Expressive rule-based modeling and fast simulation for dynamic compartments

Till Köster[1],Philipp Henning[1,2], Tom Warnke[1,3], Adelinde Uhrmacher[1]

1 Institute for Visual and Analytic Computing, University of Rostock, Germany
2 Institute of Medical Biochemistry and Molecular Biology, University Medicine Rostock, Germany
3 Limbus Medical Technologies GmbH, Rostock, Germany

Abstract

Compartmentalization is vital for cell biological processes. The field of rule-based stochastic simulation has acknowledged this, and many tools and methods have capabilities for compartmentalization. However, mostly, this is limited to a static compartmental hierarchy and does not integrate compartmental changes. Integrating compartmental dynamics is challenging for the design of the modeling language and the simulation engine. The language should support a concise yet flexible modeling of compartmental dynamics. Our work is based on ML-Rules, a rule-based language for multi-level cell biological modeling that supports a wide variety of compartmental dynamics, whose syntax we slightly adapt. To develop an efficient simulation engine for compartmental dynamics, we combine specific data structures and new and existing algorithms and implement them in the Rust programming language. We evaluate the concept and implementation using two case studies from existing cell-biological models. The execution of these models outperforms previous simulations of ML-Rules by two orders of magnitude. Finally, we present a prototype of a WebAssembly-based implementation to allow for a low barrier of entry when exploring the language and associated models without the need for local installation.

Author summary

Biochemical dynamics are constrained by and influence the dynamics of cellular compartments. Basic constraints are considered by many modeling and simulation tools, e.g., certain reactions may only occur in specific cellular compartments and at a speed influenced by the compartmental volume. However, to capture the functioning of complex compartmental dynamics such as cell proliferation or the fission or fusion of mitochondria, additional efforts are required from tool designers. These refer to how the modeler can specify these dynamics succinctly and unambiguously and how the resulting model can be executed efficiently. For modeling, we rely on ML-Rules, an expressive, formal rule-based language for modeling biochemical systems, which ships with the required features and which we only slightly adapt in our re-implementation. We design a new simulation engine that combines efficient data structures and various algorithms for efficient simulation. The achieved efficiency will enable thorough analysis, calibration, and validation of compartmental dynamics and, thus, allow the “in-silico” pursuit of research questions for which compartmental dynamics are essential. To further facilitate exploring the interplay of compartmental and non-compartmental

dynamics, we exploit recent advances in web technology so that ML-Rules models can be run efficiently in the web browser.

1 Introduction

Compartmentalization is a central aspect of cell biological systems [1]. Modeling such systems as interrelated compartments in and between which processes occur allows approximating the system structure and its effect on the processes. To include the causal influence of compartments on the processes, it suffices to support *static compartments*, that is, compartments that do not change. However, to account for the causal influence of the occurring processes on the compartmental structure and interactions, the model must support *dynamic compartments*. This includes the creation of new compartments, their deletion, merging, and splitting of compartments, as well as one compartment entering or leaving another (Figure 1).

The reciprocal influence of compartments and the processes that occur in and between them can be observed in many cell biological systems. For example, the life cycle of a cell can be modeled as intracellular processes affecting cell growth, ultimately leading to cell division [2]. Intercellular communication in cell populations affects the intracellular behavior of individual cells. Signaling pathways include intracellular compartmental processes such as endocytosis or the transfection of lipoplexes. Although such processes can sometimes be approximated with static compartments, dynamic compartments are the natural method for precise and expressive models.

Several modeling approaches that support dynamic compartments have been proposed in the past ~~[3–5]~~ [3–6]. However, in contrast to approaches limited to static compartments, running simulations for these modeling approaches proved too computationally challenging for usage in relevant biological applications. In fact, some of the approaches developed were never equipped with a (publicly available) simulator implementation.

In the following, we present an approach that overcomes these performance problems and makes dynamic compartments feasible for studying cellular processes. ~~The approach builds on the modeling language~~In particular, our contributions are:

- an analysis of rule-based language design choices on efficiently executing dynamic compartments,
- an analysis of execution requirements for an expressive rule-based language, such as ML-Rules~~ [5]~~, which allows compact and expressive specifications of models with dynamic compartments [7]~~. We devise an~~ ,
- a design of a new efficient stochastic simulation algorithm tailored to models with dynamic compartments ~~. The resulting~~ combining various algorithms with code generation,
- an implementation of a new modeling and simulation tool ~~combines an external domain-specific language and a simulation engine realized in the programming language Rust. Our software design also enables relatively simple deployment of a~~ ML-Rules 3 in Rust based on the new efficient stochastic simulation algorithm, and slight adaptations in comparison to earlier implementations of ML-Rules; the latter includes syntactic enhancements to improve usability, like type checking of units of measurements, and a new type of attributes to facilitate efficient simulation,
- a realization of a WebAssembly-based web simulation tool at `http://mlrules.pages.dev` ~~. The~~ to enable simple deployment of the tool,

- and an evaluation of the expressiveness and performance of ~~our approach are illustrated~~ ML-Rules 3 based on case studies from cell biology~~.~~

  ~~As this is the third version of a simulator implementation for~~ , including an mRNA delivery model, where based on dynamic compartments, due to ML-Rules ~~[5,8], the prototypical re-implementation of ML-Rules is called ML-Rules~~ 3~~and is available at .~~ , now closer matches to biological results become possible.

1.1 Compartmental dynamics

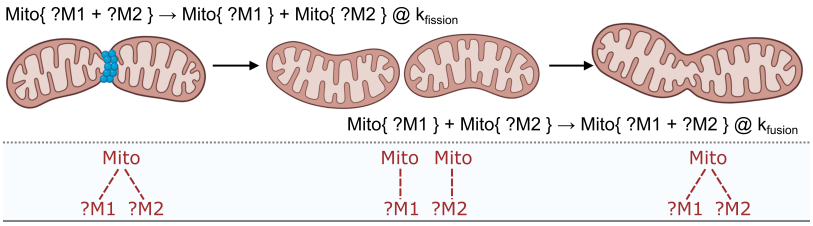
Cell biology considers compartments essential elements of cell behavior [9]. Compartments affect cellular processes by controlling both the reactions that can occur and the rate at which they do. They are considered important to modeling signaling pathways, in which extracellular ligands trigger a cascade of biochemical reactions that span various cellular compartments, such as membrane, cytosol, and nucleus [10]. A prominent example is the canonical Wnt/β-catenin signaling pathway, which is essential for cellular functions such as proliferation and differentiation and is involved in several diseases, including cancer [11]. In the last 20 years, more than 20 quantitative models have been built to analyze the Wnt/β-catenin signaling pathway [12]. Many of these models express the causal influence of compartments on processes, e.g., constrain processes to specific compartments or study proteins shuttling between compartments [13, 14]. Also, the volume of the compartment may influence the reactants' density and, consequently, the kinetics within the different compartments [15, 16]. Compartmental constraints on reactions are supported by most simulation tools and standardized modeling exchange formats such as SBML [16].

~~More generally, treating compartments~~ Considering dynamic compartments opens up new possibilities. Treating compartments as regular species with attributes ~~enable causal effects of~~ enables causal effects between compartments and of the compartmentalization on underlying processes, called *downward causation* [5]. Causality also occurs in the opposite direction— intra-compartmental dynamics can influence the compartmental level, including its attributes such as volume. This *upward causation* can also result in changes to the compartmental structure, such as fission and fusion, or the creation and deletion of compartments.

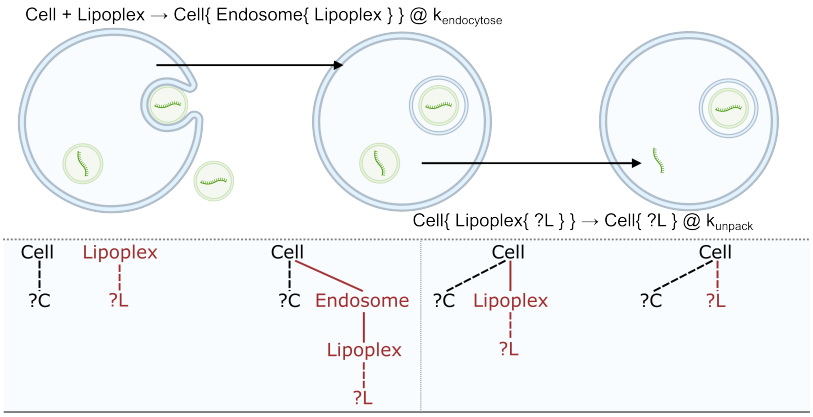
The changes in the compartmental hierarchy occur during the simulation execution. Therefore, they have been called *dynamic compartments* [17]. Dynamic compartments are one form of variable structure model [18] and might occur at and involve various levels of cellular organization [5]. Signaling interactions within multi-cellular populations synchronize and organize the individual cells' behavior in dynamic processes such as tumor growth [19] or morphogenesis [20]. Cell proliferation, migration, and differentiation are regulated by several signaling pathways, which again react to changes in the cells' environment. Dynamic compartments are also necessary for intracellular dynamics; during the last decades, it became evident that a static view of intracellular compartments does not suffice [9]. The internalization of receptors is crucial in regulating signaling pathways [21]. Receptors are sorted into different endosomal compartments when entering the endocytic pathway. To accomplish this sorting, the organelles undergo frequent compartmental dynamics such as maturation, transformation, fusion, fission, and degradation [22]. More recently, the development of mRNA vaccinations has increased the interest in studying liposomal dynamics [23] and its modeling [24] to provide effective drug delivery systems.

Compartmental dynamics take on various forms (cp. Bioambients [6]), including a) the fission and fusion processes of compartments, as in the case of mitochondria [25] (Figure 1a), b) the formation and disintegration of cellular compartments, as in the case of liposomal dynamics [26] (Figure 1b), and c) a compartment entering or leaving

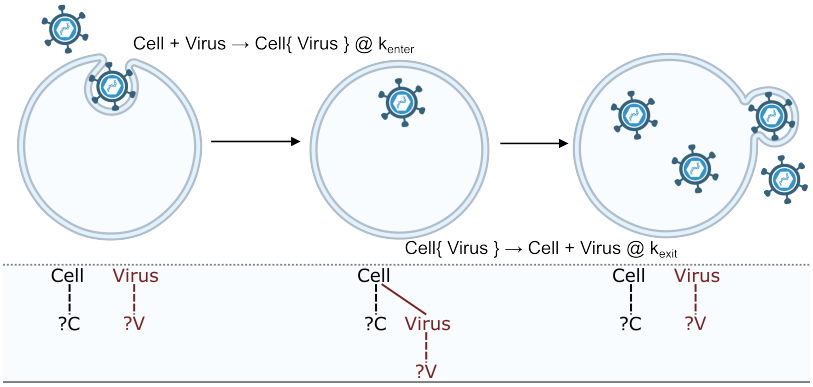
another compartment, as in the case of a viral entry or release [27] (Figure 1c).



(a) Fission and fusion of compartments: A large mitochondrium can divide into two smaller ones, whereby its content (here ?M1 and ?M2) is distributed between the two new mitochondria. Two mitochondria can also fuse into a large one containing the solution of both smaller ones.



(b) Creation and removal of compartments: During the endocytosis of the lipoplex, an endosome is formed around it. Inside the cell, the lipoplex can unpack its content (?L) into the cell.



(c) Shuttling of compartments: A virus carrying DNA or RNA can enter a cell. New viruses produced in the cell can level it and spread.

Fig 1. Examples of compartmental dynamics in cell biology. The figure shows a sketch of the processes, the specification in ML-Rules 3, and the n-ary tree structure of the term rewriting that underlies ML-Rules interpretation. The red lines in the n-ary tree structure indicate structures that are changed during the reaction. For the rules, we use the ML-Rules notation. ~~We have a~~ The left side is transformed (using the →) into the right side at a rate (following the @ symbol). The {} denote nesting relationships.

1.2 Stochastic Simulation

To simulate these compartmental dynamics, we will interpret compartments as discrete entities. We assume that the discrete compartments can be arbitrarily nested, that entities can exist within and outside of compartments, that compartmental dynamics rely on the explicit definition of reactions, and that the reactants are well-mixed within and outside of each compartment.

Based on these assumptions, adopting stochastic simulation algorithms (SSA), popularized under the term of Gillespie algorithms [28], appears most appropriate. These algorithms can be formalized using Continuous-Time Markov Chains (CTMCs) [29]. In CTMCs, the distributions for the waiting time until the next state transition and the successor state only depend on the current state. The sojourn time in a state is exponentially distributed, similar to a physical decay process. The propensity is the expected rate of firing for a particular transition, given a specific model state. Each potential transition i between states in the CTMC has a propensity p_i.

In the *direct method* family of methods [30], a timestep is sampled using the exponential distribution. The total propensity sum $\sum_i p_i$ is used as the rate parameter of the exponential distribution. The reaction is selected by weighted random choice ($\mathcal{P}(i)$ p_i). The *first reaction* family of methods [28] computes a time for every possible reaction by sampling from the exponential distribution (with the respective p_is) and then selecting the smallest one. Both original approaches are wasteful, as many computations might be needlessly redone. Major performance benefits result from storing results, such as propensities [31] or the time of the next event of a reaction [32], and updating this information on demand. For this update on demand, a dependency graph stores the dependencies between reactions and the associated propensities. The next reaction method [32] builds on the first reaction method and maintains a schedule of reactions and the time they will occur. After a reaction is executed, based on the dependency graph, the affected reactions are rescheduled. Another approach builds on the direct method, stores the propensities of reactions, and, based on the dependency graph, updates the affected propensities in each step [31].

1.3 Rule-based modeling and compartments

The modeler enumerates all possible species and reactions in species reaction models. This modeling approach is limited when dealing with complex systems, like proteins involving multiple binding sites, as the number of possible species and reactions can grow exponentially in these cases [33]. The fundamental idea of rule-based modeling (combined with CTMC semantics) is to use rewriting rules to specify transition classes of a CTMC. The left side of a rewriting rule specifies the reactants and their context as a pattern matched to the current state. Using patterns, a single rule can express a large set of reactions, which can be parametrized with the variables matched in the pattern. The patterns on the left rule side constrain the reactants participating in a reaction, for example, a pattern like $A(x) + B(x) \rightarrow \ldots$ expresses that one entity A and one entity B can only react if they share an attribute value x. Each successful match contributes one transition within the CTMC. The rewriting rule is annotated with a rate expression, which is evaluated based on the left side's match. This can include factoring mass action kinetics into the transition rate (based on the multiplicities of the reactants) as well as employing attribute values of the reactants or their context. The successor state of the resulting transition is computed by applying the rewriting operation: removing the reactants and adding the products.

Including dynamic compartments poses a challenge to rule-based modeling languages. Languages like Kappa [34,35] or BNGL (as part of BioNetGen) [36,37], for example, are based on graph rewriting rules. Their focus is on modeling interacting entities between

which bonds are created and destroyed. ~~These bonds~~ Bonds are defined for exactly two entities, which precludes using them to express compartments. Static compartments can be integrated ~~into these approaches~~ [15, 38]. They are used to restrict the scope within which a reaction takes place. Shuttling of simple (or bound) molecules between static compartments as they are commonly supported also by the modeling standards [16], can be easily expressed, e.g., by $Protein@Cytosol \rightarrow Protein@Nucleus$ in BioNetgen. However, here, neither the cytosol nor the nucleus forms a species and, as such, a potential reactant or product of a rule or reaction.

Dynamic compartments can be expressed using graph rewriting when extending the formalism with hyperedges. Hyperedges are those that can connect to more than two vertices. This approach is used in the modeling formalism React(C) [3]. Here, a compartment can be denoted by a hyperedge, which can join any number of entities. In addition, the compartment itself can be represented as ~~an entity~~a species. The following rule describes a cell containing a nucleus and a protein ~~. The protein~~ that shuttles into the nucleus (Figure 1c):

$$Cell(c) + Nucleus(n, c) + Protein(c) \rightarrow Cell(c) + Nucleus(n, c) + Protein(n) \quad .$$

Note that the containment relation is expressed purely via variables c and n, identifying the cell and nucleus as a compartment (hyperedge). So the values of the variables c and n are identifiers for the respective compartments and as such unique. Entities not affected by the rule can be omitted on both rule sides, Therefore, we could have omitted $Cell(c)$. However, this would have made it harder to understand the role of the value of variable c. ~~It should be noted that shuttling of simple (or bound) molecules between compartments is commonly supported by systems biology modeling and simulation tools, e.g., in BioNetgen [15],~~

$$Protein@Cytosol \rightarrow Protein@Nucleus$$

~~and the corresponding standards [16]. However, here, neither the cytosol nor the nucleus forms a species and, as such, a potential reactant or product of a rule or reaction.~~

In the approaches based on hypergraphs, new compartments can be created by introducing new, unused ("fresh") values for variables on the right side. Similar solutions to model dynamic compartments can be found in process algebras, e.g., the attributed PI calculus [39]. The following rule expresses the fission of an organelle, with the ν operator yielding fresh values (identifiers) for the two new cells, i.e., c_1 and c_2:

$$Cell(c) \rightarrow (\nu c_1, c_2) Cell(c_1) + Cell(c_2)$$

However, this cell fission example also shows a fundamental problem with the hypergraph approach: ~~The~~ the effects on the contained entities can not be modeled as easily as the effect on the compartments. In the above reaction, all entities referencing the value of c as their containing compartment need to be updated to either c_1 or c_2. Also the fusion of compartments

$$Cell(c_1) + Cell(c_2) \rightarrow (\nu c) Cell(c)$$

would require that all entities referencing the values c_1 and c_2 would now need to reference c. These changes can be expressed with workarounds, for example, by defining intermediate states and additional rules for each species contained within the original compartment, with infinite propensities, but this clutters the model description and is computationally rather expensive. An additional problem with the extension of graph rewriting to hypergraphs is that finding occurrences of the left rule side in the current

state is harder. Whereas graph rewriting can exploit some properties (e.g., Kappa exploits "rigidity" [40]) to speed up this process, similar optimizations for hypergraphs are not known. As a consequence, no simulation tool based on hypergraph rewriting has been published. In particular, no simulator for React(C) is available.

An alternative approach to graph rewriting is multiset rewriting, a special case of term rewriting [41,42]. Here, the ~~hypergraphs representing~~ containment in compartments are essentially n-ary trees (see also Figure 1). Therefore, they can be expressed as terms with a variadic function symbol or, equivalently, an associative and commutative function symbol for forming multisets [43,44]. The contents of a compartmental entity can then be represented by a multiset, usually denoted with the symbol + or , in infix notation. The protein movement rule above (Figure 1c) can be written as

$$Cell(Nucleus(?n) + Protein{+}?c) \to Cell(Nucleus(Protein{+}?n){+}?c) \ ,$$

where $?c$ captures the remaining multiset within the cell after finding a nucleus and a protein in a cell, and $?n$ captures the contents of the nucleus. These variables starting with ? (called "sequence variables" in the rewriting literature [45]) play a central role, as they denote the entities unaffected by a rule and allow operations on those multisets of entities, i.e., the content of the compartments [46]. One line of work that very closely follows the idea of multiset rewriting is Colored Stochastic Multilevel Multiset Rewriting (CSMMR) [4,47]; another is ML-Rules ~~[8]~~ [5,8].

In multi-set rewriting, the cell fission rule could be written as

$$C(?c) \to C(?c_1) + C(?c_2) \ \textbf{where} \ (?c_1, ?c_2) = \mathit{splitHalf}(?c) \ ,$$

binding the result of the function *splitHalf* to a pair of variables in a **where** block [46]. In the syntax of CSMMR, the rule would read as

$$C(x) \to let(z, y) = x \ in \ C(z) + C(y) \ .$$

In the syntax of ML-Rules 3, the operator "+" is overloaded to realize an inline function on cell content, which randomly assigns each element of C to bind either to the sequence variable $?c_1$ or $?c_2$, i.e.,

$$C(?c_1{+}?c_2) \to C(?c_1) + C(?c_2)$$

(see Figure 1a). In some biological processes, fission events occur asymmetrically. For example, such asymmetrical fission events are observed in mitochondria. The small mitochondria might include many damaged parts and can be removed from the cell, thereby contributing to the health of the mitochondrial network. To model this process weights can be assigned to the sequence variables

$$Mito(?m_1[R_1]{+}?m_2[R_2]) \to Mito(?m_1) + Mito(?m_2)$$

where $R_1/(R_1 + R_2)$ and $R_2/(R_1 + R_2)$ are the ratios of elements assigned to $?m_1$ and $?m_2$, respectively.

While multi-set rewriting provides an expressive and readable way to model dynamic compartments (see Figure 1), languages that are built on multi-set rewriting, such as ML-Rules~~are~~ , are still hard to execute efficiently precisely because of their expressive power. One approach to alleviate this is to provide specialized simulators that exploit that some language features are not used in a given model [8], thus working on a subclass of models. Another subclass is considered in the modeling approach presented in [17]: it only considers models where compartments are not nested, and entities do not exist outside compartments.

2 Simulation engine

We implemented version 3 of ML-Rules using the Rust programming language to develop and test an efficient simulator for dynamic compartments. The model is specified within an external domain-specific language which is parsed into Rust code. The structure of the implementation and its components is shown in Figure 2. We have tuned the implementation for performance on the main expected code paths. That means we specifically applied several optimizations to the typical main loop at the potential cost of the unusual behavior. Overall optimizations include minimizing allocations, a flat, indexed-based data layout, and partial evaluation of repeated computations. The following sections will discuss differences from the previous versions and their implications on runtime performance and user experience.

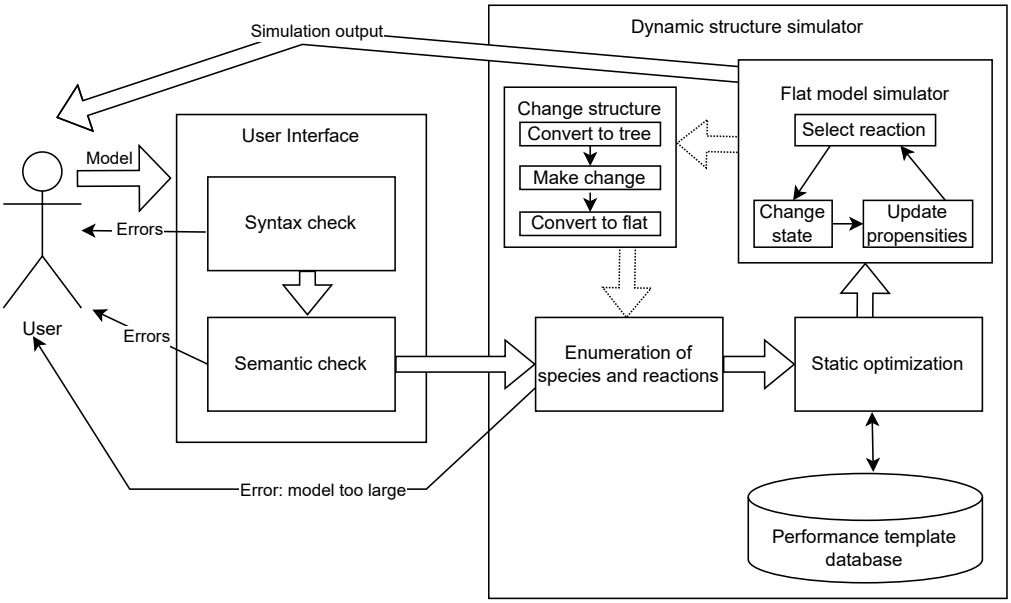


Fig 2. This Figure shows the flow of information between components when running a simulation. The user specifies the model, which is checked for syntactic and semantic consistency (like units or variable names). This is done through interaction with the web editor or command line interface. If these checks fail, errors are returned. Otherwise, the simulation loop starts by enumerating all possible species and reactions within the system into a flat representation. This can lead to errors if the system is too large. This flat representation is then optimized and put into the flat model (non-compartmental) simulator. Whenever a dynamic structural change is needed, the flattened model is transformed into a hierarchical, compartmental representation to execute those changes, and the loop starts again with the enumeration.

Compared to the previous implementation, we introduced a few changes to the syntax of ML-Rules (Section 2.1). However, the focus of our research has been to develop a simulation approach that is able to handle models that may exhibit a wide range of compartmental dynamics efficiently. It incorporates a new variant of an SSA specifically designed for large SSA systems (Section 2.3). This variant of the SSA (labeled flat model simulator in Figure 2) is not specific to compartmental models, as compartmentalization is abstracted away in a previous step. One significant optimization is code generation for repeated expression evaluation 2.4. We also incorporated a hybrid rule-based simulation method, allowing individual attributes to be simulated in a network-free manner (Section 2.1). Finally, through the use of recent

advances in web technology (namely WebAssembly), we can provide a simple web editor that enables fast local execution using the same codebase (Section 2.5).

2.1 Language

ML-Rules 3 builds on and adapts the language ML-Rules for rule-based multi-level modeling of cell biological systems [5] and its formal semantics [48]. The model is comprised of

- constants that can be used as input parameters for simulation experiments,
- definitions of functions for simplification of repeated notations,
- the initial model state,
- rewriting rules, that can also change the structure of compartments and may use complex expressions for the rates and attribute values, and
- a definition of potential outputs of the model.

The rewriting rules have the form

~~<left> -> <right> @ <rate>~~

~~.~~:

```
<left> -> <right> @ <rate>
```

We have a left side transformed (using the arrow) into the right side at a rate (following the @ symbol). The {} denote nesting relationships.

In the following, we describe some key aspects of ML-Rules and some slight changes compared to earlier implementations. The syntactic enhancements (with the exception of the network-free attributes) were not intended to improve the simulation performance but modeling in ML-Rules.

A core aspect of the language design is the nature of the patterns on the rules' left side. Modeling languages based on graph rewriting, such as Kappa or BNGL, employ graph patterns. Modeling languages based on multiset rewriting, such as CSMMR or ML-Rules, employ term patterns whose matching can be considered a specific case of unification [49]. The matching relates to compartmental structures and attributes [5, 8]. Attributes can be addressed either by position (structural pattern matching) or by name.

For a species

~~S(att_1: int, att_2: String)~~

```
S(att_1: int, att_2: String)
```

the previous ML-Rules versions required to ~~enlist~~ list all attributes of a species:

~~S(a1,a2) -> S(a1 + 1, a2)~~

~~,~~

```
S(a1,a2) -> S(a1 + 1, a2)
```

as attributes were identified by their position. Instead, we now have named attributes, where, in the case above, only the changed attribute needs to be listed:

~~S() -> S(a1=S.a1 + 1)~~

.

```
S() -> S(a1=S.a1 + 1)
```

From a formal perspective, rule-based approaches implicitly match omitted attributes. For example, given a cell species that has two attributes denoting its phase and volume, the rule

$$Cell(phase == 1) \to Cell(phase := 2)$$

```
Cell(phase == 1) -> Cell(phase = 2)
```

would be implicitly extended to

$$Cell(phase == 1, volume == v) \to Cell(phase := 2, volume := v)$$

```
Cell(phase == 1, volume == v) -> Cell(phase = 2, volume = v)
```

to express that the volume does not change. Named attributes have been used in other tools like BioNetGen [37] or Chromar [50].

The use of named attributes is obviously particularly useful if species have many attributes. For example, a simulation model of the Baltic Cod, which was developed in a previous version of ML-Rules, relied on structural pattern matching [51], and is available at `http://github.com/Baltic-Cod/EBC_IBM/blob/main/Basic_asph/Basic_asphyx.mlrj`, has been rewritten using ML-rules 3 to exploit the more efficient execution. This also resulted in a more succinct representation due to the named attributes `http://mlrules.pages.dev/gm/4/day`.

Typically, the simulation engine for rule-based models transforms rules into reaction networks by enumerating all possible values of attributes in advance to speed up the actual simulation [52]. Attributes that may assume many different and potentially unbound possible values are a challenge. In the case of unbound continuous attribute values, e.g., if the size of a lipid raft changes depending on the number of membrane proteins being aggregated within the raft or due to merging [53], this *in-advance-enumeration* becomes impossible. Even finite, categorical values can lead to a combinatoric explosion in the number of reactions and thus increase the reaction network size beyond tractable limits. This results in a *model too large* error in Figure 2. We have introduced specific types for this kind of attribute called network-free-integer and network-free-continuous. If an attribute is defined as one of these types, its values will not be explicitly enumerated before simulation execution (to generate the reaction network). For simulation, the attribute of type network free is equipped with a vector that stores the currently existing attribute values when a reaction fires. Once a rule fires, the species is instantiated with appropriate values. The approach is similar to the network-free simulation in BioNetGen [54] or CSMMR [4], particularly to the hybrid approach introduced in [55] that combines network-free and network-based calculations. However, instead of specifying entire species as network-free species, the modeler declares individual attributes to be network-free, and the simulator handles only those attributes or combinations as network-free. This distinction between being executed as network-based or network-free applies only to attributes of species that do not form a compartment. All compartments are handled as individual entities, as compartments are typically characterized by an attribute of continuous type, i.e., the volume, and they may contain an arbitrary number of diverse species (which again might be attributed), so one compartment's state is very likely different from the next and is (including its attributes' values) treated individually.

Other adaptations compared to earlier ML-Rule implementations are motivated by further increasing readability. In the previous ML-Rules versions, only equality constraints on the attribute values could be expressed on the left side of the rule. For example,

~~`Cell(vol, "M") -> ... if vol > 100 than k1 else 0`~~

```
Cell(vol, "M") -> ... if vol > 100 than k1 else 0
```

can be expressed in ML-Rules 3 as

~~`Cell(vol > 100, phase == "M") -> ... @ k1`~~

```
Cell(vol > 100, phase == "M") -> ... @ k1
```

Further adaptations aim to provide further support to define correct models. Similarly, as in other tools [56], modelers are now asked to assign units of measurement to numerical variables and constants, like micrometer, 1/second, and liter. The simulator performs the proper conversion and automatic type checking (Figure 2). This means a meter is correctly added to centimeters but not seconds. In addition, we introduced an enum type, allowing the modeler to constrain the admissible string values for a specific attribute to a specific set. Type inference, as well as constraint checks, occur ahead of simulation execution.

2.2 Efficient handling of dynamic structure reactions

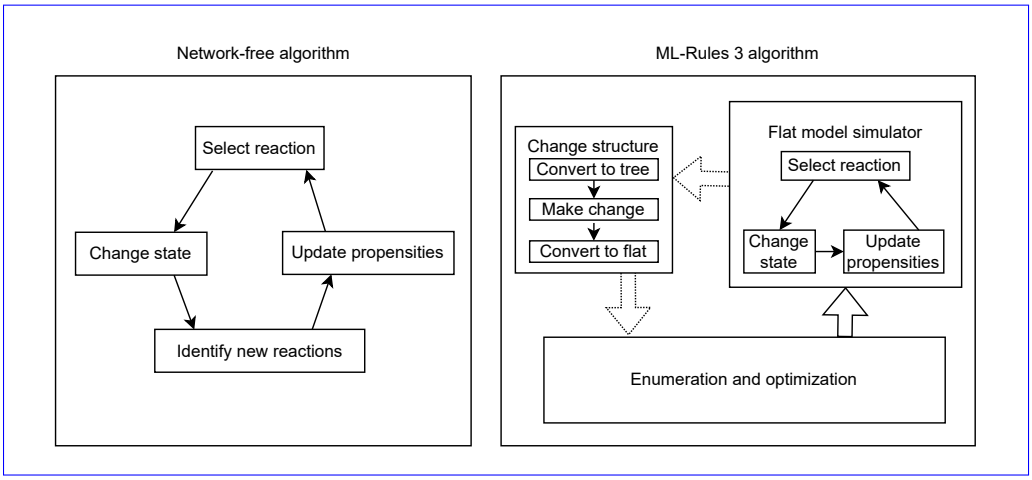


Fig 3. Comparison of the dynamic structure approaches. On the left, we have the previous network-free method. It is simpler to implement, but due to the dynamic structure, each step is significantly more costly in terms of performance. For example, neighborhood relations are dynamic, and therefore, the nesting tree structure needs to be traversed. Our approach (a subset of Figure 2) is on the right. We can utilize a faster flat (i.e., static) model simulator and have a costly but amortizing transformation and optimization process on the rare occasion of a structural change. For example, all relations can be encoded by static indices.

As discussed in Sections 1.1 and 1.3, dynamic changes in the system's structure are core to the ML-Rules modeling language. Their efficient execution has been a challenge in the past. Most simulation tools assume static compartmental structures, and their

simulators have been optimized accordingly [35,37,56]. ~~To integrate~~ Integrating compartmental dynamics into these simulators without significant loss of efficiency has been identified as a daunting challenge [57].

~~To~~ One option to solve the problem of dynamic structural changes ~~, one option~~ is to write a fully dynamic simulator~~, which~~. This simulator traverses the tree structure of the model after each reaction execution to instantiate new reactions and re-calculate propensities on demand~~, and~~. It also checks whether the dependency graph requires any updates. This is how a previous implementation of ML-Rules was realized [8] and what is shown in the left panel of Figure 3. However, compared to methods optimized for static networks [52], this has a significant overhead.

For ML-Rules 3, we implemented a new hybrid simulator approach shown on the right panel of Figure 3. During an additional analysis step, every reaction is analyzed and marked if its execution would result in a structural change. In the models we encountered, most reactions do not induce a structural change~~, these~~. These reactions are called regular reactions. A similar distinction between regular and complex reactions has been made when simulating part of the system using deterministic numerical integration methods [58] or by tau-leaping [59].

The primary mode of our simulator is running an SSA (Section 2.3) that executes only regular (non-dynamic) reactions. The model representation has been flattened, enabling static index-based access to the model's state, i.e., every species' amount is stored in an array. Hierarchical relations are only preserved to the point where they are needed to reverse the system representation to a tree form. But they are neither accessed nor needed during the simulation as long as no transformation to a tree form is required. Therefore, we call this the *Flat model simulator* in ~~Figure 2 .~~ Figures 2 and 3. When we encounter a structure-changing reaction, the model is transformed back into the tree representation reflecting the compartmental structures. The reaction is applied to this structural representation of the model. The resulting structured state is then transformed back into a flat representation, and processing is continued. This circular process is shown on the right side of Figure ~~2~~3. This Figure shows the inner loop within the flat model simulator and the loop involving the dynamic structure changes. Every dynamic structure change requires an entire rebuild of the simulator, including re-enumerating all potential reactions and a do-over of the static optimization phase. Especially for larger systems, this can be relatively costly; however, as long as dynamics structural changes are rare, the costs amortize. We will analyze this in the evaluation section 3.

2.3 Stochastic Simulation Algorithm for static compartments

Based on our experience with different SSAs, we developed our variant, which is suited particularly well for typical problems in ML-Rules, where systems can be very large, e.g., for simulating multi-cellular models. Systems become particularly large when identical reactions are replicated across many different compartments. This is the *Flat model simulator* in Figure 2. The implementation is based on storing the propensities in a binary tree with cumulative sum tracking [60]. Each node in the tree corresponds to one potential reaction and stores two values: The propensity of that reaction and the sum of all propensities of its child nodes. This allows for updates of the total propensity sum with logarithmic complexity changing a single propensity. Furthermore, reaction selection (a weighted random choice based on the reaction propensities) can also be completed in logarithmic time complexity. Similar approaches are used in other algorithms like the logarithmic direct method [61] or the simulator implementation for $S\pi@$ [62].

An additional advantage of the tree-based approach is its improved numerical stability due to the fewer operations needed to update the propensities. Another way to

minimize total propensity calculation time is by keeping track of the total propensity sum and only adding and subtracting the changes to reaction propensities after reaction execution as done by the Optimized Direct Method [31]. However, here, numerical errors from floating-point addition and subtraction accumulate over time and need to be dealt with. In our tree-based approach, numerical rounding errors only originate from the single summation, similar (arguably even better) to a single linear summation approach. The numerical error does not accumulate over time. It is independent of the number of steps and depends only on the number of reactions.

After a reaction has fired, multiple propensities in the tree must be updated. In principle, each update in the tree could be done individually based on the dependency graph via updating the cumulative sum until we reach the root. Instead, we roll these changes out in two phases. First, the node values are changed based on the dependency graph. Afterwards, the cumulative sums are updated as needed. If one were to update all sums individually, some nodes near the root might perform repeated summation updates. What total summation operations need to be done for each reaction is calculated ahead of time in the static optimization component (Figure 2). This computation has been accelerated via the use of bitsets.

The generation of the dependency graph and the propensity tree must be efficient, as it needs to be redone after every non-regular reaction.

Reaction selection is based on a weighted random choice based on the reaction propensities stored in the tree. The performance of reaction selection is improved by sorting the more likely reactions towards the root of the tree.

2.4 Performance Templates

ML-Rules is an external domain-specific language. This means that all expressions are written in a custom text format. This format is parsed by the simulation tool. One of the main downsides of using an external domain-specific language is the significant computational overhead in the repeated evaluation of mathematical expressions. Generally, the more expressive and generic a domain-specific language is, the harder it is to execute efficiently. In a more constrained language, code and algorithmic variations that are tailored to the specific requirements of the application and hardware architecture are easier to achieve. However, a domain-specific language is typically more useful if it is more generic and expressive.

In [8], we developed specialized simulators for specific sub-classes of ML-Rules models, e.g., those that do not exploit compartmental dynamics. In [63], we developed an approach generating an entire simulator optimized for a specific model defined in a rule-based language, such as BioNetGen [36]. After parsing the model, custom C or Rust code was generated. This code was then compiled and optimized using existing compiler software, resulting in a high throughput performance. This technique (called *partial evaluation* or *Futamura projection*) [64, 65] is an established technique to deal with the problem that genericity does not go along well with achieving performance.

The central idea of partial evaluation is that a function with multiple inputs can be reduced to a simpler version if some of the inputs are known. For example, the power function $f(base, exponent)$ may be simplified to $f(base, 2) = base \cdot base$ for the case of a known exponent.

Adopting the approach developed in [63] for dynamic compartments appears impractical. Its performance gain relied to a large degree on optimizing the updates of the dependency network. With dynamic compartments, these reaction networks change during execution; thus, optimization (recompilation) would need to be repeated after each structural change. However, the (re-)compilation induces a significant overhead. Especially for large models (which is typical for ML-Rules models), the additional compilation steps on every model run increase runtime significantly, independently of

model execution duration. Therefore, we developed an approach that combines dynamic interpretation and partial evaluation and does not require repeated compilation on every change. Our approach focuses on the rate evaluation (and thus propensity updates). This is costly for most dynamic compartmental models, as rate expressions involving dynamic compartments are typically complex. The complex expressions result in large abstract syntax trees (AST) that need to be parsed during execution. When using an interpreted or reflection-capable language like Java, code generation for these expressions can be introduced relatively simply [66]. For compiled languages such as Rust, a different approach is required. Our approach developed for ML-Rules 3 generates performance templates during the simulation and stores them for later reuse. Every time the simulator encounters a new AST, it checks whether generated code in the form of a performance template is available to replace this tree. Each performance template presents a partially evaluated AST and can be parameterized with numerical values (like model-specific constants) and indices (as used to identify involved species). If such an optimized previous version is found, it is used instead of the AST. If not, the simulator dynamically interprets the expression (by evaluating the AST). It generates an optimized code for future use and stores the corresponding performance template for later reuse. With every new model developed or recompilation of the model, the simulator can reuse previously created templates.

2.5 Web editor

The software tool is primarily designed as a command line tool. However, as a proof of concept, we also built a web-based version. The idea to run simulations on the web is not new and has been around almost as long as the web itself [67]. There have also been attempts at running stochastic simulations of biochemical models using a web interface [68]. ~~However, efficient simulation was usually delegated to a server in the background or the cloud.~~

However, the execution has been typically delegated to a server in the background or the cloud to enable an efficient simulation of models [69]. WebAssembly is a binary format recently developed that enables fast execution in web browsers [70]. WebAssembly is executed on a stack-based virtual machine similar to other native code. Additionally, it interoperates well with JavaScript. WebAssembly allows reasonable performance for simulation without installation in a web browser or major changes to the underlying simulator code [71].

As the Rust language can be compiled into WebAssembly, we can use the same code base for the simulator and only need to add a small frontend. We used the Rust Yew framework [1] for this. A two-panel user interface is provided at `http://mlrules.pages.dev`, which includes a code editor with some syntax highlighting based on the Monaco editor from Visual Studio Code and another panel to display simulation results or any errors and instructions from the simulator.

A significant advantage of this WebAssembly approach is that there are next to no host costs. The simulation is executed on the end user's machine. The web page is static and has a size of roughly 10 Megabytes, which is very cheap or free to host compared to other approaches that run the simulation on the server. We also added the possibility to link to specific models. We use this in our case studies to provide links to the various models, which facilitates the reproduction of results and allows users to change and adapt models and conduct basic experiments with no additional setup. As done throughout this paper, linking to an executable model with a custom URL is possible.

[1] `http://yew.rs/`

3 Evaluation

In this section, cell biological simulation models showcase the capabilities and performance of ML-Rules 3. All performance experiments were conducted using an intel i9-13900K CPU, running Rust 1.74. WebAssembly was run in Firefox version 120 using its spidermonkey engine.

3.1 Fission Yeast Model

The first case study is based on a multi-cellular model of fission yeast, including cell division and mating type switching depending on intracellular dynamics. The model is available at `http://mlrules.pages.dev/yeast/300/min`. The model (see Figure 4) has been adopted from the original ML-Rules publication [5] to show the expressiveness of ML-Rules. The fission yeast model includes an early cell cycle model [2].

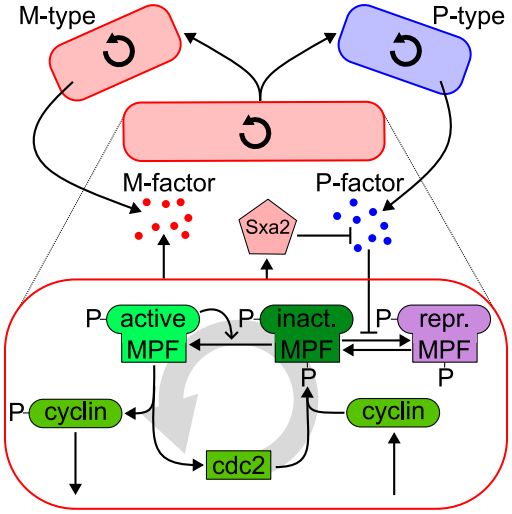


Fig 4. Fission Yeast Model. Inside the cell, proteins oscillate. Outside the cell, Sxa2 and pheromones (P and M-type) can inhibit the cell cycle of cells of the opposite mating type. Triggered by the cell cycle, a yeast cell can fission into two daughter cells.

The cell cycle model consists of two proteins (cyclin and cdc2) that can form the maturation-promoting factor (MPF). MPF exists in three versions (active, inactive, and repressed), which oscillate with a cycle duration of approximately 200 minutes (see Figure 4 left). This oscillation triggers the cell to change between its phases (G1, SG2, and M), and eventually, a spike in inactive MPF causes cells in its M-phase to divide into two daughter cells. In addition, cells are characterized by a mating type (P or M) that might differ in one daughter cell from the type of the mother cell. Based on the mating type, cells produce and secrete pheromones (M- or P-factor) to inhibit the cell cycle of the opposite mating type cells by changing MPF from its inactive to the repressed variant. M-type cells also release P-factor-specific protease (Sxa2) that inhibits the P-factor pheromone.

This model relies on a unique combination of ML-Rules features and its expressiveness. Individual cells are defined as compartments that frequently divide. Compartments and proteins are equipped with attributes, e.g., to denote the cell cycle phase, the cyclin's phosphorylation state, or the cell's volume. The cells secrete pheromones into the extracellular environment. They influence the cell cycle of other cells of the opposite mating type. The kinetics depend on rate factors that require

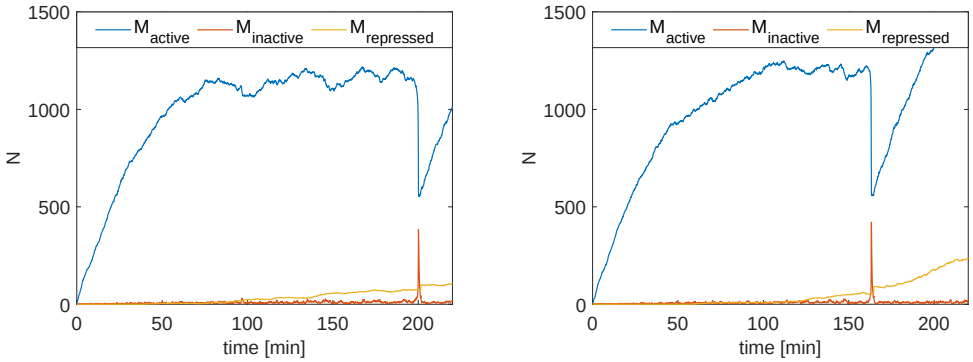


Fig 5. Two sample replications of the fission yeast model show the oscillation of active, inactive, and restricted MPF. At 200 minutes and 163 minutes, respectively, a fission event is triggered by the spike of inactive MPF.

simulator	runtime [s]	throughput [1000/s]
ML-Rules 2	12.4	40.8
ML-Rules 3	0.124 (0.190)	4190 (2690)
ML-Rules 3 - network free	0.321 (0.352)	1590 (1440)
ML-Rules 3 - w/o templates	0.259 (0.331)	1950 (1530)

Table 1. Run time and reactions per second for different simulators. The fission yeast model is executed until 1000 minutes (20 replications). The values in parenthesis refer to executing ML-Rules 3 as WebAssembly code. The ML-Rules 2 implementation of the model contains ten species and 20 rules. The ML-Rules 3 version of the model consists of 7 species and 20 rules. The model starts with two cell compartments and ends with about 12. During the simulation time, the model undergoes about 10 structural changes.

complex expressions, e.g., a Hill-type sigmoidal response curve defines the MPF repression (depending on the number of pheromones).

We have used this model as a performance benchmark. The reaction throughput rates for a run until 1000 minutes (simulation time) are shown in table 1. ML-Rules 3 is significantly faster than the previous Java implementation, ML-Rules 2. If performance templates are used, we observe a roughly 2-order speedup. Even without the templates, the performance is 26 to 18 times faster. The 18x speedup is observed when enabling network-free attributes (see Section 2.1). The model does not use network-free attributes; even only enabling this capability introduces more branching in the critical code and slows down execution. We also tested the WebAssembly version. The data for this plot can be generated locally by visiting `http://mlrules.pages.dev/benchmark/yeast/3/1000`. The first number is the number of replications, and the second is the longest test duration in minutes of simulation time. The WebAssembly has a performance penalty of only about 10% to 50%.

3.2 mRNA Delivery Model

Our second case study is centered around the delivery of mRNA into cells. Understanding how to deliver mRNA into cells is crucial for its use as a drug or vaccine [23]. Ligon et al. [72] published a simulation model capable of simulating the mRNA delivery based on lipoplexes (small lipid spheres containing mRNA).

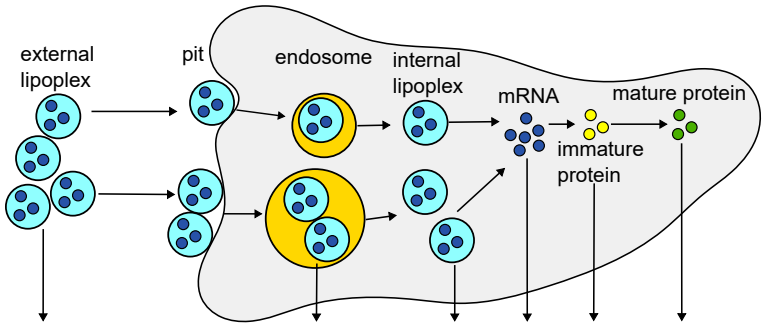


Fig 6. Sketch of the mRNA delivery model. Lipoplexes are present in the cell's environment and can accumulate in pits in the cell membrane. The lipoplexes in the pit can enter the cell via endocytosis, and the endosome can lyse to release the lipoplexes into the cell. Once they unpack their mRNA, the mRNA gets translated to proteins. The arrows that point down indicate the degradation of the species.

In their model, shown in Figure 6, lipoplexes are present within the extra-cellular environment for an hour before being removed by an event mimicking the cell's washing. During this time, clathrin-coated pits containing one or more lipoplexes form at the cell's surface. Via the invagination of the plasma membrane, endosomes are formed, and the lipoplexes enter the cell. Inside the cell, the endosome releases the lipoplexes, which then unpack their mRNA. Afterward, the mRNA can be translated into proteins.

With their model, the authors could gain insight into mRNA delivery by lipoplexes and the dose-response relationship. However, they also state a few simplifications and workarounds needed in their model, as the used modeling and simulation method only supported static compartments. One simplification is that all lipoplexes carry the same amount of mRNA in the model. In wet lab experiments, this number has been found to vary due to the different sizes of the lipoplexes [72, 73].

One possibility to model lipoplexes more realistically is to model each lipoplex as a compartment that contains the mRNA and enters the cell (which is also represented as a compartment). For this, support of dynamic compartments is required. Another possibility is to model the amount of mRNA a lipoplex contains as a specific attribute of type integer. Both solutions are possible in ML-Rules but were not possible in the tool(s) used by the authors. This led to the simplification of assuming a fixed number of mRNA per lipoplex (i.e., 350) in the model. As a result, the simulation of the protein expression in the original model shows narrow bands, depending on the number of lipoplexes that could enter the cell and unpack their mRNA (see Figure 7 A), which is not the case in the wet-lab data [73]. The authors tried to use the modeling and simulation tool SPim [74], which is based on the stochastic π calculus [75] and allows to assign an attribute to a ~~lipoplexe~~ lipoplex that states how much mRNA it carries and unpacks this amount into the cell. However, they ran into performance issues, which made it impossible to use SPim for their study (see supplement TextS001 p. 7 from [72]).

As stated above in ML-Rules, this simplification is unnecessary, and a varying number of mRNA can be assigned to the lipoplexes modeled as compartments. The modified model can be found under `http://mlrules.pages.dev/lipoplex_ext/30/h`. At the beginning of the simulation, the simulation state is set to contain one cell and 200 lipoplexes containing mRNA. The amount of mRNA per lipoplex (L) is calculated based on the lipoplex size sampled from a normal distribution. The lipoplex can then move into the cell, like in the original model. Inside the cell, the lipoplex compartment L can unpack its content (?solL).

~~Cell{L{?solL}+?solC} -> Cell{?solL + ?solC} @ kU;~~

```
Cell{L{?solL}+?solC} -> Cell{?solL + ?solC} @ kU;
```

The ?solL denotes the content of the lipoplex, here a population of mRNA, and ?solC is the cell's content, including other lipoplexes, mRNA, and proteins. Alternatively, we could have modeled the mRNA as an attribute of type integer. In such an implementation, the lipoplex would not be a compartment carrying the mRNA but a simple species equipped with an attribute (NmRNA) that denotes the amount of mRNA inside. The unpack rule shown above would change:

~~Cell{L+?solC} -> Cell{(L.NmRNA) mRNA + ?solC} @ kU;~~

```
Cell{L+?solC} -> Cell{(L.NmRNA) mRNA + ?solC} @ kU;
```

A second minor simplification is that lipoplexes can unpack their mRNA not only in the cell but also in the endosome, where the unpacked mRNA can start to deteriorate. This mechanism, called the "fully nested transfection model" by the authors (see Ligon et al. [72] Figure 8), is missing in the original model.

In the ML-Rules 3 model, it is realized by the two reactions:

~~Cell{E{L{?solL} + ?solE}+?solC} -> Cell{E{?solL + ?solE} + ?solC} @ kU;~~

for the unpacking inside the endosome, and

~~E{mRNA:m + ?solE} -> E{?solE} @ dM*#m;~~

```
E{mRNA:m + ?solE} -> E{?solE} @ dM*#m;
```

for the degradation of mRNA in the endosome.

Besides the simplification in the original model, some reactions, like the formation of pits, are lengthy to write down. Due to the lack of attributes or dynamic compartments, the number of lipoplexes that reside in a pit is stated in the name of the species. The original model uses ten pit species (P1 - P10) denoting 1 to 10 lipoplexes inside the pit. Consequently, ten reactions need to be specified for all P_i regarding the formation of pits, the endocytosis, the lysis, and the degradation of pits. This workaround only works for a low number of P-species. The growing number of reactions makes writing the model down for larger numbers increasingly harder. In ML-Rules 3, the dynamic nesting allows us to write down reactions more compactly. For example, the ten endocytosis reactions in the original model:

~~P1 -> E1~~
~~...~~
~~P10 -> E10~~

```
P1 -> E1
...
P10 -> E10
```

translate into a single rule in ML Rules 3.

~~Cell(?solC) + P{?solP} -> Cell{E{?solP}+?solC}~~

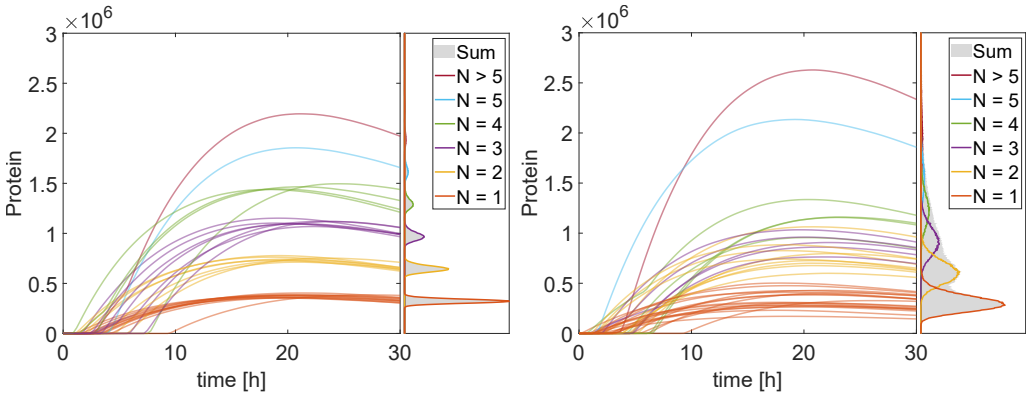


Fig 7. Protein amounts over time for the original (left) and modified (right) mRNA delivery model (N=100). The colors indicate the number of lipoplexes that unpack their mRNA in the cell. The histograms next to the time courses show the distribution of proteins after 30 hours without the cells that express no protein (N=1000000). Runs where no proteins are generated are not shown.

simulator	runtime [s]		
	20% quantile	average	80% quantile
Copasi	0.13	4.8	11.2
ML-Rules 3	0.59	0.6	0.84

Table 2. Run time for different simulators. The mRNA delivery model is executed until 30 hours (1000 replications). The ML-Rules 3 model is initialized with 201 compartments (1 cell and 200 lipoplexes) and ends with one compartment. On average, about 207 structural changes are executed during a simulation run.

```
Cell(?solC) + P{?solP} -> Cell{E{?solP}+?solC}
```

By applying the abovementioned changes, the model produces more realistic results, as can be seen by the protein expression over time in Figure 7. As stated above, the narrow bands as observed in the original model (Figure 7 A) are not observed in the wet lab experiments [72, 73]. By varying the mRNA number inside the lipoplexes, these bands in the protein expression broaden and overlap (Figure 7 B), resulting in a more realistic model behavior.

Finally, we have a look at the model's runtime performance. Therefore, we compared the runtime for executing the original model written in COPASI (version 4.41) and its ML-Rules equivalent (available at `http://mlrules.pages.dev/lipoplex_orig`. The original COPASI model consists of 48 reactions and 26 species compared to the ML-Rules model with its 12 rules and 8 species. This is due to the manual unrolling of the underlying nesting processes and illustrates ML-Rules' expressiveness, which also results in succinct models. We measured the runtime for 1000 replications.

As the simulation model contains events to describe the removal of external lipoplexes, only the Direct Method implementation within COPASI is able to execute the entire simulation model. This is not a limitation of the method but of the implementations. Timed events can be added relatively easily to most methods as an implementation feature. The difference in methods should be considered when interpreting the runtime measurements. For larger systems, the direct methods used generally perform worse than a more optimized logarithmic method, like the next reaction method. We found the average runtime for a single execution to be 4.8 seconds. However, in two-thirds of the simulation runs, no lipoplex can unpack their mRNA into

the cell, and consequently, no proteins are created. The 20% quantile takes 0.13 seconds, and the 80% quantile is 11.2 seconds. The equivalent simplified model in ML-Rules takes only about 0.6 seconds on average but 0.59 seconds for 20% and 0.84 seconds for the 80% quantile, respectively. ~~This is an indication of the higher initialization cost but better throughput of the ML-Rules implementation.~~ Most of the runtime (87% on average) in the ML-Rules model is spent on ~~rebuilding the simulator after~~ a dynamic structure ~~reaction~~reactions, something that COPASI does not need to consider. We find similar runtimes for the adapted version of the model in ML-Rules.

4 Conclusion

We built ML-Rules version 3, making concise formulations of dynamic structure models run with high performance. A performance-oriented implementation of various data structures, including a partial summation tree, made this possible. We also made some changes to the language, like introducing named attributes and units of measurement. We found that this implementation outperforms the previous version of ML-Rules by two orders of magnitude. The evaluation of the simulator is based on two biological case studies. First, we used a fission yeast model that was also used in the original ML-Rules publication to show that we have a similar expressiveness but a higher performance than the previous implementation. Second, we rebuilt and extended an mRNA delivery model and showed how using dynamic compartments needs fewer simplifications than the original model that uses a static compartmental approach. The extended model matches the wet-lab data more closely, allowing a more compact notation and better performance. Finally, we built a prototypical web-based simulator using the same source code compiled to WebAssembly that can run locally on the end user machine. WebAssembly's ease of deployment and development using existing code bases and competitive performance are promising. We see further opportunities for simulation tool developers to make their software more accessible using this technology.

Our investigation has also raised some questions for future research. Currently, the network-free execution part of the model is determined by attributes explicitly defined as network-free. However, possibly a larger portion of the model could benefit from a network-free execution. To learn during simulation which part of the model to execute most efficiently in a network-based or network-free manner, possibly reinforcement learning approaches could be adapted [76].

The main simulator for ML-Rules 3 now conforms to standard SSA, with the potential exception of more complex rate expressions. It is easier to integrate with previous research on more advanced SSA variants like the partial propensity method [77] or approximate methods like advanced tau leaping [78]. Preliminary experiments with approximate tau leaping showed significant improvement for some models. However, frequently, the limiting factor is the repeated structural conversion. Moving forward, only a subset of the simulator could be transformed if changes to the dynamic structure are localized. It would also be possible to integrate the dynamic structure more closely with the simulator, at the cost of some performance for the regular transitions. The simulator is currently applied in three different simulation studies, i.e., studying the fission and fusion of mitochondria, bone remodeling processes, and the role of endocytosis in cellular signaling.

5 Acknowledgments

Figures 1 was created with BioRender.com. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) SFB 1270 – 299150580

ELAINE and grant 225222086.

References

1. Weng G, Bhalla US, Iyengar R. Complexity in biological signaling systems. Science. 1999;284(5411):92–96. doi:10.1126/science.284.5411.92.

2. Tyson JJ. Modeling the cell division cycle: cdc2 and cyclin interactions. Proceedings of the National Academy of Sciences. 1991;88(16):7328–7332. doi:10.1073/pnas.88.16.7328.

3. John M, Lhoussaine C, Niehren J, Versari C. Biochemical Reaction Rules with Constraints. In: Proceedings of the 20th European Symposium on Programming, ESOP 2011. Berlin, Heidelberg: Springer-Verlag; 2011. p. 338–357.

4. Oury N, Plotkin GD. Multi-level modelling via stochastic multi-level multiset rewriting. Mathematical Structures in Computer Science. 2013;23(2):471–503. doi:10.1017/s0960129512000199.

5. Maus C, Rybacki S, Uhrmacher AM. Rule-based multi-level modeling of cell biological systems. BMC Systems Biology. 2011;5(1):166. doi:10.1186/1752-0509-5-166.

6. Regev A, Panina EM, Silverman W, Cardelli L, Shapiro E. BioAmbients: an abstraction for biological compartments. Theoretical Computer Science. 2004;325(1):141–167.

7. Faeder JR. Toward a comprehensive language for biological systems. BMC biology. 2011;9(1):1–5.

8. Helms T, Warnke T, Maus C, Uhrmacher AM. Semantics and Efficient Simulation Algorithms of an Expressive Multi-Level Modeling Language. ACM Transactions on Modeling and Computer Simulation. 2017;27(2):8:1–8:25. doi:10.1145/2998499.

9. Segev N, Tokarev AA, Alfonso A, Segev N. Overview of intracellular compartments and trafficking pathways. Trafficking inside cells: pathways, mechanisms and regulation. 2009; p. 3–14.

10. Harris LA, Hogg JS, Faeder JR. Compartmental rule-based modeling of biochemical systems. In: Proceedings of the 2009 Winter Simulation Conference (WSC). IEEE; 2009. p. 908–919.

11. Liu J, Xiao Q, Xiao J, Niu C, Li Y, Zhang X, et al. Wnt/β-catenin signalling: function, biological mechanisms, and therapeutic opportunities. Signal transduction and targeted therapy. 2022;7(1):3.

12. Budde K, Smith J, Wilsdorf P, Haack F, Uhrmacher AM. Relating simulation studies by provenance–Developing a family of Wnt signaling models. PLoS Computational Biology. 2021;17(8):1–21.

13. MacLean AL, Rosen Z, Byrne HM, Harrington HA. Parameter-free methods distinguish Wnt pathway models and guide design of experiments. Proceedings of the National Academy of Sciences. 2015;112(9):2652–2657.

14. Haack F, Lemcke H, Ewald R, Rharass T, Uhrmacher AM. Spatio-temporal Model of Endogenous ROS and Raft-Dependent WNT/Beta-Catenin Signaling Driving Cell Fate Commitment in Human Neural Progenitor Cells. PLOS Computational Biology. 2015;11(3):1–28. doi:10.1371/journal.pcbi.1004106.

15. Harris L, S Hogg J, R Faeder J. Compartmental rule-based modeling of biochemical systems. In: Proceedings of the 2009 Winter Simulation Conference. WSC '09. Austin, Texas: IEEE; 2009.

16. Hucka M, Bergmann FT, Chaouiya C, Dräger A, Hoops S, Keating SM, et al. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 2 Core Release 2. Journal of Integrative Bioinformatics. 2019;16(2). doi:10.1515/jib-2019-0021.

17. Duso L, Zechner C. Stochastic reaction networks in dynamic compartment populations. 2020;117(37):22674–22683. doi:10.1073/pnas.2003734117.

18. Uhrmacher AM. Dynamic structures in modeling and simulation: a reflective approach. ACM Trans Model Comput Simul. 2001;11(2):206–232.

19. Kolch W, Halasz M, Granovskaya M, Kholodenko BN. The dynamic control of signal transduction networks in cancer cells. Nature Reviews Cancer. 2015;15(9):515–527.

20. Basson MA. Signaling in cell differentiation and morphogenesis. Cold Spring Harbor perspectives in biology. 2012;4(6):a008151.

21. Sorkin A, Von Zastrow M. Endocytosis and signalling: intertwining molecular networks. Nature reviews Molecular cell biology. 2009;10(9):609–622.

22. Huotari J, Helenius A. Endosome maturation. The EMBO journal. 2011;30(17):3481–3500.

23. Hou X, Zaks T, Langer R, Dong Y. Lipid nanoparticles for mRNA delivery. Nature Reviews Materials. 2021;6(12):1078–1094.

24. Parchekani J, Allahverdi A, Taghdir M, Naderi-Manesh H. Design and simulation of the liposomal model by using a coarse-grained molecular dynamics approach towards drug delivery goals. Scientific Reports. 2022;12(1):2371.

25. Westermann B. Mitochondrial fusion and fission in cell life and death. Nature Reviews Molecular Cell Biology. 2010;11(12):872–884. doi:10.1038/nrm3013.

26. Bozzuto G, Molinari A. Liposomes as nanomedical devices. International Journal of Nanomedicine. 2015;10:975–999. doi:10.2147/IJN.S68861.

27. V'kovski P, Kratzel A, Steiner S, Stalder H, Thiel V. Coronavirus biology and replication: implications for SARS-CoV-2. Nature Reviews Microbiology. 2021;19(3):155–170. doi:10.1038/s41579-020-00468-6.

28. Gillespie DT. Stochastic simulation of chemical kinetics. Annu Rev Phys Chem. 2007;58:35–55.

29. Anderson WJ. Continuous-Time Markov Chains: An Applications-Oriented Approach (Springer Series in Statistics). Springer; 1991.

30. Gillespie DT. Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry. 1977;81(25):2340–2361. doi:10.1021/j100540a008.

31. Cao Y, Li H, Petzold L. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. The Journal of Chemical Physics. 2004;121(9):4059–4067. doi:10.1063/1.1778376.

32. Gibson MA, Bruck J. Efficient exact stochastic simulation of chemical systems with many species and many channels. The journal of physical chemistry A. 2000;104(9):1876–1889.

33. Faeder JR, Blinov ML, Hlavacek WS. Rule-based modeling of biochemical systems with BioNetGen. Systems biology. 2009; p. 113–167.

34. Danos V, Laneve C. Formal molecular biology. Theoretical Computer Science. 2004;325(1):69–110. doi:10.1016/j.tcs.2004.03.065.

35. Boutillier P, Maasha M, Li X, Medina-Abarca HF, Krivine J, Feret J, et al. The Kappa platform for rule-based modeling. Bioinformatics. 2018;34(13):i583–i592.

36. Blinov ML, Faeder JR, Goldstein B, Hlavacek WS. BioNetGen: Software for Rule-based Modeling of Signal Transduction Based on the Interactions of Molecular Domains. Bioinformatics. 2004;20(17):3289–3291. doi:10.1093/bioinformatics/bth378.

37. Harris LA, Hogg JS, Tapia JJ, Sekar JAP, Gupta S, Korsunsky I, et al. BioNetGen 2.2: advances in rule-based modeling. Bioinformatics. 2016;32(21):3366–3368. doi:10.1093/bioinformatics/btw469.

38. Pedersen M, Phillips A, Plotkin GD. A High-Level Language for Rule-Based Modelling. PLOS ONE. 2015;10(6):1–26. doi:10.1371/journal.pone.0114296.

39. John M, Lhoussaine C, Niehren J, Uhrmacher AM. The attributed pi calculus. In: International Conference on Computational Methods in Systems Biology. Springer; 2008. p. 83–102.

40. Danos V, Feret J, Fontana W, Krivine J. Scalable Simulation of Cellular Signaling Networks. In: Programming Languages and Systems. Springer Berlin Heidelberg; 2007. p. 139–157.

41. Bistarelli S, Cervesato I, Lenzini G, Marangoni R, Martinelli F. On Representing Biological Systems through Multiset Rewriting. In: Computer Aided Systems Theory - EUROCAST 2003. Springer Berlin Heidelberg; 2003. p. 415–426.

42. Cavaliere M, Sedwards S. Modeling and Simulating Biological Processes with Stochastic Multiset Rewriting. In: Nicol DM, Priami C, Nielson HR, Uhrmacher AM, editors. Simulation and Verification of Dynamic Systems. No. 06161 in Dagstuhl Seminar Proceedings. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany; 2006.Available from: `http://drops.dagstuhl.de/opus/volltexte/2006/706`.

43. Eker S. Associative-Commutative Rewriting on Large Terms. In: Rewriting Techniques and Applications. Springer Berlin Heidelberg; 2003. p. 14–29.

44. Dundua B, Kutsia T, Marin M. Variadic equational matching in associative and commutative theories. Journal of Symbolic Computation. 2021;106:78–109. doi:10.1016/j.jsc.2021.01.001.

45. Marin M, Tepeneu D. Programming with sequence variables: The Sequentica package. In: Challenging The Boundaries Of Symbolic Computation: (With CD-ROM). World Scientific; 2003. p. 17–24.

46. Warnke T, Helms T, Uhrmacher AM. Syntax and semantics of a multi-level modeling language. In: Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation; 2015. p. 133–144.

47. Oury N, Plotkin GD. Coloured stochastic multilevel multiset rewriting. In: Proceedings of the 9th International Conference on Computational Methods in Systems Biology - CMSB 2011. ACM Press; 2011.

48. Warnke T, Helms T, Uhrmacher AM. Syntax and Semantics of a Multi-Level Modeling Language. In: Proceedings of the 2015 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. New York: ACM; 2015. p. 133–144.

49. Warnke T, Uhrmacher AM. Nonlinear pattern matching in rule-based modeling languages. In: Computational Methods in Systems Biology: 19th International Conference, CMSB 2021, Bordeaux, France, September 22–24, 2021, Proceedings 19. Springer; 2021. p. 198–214.

50. Honorato-Zimmer R, Millar AJ, Plotkin GD, Zardilis A. Chromar, a language of parameterised agents. Theoretical Computer Science. 2019;765:97–119. doi:10.1016/j.tcs.2017.07.034.

51. Pierce ME, Warnke T, Krumme U, Helms T, Hammer C, Uhrmacher AM. Developing and validating a multi-level ecological model of eastern Baltic cod (Gadus morhua) in the Bornholm Basin - a case for domain-specific languages. Ecological Modeling. 2017;361:49–65.

52. Gupta A, Mendes P. An Overview of Network-Based and -Free Approaches for Stochastic Simulation of Biochemical Systems. Computation. 2018;6(1). doi:10.3390/computation6010009.

53. Simons K, Sampaio JL. Membrane organization and lipid rafts. Cold Spring Harbor perspectives in biology. 2011;3(10):a004697.

54. Sneddon MW, Faeder JR, Emonet T. Efficient modeling, simulation and coarse-graining of biological complexity with NFsim. Nature methods. 2011;8(2):177–183.

55. Hogg JS, Harris LA, Stover LJ, Nair NS, Faeder JR. Exact hybrid particle/population simulation of rule-based models of biochemical systems. PLoS computational biology. 2014;10(4):e1003544.

56. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, et al. COPASI - a complex pathway simulator. Bioinformatics. 2006;22(24):3067–3074.

57. Thompson-Walsh CD, Hayman J, Winskel G. Containment in rule-based models. Electronic Notes in Theoretical Computer Science. 2012;284:125–137.

58. Helms T, Wilsdorf P, Uhrmacher AM. Hybrid simulation of dynamic reaction networks in multi-level models. In: Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation; 2018. p. 133–144.

59. Helms T, Luboschik M, Schumann H, Uhrmacher AM. An approximate execution of rule-based multi-level models. In: Computational Methods in Systems Biology: 11th International Conference, CMSB 2013, Klosterneuburg, Austria, September 22-24, 2013. Proceedings 11. Springer; 2013. p. 19–32.

60. Köster T, Uhrmacher AM. Handling Dynamic Sets of Reactions in Stochastic Simulation Algorithms. In: ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS 2018). New York, NY, USA: ACM; 2018. p. 161–164. Available from: http://eprints.mosi.informatik.uni-rostock.de/522/.

61. Li H, Petzold LR. Logarithmic Direct Method for Discrete Stochastic Simulation of Chemically Reacting Systems; 2006.Available from: https://api.semanticscholar.org/CorpusID:7936447.

62. Versari C, Busi N. Efficient Stochastic Simulation of Biological Systems with Multiple Variable Volumes. Electronic Notes in Theoretical Computer Science. 2008;194(3):165–180. doi:https://doi.org/10.1016/j.entcs.2007.12.012.

63. Köster T, Warnke T, Uhrmacher AM. Generating Fast Specialized Simulators for Stochastic Reaction Networks via Partial Evaluation. ACM Transactions on Modeling and Computer Simulation (TOMACS). 2022;32(2):1–25.

64. Futamura Y. Partial Evaluation of Computation Process - An Approach to a Compiler-Compiler. Higher Order Symbol Comput. 1999;12(4):381–391. doi:10.1023/A:1010095604496.

65. Leißa R, Boesche K, Hack S, Pérard-Gayot A, Membarth R, Slusallek P, et al. AnyDSL: A partial evaluation framework for programming high-performance libraries. Proceedings of the ACM on Programming Languages. 2018;2(OOPSLA):1–30.

66. Meyer T, Helms T, Warnke T, Uhrmacher AM. Runtime Code Generation for Interpreted Domain-Specific Modeling Languages. In: Winter Simulation Conference (WSC 2018). IEEE; 2018. p. 605–616. Available from: http://eprints.mosi.informatik.uni-rostock.de/534/.

67. Fishwick PA. Web-Based Simulation: Some Personal Observations. In: Proceedings of the 28th Conference on Winter Simulation. WSC '96. USA: IEEE Computer Society; 1996. p. 772–779. Available from: https://doi.org/10.1145/256562.256807.

68. Ivanov S, Rogojin V, Azimi S, Petre I. Webrsim: A web-based reaction systems simulator. Enjoying Natural Computing: Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday. 2018; p. 170–181.

69. Byrne J, Heavey C, Byrne PJ. A review of Web-based simulation and supporting tools. Simulation Modelling Practice and Theory. 2010;18(3):253–276. doi:https://doi.org/10.1016/j.simpat.2009.09.013.

70. Rossberg A. WebAssembly Core Specification;. Available from: https://www.w3.org/TR/wasm-core-1/.

71. Klemenschits X, Manstetten P, Filipovic L, Selberherr S. Process Simulation in the Browser: Porting ViennaTS using WebAssembly. In: 2019 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD); 2019. p. 1–4.

72. Ligon TS, Leonhardt C, Rädler JO. Multi-Level Kinetic Model of mRNA Delivery via Transfection of Lipoplexes. PLOS ONE. 2014;9(9):e107148. doi:10.1371/journal.pone.0107148.

73. Leonhardt C, Schwake G, Stögbauer TR, Rappl S, Kuhr JT, Ligon TS, et al. Single-cell mRNA transfection studies: Delivery, kinetics and statistics by numbers. Nanomedicine: Nanotechnology, Biology and Medicine. 2014;10(4):679–688. doi:10.1016/j.nano.2013.11.008.

74. Phillips A, Cardelli L. Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus. In: Calder M, Gilmore S, editors. Computational Methods in Systems Biology. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer; 2007. p. 184–199.

75. Priami C. Stochastic π-calculus. The Computer Journal. 1995;38(7):578–589.

76. Helms T, Ewald R, Rybacki S, Uhrmacher AM. Automatic runtime adaptation for component-based simulation algorithms. ACM Transactions on Modeling and Computer Simulation (TOMACS). 2015;26(1):1–24.

77. Ostrenko O, Incardona P, Ramaswamy R, Brusch L, Sbalzarini IF. pSSAlib: The partial-propensity stochastic chemical network simulator. PLOS Computational Biology. 2017;13(12):e1005865.

78. Cao Y, Gillespie DT, Petzold LR. Efficient step size selection for the tau-leaping simulation method. The Journal of chemical physics. 2006;124(4).