



HAL
open science

Annotations for Rule-Based Models

Matteo Cavaliere, Vincent Danos, Ricardo Honorato-Zimmer, William Waites

► **To cite this version:**

Matteo Cavaliere, Vincent Danos, Ricardo Honorato-Zimmer, William Waites. Annotations for Rule-Based Models. William S. Hlavacek. Modeling Biomolecular Site Dynamics, 1945, Springer New York, pp.271-296, 2019, Methods in Molecular Biology, 978-1-4939-9100-6. 10.1007/978-1-4939-9102-0_13 . hal-04336418

HAL Id: hal-04336418

<https://hal.science/hal-04336418>

Submitted on 15 Mar 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Annotations for Rule-Based Models

**Matteo Cavaliere, Vincent Danos, Ricardo Honorato-Zimmer
and William Waites**

All authors contributed equally.

Corresponding author:

William Waites

Laboratory for Foundations of Computer Science

School of Informatics

University of Edinburgh

Edinburgh, EH8 9LE, UK

Email: wwaites@inf.ed.ac.uk

This manuscript has been prepared for inclusion in *Modeling Biomolecular Site Dynamics: Methods and Protocols* (Ed. William S. Hlavacek), part of the *Methods in Molecular Biology* series.

Summary

The chapter reviews the syntax to store machine-readable annotations and describes the mapping between rule-based modelling entities (e.g., agents and rules) and these annotations. In particular, we review an annotation framework and the associated guidelines for annotating rule-based models, encoded in the commonly used Kappa and BioNetGen languages, and present prototypes that can be used to extract and query the annotations. An ontology is used to annotate models and facilitate their description.

Key words: Rule-Based Modelling, Kappa, BNGL, KaSim, BioNetGen, RDF, Turtle, MIRIAM, SPARQL, Rule-Based Model Ontology (rbmo)

1 Introduction

1.1 The need for model annotation

The last decade has seen a rapid growth in the number of model repositories (1–5). It is also well understood that the creation of models and of repositories requires expert knowledge and integration of different types of biological data from multiple sources (6). These data are used to derive the structure of, and parameters for, models. However which data are used and how the model is derived from that data is not part of the model unless we explicitly annotate it in a well-defined way.

In general, annotations decorate a model with metadata linking to biologically relevant information (7). Annotations can facilitate the automated exchange, reuse and composition of complex models from simpler ones. Annotations can also be used to aid in the computational conversion of models into a variety of other data formats. For example, PDF documents (1) or visual graphs (8) can be automatically generated from annotated models to aid human understanding.

On the computational and modelling side, rule-based languages such as Kappa (9, 10) and the BioNetGen language (BNGL) (11) have emerged as helpful tools for modelling biological systems (12). One of the key benefits of these languages is that they can be used to concisely represent the combinatorially complex state space inherent in biological systems. Rule-based modelling languages have facilities to add comments that are intended for unstructured documentation and usually directed at the modeller or programmer. These comments are in general human and not machine-readable. This can be a problem because the biological semantics of the model entities are not computationally accessible and cannot be used to influence the processing of models.

Previous works have addressed the issue of annotations in rule-based models. In particular, Chylek et al. (13) suggested extending rule-based models to include metadata, focusing on documenting models with biological information using comments to aid the understanding of models for humans. More recently, Klement et al. (14) have presented a way to add data in the form of property/value pairs using a specific syntax. On the other hand, machine-readable annotations have been applied to rule-based models using PySB, a programming framework for writing rules using Python (15). However, this approach is restricted as annotations cannot be applied to sites or states.

In this chapter we first discuss the general idea of annotation, its relation with the concept of abstraction and then review an annotation framework for rule-based models that has recently introduced and defined by Misirli et al. (16).

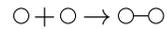
1.2 Reactions, rules, annotations and abstractions

Before entering into the technicalities of the annotation framework of interest, we would like to discuss in an informal and intuitive manner the differences between models created using reactions versus those obtained using rules, discussing the advantages of considering annotations and how they are strictly linked to the much more general notion of abstraction.

1.2.1 Reactions and rules

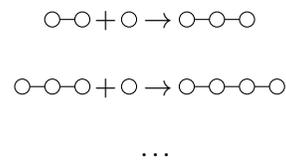
Rules as they are to be understood in the present context are a sort of generalisation of reactions of the type familiar from chemistry. The reason this generalisation is useful can

be easily seen. Consider the following toy example,

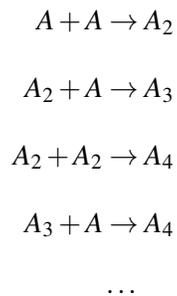


which can be understood as a step in the creation of a polymer from two monomers.

Multiple applications of this rule result in a progressively longer chain of molecules,

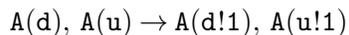


Writing this down in the notation of reaction, we would need to explicitly generate the entire unbounded sequence of reactions with an unbounded number of chemical species,



Clearly this is unworkable with finite resources. The solution is to allow a species to have *sites* at which connections can be made. In the above example, the species could be described as $A(u, d)$, that is substance A with an upstream and a downstream site. The

interaction can then be written as,



where the notation d means that the downstream site is unbound, and the $d!1$ means it is bound with a particular edge. Note that this says nothing about the state of the upstream site in the first instance of A nor the downstream site in the second, so there can be an arbitrarily long chain of molecules attached at those sites. It is easy to see that this compact notation captures both the infinite sequence of reactions and the infinite set of species that would be required to express the same interaction as a set of chemical reactions.

1.2.2 Annotations

Informally, the word “annotation” has a meaning similar to “documentation” but with a difference in specificity. Whereas documentation connotes a rather large text describing something (e.g., an object), annotation is expected to be much shorter. It also evokes proximity: it should be in some sense “near” or “on” the thing being annotated. In both cases there seems to be a sharp distinction between the text and its object. The object should exist in its own right, be operational or functional in the appropriate sense without need to refer to exogenous information. Annotation might help to understand the object but the object exists and functions on its own.

This folk theory of annotation breaks down almost immediately under inspection. A typical example is data about a book such as might be found in a library catalogue. This is a canonical example used to explain what is meant by *metadata* or data about data. The first observation is that if we look at a book and peruse the first few pages it

is almost certain that we will find information about who wrote it and where and when it was published. This information is not the book, it is metadata about the book, but it is contained within the covers of the book itself.

Perhaps this is not so serious a problem. It is possible in principle to imagine that a book, say with the cover and first few pages torn out, is still a book that can be read and enjoyed. Perhaps somehow the metadata is *separable* and that is the important idea. The book-object can exist on its own and serve its purpose independently of any annotation or metadata. While the metadata might usually be found attached to the book, can easily be removed without affecting the fundamental nature of the book itself.

But what of other things that we might want to do with a book? A favourite activity of academics is *citing* documents such as books and journal articles. This means including enough information in one work to unambiguously *refer* to another. There is an urban legend that Robarts Library at the University of Toronto is said to be sinking because the engineers charged with building it did not account for the weight of the books within. Supposing that this were true, these poor apocryphal engineers could have used metadata within the university's catalogue to sum up the number of pages of all the books and estimate their weight to prevent this tragedy. This summing is a computation that operates purely on the metadata and not on the books themselves.

More mundanely, categorising and counting books in order to plan for the use of shelf space in a growing collection, or even locating a book in a vast library seem to be a plausible things to do with metadata that do not involve any actual books. Manipulation and productive use of annotation is possible in the absence of the objects and well-defined even if the objects no longer exist. One imagines the despondent librarians and archivists of Alexandria making such lists to document and take stock of their losses after the great

fire.

Now suppose that this list created by the librarians of Alexandria itself ended up in a collection in some other library or museum. It is given a catalogue number, the year it was acquired is marked. Now what was metadata has now itself become the object of annotation! Here we arrive at the important insight: what is to be considered annotation and what is to be considered object depends on the purpose one has in mind. If the interest is the collection of books in Alexandria, the list is metadata, a collection of annotations, about them. If the interest is in the documents held by a contemporary museum, among which the list is to be found, the list is an object. The distinction is not intrinsic to the objects themselves.

Turning to the subject at hand, the objects to be annotated are rules. According to the folk theory of annotation, there should be a sharp distinction between rules and their annotation. When it comes to executing a simulation, the software that does this need not be aware of the annotations. Indeed the syntax for annotating rules described here is specifically designed for backwards compatibility such that the presence of annotations should not require any disruption or changes to existing simulation software.

So long as the *purpose* of the annotations is as an aid to understanding the rules the location of the distinction between rule and annotation is fixed in this way. The obvious question is, are there other uses to which the annotations can be put?

In the report of Misirli et al. (16), where the annotation mechanism of interest was first described, one of the motivating examples was to create a *contact map*, a type of diagram that shows which agents or species interact with each other and labels these interactions with the rule(s) implementing them (an example of a contact map is provided later in this chapter).

Use of a contact map is illustrative of how movable the separation between object and annotation is (17). The entities of interest, rules and agents, are on the one hand decorated with what seems to be purely metadata: labels, or friendly human-readable names that are suitable for placing on a diagram, preferable to the arbitrary machine-readable tokens that are used by the simulator (arbitrary because they are subject to renaming as required). On the other hand, the interactions between the substances, what we wish to make a diagram *of*, are written down in a completely different language with an incompatible syntax.

A minor change of perspective neatly solves this problem. It is simply to rephrase the rule, saying “A and B are related, and the way they are related is that they combine to form C”. This has the character of annotation: the rule itself is a statement about the substances involved. More particularly it describes a *relation* between the substances. On close inspection, giving a token used in a rule a human-readable name is also articulating a relation, that is the relation called “naming” between the substance and a string of characters suitable for human consumption.

With this change of perspective, all of the information required to make the diagram is now of the same kind. The only construct that must be manipulated is sets of relations between entities (and strings of text, which are themselves a kind of entity). Fortunately there exist tools and query languages for operating on data stored in just this form. Having worked out the correct query to extract precisely what is needed to produce the diagram, actually generating it is trivial.

1.2.3 Abstractions and annotations

The preceding section on annotation, describes what can be thought of as a “movable line”. “Above” this line are annotations and “below” it are the objects. The sketch of a

procedure for producing a diagram to help humans understand something about a system of rules as a whole illustrated that it can be convenient to place this line somewhere other than might be obvious at first glance — and this example will be considered in more detail below to demonstrate how this happens in practice. However the idea of such a line and how it might be moved and what exactly that means is still rather vague. Let us now make this notion more precise.

Formally, a *relation* between two sets, X and Y , is a subset of their Cartesian product, $X \times Y$. In other words it is a set of pairs, $\{(x,y) | x \in X, y \in Y\}$, and it is usually the case that it is a proper subset in that not all possible pairs are present in the relation. In order to compute with relations, the sets must be *symbols*, $X, Y \subseteq \mathbb{S}$, ultimately realised as sequences of bits because a computer or Turing machine is defined to operate on such sequences and not on every day objects such as books, pieces of fruit, molecules or sub-atomic particles, or indeed concepts and ideas.

This last point is important. It is not possible to compute with objects in the world, be they concrete or abstract, it is only possible to compute with symbols representing these objects. Another kind of relation is required for this, $\mathbb{R} \subseteq \mathbb{S} \times \mathbb{W}$ where \mathbb{W} is the set of objects in the world. It is not possible to write down such relations between symbols and real-world objects any more than it is possible to write down an apple. So we have two kinds of relations to work with: annotations which are relations among symbols in $\mathbb{S} \times \mathbb{S}$ and representations which map between symbols and the world, $\mathbb{S} \times \mathbb{W}$.

Some observations are in order. First, the representation relation has an inverse, $\mathbb{W} \times \mathbb{S}$. This is trivial and is simply “has the representation” as opposed to “represents”. Second, of course, symbols are themselves objects in the world, so $\mathbb{S} \subset \mathbb{W}$. Finally, relations among symbols—annotations—are likewise objects in the world, so $\mathbb{S} \times \mathbb{S} \subset \mathbb{W}$ also.

This is useful because it means that it is possible to represent annotations with symbols and from there articulate relationships among them using more annotations, constructing a hierarchy of annotation as formalised by Buneman et al. (17). We run into trouble though if we try to say that representations are in the world because $\mathbb{S} \times \mathbb{W}$ is larger than \mathbb{W} , and this is why they cannot be written down. Symbols represent, annotations are relations among symbols, and the character of representation is fundamentally different from that of annotation.

We have enough background to explain the intuition behind the folk theory of annotation, that there is a difference of kind between the annotation and its object. This difference is just the same as considering a notional pair $(x \in \mathbb{S}, -)$ *qua* annotation or *qua* representation, that is, deciding the set from which the second element of the tuple should be drawn. A similar choice is available, *mutatis mutandis*, for the inverse, $(-, x \in \mathbb{S})$. If the unspecified element is in $\mathbb{W} \setminus \mathbb{S}$ (i.e., those objects in the world that are not symbols), there is only one choice: the relation can only be treated as representation. If it is in $\mathbb{W} \cap \mathbb{S}$ then either interpretation is possible, and one or the other might be more appropriate depending on the purpose or question at hand.

The ability to make this choice is no more than the ability to select an appropriate *abstraction*. Selecting an abstraction means deciding to interpret a relation as representation and not annotation. This is best illustrated with an example. Here is a (representation of an) agent or substance:



Perhaps it is a fragment of DNA which can be connected up-stream and down-stream to

other such fragments, and it has a binding site where RNA polymerase can attach as part of the transcription process. Some annotations involving A might be,

$$(A, \text{"Promoter"}) \in \mathbb{L}$$

$$(A, \text{TTGATCCCTCTT}) \in \mathbb{M}$$

where the first is from the set of labellings, \mathbb{L} , and the second is from the set of correspondences with symbols representing nucleotide sequences, which we will call \mathbb{M} . A more conventional way of writing these correspondences more closely to the Semantic Web practice is,

A label "Promoter"

A has sequence TTGATCCCTCTT

The labelling annotation is easy to understand. It simply provides a friendly string for humans.

The second annotation is more challenging. It says that the DNA fragment represented by A corresponds to a certain sequence of nucleotides. On the one hand the symbol for that sequence could simply be taken as-is, if it does not play an explicit role in the computer simulation of whatever interactions A is involved in. That corresponds to treating the symbol TTGATCCCTCTT as a representation. It is the end of the chain; there only remains the relation from that symbol to something in the world, which is not something that we can write down or compute with.

On the other hand, it is equally possible to write down an annotation on the sequence symbol that specifies the list of (symbols representing) the nucleotides that it consists of,

TTGATCCCTCTT consists [T, T, G, A, T, C, C, C, T, C, T, T] .

Such a verbose formulation might be useful if one had, for example, a machine for synthesizing DNA molecules directly to implement an experiment *in vitro* for a genetic circuit that had already been developed and tested by simulation *in silico*, or a computer simulation that worked at a very detailed level. In this case the symbols, A, C, T and G play the role of representing real-world objects and the symbol TTGATCCCTCTT is merely a reference that can be used to find the (list-structured) relations among them. By making this choice, the selected abstraction has become more granular.

Another example, pertinent because while we do not yet have machines for arbitrarily assembling DNA molecules from individuals, we do have tools for drawing contact map diagrams, is a rule involving this agent. This agent has a binding site which may be occupied by an RNA-polymerase molecule at a certain rate. This could be expressed as,

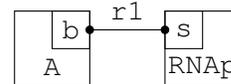
```
#^ r1 label "Binding of RNAP to A"
```

```
'r1' A(b!_), RNAP(s!_) -> A(b!1), RNAP(s!1) @k
```

where now we have introduced a little bit more of the syntax that will be more fully elaborated later for annotating rules written in a file using the Kappa language. Here a rule is simply given a useful human-readable label, the canonical example of annotating something. On its own, it is useful. Imagine a summary of the contents of a set of such rules using labels like this. For that purpose the symbol `r1` can be considered just to represent the rule without looking any deeper.

For a contact map diagram, more information is needed.

At right is the diagram that corresponds to the example rule.



It shows that A and RNAP interact, that it happens through

the action of the rule `r1` and in particular involves the sites `b` and `s`. Perhaps including which sites are involved in the interaction is too granular and it might be desirable in some circumstances to have a similar diagram involving just the agents and the rules. Or

perhaps more information is desired to be presented in the diagram such as whether the rule involves creation or annihilation of a bond, say using arrows or a broken edge. No matter the level of granularity required, it is clear that the necessary information is contained within the rule itself, so simply considering the symbol r_1 to opaquely represent to the rule as an object is not enough. Such a level of abstraction would be too coarse, it must be elaborated further. Instead it should be considered to represent annotations that themselves represent the structure of the rule.

This discussion illustrates the idea of a contact map and how it can be generated from annotations, but to elaborate the rule sufficiently to support the production of such a diagram in practice involves a much greater amount of annotation structure than we have seen so far. A rule has a left and a right side. Each of those has zero or more agent *patterns*. A rule does not involve agents as such, rather it involves patterns that can match configurations of agents, so patterns then relate, *intra alia*, to agents and sites, and finally bonds between sites that are either to be matched (on the left-hand side) or created or annihilated (on the right-hand side). It involves some work to represent a rule as annotation in sufficient detail, but it is straightforward to do within the framework that we have given.

2 Annotation of Rule-Based Models

We focus our attention on annotating models written using either the Kappa or BioNetGen language. Software tools compatible with these modeling languages are available at the following URLs:

1. <https://kappalanguage.org>

2. <https://github.com/RuleWorld>

2.1 Rationale for recommended annotation conventions

Following our general discussion above about annotations and rule-based models, here we move to the more technical aspects (focusing on two languages, Kappa and BNGL) and follow the terminology and the definitions provided in Ref. (16).

Biological entities are represented by agents in Kappa and molecule types in BNGL (we use ‘agent’ to generically refer to both types). Agents may include any number of sites that represent the points of interactions between agents. For example, the DNA binding domain of a transcription factor (TF) agent can be connected to a TF binding site of a DNA agent. Moreover, sites can have states. For instance, a TF may have a site for phosphorylation and DNA binding may be constrained to occur only when the state of this site is phosphorylated.

For an agent with two sites, of which one with two internal states and the other with three, the number of possible combinations is six (Figure 1A, B). A pattern is an (possibly incomplete) expression of an agent in terms of its internal states and binding states. Rules specifying biological interactions consist of patterns on the left-hand side which, when matched, produce the result on the right-hand side (Figure 1C). Specific patterns of interest can be declared as an observable of a model (i.e., a simulation output).

It is important to highlight that while the syntactic definition of an agent identifies sites and states in rule-based models, the semantics of sites and states is usually clear only to the modeller. Clearly, if one wishes to have machine access, then this information must be exposed in a structured way. The key idea of the approach presented in Ref. (16) and that we review in what follows, is to extend the syntax of rule-based models to incorporate

```

A: An agent definition
A(site1~u~v, site2~x~y~z)

B: Possible combinations of internal states
A(site1~u,site2~x)
A(site1~u,site2~y)
A(site1~u,site2~z)
A(site1~v,site2~x)
A(site1~v,site2~y)
A(site1~v,site2~z)

C: An example binding rule
A(site1~v,site2~z),A(site1~v,site2~y)
  -> A(site1~v!1,site2~z),A(site1~v!1,site2~y) @kf

```

Figure 1: **A.** An agent with two sites. *site1* has two possible internal states while *site2* has three. **B.** This agent can be used in six different ways depending on the internal states of its sites. **C.** A rule that specifies how agent *A* forms a dimer when the state of *site1* is *v* and the states of *site2* are *z* and *y*, respectively. The symbol *!n* means that the sites where it appears are bound (connected) together. The constant *k_f* denotes the kinetic rate associated with the rule.

annotations.

Existing metadata resources include machine readable controlled vocabularies and ontologies and Web services providing standard access to external identifiers and guidelines for the use of these resources. For example, the Minimum Information Requested in the Annotation of Models (MIRIAM) standard (18) provides a standard for the minimal information required for the annotation of models.

Following Ref. (16) we suggest that entities in models should be linked to external information through the use of unique and unambiguous Uniform Resource Identifiers (URIs), which are embedded within models. The uniqueness and global scope of these URIs are then crucial for *disambiguation* of model agents, variables and rules.

We also choose to represent annotations using the Resource Description Framework (RDF) data model (19, 20) as statements or binary predicates. A statement can link a modelling entity to a value using a standard qualifier term (predicate), which represents the

relationship between the entity and the value. These qualifiers often come from controlled vocabularies or ontologies in order to unambiguously identify the meaning of modelling entities. URIs are used as values to link these entities to external resources, and hence to a large amount of biological information by keeping the number of annotations minimal. The links themselves are typed, again with URIs. The qualifiers and resources to which they refer are drawn from ontologies that encode the Description Logic (21) for a particular domain.

Semantics can be unified by means of metadata with controlled vocabularies. There are several metadata standard initiatives that provide controlled vocabularies from which standard terms can be taken. For instance, metadata terms provided by the Dublin Core Metadata Initiative (DCMI) (22) or BioModels qualifiers can be used to describe modelling and biological concepts (1, 23). On the other hand, ontologies such as the Relation Ontology provide formal definitions of relationships that can be used to describe modelling entities (24). There are also several other ontologies and resources that are widely used to classify biological entities represented in models with standard values (25): the Systems Biology Ontology (SBO) (26) to describe types of rate parameters; the Gene Ontology (GO) (27) and the Enzyme Commission (EC) numbers (28) to describe biochemical reactions; the Sequence Ontology (SO) (29) to annotate genomic features and unify the semantics of sequence annotation; the BioPAX ontology (30) to specify types of biological molecules and the Chemical Entities of Biological Interest (ChEBI) (31) terms to classify chemicals. URIs of entries from biological databases, such as UniProt (32) for proteins and KEGG (33) for reactions, can also be used to uniquely identify modelling entities.

Access to data should be unified and this can be done by accessing external re-

sources through URIs using MIRIAM or Identifiers.org URIs (34). It should be noted that MIRIAM identifiers are not resolvable directly over the Internet and require out of band knowledge to retrieve additional information though they are unique and unambiguous. These URIs consist of collections and their terms, which may represent external resources and their entries respectively. For example, the MIRIAM URI `urn:miriam:uniprot:P69905` (see **Note 1**) and the Identifiers.org URI `http://identifiers.org/uniprot/P69905` can be used to link entities to the P69905 entry from UniProt. The relationships between modelling entities, annotation qualifiers and values can be represented using RDF graphs.

We recommend to use RDF syntax that represents knowledge as (subject, predicate, value) triples, in which the subject can be an anonymous reference or a URI, the predicate is a URI and the object can be a literal value, an anonymous reference or a URI.

Subjects and objects may refer to an ontology term, an external resource or an entity within a model. RDF graphs can be then serialized in different formats such as XML or the more human readable Turtle format (35). Modelling languages such as the Systems Biology Markup Language (SBML) (36), CellML (37, 38) and Virtual Cell Markup Language (5) are all XML-based and provide facilities to embed RDF/XML annotations (6).

Moreover, there are also other exchange languages, such as BioPAX and the Synthetic Biology Open Language (SBOL) (39, 40), that can be serialised directly as RDF/XML allowing custom annotations to be embedded.

Following the suggestion of Misirli et al. (16) one can extend the use of RDF and MIRIAM annotations to describe a syntax to store machine-readable annotations and an ontology to facilitate the mapping between rule-based model entities and their annotations. We illustrate annotations using terms from this ontology and propose some exam-

ples.

2.2 Conventions for annotating Kappa- and BNGL-formatted models

Here, we review the syntax originally defined by Misirli et al. (16) for storing annotations. We start by noticing that a common approach, when trying to add additional structured information to a language where it is undesirable to change the language itself, is to define a special way of using comments. This practice is established for structured documentation or “docstrings” in programming languages (41, 42). The idea is to use this same approach so that models written using the conventions that we describe here do not require modification of modelling software, such as KaSim (43) or RuleBender (44).

For this reason, we use the language’s comment delimiter followed by the ‘^’ character to denote annotations in the textual representation of rule-based languages. Kappa and BNGL both use the ‘#’ symbol to identify comment lines, so in the case of these languages, comments containing annotations are signalled by a line beginning with ‘#^’. This distinguishes between comments containing machine-readable annotations and comments intended for direct human consumption. Annotation data for a single modelling entity or a model itself can be declared over several lines and each line is prefixed with the ‘#^’ symbol.

Annotations are then serialised in the RDF/Turtle format. We claim that this leads to a good balance between the need for a machine-readable syntax and a human readable textual representation. Rule-based modelling languages are themselves structured text formats designed for this same balance, so RDF/Turtle is more suitable than the XML-based representations of RDF.

Annotations for a single rule-based model entity are a list of statements. It is important to stress that annotations may refer to other annotations within the same model. When all the lines corresponding to a rule-based model and the annotation delimiter symbols are removed, the remaining RDF lines can represent a single RDF document. This enables annotations to be quickly and easily extracted without special tools (*see Note 2*).

In textual rule-based models, it is difficult to store annotations within a modelling entity since Kappa and BNGL represent modelling entities such as agents and rules as single lines of text. As a result, there is no straightforward location to attach annotations to an entity. Following Ref. (16) we achieve the mapping between a modelling entity and its annotations by defining an algorithm to construct a URI from the symbol used in the modelling language. The algorithm generates unique and unambiguous prefixed names that are intended to be interpreted as part of a Turtle document. The algorithm simply constructs the local part of a prefixed name by joining symbolic names in the modelling language with the ‘.’ character, and prepending the empty prefix, ‘.’. This means that one must satisfy the condition that the empty prefix is defined for this use. Using this algorithm, we can derive a globally unique reference for the y internal state of site `site2` of agent `A` from $A(\text{site1} \sim u \sim v, \text{site2} \sim x \sim y \sim z)$ as `:A:site2:y`.

In Kappa, rules do not have symbolic names but each rule can be preceded by free text surrounded by single quotes. We require this free text to be consistent with the local name syntax in the Turtle and SPARQL (45) languages. If this requirement is satisfied, identifiers for subrules are created by just adding their position index, based on one, to the identifier for a rule (see Figure 4B). A similar restriction is placed on other tokens used in the models; agent and site names, variable and observable names must all conform to the local name syntax.

Controlled vocabularies such as BioModels.net qualifiers are formed of *model* and *biology* qualifiers. The former offers terms to describe models. BioModels.net qualifiers are also appropriate to annotate rule-based models, but additional qualifiers are needed to fully describe rule-based models. These are specific to the annotation of rule-based models and this is done by using a distinct ontology – the *Rule-Based Model Ontology* – in the namespace <http://purl.org/rbm/rbmo#> conventionally abbreviated as `rbmo` (we omit the prefix if there is no risk of ambiguity). Each qualifier is constructed by combining this namespace with an annotation term. A subset of significant terms are listed in Table 1 while the full ontology is available online at the namespace URI.

In the `rbmo` vocabulary, the `Model` classes such as `Kappa` and `BioNetGen` specify the type of the model being annotated. The term `Agent` is used to declare physical molecules. Hence, the `Agent` class can represent agents and tokens in Kappa, or molecule types in BioNetGen. `Site` and `State` represent sites and states in these declarations respectively. Rules are identified using `Rule`. The predicates `hasSite` and `hasState` and their inverses are used to annotate the links between agents, sites and internal states declarations. Table 1 reviews the terms related to the declaration of the basic entities from which models are constructed. We assume that the terms that start with an uppercase letter are types (In the sense of `rdf:type`, and also in this instance `owl:Class`) for the entities in the model which the modeller could be expected to explicitly annotate. The predicates begin with a lowercase letter and are used to link entities to their annotations.

Table 2 includes terms to facilitate representation of rules in RDF. This change of representation (materialization), from Kappa or BNGL to RDF is something that can easily be automated and a tool is already available (for models written in Kappa).

This representation in RDF is helpful for analysis of models because it merges the

model itself with the metadata in a uniform way easy to query. Annotations that cannot be derived from the model (as well as the model itself) are written explicitly in RDF/Turtle using the terms from Table 1 embedded in comments using a special delimiter. Extra statements can then be derived by parsing and analyzing the model using terms from Table 2 and the same naming convention from the algorithm previously described. These statements are then merged with the externally supplied annotations to obtain a complete and uniform representation of all the information about the model.

The open-ended nature of the RDF data model means that it is possible to freely incorporate terms from other ontologies and vocabularies, including application-specific ones. In this respect, two terms are crucial. The `dct:isPartOf` predicate from DCMI Metadata Terms is used to denote that a rule or agent declaration *is part of* a particular model (or similarly with its inverse, `dct:hasPart`).

The `bqiol:is` predicate from the *Biomodels.net Biology Qualifiers* is used to link internal states of sites to indicate their biological meaning. This term is chosen because it denotes a kind of identification that is much weaker than the logical replacement semantics of `owl:sameAs`. Using the latter would imply that everything that can be said about the site *qua* biological entity can also be said about the site *qua* modelling entity. Clearly, these are not the same and identifying them in a strong sense would risk incorrect results when computing with the annotations.

Table 3 enumerates useful ontologies and vocabularies with their conventional prefixes to annotate rule-based models. This list is not exhaustive and can be extended.

2.3 Adding annotations to model-definition files

Here, we demonstrate how the suggested annotations can be added to rule-based models.

Again we follow the methodology originally presented in Ref. (16).

```
#^@prefix : <\protect\vrule width0pt\protect\href{http://.../tcs.kappa#}{http://.../tcs.kappa#}>.
#^@prefix rbmo: <\protect\vrule width0pt\protect\href{http://purl.org/rbm/rbmo#}{http://purl.org/rbm/rbmo#}>.
# ... other prefixes elided ...
#^@prefix dct: <\protect\vrule width0pt\protect\href{http://purl.org/dc/terms/}{http://purl.org/dc/terms/}>.
#^@prefix foaf: <\protect\vrule width0pt\protect\href{http://xmlns.com/foaf/0.1/}{http://xmlns.com/foaf/0.1/}>.

#^ :kappa a rbmo:Kappa ;
#^ dct:title "TCS_PA Kappa model" ;
#^ dct:description
#^ "Two component systems and promoter architectures" ;
#^ dct:creator "Goksel Misirli", "Matteo Cavaliere";
#^ foaf:isPrimaryTopicOf <https://.../tcs.kappa> .
```

Figure 2: An example model annotation (as in (16)), with details about its name, description, creators and online repository location. The prefix definitions required to annotate the model are defined first, and the empty prefix is defined for the model namespace itself.

Annotations are added by simply adding a list of prefix definitions representing annotation resources providing relevant terms for the annotation of all model entities (such as agents and rules). These definitions are followed by statements about the title and description of the model, using the `title` and `description` terms from *Dublin Core*. Annotations can be expanded to include model type, creator, creation time, and its link to an entry in a model database (Figure 2).

Table 4 shows how distinct entities in a model can be annotated using terms from `rbmo` and from other vocabularies. Figure 3 shows examples of Agent annotations. In Figure 3A the ATP token is annotated as a small molecule with the identifier 15422 from ChEBI. Agents without sites can also be annotated in a similar way. In Figure 3B, the agent is specified to be a protein using the `biopax:Protein` value for the `biopax:physicalEntity`

```

A:
#^:ATP a rbmo:Agent ;
#^ bqbiol:isVersionOf chebi:CHEBI:15422 ;
#^ biopax:physicalEntity biopax:SmallMolecule .
%token: ATP()

B:
#^:Kinase a rbmo:Agent ;
#^ rbmo:hasSite :Kinase:psite ;
#^ bqbiol:is uniprot:P16497 ;
#^ biopax:physicalEntity biopax:Protein ;
#^ ro:hasFunction go:GO:0000155 .
#^:Kinase:psite a rbmo:Site ;
#^ rbmo:hasState :Kinase:psite:u, :Kinase:psite:p .
#^:Kinase:psite:u a rbmo:State ;
#^ bqbiol:is pr:PR:000026291 .
#^:Kinase:psite:p a rbmo:State ;
#^ bqbiol:is psimod:MOD:00696 .
%agent: Kinase(psite~p~u)

C:
#^:pSpo0A a rbmo:Agent ;
#^ rbmo:hasSite :pSpo0A:tfbs ;
#^ bqbiol:isVersionOf so:SO:0000167 ;
#^ biopax:physicalEntity biopax:DnaRegion ;
#^ sbol:nucleotides "ATTTTTTTAGAGGGTATATAGCGGTTTTGTCGAATGTAACATGTAG" ;
#^ sbol:annotation :pSpo0A_annotation_28_34 .
#^:pSpo0A:tfbs a rbmo:Site ;
#^ bqbiol:isVersionOf so:SO:0000057 ;
#^ biopax:physicalEntity biopax:DnaRegion ;
#^ sbol:nucleotides "TGTCGAA" .
#^:pSpo0A_annotation_28_34 a sbol:SequenceAnnotation ;
#^ sbol:bioStart 28;
#^ sbol:bioEnd 34 ;
#^ sbol:subComponent :pSpo0A:tfbs .
%agent: pSpo0A(tfbs)

D:
#^:Spo0A a rbmo:Agent .
%agent: Spo0A(psite~p~u)
#^:Spo0A_p a rbmo:Observable ;
#^ ro:has_function go:GO:0045893 .
%obs: 'Spo0A_p' Spo0A(psite~p)

```

Figure 3: Examples of agent annotations for **A**. An ATP token agent. **B**. A kinase agent with phosphorylated and unphosphorylated site. **C**. A promoter agent with a TF binding site. **D**. An agent and an associated observable for the phosphorylated Spo0A protein, which can act as a TF.

term. This protein agent is annotated as P16497 from UniProt, which is a protein kinase (i.e., an enzyme that phosphorylates proteins) involved in the process of sporulation. It has a site with the phosphorylated and unmodified states, which are annotated with corresponding terms from the Protein Modification Ontology (46).

The `ro:hasFunction` term associates the agent with the GO's histidine kinase molecular function term `GO:0000155`. In Figure 3C, a promoter agent with a TF binding site is represented. Both the promoter and the operator agents are of "DnaRegion" type, and are identified with the `SO:0000167` and `SO:0000057` terms. Although the nucleotide information can be linked to existing repositories using the `bqbiol:is` term, for synthetic sequences agents can directly be annotated using SBOL terms. The term `sbol:nucleotides` is used to store the nucleotide sequences for these agents. A parent-child relationship between the promoter and the operator agents can be represented using an `sbol:SequenceAnnotation` RDF resource, which allows the location of an operator subpart to be specified.

This approach can be used to annotate a pattern with a specific entry from a database (patterns can also be stated as observables of the model). For instance, Figure 3D shows an example of such an observable. `Sp00Ap` represents the phosphorylated protein, which acts as a TF and is defined as an observable.

Figure 4 demonstrates annotation of rules. The first rule (Figure 4A) describes the binding of the LacI TF to a promoter. This biological activity is described using the `GO:0008134` (*transcription factor binding*) term. In the second example (Figure 4B), a phosphorylation rule is annotated. The rule contains a subrule representing ATP to ADP conversion. This subrule is linked to the parent rule with the `hasSubrule` qualifier. Moreover, the annotation of the rate for this rule is presented in Figure 4C. The anno-

tated Kappa and BNGL models for a two-component system (TCS), controlling a simple promoter architecture can be found online (*see Note 3*).

Finally, in Figure 5 we present the fragment of a specific rule (taken from the TCS Kappa model) materialised using the `krdf` tool. The tool generates a version of the rules themselves in RDF together with the annotations (in this way the entire model is presented in a more uniform way).

```

A:
#^:LacI.pLac a rbmo:Rule ;
#^ bqbiol:isVersionOf go:GO:0008134 ;
#^ dct:title "Dna binding" ;
#^ dct:description "TF1 binds to the promoter" .
'LacI.pLac' Target(x~p), Promoter(tfbs1,tfbs2) <-> Target(x~p!1), Promoter(tfbs1!1,tfbs2)
@kf,kr

B:
#^:S_phosphorylation a rbmo:Rule ;
#^ bqbiol:isVersionOf sbo:SBO:0000216 ;
#^ dct:title "S Phosphorylation" ;
#^ dct:description "S is phosphorylated" ;
#^ rbmo:hasSubrule :S_phosphorylation:1 .
#^:S_phosphorylation:1 a rbmo:Rule ;
#^ bqbiol:isVersionOf sbo:SBO:0000216 ;
#^ dct:title "ATP -> ADP" ;
#^ dct:description "ATP to ADP conversion" .
'S_phosphorylation' S(x~u!1), K(y!1) | 0.1:ATP -> S(x~p), K(y) | 0.1:ADP @kp

C:
#^:kp a sbo:SBO:0000002 ;
#^ bqbiol:isVersionOf sbo:SBO:0000067 ;
#^ dct:title "Phosphorylation rate" .

```

Figure 4: Annotating rules and variables. **A.** TF DNA binding rule. **B.** Phosphorylation rule with a subrule for the ATP to ADP conversion. **C.** Annotation of a phosphorylation rate variable.

2.4 How to use annotations

The framework we have described can be coupled to the development of tools that allow one to extract and analyze the annotations embedded in a model. Several tools are currently under development. We demonstrate here the `krdf` tool that can be used for checking duplication of rules and inconsistencies between different parts of a model, basic

```

:As1As2Spo0A_to_As2Spo0A a rbmo:Rule ;
dct:title "Cooperative unbinding" ;
rbmo:lhs [
  a rbmo:Pattern ;
  rbmo:agent :Spo0A ;
  rbmo:status [
    rbmo:isBoundBy :As1As2Spo0A_to_As2Spo0A:left:1 ;
    rbmo:isStatusOf :Spo0A:DNAb ;
    a rbmo:BoundState ;
  ], [
    rbmo:internalState :Spo0A:RR:p ;
    rbmo:isStatusOf :Spo0A:RR ;
    a rbmo:UnboundState ;
  ] ;
].

```

Figure 5: Fragment of the RDF representation of a materialised rule obtained by merging the metadata supplied by the model author with an RDF representation of the rule. The left hand side of the rule contains a pattern involving `:Spo0A` and that there are two pieces of state information: The first one refers to the `:Spo0A:DNAb` site, and it is bound to something (that can only be recovered using the rest of the model, not presented here). The second refers to the `:Spo0A:RR` site, it has a particular internal state, and it is unbound.

problems encountered when composing and creating biological models (47, 48). Another application is to draw an annotated contact map visualising the entities involved, the interactions and the biological information stored in the annotations – this merges the classical notion of contact map used to illustrate Kappa and BNGL models (9, 49) with biological semantics.

The `krdf` tool operates on Kappa models and has several modes of operation that can provide increasingly more information about a model. The first, selected with the `-a` option, extracts the modeller’s annotations. The second mode, selected with the `-m` option, *materialises* the information in the rules themselves into the RDF representation (as illustrated in Figure 5). Finally the `-n` option *normalises* the patterns present in the rules according to their declarations.

Once a complete uniform representation of the model in RDF has been generated, one

can query it using SPARQL with a tool such as `roqet` (50). For example, a SPARQL query can deduce a contact map – pairings of sites in agents that undergo binding and unbinding according to the rules in a model. These pairings form a graph that can be visualised using tools such as GraphViz (51). With an appropriate query (see Note 4), `roqet` can output the result in a GraphViz-compatible format. A more sophisticated manipulation (see Note 5) can extract annotations from the RDF representation of the TCS example model and easily create a richly annotated contact map diagram (Figure 6). In this way, biological information extracted from the annotations can be added to the agents, sites and interactions (using GraphViz for rendering) (see Note 6).

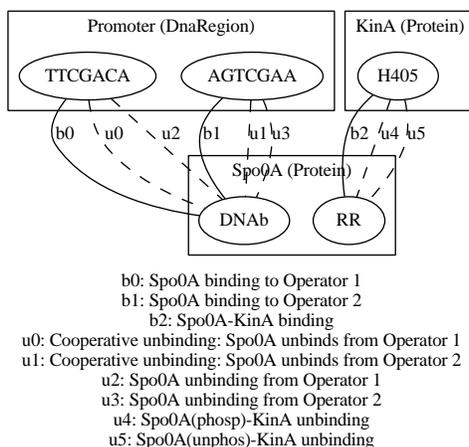


Figure 6: Contact map generated by a SPARQL query on the RDF materialisation of the TCS example in Kappa. Biological information concerning the agents, rules and sites, types of the molecules, DNA sequences and typology of the interaction, are extracted automatically from the model annotations. This figure is a reproduction of Fig. 6 in Ref. (16); no changes have been made. The figure is used under the terms of the CC-BY license (<https://opendefinition.org/licenses/cc-by/>).

Moreover, one can easily create a query that implements a join operation on the prop-

erty of `bqbiol:is`, enforcing a stronger form of identity semantics than this predicate is usually given. A filter clause is necessary to prevent a comparison of a rule with itself (see the SPARQL query in Figure 7). In this way, the discussed annotations could also be used to detect duplication of rules (e.g., obtained when combining different biological models).

```
SELECT DISTINCT ?modelA ?ruleA ?modelB ?ruleB
WHERE {
  ?ruleA a rbmo:Rule;
  dct:isPartOf ?modelA;
  bqbiol:is ?ident.
  ?ruleB a rbmo:Rule;
  dct:isPartOf ?modelB;
  bqbiol:is ?ident.
  FILTER (?ruleA != ?ruleB)
}
```

Figure 7: Detection of duplicate rules.

Another possible application of the presented annotation schema is the checking of inconsistencies in a rule-based model. This can be done in several different ways. A simple way is to use the replacement semantics of `owl:sameAs`. A statement of the form `a owl:sameAs b` means that every statement about `a` is also true if `a` is replaced by `b`. In particular if we have statements about the types of `a` and `b`, and these types are disjoint, the collection of statements is unsatisfiable (hence, the model has been found to be inconsistent). Then, an OWL reasoner such as HermiT (52) or Pellet (53) can derive that `a` and `b` have type `owl:Nothing`.

This can be implemented with the following work-flow (here only sketched): (i) generate the fully materialised RDF version of a model using `krdf`. For each use of `bqbiol:is`, add a new statement using `owl:sameAs`; (ii) retrieve all ontologies that are used from the Web. For each external vocabulary term with `bqbiol:is` or `bqbiol:isVersionOf` retrieve a description and any ontology that it uses (recursively). Merge all of these into

a single graph. This graph contains the complete model and annotations, with entities linked using a strong form of equality to external vocabulary terms, and descriptions of the meaning of these vocabulary terms; (iii) the reasoner can be used to derive terms that are equivalent to `owl:Nothing` and if any of these terms is found then an inconsistency has been identified. Using the proof generation facilities of OWL reasoners, the sequence of statements required to arrive at `foo rdf:type owl:Nothing` can be reproduced (in this way, the initial source of the inconsistency can be also identified).

2.5 Closing remarks

In this chapter we have reviewed the recent proposal to incorporate annotations into rule-based models, following the approach recently presented in Ref. (16). We have also discussed in a more general way the role of annotations and how they are strongly related to the notion of abstraction. In general, for consistency, we have followed the terms originally defined in Ref. (16). However, the suggested standardized terms can be used in a complementary manner with existing metadata resources such as MIRIAM annotations and URIs, and existing controlled vocabularies and ontologies. Although, the approach has only described the annotations of Kappa- and BNGL-formatted model-definition files, it can be easily applied to other formats for rule-based models.

In particular, PySB (15) already includes a list of MIRIAM annotations at the model level, and can be extended to include the type of annotations described here. SBML's `multi` package (see Note 7) (54) is intended to standardise the exchange of rule-based models. The entities in this format inherit the annotation property from the standard SBML and can therefore include RDF annotations. These SBML models could thus be imported or exported by tools such as KaSim or BioNetGen/RuleBender, avoiding the

loss of any biological information.

It is important to remark that annotations are also useful for automated conversions between different formats. Conversion between rules and reaction networks is already an ongoing research subject (47), and the availability of annotations can play an important role for reliable conversion and fine-tuning of models (55, 56). It is straightforward to use the framework presented and automatically map agents and rules to glyphs (13) or to convert models into other visual formats such as SBGN or genetic circuit diagrams (57).

More generally, annotations are designed for machine readability and can be produced computationally (e.g., by model repositories). This can be done by developing APIs and tools to access a set of biological parts (4, 58) that will incorporate rule-based descriptions and will be annotated with the proposed schema. This will open the possibility of composing (stitching together) rule-based models extracted from distinct repositories. Tools such as Saint (48) and SyBIL (7) could be extended to automate the annotation of rule-based models. In this way, the extensive information available in biological databases and the literature can be integrated and made available via rule-based models, taking advantage of the syntax and the framework presented here and elsewhere.

One of the ultimate goals is to use annotations as a facilitator of automatic composition of rule-based models. As recently suggested by Misirli et al. (59) the proposed schema can be used to automate the design of biological systems using a rule-based model with a workflow that combines the definition of modular templates to instantiate rules for basic biological parts. The templates, defining rule-based models for basic biological parts (see **Note 8**), can be associated with quantitative parameters to create particular parts models, which can then be merged into executable models. Such models may be annotated using the reviewed schema leading to a feasible protocol to automate their composition

for the scalable modelling of synthetic systems (59).

The described annotation ontology for rule-based models can be found at <http://purl.org/rbm/rbmo> while the tool and all the presented examples can be found at <http://purl.org/rbm/rbmo/krdf>.

3 Notes

1. A dereferenceable URI using the MIRIAM Web service is <http://www.ebi.ac.uk/miriamws/main/rest/resolve/urn:miriam:uniprot:P69905>
2. For example, on a UNIX system, the following pipeline could be used:

```
grep '^#\^'| sed 's/^#\^//'
```

3. The files `tcs.kappa` and `tcs.bngl` are available in the <http://purl.org/rbm/rbmo/examples> directory.
4. See the `binding.sparql` file in the `krdf` directory.
5. See the `contact.py` script in the `krdf` directory.
6. The tool assumes that only single instances of an agent are involved in a rule. It can be generalized.
7. See http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/multi for details.
8. These are available at <http://github.com/rbm/composition>.

Acknowledgement

The Engineering and Physical Sciences Research Council grant EP/J02175X/1 (to V.D. and M.C.), the European Union's Seventh Framework Programme for research, technological development and demonstration grant 320823 RULE (to W.W., R.H-Z, V.D.).

References

- (1) Li C, Donizelli M, Rodriguez N, et al (2010) BioModels Database: an enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Syst Biol* 4:92
- (2) Yu T, Lloyd CM, Nickerson DP, et al (2011) The Physiome Model Repository 2. *Bioinformatics* 27:743–744
- (3) Snoep JL, Olivier BG (2003) JWS online cellular systems modelling and microbiology. *Microbiology* 149:3045–3047
- (4) Misirli G, Hallinan JS, Wipat A (2014) Composable modular models for synthetic biology. *ACM J Emerging Technol Comput Syst* 11:22
- (5) Moraru II, Schaff JC, Slepchenko BM, et al (2008) Virtual Cell modelling and simulation software environment. *IET Syst Biol* 2:352–362
- (6) Endler L, Rodriguez N, Juty N, et al (2009) Designing and encoding models for synthetic biology. *J R Soc Interface* 6:S405–S417
- (7) Blinov ML, Ruebenacker O, Schaff JC, Moraru II (2010) Modeling without bor-

- ders: creating and annotating VCell models using the Web. *Lect Notes Comput Sci* 6053:3–17
- (8) Funahashi A, Jouraku A, Matsuoka Y, Kitano H (2007) Integration of CellDesigner and SABIO-RK. *In Silico Biol* 7:81–90
- (9) Danos V, Laneve C (2004) Formal molecular biology. *Theor Comput Sci* 325:69–110
- (10) Danos V, Feret J, Fontana W, Krivine J (2007) Scalable simulation of cellular signaling networks. *Lect Notes Comput Sci* 4807:139–157
- (11) Faeder JR, Blinov ML, Hlavacek WS (2009) Rule-based modeling of biochemical systems with BioNetGen. *Methods Mol Biol* 500:113–167
- (12) Köhler A, Krivine J, Vidmar J (2014) A rule-based model of base excision repair. *Lect Notes Comput Sci* 8859:173–195
- (13) Chylek LA, Hu B, Blinov ML, et al (2011) Guidelines for visualizing and annotating rule-based models. *Mol BioSyst* 7:2779–2795
- (14) Klement M, Děd T, Šafránek D, et al (2014) Biochemical Space: a framework for systemic annotation of biological models. *Electron Notes Theor Comput Sci* 306:31–44
- (15) Lopez CF, Muhlich JL, Bachman JA, Sorger PK (2013) Programming biological models in Python using PySB. *Mol Syst Biol* 9:646
- (16) Misirli G, Cavaliere M, Waites W, et al (2016) Annotation of rule-based models with

formal semantics to enable creation, analysis, reuse and visualisation. *Bioinformatics* 32:908–917

- (17) Buneman P, Kostylev EV, Vansummeren S (2013) Annotations are relative. In: Proceedings of the 16th International Conference on Database Theory, ACM, New York, pp 177–188
- (18) Le Novère N, Finney A, Hucka M, et al (2005) Minimum information requested in the annotation of biochemical models (MIRIAM). *Nat Biotechnol* 23:1509–1515
- (19) Cyganiak R, Wood D, Lanthaler M (2014) RDF 1.1 concepts and abstract syntax. URL <https://www.w3.org/TR/2014/REC-rdf11-concepts>, Accessed 17 Aug 2016
- (20) Gandon F, Schreiber G (2014) RDF 1.1 XML syntax. URL <http://www.w3.org/TR/rdf-syntax-grammar>, Accessed 17 Aug 2016
- (21) McGuinness DL, van Harmelen F (2004) OWL Web Ontology Language. URL <http://www.w3.org/TR/owl-features>, Accessed 17 Aug 2016
- (22) DCMI Usage Board (2012) DCMI metadata terms. URL <http://www.dublincore.org/documents/dcmi-terms>, Accessed 17 Aug 2016
- (23) Le Novère N, Finney A (2005) A simple scheme for annotating SBML with references to controlled vocabularies and database entries. URL <http://www.ebi.ac.uk/compneur-srv/sbml/proposals/AnnotationURI.pdf>, Accessed 17 Aug 2016

- (24) Smith B, Ceusters W, Klagges B, et al (2005) Relations in biomedical ontologies. *Genome Biol* 6:R46
- (25) Swainston N, Mendes P (2009) libAnnotationSBML: a library for exploiting SBML annotations. *Bioinformatics* 25:2292–2293
- (26) Courtot M, Juty N, Knüpfer C, et al (2011) Controlled vocabularies and semantics in systems biology. *Mol Syst Biol* 7:543
- (27) The Gene Ontology Consortium (2001) Creating the Gene Ontology Resource: design and implementation. *Genome Res* 11:1425–1433
- (28) Bairoch A (2000) The ENZYME database in 2000. *Nucleic Acids Res* 28:304–305
- (29) Eilbeck K, Lewis S, Mungall C, et al (2005) The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biol* 6:R44
- (30) Demir E, Cary MP, Paley S, et al (2010) The BioPAX community standard for pathway data sharing. *Nat Biotechnol* 28:935–942
- (31) Degtyarenko K, de Matos P, Ennis M, et al (2008) ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Res* 36:D344–D350
- (32) Magrane M, UniProt Consortium (2011) UniProt Knowledgebase: a hub of integrated protein data. *Database (Oxford)* 2011:bar009
- (33) Kanehisa M, Araki M, Goto S, et al (2008) KEGG for linking genomes to life and the environment. *Nucleic Acids Res* 36:D480–D484
- (34) Juty N, Le Novre N, Laibe C (2012) Identifiers.org and MIRIAM Registry: community resources to provide persistent identification. *Nucleic Acids Res* 40:D580–D586

- (35) EPrud'hommeaux E, Carothers G (2014) RDF 1.1 Turtle. URL <http://www.w3.org/TR/turtle>, Accessed on 17 Aug 2016
- (36) Hucka M, Finney A, Sauro HM, et al (2003) The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19:524–531
- (37) Cuellar AA, Lloyd CM, Nielsen PF, et al (2003) An overview of CellML 1.1, a biological model description language. *SIMULATION* 79:740–747
- (38) Hedley WJ, Nelson MR, Bellivant DP, Nielsen PF (2001) A short introduction to CellML. *Philos Trans A Math Phys Eng Sci* 359:1073–1089
- (39) Galdzicki M, Wilson ML, Rodriguez CA, et al (2012) Synthetic Biology Open Language (SBOL) version 1.1.0. URL <http://hdl.handle.net/1721.1/73909>, Accessed 17 Aug 2016
- (40) Galdzicki M, Clancy KP, Oberortner E, et al (2014) The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat Biotechnol* 32:545–550
- (41) Acuff R (1988) KSL Lisp environment requirements. URL https://profiles.nlm.nih.gov/BB/G/H/S/D/_/bbghsd.pdf, Accessed 14 Aug 2018
- (42) Stallman R, other GNU Project volunteers (1992) GNU coding standards. URL <https://www.gnu.org/prep/standards/>, Accessed 17 Aug 2016

- (43) Krivine J (2014) KaSim. URL <https://github.com/Kappa-Dev/KaSim>, Accessed 17 Aug 2016
- (44) Xu W, Smith AM, Faeder JR, Marai GE (2011) RuleBender: a visual interface for rule-based modeling. *Bioinformatics* 27:1721–1722
- (45) Prud'hommeaux E, Seaborne A (2013) SPARQL query language for RDF. URL <http://www.w3.org/TR/rdf-sparql-query>, Accessed 17 Aug 2016
- (46) Montecchi-Palazzi L, Beavis R, Binz PA, et al (2008) The PSI-MOD community standard for representation of protein modification data. *Nat Biotechnol* 26:864–866
- (47) Blinov ML, Ruebenacker O, Moraru II (2008) Complexity and modularity of intracellular networks: a systematic approach for modelling and simulation. *IET Syst Biol* 2:363–368
- (48) Lister AL, Pocock M, Taschuk M, Wipat A (2009) Saint: a lightweight integration environment for model annotation. *Bioinformatics* 25:3026–3027
- (49) Danos V, Feret J, Fontana W, Harmer R, Krivine J (2009) Rule-based modelling and model perturbation. *Lect Notes Comput Sci* 5750:116–137
- (50) Beckett D (2015) Redland RDF libraries. URL <http://librdf.org>, Accessed 17 Aug 2016
- (51) Ellson J, Gansner E, Koutsofios L, North SC, Woodhull G (2001) Graphviz—open source graph drawing tools. *Lect Notes Comput Sci* 2265:483–484
- (52) Shearer R, Motik B, Horrocks I (2008) HermiT: a highly-efficient OWL reasoner. In:

Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED)

- (53) Sirin E, Parsia B, Cuenca Grau B, Kalyanpur A, Katz Y (2007) Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web* 5:51–53
- (54) Zhang F, Meier-Schellersheim M (2018) SBML Level 3 package: multistate, multi-component and multicompartment species, version 1, release 1. *J Integr Bioinform* 15:20170077
- (55) Tapia JJ, Faeder JR (2013) The Atomizer: extracting implicit molecular structure from reaction network models. In: *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics*, ACM, New York
- (56) Harris LA, Hogg JS, Tapia JJ, et al (2016) BioNetGen 2.2: advances in rule-based modeling. *Bioinformatics* 32:3366–3368
- (57) Misirli G, Hallinan JS, Yu T, et al (2011) Model annotation for synthetic biology: automating model to nucleotide sequence conversion. *Bioinformatics* 27:973–979
- (58) Cooling MT, Rouilly V, Misirli G, et al (2010) Standard virtual biological parts: a repository of modular modeling components for synthetic biology. *Bioinformatics* 26:925–931
- (59) Misirli G, Waites W, Cavaliere M, et al (2016) Modular composition of synthetic biology designs using rule-based models. In: *Proceedings of 8th International Workshop on Bio-Design Automation (IWBD A 2016)*

- (60) Natale DA, Arighi CN, Barker WC, et al (2011) The Protein Ontology: a structured representation of protein forms and complexes. *Nucleic Acids Res* 39:D539–D545
- (61) Mulder NJ, Apweiler R (2008) The InterPro database and tools for protein domain analysis. *Curr Protoc Bioinformatics* 21:2.7.1–2.7.18

Tables

Table 1

Term	Description
Kappa , BioNetGen	Model types.
Agent	Type for declarations of biological entities.
Site	Type for sites of Agents .
State	Type for internal states of Sites .
hasSite , hasState , siteOf , stateOf	Predicates for linking Agents , Sites and States .
Rule	Type for interactions between agents.
hasSubrule , subruleOf	Specifies that a rule has a subrule (i.e., KaSim subrules).
Observable	Type for agent patterns counted by a simulation.

Table 2

Term	Description
Pattern	Type of a pattern as it appears in a Rule or Observable .
lhs, rhs	Predicates for linking a Rule to its left and right hand side Patterns .
pattern	Predicate for linking an Observable to the patterns that it matches.
agent	Predicate for linking a Pattern and a site within it to the corresponding Agent .
status	Specifies a status of a particular Site (and State) in a Pattern .
isStatusOf, internalState	Predicates for linking a status in a Pattern to corresponding Site and State declarations.
isBoundBy	Specifies the bond that a Site is bound to in a particular Pattern . Bonds are identified via URIs.
BoundState, UnboundState	Terms denoting that a Site in a Pattern is bound or unbound.

Table 3

Prefix	Description
rbmo	Rule-based modelling ontology (presented in this paper)
dct	Dublin Core Metadata Initiative Terms (http://www.dublincore.org/documents/dcmi-terms)
bqiol	BioModels.net Biology Qualifiers (<i>1</i>)
go	Gene Ontology (<i>27</i>)
psimod	Protein Modification Ontology (<i>46</i>)
so	Sequence Ontology (<i>29</i>)
sbo	Systems Biology Ontology (<i>26</i>)
chebi	Chemical Entities of Biological Interest Ontology (<i>31</i>)
uniprot	UniProt Protein Database (<i>32</i>)
pr	Protein Ontology (<i>60</i>)
ro	OBO Relation Ontology (<i>24</i>)
owl	Web Ontology Language (http://www.w3.org/TR/owl-features)
sbol	The Synthetic Biology Open Language (<i>39, 40</i>)
foaf	Friend of a Friend Vocabulary (http://xmlns.com/foaf/spec)
ipr	InterPro (<i>61</i>)
biopax	Biological Pathway Exchange Ontology Ontology (<i>30</i>)

Table 4

Term	Annotation Values
<u>Agent declarations:</u>	
rdf:type	Agent
dct:isPartOf	Identifier for the Model .
hasSite	Identifier of a Site .
biopax:physicalEntity	A biopax:PhysicalEntity term, e.g. DnaRegion or SmallMolecule .
bqbiol:is	A term representing an individual type of an Agent entity, e.g. a protein entry from UniProt.
bqbiol:isVersionOf	A term representing the class type of an Agent entity, e.g. a SO term for a DNA-based agent.
<u>Site declarations:</u>	
rdf:type	Site
hasState	Identifier for an internal state.
bqbiol:isVersionOf	A term representing the type of the site, e.g. A SO term for a nucleic acid-based site or an InterPro term for an amino acid-based site.
<u>Internal state declarations:</u>	
rdf:type	State
bqbiol:is	A term representing the state assignment, e.g. a term from the PSIMOD or the PO.
<u>Rules:</u>	
rdf:type	Rule
dct:isPartOf	Identifier for the Model .
bqbiol:is	A term representing an individual type of a rule, e.g. a KEGG entry.
bqbiol:isVersionOf	A term representing a class type of a rule, e.g. an EC number, a SO term or a GO term.
subrule	Identifier for a Rule entity.
lhs[†] rhs[†]	References to the patterns forming the left and right hand side of the rule.
<u>Observables:</u>	
rdf:type	Observable
dct:isPartOf	Identifier for the Model .
pattern[†]	References the constituent patterns.
<u>Patterns:</u>	
rdf:type	Pattern
ro:hasFunction	A GO term specifying a biological function.
agent[†]	Reference to the corresponding Agent declaration
internalState[†]	Reference to a representation of a site's state
isStatusOf[†]	Reference from a site's state to the corresponding site
<u>Variables:</u>	
rdf:type	sbo:SBO:0000002 (<i>quantitative systems description parameter</i>)
dct:isPartOf	Identifier for the Model .
bqbiol:isVersionOf	A term representing a variable type. If exists, the term should a subterm of SBO:0000002 .