to another. Therefore, we parameterize the extracellular space model so it can handle different metabolites and reactions. Similarly, all cells have organelles, but the number of organelles varies from one cell to another. The number and type of organelles in the model will be a parameter of the model architecture as we detail in step 3. We conduct a similar analysis for all the elements in a cell.

3.2. Devs modeling

After identifying the common and variable aspects of a cell, we map the cell components and its functions (carried by specific reactions) into DEVS models. We consider that a cell has two main types of elements: (1) those with an associated behavior such as the organelles, the periplasm, the cytoplasm, and the extracellular space, and (2) those static, like the metabolites. The elements with associated behavior consume metabolites to perform their functions (i.e., reactions). We model each of these elements as a parameterized DEVS model to capture the various behaviors. We then define the connections between these elements as a coupled model.

Fig. 1 shows the general structure of the main coupled model. The biological cell hierarchy is defined by the links between the submodels of a cell: extracellular space, periplasm, cytoplasm, and organelles. The extracellular space only communicates with the periplasm; the periplasm communicates with both the extracellular and the cytoplasm. Finally, the cytoplasm communicates with the periplasm and with the organelles. We model metabolites as independent entities that move from one component to another (and in DEVS, they are modeled as the model's inputs/outputs).

3.3. Model structure

The top model architecture, shown in Fig. 2, represents an abstraction of a biological cell interacting with the extracellular space. It

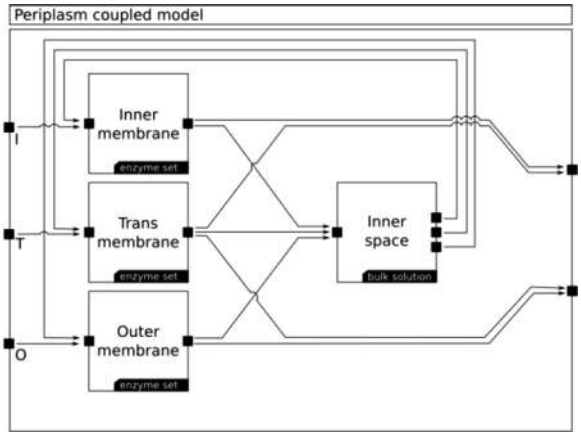


Fig. 3. DEVS coupled structure of the periplasm.

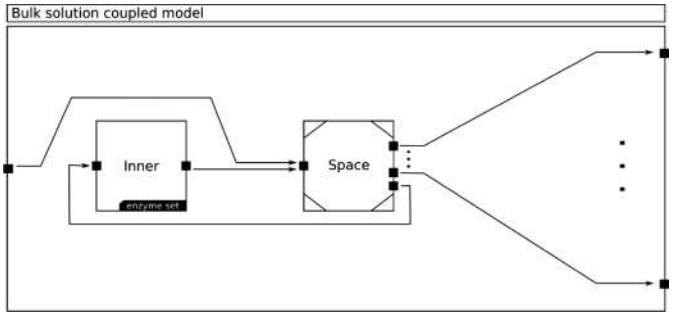


Fig. 4. DEVS coupled structure of the bulk solution.

includes models for the extracellular space, the periplasm (which includes three sub-models representing the outer, inner and trans membranes), the cytoplasm and multiple organelles.

Each sub-model represents a physically separated compartment, and the links between two submodels represent that those compartments exchange metabolites. Since a compartment may have one or multiple membranes, each sub-model has one or multiple input ports to receive metabolites through its different membranes. For example, the Periplasm model includes three membranes and therefore it uses three input ports (one for each membrane). In that way, other models can send metabolites to the various membranes of the periplasm by sending the metabolites to the corresponding port. The Periplasm outer membrane is connected to the port named "O", the inner membrane is connected to the port named "I" and the trans membrane is connected to the port named "T". Therefore, as in biological cells, the extracellular space only interacts with the Periplasm outer and trans membranes. Similarly, each model uses one output port for each external membrane (membranes of other models) to which it sends metabolites.

Although the Nucleus is an important part of biological cells, not all cells have one. By explicitly modelling the Nucleus, we would restrict which cells can fit in the proposed architecture. Therefore, the cells that have a Nucleus include a new organelle representing it.

Following, we discuss the description of each of these components in detail.

3.3.1. Periplasm

The periplasm transports metabolites from the extracellular space to the cytoplasm through different membranes. It can also catalyze reactions in its inner membrane to transform metabolites as part of the cell metabolism. We model the periplasm as a coupled model with four components:

- Three *membrane* models representing the outer, inner and trans membranes. These models are different instances of the parameterized *enzyme sets* coupled model defined below.
- An *inner space* model where we handle the reactions occurring within the periplasm. The *Inner space* is an instance of the *bulk solution* coupled model defined below.

As seen in Fig. 3, *inner, trans and outer membrane* models connect the different compartments of the cell to exchange metabolites through transport reactions. Reactions in the *outer membrane* model consume and produce metabolites from/to the *extracellular space* and *periplasm* models. The flow of a metabolite within the *periplasm* model is as follows:

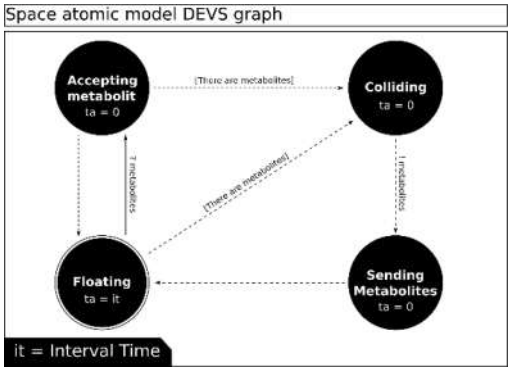


Fig. 5. The *Space* DEVSGraph.

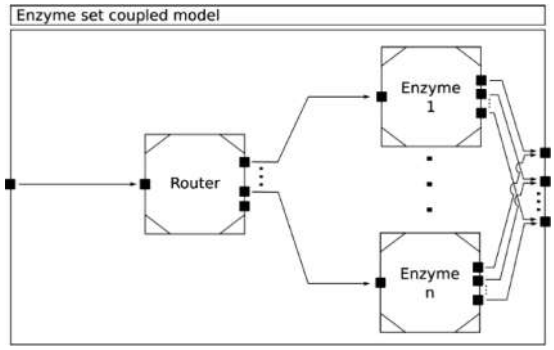


Fig. 6. An *enzyme set* coupled model with the *router* and the *enzyme* atomic models.

- Reactions within the periplasm that are not transport reactions are modelled in the *inner space* model.
- When *inner space* finds that metabolites collided with enzymes in the *membrane* models, it sends them back to the *membranes* (as is or modified)
- The *membranes* receive metabolites from other compartments and generate the metabolite exchange process between other compartments and the periplasm *inner space*.

Inner space is an instance of *bulk solution*, a parameterized coupled model shown in Fig. 4. In the *periplasm*, the model interacts with its own compartment membranes. *Bulk solution* can represent either a compartment without membranes (e.g., extracellular space, cytoplasm) or a sub-model of a compartment with membranes with its own inner space (e.g., periplasm, organelles). In both cases, *bulk solution* only exchanges metabolites with membrane models.

Within a *bulk solution* model, the metabolites flow as follows:

- The *space* atomic model models the spatial distribution and movement of metabolites and enzymes inside a compartment. It also has information of the enzymes of related compartments, which is a parameter of the model.
- Each time metabolites collide with enzymes, *space* sends the metabolites to the corresponding instance of *enzyme set*. This represents metabolite binding and reacting.
- *Enzyme set* can represent a membrane (i.e., enzymes catalyzing transport reactions) or the *inner* model of the bulk solution (i.e., the enzymes catalyzing non-transport reactions).

The *space* atomic model computes collisions between metabolites and enzymes in the *bulk solution* as described in Fig. 5.

The model starts in the *floating* state. Using fixed-length intervals, it calculates which metabolites have collided. At the end of each interval, the model sends each colliding metabolite to the corresponding *enzyme set* model. For this purpose, at the end of each interval, the model triggers a sequence of two instantaneous internal transitions. In the first transition, the model goes to the *colliding* state, where it calculates all the collisions that occurred since the last internal transition using the algorithm detailed in Appendix III. Then, following an internal transition, it switches to *sending metabolites* and it sends the metabolites that have collided to the corresponding *enzyme set* models though the output function. Finally, the model returns to the *floating* state where it will remain for the duration of the interval time. Each time metabolites (with bound enzymes) are sent, and the *space* model returns to the *floating* state, it decrements the number of enzymes of that type until the enzyme returns.

Each time an external event occurs, the model switches to *accepting metabolites* state and it increases the number of metabolites in

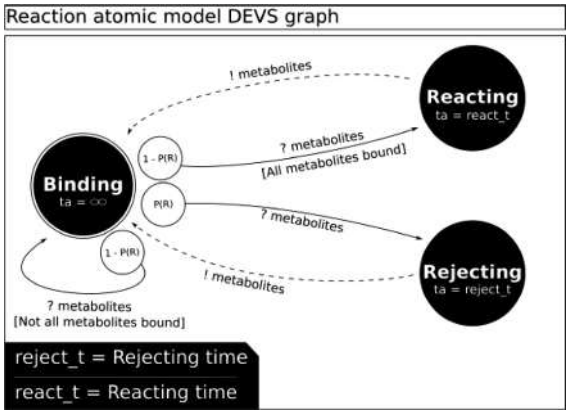


Fig. 7. The DEVSGraph of one enzyme within the *enzyme* atomic model.

the compartment. Then, the *space* state is updated to consider the incoming metabolites, and they will be available in the next scheduled internal function when collisions are calculated. The time advance of the *floating* state is set to the interval time only when transitioning from the *sending metabolites* state. When transitioning from *accepting metabolites,* the time advance of *floating* state is set to the interval time minus the elapsed time since the last *sending metabolites* state. This is because we want fixed time steps to calculate collisions.

The *enzyme set* is a simple coupled model (see Fig. 6) where a set of *enzyme* atomic models are grouped under the same compartment. The grouping of the enzymes allows the *space* models to direct the metabolite messages using a routing system like a network protocol. Instead of sending each message to each atomic *enzyme* model, the routing protocol is used to only send the message to the corresponding *enzyme set*. This reduces number of outgoing links in the *Space* model because all the atomic *enzyme* models are linked to the same output ports. *Enzyme* models use a similar routing system to route their messages to the *space* models outside the *enzyme set*.

As we already mentioned, an enzyme can handle multiple reactions. We need to consider the following restrictions:

- Each reaction consumes and produces different metabolites and has different rates and bounding constants.
- Reactions are not physical elements of a biological cell; they are the behavior of an enzyme when it is catalyzing a reaction.
- We used model components to only represent physical elements of a biological cell.

Each time an *enzyme* atomic model receives metabolites, it determines which reaction should compute. Because two reactions cannot occur at the same time in one enzyme, the *enzyme* model can only execute one of its reactions at a time. The general attributes of the *enzyme* atomic model are:

- Enzyme reaction: (defined for each reaction that the enzyme can handle)
  - Reaction rate: a constant that indicates how long the reaction takes.
  - Substrate stoichiometry: a set of metabolites associated with amounts that indicate the left side of the reaction stoichiometry formula.
  - Product stoichiometry: a set of metabolites associated with the right side of the reaction stoichiometry formula.
  - K_{offSTP}: the K_{off} constant for rejecting Substrate To Product (STP) metabolites.
  - K_{offPTS}: the K_{off} constant for rejecting Product To Substrate (PTS) metabolites.
  - Reject rate: a constant that indicates the rate at which metabolites are rejected for that reaction.
- Enzyme: (not related with the enzyme reactions above)
  - Routing table: a table that indicates where we send must each rejected or produced metabolite.

The *product* and *substrate stoichiometry* attributes are instances of the formula to compute the reactions for the substrate and the product. The *enzyme* atomic model uses the reactions stoichiometries to determine which reaction must be triggered based on which stoichiometry consumes the received metabolites. Once the *enzyme* atomic model determines the reaction that must be handled, it will use the attributes and state of that reaction until the reaction finishes.

If Fig. 7 we show a DEVSGraph of one *enzyme* within the *enzyme set* coupled model.

The general enzyme model state flow is as follows:

- The *enzyme* model starts in the state *binding* with the *bound substrate* and *bound product* variables of all the reactions set as false (for all compartments that send metabolites to the enzyme).
- When a metabolite is received, the state is updated through the external transition function to determine the next state: *rejecting* if incoming metabolites are rejected (Probability P(R)); *reacting* if all the related compartments already sent their metabolites and

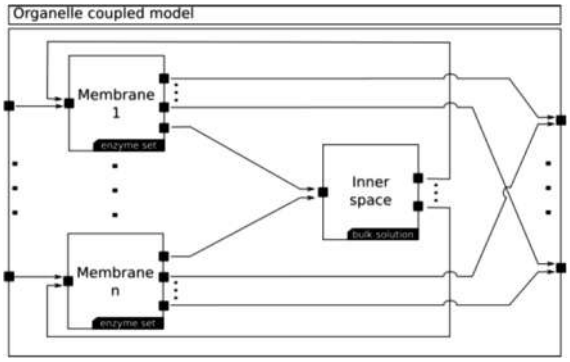


Fig. 8. The general *organelle* coupled model with multiple *membranes* and the *inner space*.

incoming metabolites are not rejected (Probability (R)-1); *binding* if not all the related compartments have already sent their metabolites and incoming metabolites are not rejected (Probability (R)-1). The circle in the continuous lines (external transitions) shows the probability that the model will use that path if all conditions are satisfied. Note that the transition from *binding* to *binding* and from *binding* to *reacting* are disjoined conditions (i.e., they cannot be satisfied at the same time).

- The enzyme atomic model can only accept incoming metabolites in the *binding* state. If the state is *reacting* or *rejecting* then, it will remain in that state for the time specified by the *reacting* or *rejecting rate* attribute.
- Once the rate time is over, the reaction sends the product or rejected metabolites back to the corresponding compartments and comes back to the *binding* state. The enzyme rejects the incoming metabolites with a probability based on the K_{off} disassociation constant following this rule: $P(X > K_{off})$ where $X \sim U(0; 1)$. Once the reaction starts, the *reaction rate* constant indicates how long the reaction will take until the produced metabolites are ready.
- The enzyme rejects the incoming metabolites if there is a reactions conflict. This occurs when the enzyme is in *binding* state, but some compartments already sent their metabolites. Then, the incoming metabolites must belong to the same reaction of the already bound metabolites. If the incoming metabolites and the already bound metabolites are from different reaction stoichiometries, then the metabolites are rejected.
- In some cases, an enzyme can handle reactions in both directions (substrate to product and product to substrate). When this is the case, each direction is considered a different reaction and therefore they cannot be handled at the same time. Thus, once metabolites from a compartment are bound as substrate, no product can be accepted until the reaction is over and vice versa.

3.3.2. Extracellular space and cytoplasm

The *extracellular space* and the *cytoplasm* coupled models are different instances of the parameterized *bulk solution* coupled model already explained.

3.3.3. Organelle

Each organelle is modeled as a coupled model with multiple membranes that communicates with the organelle inner space and other compartments (i.e., other organelles and the cytoplasm). Fig. 8 shows the *organelle* coupled model.

As shown in Fig. 8, the *organelle* coupled model has an *inner space,* an arbitrary number of *membrane* models (i.e., n), and an arbitrary number of output ports (i.e., m). The *inner space* is an instance of the *bulk solution* parameterized coupled model and each *membrane* is an instance of the *enzyme set* parameterized coupled model. Each *membrane* interacts with different compartments; therefore, the output ports of a *membrane* model are only connected with those output ports that connect the organelle with those specific compartments. An *organelle* coupled model has one input port for each membrane.

In a biological cell, organelles are placed in the cytoplasm; consequently, *organelle* models send and receive metabolites from/to the *cytoplasm* model. In the proposed structure, an organelle can be considered as a simple cell model with its own metabolic network. Therefore, we encapsulate the flow of metabolites; the cytoplasm and the organelles do not know what happens in any specific organelle or in the cytoplasm.

It is worth to remark that in the architecture proposed in this research, every component can be modified, replaced, added, or even removed. For example, if a cell has an organelle that does not fit this proposed structure, or if the modeler wants to use a macro-view modeling approach for one or more organelles, they are able to add their model to the structure and connect that model as a new *organelle* model.

3.4. Model implementation

The architecture was implemented in the *Parser and Model Generation of Metabolic Pathways* (PMGMP) tool, which manages SBML files and use their information to automatically instantiate and generate DEVS models implemented in Cadmium. Likewise, each time a new model is generated, PMGMP generates a JSON model.

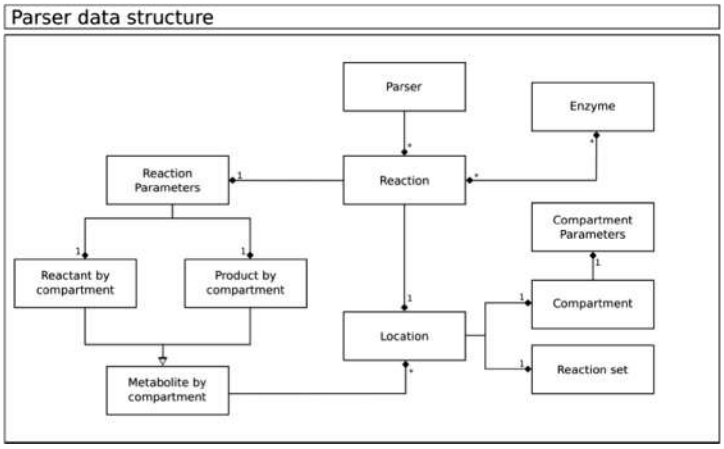


Fig. 9. UML diagram of the parser data structure obtained from the SBML file.

3.4.1. Parsing SBML files

First, we use the *ListOfCompartmentTypes* attribute in the SBML files along with their type to obtain the coupled model structure information (Extracellular, Periplasm, Cytoplasm and Organelle). To instantiate each component, we need to know which reactions occur in each compartment; this is defined in the *ListOfReactions* element in the SBML files. From this element, we get the reaction id, if it reversible or not, the stichometry and the main properties such as K_{onSTP}, K_{onTPS}, K_{offSTP}, K_{offSTP}, reaction rate, and rejection rate. We also need to know which enzymes catalyze each reaction, and in which compartment each of the enzymes is. SBML specifications do not consider enzymes: they are specified in the reactions using logical expressions to denote the enzymes' name. To know to which compartments each enzyme belongs, we could use the compartment where the reactions happen. However, in SBML all the reactions are listed in the *ListOfReaction* attribute without being separated by compartments. We need to deduct that information from the reaction stoichiometry in the SBML structure. In SBML, species are compartment-specific elements. Therefore, if a metabolite can be found in multiple compartments, the metabolite must be declared multiple time using different species IDs for each compartment. This is normally done by adding the compartment name as suffix. Additionally, the stoichiometry of the reactions determines the species a reaction consumes and produces. With this information we can deduct where the reaction occurs.

The information retrieved from the SBML file is stored in a data structure called *parser data*, shown in Fig. 9. The white arrow indicates that the product and reactant have the same structure (i.e., *metabolite by compartment*). As a reaction is handled by multiple enzymes and an enzyme can handle multiple reactions, this is represented in the diagram with an asterisk. We use the values stored in the *reaction parameter* structure to create instances of the enzyme atomic models. If when we attempt to create a model of an enzyme, the model already exists, we add the reaction to the list of reactions of that the existing enzyme atomic model can handle.

3.4.2. Model generation

Fig. 10 shows a diagram of the top-down recursive strategy to generate the models (i.e., the compartments of the cell) as a rooted tree graph where the root is the top model generation process, and each level defines the process recursive hierarchy. Each line link in the figure represents a recursive call to a function that generates a sub-model.

The data needed to define each compartment (i.e., DEVS model) is retrieved from the data structure presented in Fig. 9. A new data structure call *ModelStructure* is created for each atomic component with the following attributes: the compartment model *id*; the *enzyme sets* within the same membrane or the compartment inner enzyme set; a *routing table* that determines how metabolites are sent to the enzymes; the *membrane external input coupling* between the membranes and the model input ports (if there is no membrane the input ports are all directed to the *compartment space* model); and the *space parameters*. The *space parameters* include the compartment id, the interval time (model time step; it is not specified in the SBML file and must be provided by the modeler), metabolites (the compartment species list) and enzymes (all the enzymes where the compartment send and/or receive metabolites from).

We use the *ModelStructure* class to generate the DEVS model as shown in Fig. 10. It is a top-down process that starts creating the top cell coupled model and continues constructing the sub-components recursively until the bottom elements are built. Once all sub-components are ready, the cell top model is finished.

The cell top model is composed by the *extracellular, periplasm, cytoplasm* and *organelles* models as well as their connections.

3.4.3. Model parameter extraction

The final step is to write the parameters of the model in an external file (in this case, an XML file) to define initial conditions of the simulation to obtain different scenarios without the need for regenerating and compiling the model. The XML schema for the parameters of the model is shown in Appendix IV. All the model parameters are defined using three main lists: *spaces, reactions,* and *routers*.

Space represents all the parameters for atomic models of type space, and it has a list with all the *compartments.* Each *compartment* has a model id, an interval time, a list of metabolites, a rooting table that which output port must be used to send metabolites to an

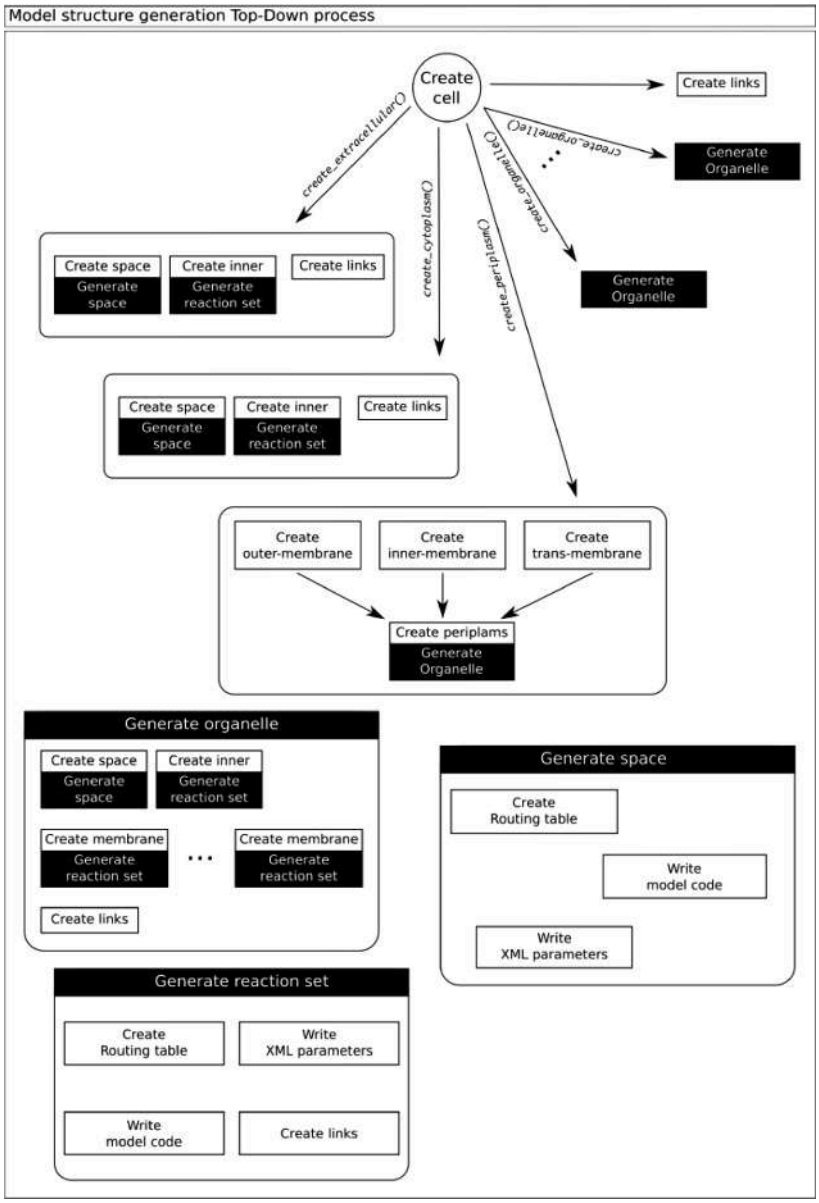


Fig. 10. The Top-down model generation process structure.

enzyme, and a list of enzymes. Each *enzyme* represents the all the parameters for atomic models of type *enzyme*. An *enzyme* is basically a list of handled *reactions*.

All the reactions are stored in a list of ***reactions***. Each reaction element has an id, a reaction rate, the Ko constant for the STP direction, the Ko constant for the PTS direction, a routing table that indicates which port must be used to send each metabolite to its corresponding space atomic model, and the stoichiometry.

Router represents all the parameters for the atomic models of type *router*. A *router* element has an *id* attribute to identify the model and a *routing table* element. Each child of the *routing table* element is an *entry* element that specifies, for each enzyme id (*eid* attribute) the port where the message must be routed to reach the enzyme (*port* attribute).

If we want to run new simulations with different parameters, we call the compiled Cadmium model with the path to the XML file. The constructor of the atomic model classes, which take as input parameter the path to the XML file, uses the field of called *cid* (i.e., the compartment if) to find the parameters within the XML file.

The complete model generator process implemented in *Parsing and Model Generation of Metabolic Pathways* (PMGMP) is summarized in Fig. 11. First, the *SBMLParser, ModelCodeWriter* and *XMLParameterGenerator* components are initialized. Then, we initialize the *ModelStructures,* where the atomic components are defined, using the top-down strategy in Fig. 10. Each time an atomic model is defined, its parameters are added to the XML structure. Once the entire model is built, *ModelCodeWriter* and *XMLParameterGenerator*

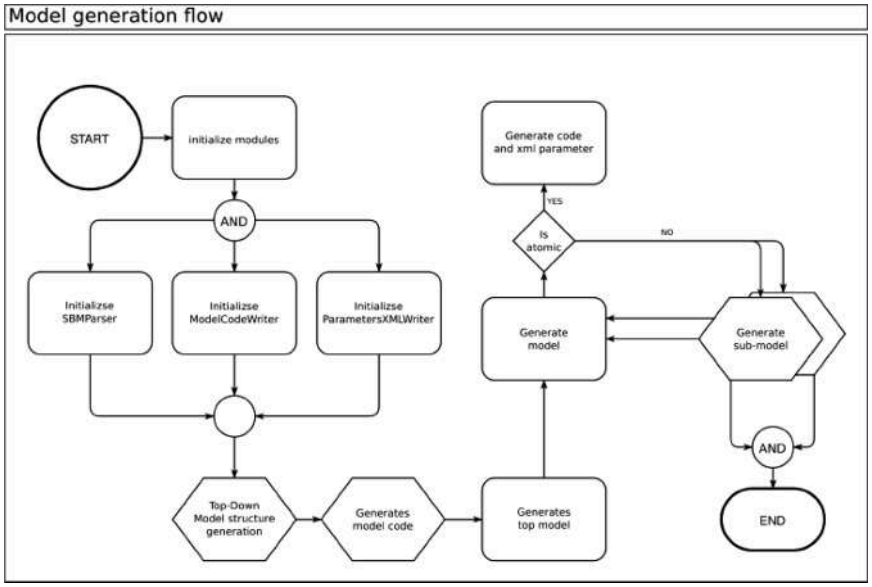


Fig. 11. The recursive model generation flow

close their files and the model is ready to be compiled.

4. Modeling and simulating metabolic pathways

In this section we show how to apply our method and tools to build models and run simulations of metabolic pathway. The implementation and the data to replicate these case studies are publicly available in https://github.com/SimulationEverywhere/PMGBP-PDEVS.

4.1. Synthetic model

We defined a synthetic model in SBML that defines a theoretical cell with one enzyme (called **b0000**) that catalyzes all the cell reactions. In each compartment (periplasm and cytoplasm) there are multiple b0000 enzymes. The model only considers reactions that are catalyzed by the b0000 enzymes. To do so, we defined the following reactions under the <reactions> tag:

- **A to 2A in cytoplasm**: this reaction occurs in the cytoplasm. It consumes one metabolite (A_c) and produces two metabolites (A_c).
- **B to 2B in extracellular space**: this reaction occurs in the extracellular space. It consumes one metabolite (B_e) and produces two metabolites (B_e).
- **C and C to 2C and 3C in periplasm inner**: it occurs in the periplasm inner membrane, and it consumes one metabolite (C_c) from the cytoplasm and one from the periplasm (C_p). It produces two metabolites (C_c) in the cytoplasm and three in the periplasm (C_p).
- **D and D to 2D 3D in periplasm outer:** it occurs in the periplasm outer membrane, and it consumes one metabolite (D_e) from the extracellular space and one (D_p) from the periplasm. It produces two metabolites (D_e) in the extracellular space and three (D_p) in the periplasm.
- **E, E and E to 2E, 3E and 4E in periplasm trans:** it occurs in the trans-membrane, consuming one metabolite (E_e) from the extracellular space, one (E_c) from the cytoplasm, and one (E_p) from the periplasm. It produces two metabolites (E_e) in the extracellular space, three (E_c) in the cytoplasm, and four (E_p) in the periplasm.
- **No product in periplasm:** This reaction occurs in the periplasm. It consumes one metabolite (F_p) and does not produce anything.
- **No substrate from related compartment in periplasm outer:** this transport reaction occurs in the periplasm outer membrane. It transports a metabolite (G_e) from the extracellular space to the periplasm, obtaining a metabolite (G_p) in the periplasm.

Once the SBML file is defined, the *PMGMP* application generates atomic models in Cadmium parameterized using the parameters in the SBML file. These models are automatically built based on the parameters provided. We first generate a parameterized coupled model in Cadmium (ready to compile without any changes). We can also use different external parameters without recompiling the model to test the effect of different parameters. In this case, we use an external parameter to define 1,000 enzymes b0000 in each enzyme set and 600,000 metabolites of each type (A to G). Because each reaction uses different metabolites, all the metabolic pathways are composed by one reaction. We also set the volume of each compartment to be equal to the estimated volume of the E. Coli bacteria (0.528 cubic micrometers for the cytoplasm, and 0.072 cubic micrometers for the periplasm and extracellular space). We set the K_{on}

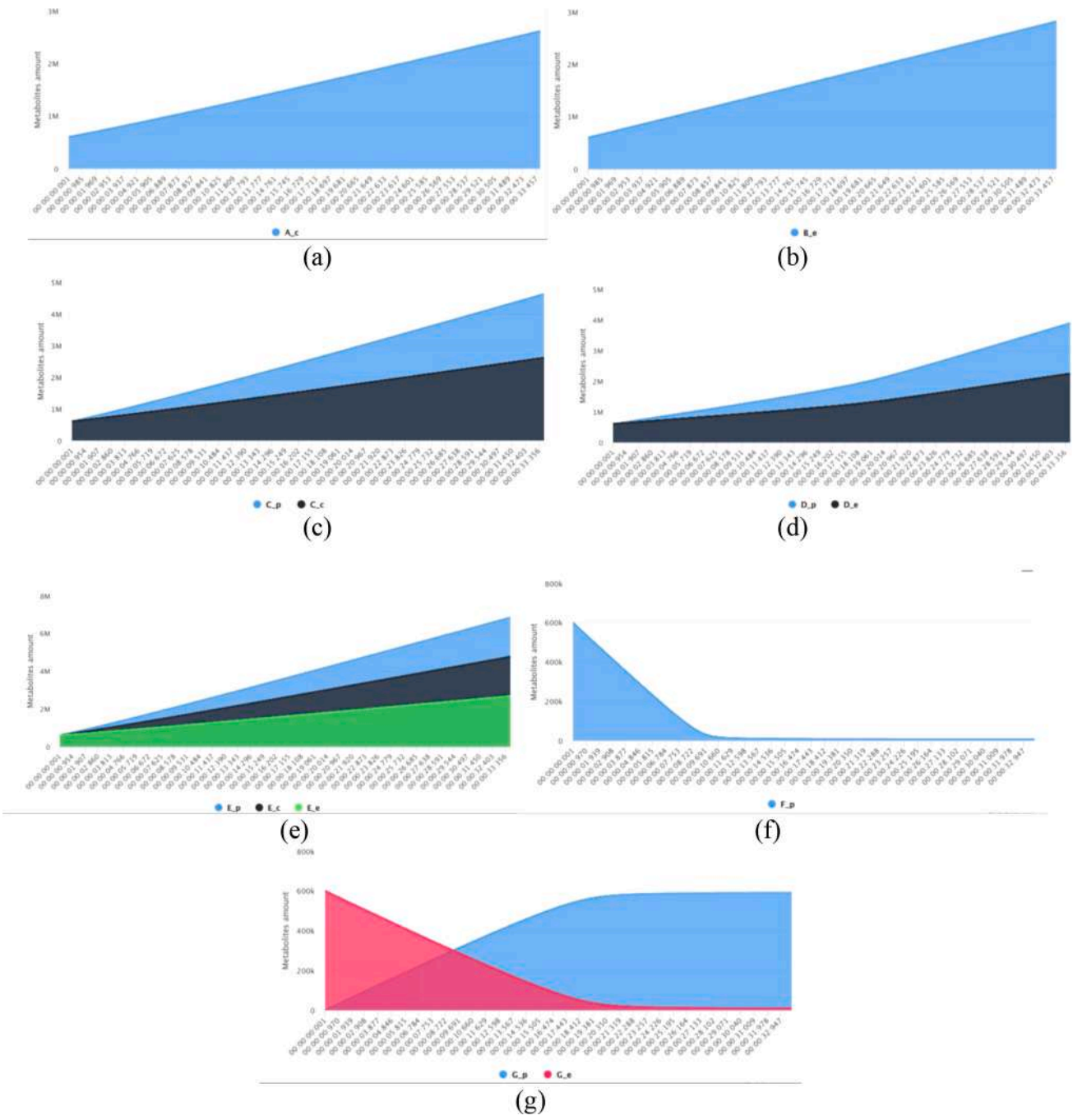


Fig. 12. Number of metabolites for the different reactions occurring in the synthetic model. (a) amount of metabolites A_c in time; (b) metabolites B_e; (c) metabolites C_p and C_c; (d) metabolites D_p and D_e (e) metabolites E_p, E_c and E_e (f) metabolites F_p and (g) metabolites G_p, and G_e.

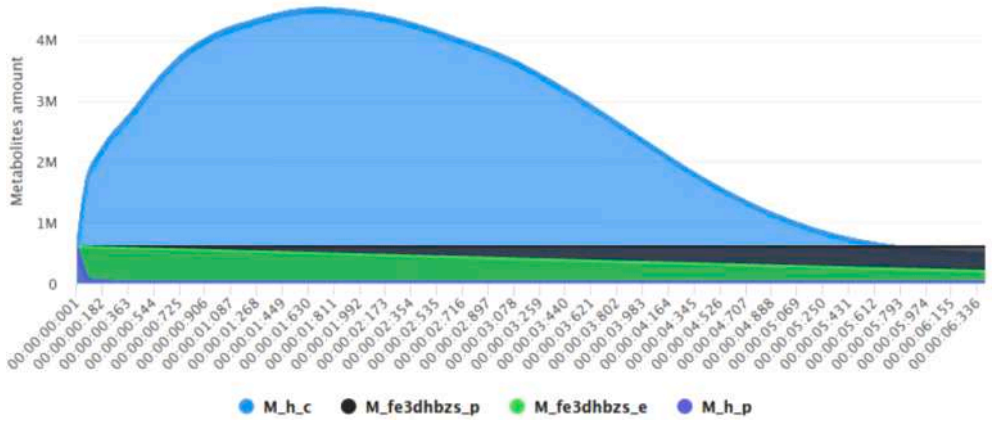


Fig. 13. Number of metabolites in the FE3DHBZStonex transport reaction for E. Coli bacteria

Table. 1
Metabolite productions and consumption rates in the entire E. Coli bacteria.

Metabolite	Total production	Total consumption	Final production rate
M_fe3dhbzs_e	0	2	-2
M_fe3dhbzs_p	1	1	0
M_h_p	130	232	-102
M_h_c	920	353	567

and K_{off} constants equal to 0.8 for all the reactions to increase the probability of binding metabolites and decrease the probability of rejecting bound metabolites. Finally, we set the enzymes reacting, ejecting and the spaces intervals of time in one millisecond, which is an arbitrary value for our synthetic model, but it does not affect the results as all the reactions are independent of each other. The rest of the parameters are not modified. Fig. 12 shows the simulation results of the metabolic pathways from the proposed synthetic model.

From the simulation results we can extract the following conclusions:

- All the metabolites that increased by one in a reaction have a similar linear curve with a mean final number of 2,500,000 metabolites: metabolites A_c, B_e, C_c, D_e, and E_e.
- All the metabolites that increased by two in a reaction have a similar linear curve with a mean final number of 4,500,000 metabolites, which is more than the metabolites produces by reactions that increased the number by one. These are E_c, D_p, and C_p.
- The metabolite E_p, which is produced by a reaction that increased by three the number of metabolites, has a similar linear curve with a mean final number of 7,000,000 metabolites. This is almost 3 times the value obtained by a reaction that increases the metabolites by one.
- The metabolite F_p decreases until there are none left. As shown in the figure, the decreasing curve is not linear because each time a "*No product in periplasm*" reaction occurs, the number of metabolites F_p decreases, and therefore, the probability of binding also decreases, making the reaction flow rate slower.
- The metabolite G_e decreases at the same rate than the metabolite G_p increases, which is consistent with the transport reaction. In this case, we see the same effect shown as with metabolite F_p: the reaction flow rate is not linear as the amount of metabolite G_e decreases because the probability of getting the reaction also decreases.

4.2. Modeling E. COLI

In this section, we present a case study where we build a model for the E. Coli bacteria. We use an SBML model available in https://github.com/SimulationEverywhere/PMGBP-PDEVS/blob/master/msb201165-sup-0003.xml, which provides the cell structure, enzymes and reactions involved in the E. Coli metabolic pathways. However, this model does not have information about the K_{on} and K_{off} constants or the reaction and reject rates (as the objective of this research is to show how to build these models with the proposed method and obtaining the real values of these parameters is out of the scope, we cannot compare the results with in-vitro or in-situ results. The advantage of our approach is that once these values become available, we just need to update those values in the XML parameter file and re-run the simulation and the analysis).

For the analysis presented in this section, after the parameterized coupled model and the XML file with the model parameters were generated using *PMGMP*, we updated the XML with the following parameters: $K_{on} = 0.8$ for all the reactions; $K_{off} = 0.8$ for all the reactions; reaction time = 1 millisecond for all the reactions; reject time = 1 millisecond for all the reactions; space interval of time = 1 millisecond for all the compartments; initial enzyme amount = 1,000 for all the enzymes; initial metabolite amount = 600,000 for all

the metabolites. During the simulation, 2581 types of reactions were handled by 1577 types of enzymes consuming and producing a total of 1805 types of metabolites. Here we analyze a reaction and their metabolites as example. We have chosen the reaction *R_FE3DHBZStonex*. This is a transport reaction located in the periplasm trans-membrane of the E. Coli cell. The reaction R_FE3DHBZStonex has the following stoichiometry:

$$M\_h\_p + M\_fe3dhbzs\_e \rightarrow M\_h\_c + M\_fe3dhbzs\_p$$

This reaction transports metabolite h from the periplasm (M_h_p) to the cytoplasm (M_h_c), and metabolite fe3dhbzs from the extracellular space (M_fe3dhbzs_e) to the periplasm (M_fe3dhbzs_p).

Fig. 13 shows the result of the metabolites handled by this reaction.

As shown in Fig. 13, the substrate metabolites M_h_p and M_fe3dhbzs_e decrease, but the product metabolites M_h_c and M_fe3dhbzs_p do not always increase. An interesting result is that the metabolite M_h_c (metabolite h in the cytoplasm) has a rapid increasing rate until the virtual time 00:00:01:811. Then it starts decreasing at almost the same rate. Metabolite M_fe3dhbzs_p remains almost constant.

To understand this phenomenon and considering we have used the same parameters for all the reactions, we studied the total production and consumption rates of these metabolites in the entire bacteria. Table 1 shows for each metabolite the total production rate considering all the reactions that produce that metabolite and the total consumption rate considering all the reactions that consume that metabolite. The final production rate is the difference between the production and the consumption rates. If a metabolite has a higher consumption than production rate, the metabolite tends to decrease its number over the time, while a metabolite with a higher production than consumption rate tends to increase its number over the time.

M_h_p has a significant negative final production rate. This result matches what we can see in Fig. 13. The metabolite is almost completely consumed in the first milliseconds. M_fe3dhbzs_e has a final production rate of -2, which is consistent with the simulation results where it decreases slower than the M_h_p metabolite. The metabolite M_fe3dhbzs_p has a final production rate of 0 and as shown in the simulation result it remains almost constant over the time. M_h_c has a final production rate of 567 which is consistent with the first 1.811 simulation virtual seconds, but it is contradictory with the remaining simulation virtual time.

If we look at substrate of all the reactions producing the metabolite M_h_c, and we analyze the mean of the final production rate of these metabolites, we have that the rate is approximately -0.037. This result shows that, in general, the metabolites needed to produce M_h_c tend to decrease. If look at how M_h_c is consumed (i.e., the substrate of all the reactions consuming the metabolite M_h_c), and we analyze the mean of their final production rate, we have that the rate is approximately 0.7889. This means that the metabolites needed to consume M_h_c tend to increase. When the simulation starts, we have enough metabolites, and the final production rate of the metabolite M_h_c is the one shown in table 1. However, with time, the remaining metabolites needed to produce M_h_c decrease, and the metabolites needed to consume M_h_c increase. Therefore, M_h_c decreases. This explains the original unexpected result where, the metabolite M_h_c decreases even though its final production rate is high.

5. Conclusions

We introduced a flexible structure for modeling reactions occurring in biological cells. This structure was implemented using parameterized DEVS models. This framework can be improved and adapted for different projects, allowing to integrate new models when needed. The proposed structure considers general aspect of cells, allowing the user to modify the components.

To automate the generation of models, the *PMGMP* software takes a biological model defined in SBML and uses it to generates a DEVS model reading to compile. It also generates an XML file with the model parameters. The user can modify this file to run different simulation scenarios without re-compiling the model. Additionally, most users can use the application if they have a biological model defined in SBML available.

We introduced two case studies: a simple synthetic model and a real model of the E. Coli. The reactions in the synthetic model are stochastic processes. Therefore, even for those metabolites with the same parameters in their corresponding reactions, there are some differences in the curves. However, because we set a considerable number of enzymes, the probability reaches values close to the mean and we obtain almost linear curves without considerable deviations. As future work, instead of calculating the binding probability individually for each enzyme, we could consider a general formula that calculates the final total number of enzymes that bound metabolites. This would be a major improvement in the simulation time complexity from linear to almost constant.

Acknowledgements

This Research has been partially funded by NSERC. The complete research materials used as source for the article can be found in [45]

Appendix I. SBML representation of models

SBML is a generic specification language to store information of different biological phenomena. The SBML general structure is presented in Fig. A1.

SBML models use a simple hierarchy contained under the *Model* tag as shown in Fig. A1 (where black diamond arrows indicate element relations with their cardinality; * means that cardinality can be zero, one or multiple elements; each element name is delimited by <> to specify they are XML tag elements). In the first level, there can be different lists of elements: such as compartments, species, and reactions. In the second level, each list stores several single elements. For example, *ListOfReactions* holds multiple instances of

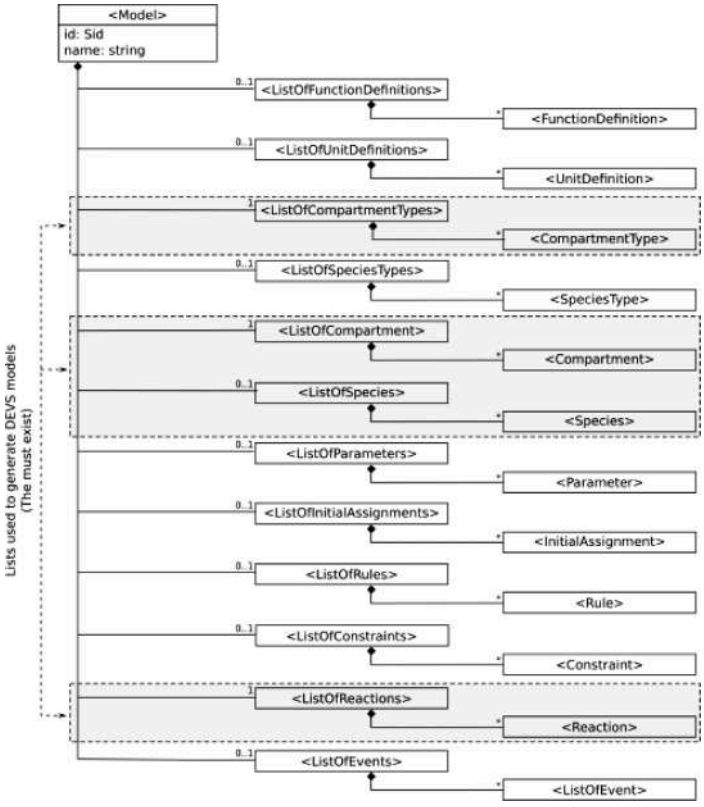


Fig. A1. Structure of general structure of an SBML v2.5 file

Reaction elements. Every element has a sub-structure to store its information. Additionally, an element can have a "note" attribute where we can store any valid HTML structure to include any information that does not fit into the SBML specification structure (e.g., a list of parameters, which are not usually included in macro view models).

ListOfCompartmentTypes is used to define the compartment types present: Extracellular space, Periplasm, Cytoplasm and Organelle can be declared in this list. Then, *ListOfCompartments* is used to specify the different compartments present in a cell. For example, we may have a compartment called Vacuole that is of type Organelle. The properties of the reactions occurring within a cell are defined in *Reaction* elements within the tag *ListOfReaction*. The structure of the reaction element is shown in Fig. A2. Among other elements, a reaction includes an *id*, if the reaction is reversible or not, and if it is a fast reaction. Similarly, it references to the list of species in the reaction, which are defined under the tag *ListOfSpecies*.

Each *Species* element in *ListOfSpecies* includes various attributes that include:

- *Id*: an identification that allows us to differentiate each metabolite (species) in the model
- *Name*: it allows us to show the biological name of each metabolite.
- *Compartment*: it holds the compartment ID where the species belongs.
- *Initial Amount*: Specifies the total number of metabolites of that species existing at the initial time.

The reaction stoichiometry is a quantitative relationship between reactants and products. It is defined using *listOfReactants* and *listOfProducts*, which are lists of species belonging to the stoichiometry. Each species within these lists has a stoichiometry attribute (a number showing the amount of that species contained in the stoichiometry). The remaining properties, such as the reaction rate, are defined in the *ListOfParameters*.

Enzymes that catalyze the different reactions are an important aspect in biological models. However, the SBML specification structure does not consider enzymes. Enzymes are specified in the reactions using logical expressions to denote the enzymes' name. To know to which compartments each enzyme belongs, we could use the compartment where the reaction happens. Nevertheless, in SBML all the reactions are listed in the *ListOfReaction* without being separated by compartments.

Appendix II. DEVS models formal specification

DEVS Atomic models are used define the behavior of the system. The formal definition of an atomic model is as follows:
$AM = \langle X, Y, S, ta, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda \rangle$

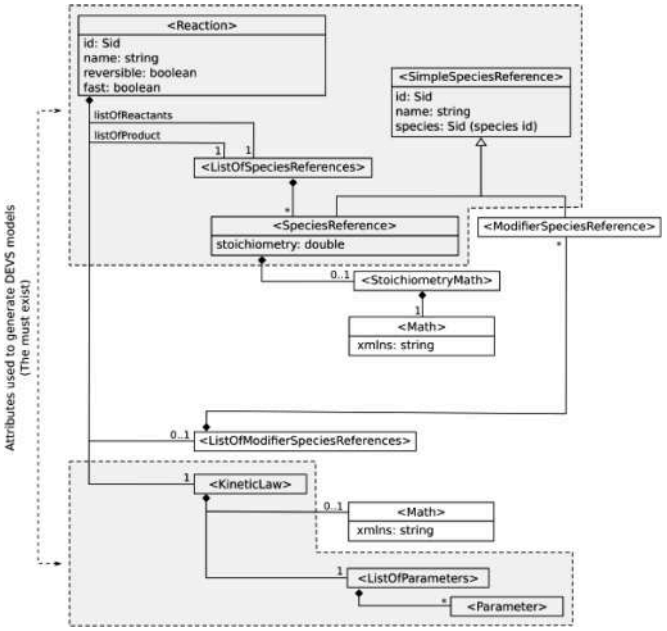


Fig. A2. Structure of the reaction element in an SBML Model

Where:

X is the set of input events.

Y is the set of output events.

S is the set of sequential states.

$ta : S \rightarrow \mathbb{R}_0^+ \cup \infty$ is the time advance function that determines the time until the next internal transition.

$\delta_{ext} : QxX^b \rightarrow S$ is the external transition function that determines the next state when external events arrive, where $Q = \{(s,e)s \in S,\ 0 \leq e \leq ta(s)\}$, e is the elapsed time since the last state transition and X^b is a set of bags over elements in X.

$\delta_{int} : S \rightarrow S$ is the internal transition function that determines the state transition of the model when the state duration is over, and no external event has arrived.

$\delta_{con} : QxX^b \rightarrow S$ is the confluence transition function that determines the next state when and external events arrive at the same time than an internal transition is triggered.

$\lambda : S \rightarrow Y^b \cup \varnothing$ is the output function that determines the output of the model based on its current state. Y^b is a set of bags over elements in Y and $\varnothing$ is the empty set.

Coupled models are defined connecting multiple DEVS models (coupled or atomic) linking the models' inputs and outputs. A coupled model is defined as the next 7-tuple:

$CM = \langle X, Y, D, \{M_d d \in D\},\ EIC, EOC, IC\rangle$

Where:

X : Is the set of input events.

Y : Is the set of output events.

D : Is the set of the names of the sub-components.

$\{M_d\}$: Is the set of sub-components where $d \in D$. Each M_d is a DEVS model (either atomic or coupled)

EIC: is the set of external input couplings

EOC: is the set of external output couplings

IC: is the set of internal couplings

Appendix III. – Algorithm to calculate collisions between enzymes and metabolites in the *space* atomic model

To understand the logic of the algorithm used in the *Space* atomic model to calculate collisions between enzymes and metabolites, we will first explain the biological binding mechanism that we modelled.

There are multiple binding mechanisms; one of the most common (which we used here) is as follows. Enzymes and metabolites are in the bulk solution, and when a metabolite is near an enzyme, they can collide. When a metabolite collides with an enzyme there is a chance that the metabolite will bind with the enzyme. The probability of this event is not independent of the current enzyme state: every time an enzyme binds to a new metabolite, the chances to bind to new ones increase exponentially.

To model this binding mechanism, we consider that once an enzyme binds to the first metabolite, the probability to bound the remaining needed metabolites is close to one (because it increases exponentially). Therefore, once an enzyme binds the first

metabolite, we can assume that it will bound all the remaining metabolites it needs to react. Thus, we use a binary probability function where the enzyme binds all the metabolites it needs to react to, or it does not bind any metabolite at all.

Some enzymes consume metabolites from different compartments (i.e., different *Space* atomic models). In this case, the binding process is independent for all the compartments and each *Space* model will calculate the binding process without knowing the state of other compartments.

To calculate the binding probability, we use the following spatial stochastic formula that considers the space volume to calculate the metabolite concentrations $[A_i]$:

$$[A_i] = \frac{|A_i|}{L * compartment\_volume}$$

Where

L is the Avogadro constant used to deal with the fact that concentrations are calculated in moles unit and in our micro-view model we have the metabolites in metabolite amount units.

Multiple enzymes can bind metabolites of the same type. Then, if an enzyme binds a metabolite, the concentrations of that metabolite type decrease. To model this, we use a uniform random distribution to determine in which order the enzymes will bind the metabolites. Each time an enzyme binds some metabolites, the free metabolites amount is updated $|A_i|$, and the concentrations decrease for the remaining free enzymes.

Similarly, an enzyme can handle multiple reactions depending on which metabolites the enzyme bounds. To model this, we use a two-step process where we first calculate the partial probability of the enzyme to bind the metabolites of each reaction that it handles using the following formula:

$$P_{partial\_binding}(reaction, k_{on}) = e^{\frac{-1}{\prod_{i=0}^{n} |A_i| * k_{on}}}$$

$A_i \in$ reaction stoichiometry.

n : number of elements in the stoichimetry.

K_{on} : association constant.

Then, we integrate all the partial probabilities and determine which metabolites will bind the enzyme using the following formula:

$$P_{bind}(r_i) = P(\sum_{j=0}^{j=i-1} P_{partial\_binding}(r_j, K_{on_j}) \leq X < \sum_{j=0}^{j=i} P_{partial\_binding}(r_j, k_{on_j}))$$

$X \sim U[0,1]$

$r_j : j \in [0, n]$ the j-th reaction handled by the enzyme in some arbitrary order.

n : number of handled reactions.

As $P_{bind}(r_i)$ is within the interval [0, 1], we normalize the single probabilities to fit in the interval when the sum of all the single probabilities is greater than 1.

The formula to calculate $P_{bind}(r_i)$ splits the interval [0, 1] into n+1 disjoint subintervals. Each interval represents a different reaction handled by the enzyme. The duration of each interval is determined by $P_{partial\ binding}$ (reaction; k_{on}) according to the concentration (in the compartment) of the metabolites involved in the reaction. Once the interval [0, 1] is divided, we use a uniform random

Algorithm 1
Calculates the enyzme and metabolites collisions.

```
procedure CALCULATE_COLLISIONS(state, K_onPTS, K_onSTP)
    collisions ← {}
    for enzyme ∈ state.enzymes do
        p_kons ← []
        for reaction ∈ enzyme.handled_reactions do
            P_kons ← P_kons · [< reaction, STP, P_partial_binding(reaction, K_onSTP) >]
            if reaction is reversible then
                P_kons ← P_kons · [< reaction, PTS, P_partial_binding(reaction, K_onPTS) >]

        if Σ_{p∈p_kons} p > 1 then
            p_kons ← [p_kon / Σ_{p∈p_kons} p | p ∈ p_kons]
        X ← U(0,1)                                   ▷ Uniform random value
        for i ∈ [0, ..., |P_kons|] do
            if Σ_{j=0}^{j=i-1} P_kons_j < X < Σ_{j=0}^{j=i} P_kons_j then
                collisions ← collisions ∪ {p_kons_i}
    return collisions
```

variable to choose one of the sub-intervals. Finally, the enzyme binds the metabolites of the reaction represented by the chosen interval. The last interval (the number n + 1) represents a case where the enzyme does not bind any metabolite.

The length of the last interval (the interval n + 1) is not obtained by $P_{\text{partial binding}}$ (reaction; k_{on}); instead, it is the remaining space once the first n intervals are calculated.

This collision process is summarized in Algorithm 1.

Appendix IV. XML schema for the parameters of the model

The Fig. A3 represents the UML diagram of the XML parameters file.

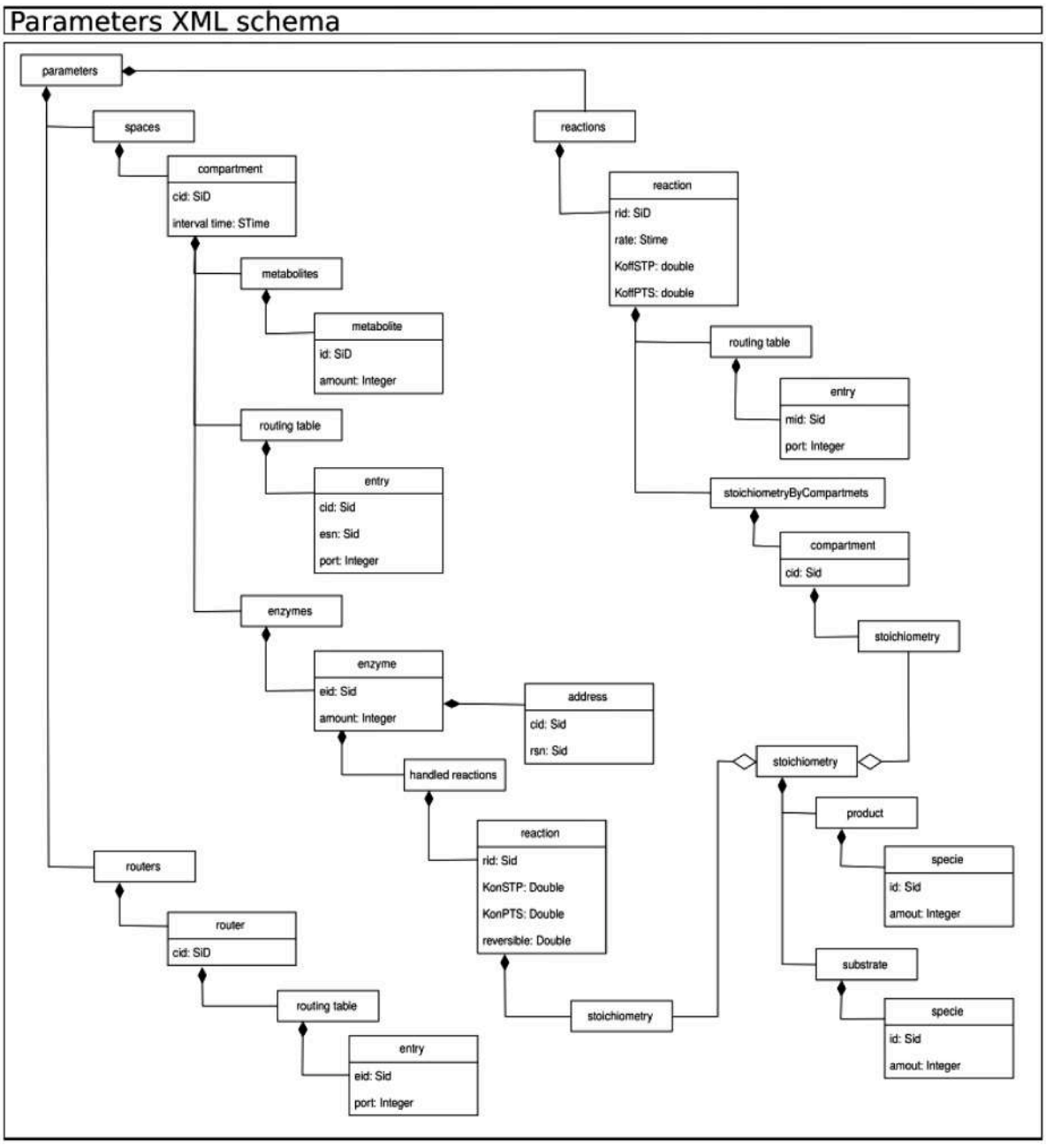


Fig. A3. UML diagram representation of the parameters in the XML file.

The root element is *parameters*. This element contains 3 main lists: *spaces, reactions,* and *routers*.

Space represents all the parameters for atomic models of type space. *Space* has a list with all the *compartments.* Each *compartment* has the following elements:

- *cid*: model ID used by each model to find its corresponding parameters.
- *interval time:* fixed-length interval time steps used by space atomic models. This attribute is a string to define time with the format "hours:minutes:seconds:milliseconds".
- *metabolites*: list containing the information of the metabolites that belong to the compartment. Each metabolite information is specified in a *metabolite element* within this list and has the following attributes:
  - *id*: identifies the metabolite across the models. When a metabolite is sent from one model to another, the id is sent to identify it.
  - *amount*: is an integer attribute that defines the initial amount of metabolite available in the compartment.
- *routing table*: specifies for each pair of compartment ID and enzyme set ID, which output port must be used to send metabolites to an enzyme. Each entry element has the following attributes:
  - *cid*: id of the compartment where the enzyme set is located.
  - *esn*: id of the enzyme set.
  - *port*: number of the port that must be used to reach the enzyme set.
- *enzyme* contains the information of the enzymes that are related to the *compartment*. Each enzyme information is specified in an *enzyme* element within this list and has the following attributes and children:
  - *eid*: identifies the enzyme across the models.
  - *amount*: is an integer attribute that defines the initial number of free enzymes (i.e., an enzyme without any metabolite bound) in the compartment.
  - *address*: It is a child element containing the compartment and enzyme set IDs where the enzyme belongs.
  - *handled reactions*: it is a child element organized as a list with the information of all the reactions handled by the enzyme. For each reaction, it includes a reaction element with the following attributes and children:
    - *rid*: id of the reaction.
    - *KonSTP*: Kon constant for the Substrate to Product direction.
    - *KonPTS*: Kon constant for the Product to Substrate direction.
    - *reversible*: Boolean indicating if the reaction is reversible or not.
    - *stoichiometry*: A child element that has two lists of species (Product and Substrate) that specifies each one of the species id and amount that belongs to the reaction stoichiometry.

Enzyme represents the all the parameters for atomic models of type Enzyme. *Enzyme* is basically a list of handled *reactions*. Thus, to instantiate an enzyme atomic model we need all the parameter of the enzyme handled reactions. Each reaction information is contained in a *reaction* element within the list *reactions.* It has the following attributes and children elements:

- *id*: reaction id.
- *rate*: time that takes the reaction to convert the consumed metabolites to the produced metabolites.
- *Ko_STP*: Ko constant for the Substrate to Product direction.
- *Ko_PTS*: ko constant for the Product to Substrate direction.
- *routing table*: list containing routing table entries that indicates which port must be used to send each metabolite to its corresponding space atomic model. Each entry has a *mid* (metabolite id) and a port number.
- *stoichiometryByCompartment*: list of stoichiometry divided by compartments as we already explained. As we can see in Fig. A3, this list has a child for each compartment, and each compartment has one child where the reaction stoichiometry related to that compartment is specified.

Router represents all the parameters for the atomic models of type router. All the parameters of a router atomic model are contained in a *router* element within the *routers* list element. A *router* element has an *id* attribute to identify the model and a *routing table* element. Each child of the *routing table* element is an *entry* element that specifies, for each enzyme id (*eid* attribute) the port where the message must be routed to reach the enzyme (*port* attribute).

References

[1] S. Klamt, J. Stelling, Combinatorial Complexity of Pathway Analysis in Metabolic Networks, Molecular Biology Reports 29 (1-2) (2002) 233–236.
[2] A.M. Uhrmacher, D. Degenring, B. Zeigler, Discrete Event Multi-Level Models for Systems Biology, **Transactions on computational systems biology I** (2005) 66–89.
[3] R. Ewald, J. Himmelspach, M. Jeschke, S. Leye, A.M. Uhrmacher, Flexible Experimentation in the Modeling and Simulation Framework JAMES II. Implications for Computational Systems Biology, Briefings in bioinformatics 11 (3) (2010) 290–300.
[4] W.B. Copeland, B.A. Bartley, D. Chandran, M. Galdzicki, K.H. Kim, S.C. Sleight, C.D. Maranas, H.M. Sauro, Computational Tools for Metabolic Engineering, Metabolic engineering 14 (3) (2012) 270–280.

[5] J.R. Karr, J.C. Sanghvi, D.N. Macklin, M.V. Gutschow, J.M. Jacobs, B. Bolival Jr, N. Assad-Garcia, J.I. Glass, M.W. Covert, A Whole-cell Computational Model Predicts Phenotype from Genotype, Cell 150 (2) (2012) 389–401.
[6] SBML Team, (2010) "The History of SBML", http://sbml.org/History_of_SBML (Last Access: 21/05/2021).
[7] M. Hucka, A. Finney, H.M. Sauro, H. Bolouri, J.C. Doyle, H. Kitano, A.P. Arkin, B.J. Bornstein, D. Bray, A. Cornish-Bowden, et al., The Systems Biology Markup Language (SBML): a Medium for Representation and Exchange of Biochemical Network Models, Bioinformatics 19 (4) (2003) 524–531.
[8] A. Bauer-Mehren, L.I. Furlong, F. Sanz, Pathway Databases and Tools for their Exploitation: Benefits, Current Limitations and Challenges, Molecular systems biology 5 (1) (2009) 290.
[9] Z. Wang, Cell Biology Simulation using DEVS Combined with SBML, Department of Electrical and Computer Engineering, The University of Arizona, 2009. Master of Science.
[10] B.P. Zeigler, T.G. Kim, H. Prähofer, Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, 2nd ed., Academic Press, San Diego, California, 2000.
[11] S.M. Ruggiero, A.N. Ford Versyp, SBMLtoODEpy: A Software Program for Converting SBML Models into ODE Models in Python, Journal of Open-Source Software 4 (41) (2019) 1643.
[12] L. Belloli, G. Wainer, R. Najmanovich, Parsing and Model Generation for Biological Processes, in: 2016 Symposium on Theory of Modeling and Simulation (TMS-DEVS), SpringSim 2016, Pasadena, Ca, USA, 2016.
[13] G. Wainer, R. Djafarzadeh, DEVS Modelling and Simulation of the Cellular Metabolism by Mitochondria, Molecular Simulation 36 (12) (2010) 907–928.
[14] H.-G. Holzhutter, G. Jacobascj, A. Bisdorff, Mathematical Modelling of Metabolic Pathways Affected by an Enzyme Deficiency: A Mathematical Model of Glycolysis in Normal and Pyruvate-Kinase-Deficient Red Blood Cells, European journal of biochemistry 149 (1) (1985) 101–111.
[15] A.M. Uhrmacher, R. Ewald, M. John, C. Maus, M. Jeschke, S. Biermann, Combining Micro and Macro-Modeling in DEVS for Computational Biology, in: Winter Simulation Conference, Washington, DC, USA, 2007.
[16] V.S. Ayyadurai, C.F. Dewey, Cytosolve: a Scalable Computational Method for Dynamic Integration of Multiple Molecular Pathway Models, Cellular and Molecular Bioengineering 4 (1) (2011) 28–45.
[17] H.M. Sauro, D. Harel, M. Kwiatkowska, C.A. Shafer, A.M. Uhrmacher, M. Hucka, P. Mendes, L. Stromback, J.J. Tyson, Challenges for Modeling and Simulation Methods in Systems Biology, in: Winter Simulation Conference, Monterrey, CA, USA, 2006.
[18] C. Maus, M. John, M. Röhl, A.M. Uhrmacher, Hierarchical Modeling for Computational Biology". Formal Methods for Computational Systems Biology, in: SFM 2008. Lecture Notes in Computer Science, 5016, Springer, Berlin, Heidelberg, 2008.
[19] B.J. Bornstein, S.M. Keating, A. Jouraku, M. Hucka, LibSBML: an API Library for SBML, Bioinformatics 24 (6) (2008) 880–881.
[20] F Maggioli, T Mancini, E Tronci, SBML2Modelica: Integrating Biochemical Models within Open-Standard Simulation Ecosystem, Bioinformatics 36 (7) (2020) 2165–2172.
[21] T.-S. Jung, K.-R. Kim, S.-H. Jung, T.-K. Kim, M.-S. Ahn, J.-H. Lee, W.-S. Cho, in: SPDBS: An SBML-Based Biochemical Pathway Database System", International Conference on Intelligent Computing, Kunming, Yunnan, China, 2006.
[22] J. Scott-Brown, A. Papachristodoulou, SBML-diff: A Tool for Visually Comparing SBML Models in Synthetic Biology, ACS Synthetic Biology 6 (7) (2017) 1225–1229.
[23] L. Watanabe, J. Barhak, C. Myers, Toward Reproducible Disease Models using the Systems Biology Markup Language, Simulation 95 (10) (2019) 895–930.
[24] SBML (2019) The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 2 Core. Release 2. Available online: http://sbml.org/Special/specifications/sbml-level-3/version-2/core/release-2/sbml-level-3-version-2-release-2-core.pdf (Last Access: 21/05/2021).
[25] Y. Van Tendeloo, H. Vangheluwe, An Evaluation of DEVS Simulation Tools, Simulation 93 (2) (2017) 103–121.
[26] J. Nutaro (2014). A Discrete EVent system Simulator. Available: http://web.ornl.gov/~1qn/adevs/adevs-docs/manual.pdf. (Last Access: 21/05/2021).
[27] Y. Van Tendeloo, H. Vangheluwe, in: The Modular Architecture of the Python (P)DEVS Simulation Kernel" Symposium on Theory of Modeling & Simulation, Tampa, FL, 2014.
[28] G. Quesnel, R. Duboz, E. Ramat, M.K. Traore, VLE: a Multimodeling and Simulation Environment, in: Summer Computer Simulation Conference, San Diego, CA, 2007.
[29] G. Wainer, Discrete-Event Modeling and Simulation: A Practitioner's Approach, CRC Press, Boca Raton, FL, USA, 2009.
[30] G. Wainer, Advanced Cell-DEVS Modeling Applications, SIMULATION: Transactions of the SCS. (2018), https://doi.org/10.1177/0037549718761596.
[31] MS4 Me user guide Available Online: www.ms4system.com. (Last Access: 21/05/2021).
[32] K. Sungung, H. Sarjoughian, Vignesh Elamvazhuthi, DEVS-Suite: a Simulator Supporting Visual Experimentation Design and Behavior Monitoring, in: Spring Simulation Conference, San Diego, CA, USA, 2009.
[33] L. Belloli, D. Vicino, C. Ruiz-Martin, G. Wainer, Building Devs Models with the Cadmium Tool, in: Winter Simulation Conference, National Harbor, MD, USA, 2019.
[34] D. Heredia, V. Sanz, A. Urquia, M. Sandin, A Systemic Approach for Modeling Biological Evolution using Parallel DEVS, Biosystems 134 (2015) 56–70.
[35] N. Akhtar, N. Niazi, F. Mustafa, A. Hussain, A Discrete Event System Specification (DEVS)-based Model of Consanguinity, Journal of Theoretical Biolology 285 (1) (2011) 103–112.
[36] G. Wainer, R. Djafarzadeh, DEVS Modelling and Simulation of the Cellular Metabolism by Mitochondria, Molecular Simulation 36 (12) (2010) 907–928.
[37] Z. Wang, Cell Biology Simulation using DEVS Combined with SBML, Department of Electrical and Computer Engineering, The University of Arizona, 2009. Master of Science.
[38] G Wainer, K Al-Zoubi, R. Madhoun, Distributed Simulation of DEVS and Cell-DEVS Models in CD+ + using Web-Services, Simulation Modelling Practice and Theory 16 (9) (2008) 1266–1292.
[39] K. Al-Zoubi, G. Wainer, RISE: A General Simulation Interoperability Middleware Container, Journal of. Parallel and Distributed Computing. 73 (5) (2013) 580–594.
[40] G. Wainer, S. Wang, MAMS: Mashup Architecture with Modeling and Simulation as a Service, Journal of Computer Science 21 (2017) 113–131.
[41] NATO, (2013), "Allied Framework for Modelling and Simulation as a Service (MSaaS)" https://nmsg.sto.nato.int/themes/msaas. (Last Access: 27/09/2021).
[42] W. Kenneth (2013). "SaaS Redefined: Simulation as a Service (or) Cloud-Hosted Simulation" https://www.digitalengineering247.com/article/saas-redefined-simulation-as-a-service-or-cloud-hosted-simulation/ (Last Access: 27/09/2021).
[43] R. Barwell, G. Wainer, Strategic Airlift Operationalizing Constructive Simulations, in: SCS/ACM/IEEE Annual Simulation Symposium (ANSS), ANNSIM'21 Fairfax, VA, 2020.
[44] K Al-Zoubi, G Wainer, Fog and Cloud Collaboration to Perform Virtual Simulation Experiments, Simulation Modelling Practice and Theory 101 (2020), https://doi.org/10.1016/j.simpat.2019.102032.
[45] L. Belloli, Biological Modeling and Simulation as a Service, Facultad de Ciencias Exactas y Naturales, Departamento de Computacion, Universidad de Buenos Aires, 2019. Master of Science Equivalent.