Integration of constraints documented in SBML, SBO, and the SBML Manual facilitates validation of biological models

Allyson L. Lister[1,2], Matthew Pocock[2], Anil Wipat[1,2,*]

[1] Centre for Integrated Systems Biology of Ageing and Nutrition (http://www.cisban.ac.uk)
[2] School of Computing Science (http://www.cs.ncl.ac.uk),
Newcastle University (http://www.ncl.ac.uk)[*]

Abstract

The creation of quantitative, simulatable, Systems Biology Markup Language (SBML) models that accurately simulate the system under study is a time-intensive manual process that requires careful checking. Currently, the rules and constraints of model creation, curation, and annotation are distributed over at least three separate documents: the SBML schema document (XSD), the Systems Biology Ontology (SBO), and the "Structures and Facilities for Model Definition" document. The latter document contains the richest set of constraints on models, and yet it is not amenable to computational processing. We have developed a Web Ontology Language (OWL) knowledge base that integrates these three structure documents, and that contains a representative sample of the information contained within them. This Model Format OWL (MFO) performs both structural and constraint integration and can be reasoned over and validated. SBML Models are represented as individuals of OWL classes, resulting in a single computationally amenable resource for model checking. Knowledge that was only accessible to humans is now explicitly and directly available for computational approaches. The integration of all structural knowledge for SBML models into a single resource creates a new style of model development and checking.

1 Introduction

1.1 Background

Systems Biology Markup Language[1] (SBML) is an XML format that has emerged as the *de facto* standard file format for describing computational models in systems biology. It is supported by a vibrant community who have developed a wide range of tools, allowing models to be generated, analysed and curated in any one of many independently maintained software applications[1]. The Systems Biology Ontology[2][2] (SBO) was developed to enable a useful understanding of the biology to which a model relates, and to provide well-understood terms for describing common modelling concepts. The community is engaged in an on-going effort to develop the SBML standard in ways needed to support systems biology applications. As part of this process, a manual is maintained that describes and defines SBML and SBO[3].

The biological knowledge used to create and annotate a high-quality SBML model is typically analysed and integrated by a researcher. These modellers know and understand both the systems they are modelling and the intricacies of SBML. However, as with most areas of biology, the amount of data that is relevant to generating even a relatively small and well-

[*] Corresponding Author: anil.wipat@newcastle.ac.uk
[1] http://www.sbml.org [2] http://www.ebi.ac.uk/sbo
[3] http://prdownloads.sourceforge.net/sbml/sbml-level-2-version-3-rel-1.pdf?download

scoped model is overwhelming. In order to extend the range of modelling tasks that can be automated, it is necessary to both capture the salient biological knowledge in a form that computers can process, and represent the SBML rules in a way computers can systematically interpret. Here we address the latter issue: describing SBML, SBO and the rules about what constitutes a correctly formed model in a way suitable for computational manipulation.

The Semantic Web[4] can be seen as today's incarnation of the goal to allow computers to go beyond performing numerical computations, and to share and integrate information more easily. There are now several standards forming within the Semantic Web community that together formalise computational languages for representing knowledge and strictly define what conclusions can be reached from facts expressed in these languages. The Web Ontology Language[3][5] (OWL) is one such language that enjoys strong tools support and which is used for capturing biological and medical knowledge (e.g. OBI[6], BioPax[7], EXPO[4], and FMA[5] and GALEN[6] in OWL). Once the information about the domain has been modelled in an OWL file, a software application called a reasoner[7, 8] can automatically deduce all other facts that must logically follow as well as find inconsistencies between asserted facts.

The knowledge about a system described in SBML can be divided into two parts. Firstly, there is the *biological* knowledge. This includes information about the biological entities involved and their biological. Secondly, there is the *structural* knowledge, describing how the biological knowledge must be captured in well-formed documents suitable for processing by applications. In the case of a high-quality SBML model, the structural knowledge required to create such a model is tied up in three main locations:

- The Systems Biology Markup Language (SBML[1][8]) XML Schema Document (XSD[9]), describing the range of XML documents considered to be in SBML syntax,
- The Systems Biology Ontology (SBO[2][10]), describing the range of terms that can be used to describe parts of the model in a way understandable to the community using the Open Biological Ontologies (OBO[11]) format, and
- The "Structures and Facilities for Model Definition" document[12] (hereafter referred to as the "SBML Manual"), describing many additional restrictions and constraints upon SBML documents, and the context within which SBO terms can be used, as well as information about how conformant documents should be interpreted.

From a knowledge-engineering point of view, it makes sense to represent these sources of structural knowledge as part of a single knowledge base. Although, to a knowledge-engineer, this current separation of documents could appear arbitrary, it is in fact well-motivated according to consumers of each type of information. The portion of the knowledge codified in SBML transmits all of and only the information needed to parameterise and run a computational simulation of the system. The knowledge in SBO is intended to aid humans in understanding what is being modelled. The SBML Manual is aimed at tools developers needing to ensure that software developed is fully compliant with the specification.

[4] http://www.w3.org/2001/sw/ [5] http://www.w3.org/TR/owl-ref/ [6] http://obi.sf.net [7] http://biopaxwiki.org
[8] http://www.sbml.org [9] http://www.w3.org/XML/Schema [10] http://www.ebi.ac.uk/sbo
[11] http://www.obofoundry.org [12] http://prdownloads.sourceforge.net/sbml/sbml-level-2-version-3-rel-1.pdf?download

Only two of these three sources of structural knowledge are directly computationally amenable. SBML has an associated XSD that describes the range of legal XML documents, which elements and attributes must appear, and constraints on the values of text within the file. SBO captures a term hierarchy containing human-readable descriptions and labels for each term and a machine-readable ID for each term. Neither of these documents contains much information about how XML elements or SBO terms should be used in practice, how the two interact, or what a particular conformant SBML document should mean to an end-user. The majority of information required to develop a format-compliant model is in the SBML Manual, in formal English. Anything more than simple programmatic steps, such as XML validation, can currently only be done by manually encoding the English descriptions in the SBML Manual into rules in a program. libSBML[13] is the reference implementation of this procedure, capturing the process of validating constraints. Manual encoding provides scope for misinterpretation of the intent of the SBO Manual or may produce code that accepts or generates non-compliant documents due to silent bugs. In practice, these problems are ameliorated by regular SBML Hackathons[14] and the use of libSBML by many SBML applications. However, the need for a more formal and complete description of the information in the SBML Manual becomes more pressing as the community grows beyond the point where all of the relevant developer groups can be adequately served by face-to-face meetings.

We find that some of these issues can be avoided by combining the structural knowledge currently spread across three documents in three formats into a single computationally amenable resource. This method of constraint integration for all information pertinent to SBML will require a degree of rigour that can only improve the clarity of the specification. Once established, standard OWL tools can be used to validate and reason over SBML models, to check their conformance and to derive any conclusions that follow from the facts stated in the document, all without manual intervention.

To address this proposition, we have developed the Model Format OWL (MFO), implemented in OWL-DL and capturing the SBML structure plus a representative sample of SBO and human-readable constraints from the SMBL Manual. We demonstrate that MFO is capable of directly capturing many of the structural rules and semantic constraints documented in the SBML Manual. The mapping between SBML documents and the OWL representation is bi-directional: information can be parsed as OWL individuals from an SBML document, manipulated and studied, and then serialized back out again as SBML. We demonstrate feasibility with two simple, illustrative, examples. In future, we hope to use this as the basis for a method of automatically improving the annotation of SBML models with rich biological knowledge, and as an aid to principled automated model improvement and merging.

The integration of all structural knowledge for SBML models into a single resource creates a new style of model document development, which we believe will greatly reduce the overheads associated with computational transformations between biological knowledge and high-quality systems biology models. MFO is not intended to be a replacement for any of the APIs or software programs available to the SBML community today. It addresses the very specific need of a sub-community within SBML that wishes to be able to express their models in OWL for the purpose of reasoning, validation, and querying. It has also been created as the first step in a larger data integration strategy that will eventually encompass the biological as well as structural knowledge present in SBML documentation and models.

[13] http://www.sbml.org/software/libsbml/ [14] http://www.mas.ncl.ac.uk/~ncsg3/hackathon07/

1.2 Conventions

OWL class, property, and individual names are written in a `fixed-width font`. To distinguish between the SBML standards and MFO, XML elements and attribute names as well as SBO class names are written in a sans serif font. In MFO, names of individuals (instantiations of OWL classes) always begin with a lowercase letter, while names of OWL classes always begin with an uppercase letter.

1.3 Programmatic versus Ontological Validation

Libraries area already available for the manipulation and validation of SBML models, with libSBML being the most prominent example. The SBML website provides an online XML validator[15], and the SBML Toolbox[16] is also capable of XML validation of the structure of SBML using libSBML. The online validator uses only the SBML XSD to check the consistency of a model against the schema. The purpose of the SBML Toolbox is to integrate SBML models with the MATLAB environment, but it does have some validation features that check for the correct combination of components. The SBML Validation Suite[17] provides a set of test-cases meant to be used in software development, and can also be used to compare different software applications.

OWL provides additional mechanisms by which many of the structural constraints documented in the SMBL Manual can be captured directly in a computationally amenable form. Once represented in OWL, a reasoner can be applied in order to check for internal consistency and correct structure. There are some limitations to using ontology languages. Some of these are inherent to the logical requirements for these languages; there is an assumption that there exists an algorithm for deciding the true or false status of all possible combinations of facts that they allow to be stated. Others are due to the maturity of the current software. For example, DIG[18] reasoners (such as those used within the Protégé, a common ontology editing environment) cannot yet handle all of the expressivity of OWL-1.1. Specifically, such reasoners cannot reach all the expected conclusions based upon the value of properties connecting individuals to data-types, such as string, integer or Boolean values, even though it has been proven that algorithms to handle these cases do exist. In practice, for example, this limits the ability of a reasoner to decide if two SBML id values are identical. Such limitations will lessen with time as tools improve. With these limitations in mind, MFO currently represents some things as concepts rather than as data-types. This method of representation allows us, for example, to capture the complex relationships that hold between IDs themselves and classes that have IDs, in ways that are not possible currently in OWL if we were to represent the IDs directly as string data-types.

2 Results

2.1 Creation of the Model Format OWL

MFO is an OWL representation of the integration of the *structure* of SBML and the additional relations and constraints stored within SBO and the SBML Manual. Specific SBML model documents can be represented as sets of OWL "individuals" conforming to MFO classes.

[15] http://sbml.org/validator/ [16] http://sbml.org/software/sbmltoolbox/ [17] http://sbml.org/wiki/Validation_Suite
[18] http://dl.kr.org/dig/

Other ontology languages such as the Open Biological Ontologies[19] (OBO) format are useful for describing hierarchies of terms, but not for defining complex relationships between them. OWL is more useful when reduction of the ambiguity of interpretation is paramount[9], as it is with MFO. Facts sourced from SBML, SBO, and the SBML Manual are attributed in the OWL file using an rdfs:comment field with the phrasing described below.

- *Structural Element of SBML:* The classes with this comment have a direct association to a specific element or attribute of the SBML format. Examples from MFO include `Species`, `Reaction`, and `ModifierSpeciesReference`, all of which are named after the SBML elements of the same name.
- *Structural Element of XML:* The classes with this comment have a direct association to a specific part of an XML document. The only two OWL classes currently containing this statement are `Element` and `Attribute`, which correspond to XML elements and attributes, respectively. These classes store the information about how each SBML component is structured in XML.
- *Documented Constraint:* The classes with this comment capture documented constraints on certain types of elements or attributes. They are generally taken from the SBML Manual, and include the version of the document specifying the constraint. The `UnchangingQuantityBoundarySpecies` class, which can be inferred based on the values for the SBML attributes constant and boundaryCondition, has this comment.

Some MFO classes may have none of these comments. These classes have been created for a specific organizational purpose, but are not based on knowledge drawn from one of the three resources described. An example “helper” class is `SBMLPart`, which covers all of the SBML and SBO components, and serves to distinguish these from the XML hierarchy under the “helper” class `XMLPart` (see Supplemental Figure 1). However, many MFO classes have more than one of these comments. The most common combination is “Structural Element of SBML” combined with “Documented Constraint”, indicates that a specific SBML component is defined not only in the SBML XSD, but also in another resource such as the SBML Manual. One example is the MFO class `Reaction`, which only allows `ListOfModifiers`, `ListOfReactants`, and `ListOfProducts` classes as direct sub-elements. The documented constraint is found in Section 4.13.1 of the SBML Manual, while the SBML component itself is defined in the SBML XSD.

MFO does not yet contain all of the information described by the three SBML resources. We have begun by implementing portions which allow us to explore the utility of our OWL representation, focusing upon the necessary structures for modelling reaction *members*. Capturing the remaining information in MFO will continue as this project matures, until the first prototype of a fully usable product is available for final release.

2.1.1 Modelling the SBML XML Schema

SBML is a format for storing and exchanging information on biochemical networks. Its main encoding format is XML[1]. It is a good candidate for representation in OWL as the large amount of rules and constraints are described in detail, and the interlocking nature of those rules makes integrating them worthwhile. Also, this is a necessary first step in a much larger research effort which will see MFO layered with a number of other ontologies to create a rich picture of systems biology modelling.

[19] http://www.obofoundry.org/

It is useful to model the structure of an SBML document in terms of the XML itself and the relationships amongst the SMBL "classes" (as they are referred to in the SBML Manual). The generic structure of an XML document is represented in MFO with two classes, `Element` and `Attribute`. A high-level view of MFO, with `XMLPart` expanded, is provided in Supplemental Figure 1.

`XMLPart` and `SBMLPart` are disjoint, meaning that any given individual may not be both an `XMLPart` and an `SBMLPart` simultaneously. Each `SBMLPart` instance represents the data structure that results from processing a part of the XML document. Each `XMLPart` represents the portion of the XML document that was processed to generate that data structure. Each `SBMLPart` is realized as an `XMLPart` when serialized into an XML document. This structure is captured in OWL using the `realizedAs` object property. `SBMLPart` contains classes that are named after either SBML elements (e.g. SpeciesType) or attributes (e.g. sboTerm), and every child of `SBMLPart` must declare which `XMLPart` it is realized as in a SBML-formatted file. A graphical view of this region of the ontology is available in Supplemental Figure 2.

The relationships in classes belonging to `SBMLPart` can extend beyond a simple indication of their XML type. Nearly all of the documented constraints in the SBML manual can be added to this part of the ontology. The reward for the addition of such constraints is the creation of a rich ontology that can reason over and validate information drawn from SBML-formatted files. Some examples of this are described in detail in Section 2.1.3. In summary, the structure of an SBML document is represented in the classes under `SBMLPart`, while the documented constraints in the SBML Manual are represented mainly as relationships between and restrictions on these classes.

2.1.2 Modelling SBO

SBO is a hierarchy of terms developed by the systems biology modelling community. SBO was written to assist with model compliance to the Minimum Information Requested In the Annotation of Models (MIRIAM) checklist. By adding SBO terms, a modeller has taken an "essential step" towards such compliance[2]. SBO is intended to provide a link between a specific model component and the overall model structure. Each term in the SBO OWL file is related to its direct parent with an "is a" relationship. For instance in SBO, catalyst is_a modifier.

In SBML, each element that inherits from SBML:Sbase has an optional attribute called sboTerm, which links to a specific term in SBO. In MFO, this is represented as the object property `sboTerm`. Every individual of a class in the `SBMLPart` hierarchy may use this property except for those modelling the SBML "ListOf__" classes. Section 4.2.1 of the SBML Manual states that currently there are no sections of SBO that contain suitable terms for the ListOf__ classes, and therefore this restriction is built directly into MFO. The SBML Manual places restrictions on the location of particular SBO terms. For example, the only place that terms deriving from SBO_0000231 (event) can be used is on Reaction elements. Conversely, if Reaction elements have a term at all, it must be one deriving from event. This is one of the many constraints placed on the use of SBO terms in SBML documented within the SBML Manual, and not captured in either the SBML XSD or the SBO OBO file.

In MFO, each SBO term is a class whose ultimate parent is `SBOTerm`, and whose overall structure mirrors that of SBO**.** The "is a" property of each SBO term is main relation available within the SBO OBO file, and the only relationship available within the SBO OWL file. These relationships are mirrored directly in the MFO hierarchy underneath `SBOTerm`. For example, the physical participant SBO term is modeled in MFO as a sub-class of `SBOTerm`, `SBO_PhysicalParticipant`. A portion of the `SBMLTerm` hierarchy, and its placement within MFO, is shown in Supplemental Figure 3. Examples of classes that have restricted use of SBO terms can be found in Figure 1.

2.1.3 Modelling the SBML Manual

The SBML Manual is a highly important document that contains the majority of the rules and constraints that valid models must conform to. For example, sub-elements of the SBML ModifierSpeciesReference element should refer to only those Species which fall under the SBO modifier hierarchy (Figure 1a, SBML Manual section 4.13.4). Additionally, depending on the value of the constant and boundaryCondition attributes of the Species element, corresponding SpeciesReference elements may or may not be products or reactants in reactions (SBML Manual section 4.8.6, table 5). These kinds of correlation constraints can be captured in MFO, and some examples of these have been provided in Section 2.2.2. As the majority of constraints in the SBML Manual apply directly to SBML components, the addition of the rules does not add significantly to the hierarchy of MFO. However, it does add to the relations and properties between MFO classes, as can be seen in Figure 1.

Some classes of constraints rely upon the default value of attributes. Typically, default values are supplied by the XML parser when it processes the input file. If data is entered into MFO without the default values being filled in, a reasoner will be unable to infer that the missing values should take on the defaults. This inability is a direct result of the “open world” reasoning employed by OWL. In the future, if default values prove to be important for processing SBML files, and for some reason population of MFO with specific models does not fill these in automatically, it would be possible to make limited use of SWRL[20] to capture this information.

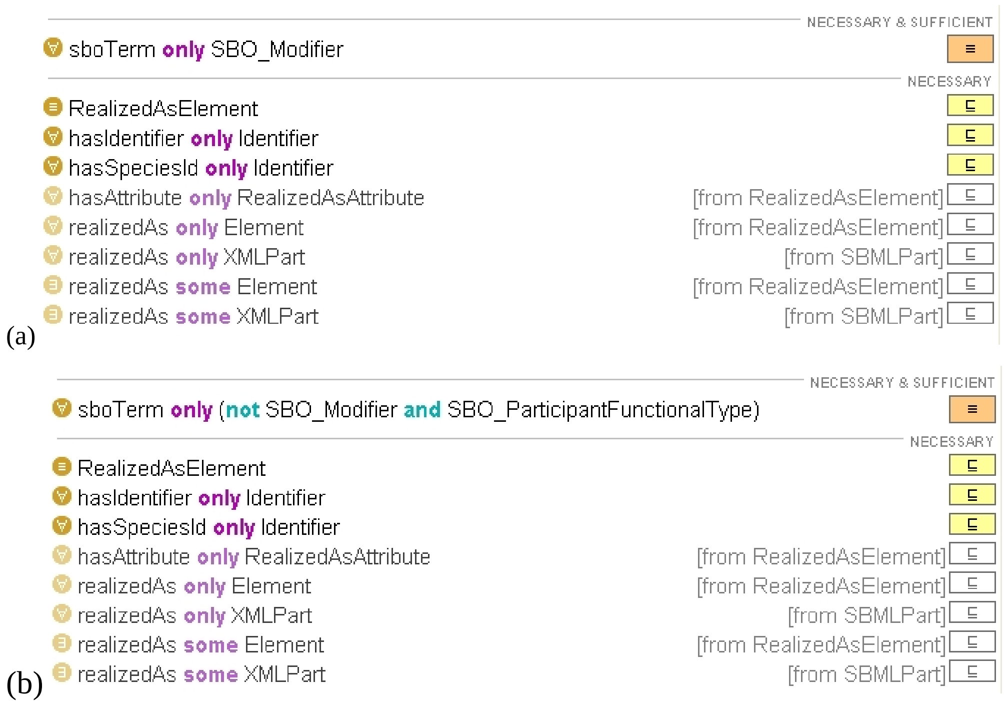
2.2 Reasoning over MFO

We illustrate the motivation of the research presented here with two illustrative examples: one is a straightforward but useful case of SBO term validation within a model, and the other is a more complex application of constraints found within the SBML Manual.

2.2.1 SBO Term Validation

The SBML Manual provides constraints on the use of SBO terms in each element. However, these constraints are in multiple sections of the document, making even manual interpretation of all of the rules difficult. For instance, table 7 in section 5.2.2 of the SBML Manual describes the limitation of the sboTerm attribute for both ModifierSpeciesReference and SpeciesReference to terms in the Functional Participant (SBO_0000003) hierarchy. In section 4.13.4, ModifierSpeciesReference has a further restriction limiting the sboTerm attribute to children of Modifier (SBO_0000019), itself a child of Functional Participant. There is no explicit mention of this term, but the manual states that the express purpose of ModifierSpeciesReference is to hold references to modifiers for a given reaction. Figure 1 shows the MFO terms and properties for these elements.

[20] http://www.w3.org/Submission/SWRL/



(a)

```
                                                    NECESSARY & SUFFICIENT
sboTerm only SBO_Modifier                                                ≡
                                                                 NECESSARY
RealizedAsElement                                                        ⊑
hasIdentifier only Identifier                                            ⊑
hasSpeciesId only Identifier                                             ⊑
hasAttribute only RealizedAsAttribute          [from RealizedAsElement]  ⊑
realizedAs only Element                        [from RealizedAsElement]  ⊑
realizedAs only XMLPart                                [from SBMLPart]   ⊑
realizedAs some Element                        [from RealizedAsElement]  ⊑
realizedAs some XMLPart                                [from SBMLPart]   ⊑
```

(b)

```
                                                    NECESSARY & SUFFICIENT
sboTerm only (not SBO_Modifier and SBO_ParticipantFunctionalType)        ≡
                                                                 NECESSARY
RealizedAsElement                                                        ⊑
hasIdentifier only Identifier                                            ⊑
hasSpeciesId only Identifier                                             ⊑
hasAttribute only RealizedAsAttribute          [from RealizedAsElement]  ⊑
realizedAs only Element                        [from RealizedAsElement]  ⊑
realizedAs only XMLPart                                [from SBMLPart]   ⊑
realizedAs some Element                        [from RealizedAsElement]  ⊑
realizedAs some XMLPart                                [from SBMLPart]   ⊑
```

Figure 1. (a) The properties of the MFO class `ModifierSpeciesReference`. (b) The properties of the MFO class `SpeciesReference`. The position of both of these classes in the MFO hierarchy is visible in Supplemental Figure 2.

The constraints in Figure 1 have been converted from a human- to a computationally-readable format. Incorporating the definitions in the SBML Manual for SpeciesReference and ModifierSpeciesReference requires the creation of a set of OWL property restrictions to describe them, such as "`sboTerm only sbo_modifier`" in Figure 1a. This means that, if there is an SBO term used at all in the ModifierSpeciesReference element in an SBML model, then it must be from the SBO modifier hierarchy. Some statements in the SBML Manual are not completely straightforward. In the same section (4.13.4), the SBML Manual makes the seemingly contradictory statement that modifiers are allowed both in ModifierSpeciesReference elements and in SpeciesReference elements, as long as they are present in both the ListOfReactants *and* ListofProducts of that reaction. However, the purpose of the ListOfModifiers is to contain ModifierSpeciesReference elements explicitly as a "way to express which species act as modifiers in a given reaction." Therefore, while one part of the manual indicates that these lists are not disjoint, the intended usage of the Modifier hierarchy and ModifierSpeciesReference indicates that they are. In the current version of MFO, we have made the arbitrary decision to make `ModifierSpeciesReference` and `SpeciesReference` properties disjoint. This is an example of where representing these constraints in OWL has revealed an ambiguity in the SBML Manual. Systematically addressing cases like this should lead to an improvement in its clarity.

The restriction on `sboTerm` is utilized by Protégé 3.3 Beta when creating individuals of `ModifierSpeciesReference` in MFO. Thus, when a user attempts to populate the `sboTerm` property of `ModifierSpeciesReference`, Protégé presents the user with the correct range of SBO terms (as shown in Supplemental Figure 4). Creation-time restriction prevents incorrect `SBOTerm` classes from being added manually. However, if the OWL file were modified outside of Protégé, or if the restriction is too complex for Protégé to interpret at creation-time, it is possible that the wrong terms could be used. These kinds of errors are the sort of things that reasoners will discover as logical inconsistencies. The main way of checking the consistency of an ontology and its individuals is by reasoning over it. Reasoners test the *asserted* hierarchy created by the ontology developer, and classify all instances into the most specific class or classes that they conform to. We have found it useful during development to reason over MFO after each change to the ontology, including the addition of new classes or individuals, and particularly when modifying restrictions over properties.

To validate that asserting an incorrect `SBOTerm` to a `SpeciesReference` triggers an inconsistency during reasoning, we introduced an individual of `SpeciesReference` called `exampleSpeciesReference,` with the `sboTerm` value of `sbo_catalyst`. `sbo_catalyst` is an individual of the class `SBO_Catalyst`, which is a `SBO_Modifier` and therefore falls outside the range allowed by the definition of `SpeciesReference`. Figure 2 shows the output of Pellet[7] when run over the ontology containing `exampleSpeciesReference.`

```
Input file: MFO.owl

 WARN [main] (KnowledgeBase.java:1527) - Inconsistent ontology. Reason:
Individual http://www.cisban.ac.uk/downloads/MFO.owl#sbo_catalyst is forced
to belong to class http://www.cisban.ac.uk/downloads/MFO.owl#SBO_Modifier
and its complement

Consistent: No
```

Figure 2: Relevant Pellet reasoner output for MFO containing `exampleSpeciesReference`, which itself contains an out-of-scope `SBOTerm`.

Therefore, by reasoning over the ontology, the error in the model is revealed. The message in Figure 2 describes the state of `sbo_catalyst`, which simultaneously fulfills the concepts:

- `sbo_catalyst` must belong to `SBO_Modifier` due to its placement in the ontology, and
- `sbo_catalyst` must *not* belong to `SBO_Modifier`, due to the restriction placed on `exampleSpeciesReference`, which does not allow individuals from the `SBO_Modifier` hierarchy.

As `sbo_catalyst` cannot fulfil both of these requirements at once, the ontology is deemed inconsistent. This works for all defined ranges in table 7 of the SBML Manual covered by MFO, as well as for those SBML elements, specifically the "ListOf__" elements, which must not have an associated SBO term (SBML Manual, section 4.2.1). Figure 3 shows the result from Pellet when an individual of type `SBO_Macromolecule` (named `sbo_macromolecule`), has been added via the `sboTerm` property to `exampleListOfModifiers`, which belongs to the class `ListOfModifiers`.

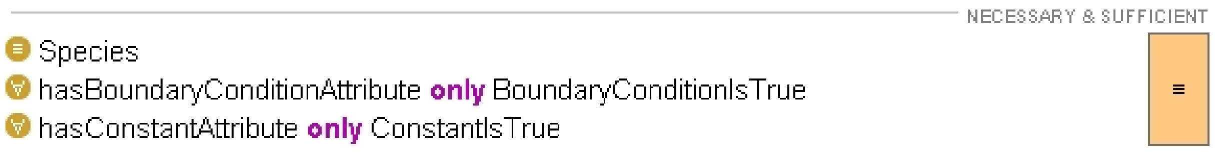
```
Input file: MFO.owl

Consistent: No

 WARN [main] (KnowledgeBase.java:1527) - Inconsistent ontology. Reason:
Individual http://www.cisban.ac.uk/downloads/MFO.owl#sbo_macromolecule is
forced to belong to class
http://www.cisban.ac.uk/downloads/MFO.owl#SBO_PhysicalParticipant and its
complement
```

Figure 3: Relevant Pellet reasoner output for MFO containing `exampleListOfModifiers`, which itself contains a disallowed `SBOTerm`.

As seen in Figure 3, the reasoner finds the ontology inconsistent, as an `SBOTerm` has been used where it is not allowed. The logic applied to MFO is the same as that for the `sbo_catalyst` example above: the definition of `sbo_macromolecule` required it to be an `SBOTerm`, while the rule against using any individual of `SBOTerm` in `ListOfModifiers` forced `sbo_macromolecule` to not belong to `SBOTerm`.

2.2.2 Inferring More Specific Subtypes of SBML Components

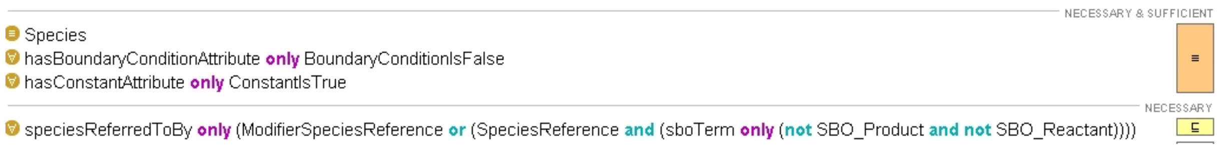
Section 4.8.6, table 5 of the SBML Manual provides a table of all possible combinations of the Boolean attributes constant and boundaryCondition for Species elements. The quantity of Species elements depends upon the values of these attributes. For instance, if both are true, the quantity of the Species can not be altered by the reaction system and therefore its value will not change during a simulation run (Figure 4). If the Species has constant set to true and boundaryCondition set to false, then the Species can not be referenced as either a product or a reactant anywhere else in the SBML document (Figure 5). MFO models all four possible combinations of these Boolean attributes as four distinct sub-types of `Species`.


NECESSARY & SUFFICIENT

Species
hasBoundaryConditionAttribute **only** BoundaryConditionIsTrue
hasConstantAttribute **only** ConstantIsTrue

≡

Figure 4: The specific relationships defining the `ConstantBoundarySpecies` class.


NECESSARY & SUFFICIENT

Species
hasBoundaryConditionAttribute **only** BoundaryConditionIsFalse
hasConstantAttribute **only** ConstantIsTrue

≡

NECESSARY

speciesReferredToBy **only** (ModifierSpeciesReference **or** (SpeciesReference **and** (sboTerm **only** (**not** SBO_Product **and** **not** SBO_Reactant))))

⊑

Figure 5: The specific relationships defining the `ConstantNonBoundarySpecies` class.

The placement of a given individual in the `Species` hierarchy can be accomplished by inferring its location from the information provided by an SBML Model. As an example, if an individual named `exampleToBeInferred` is created, and we assert that it is a `Species` and that both its boundary condition and its constant values are true, Pellet will correctly classify this individual as a `ConstantBoundarySpecies` (Figure 6).

```
owl:Thing
    MFO:RealizedAsAttribute
       MFO:BoundaryCondition
          MFO:BoundaryConditionIsTrue
          MFO:BoundaryConditionIsFalse
       MFO:Constant
          MFO:ConstantIsFalse
          MFO:ConstantIsTrue
    MFO:RealizedAsElement
       MFO:SpeciesReference
       MFO:Species
          MFO:ConstantBoundarySpecies - (MFO:exampleToBeInferred)
          MFO:ChangeableNonBoundarySpecies
          MFO:ChangeableBoundarySpecies
          MFO:ConstantNonBoundarySpecies
```

Figure 6: The output of the Pellet reasoner, showing the inference of the `exampleToBeInferred` individual into the correct class, `ConstantBoundarySpecies`.

Supplemental Table 1 shows details of the time required to check the consistency of MFO, classify the asserted hierarchy, and infer the correct placement of an individual such as the one in Figure 6 using two different reasoners.

3 Discussion

Here we present a method for the computational representation of the structural information required to build and validate an SBML model. Currently this information is found in three sources to meet the needs of distinct user groups. Whilst optimal for the existing range of human-driven applications, this disparate arrangement is not suitable for computational approaches to model construction and annotation. This is primarily because much of the knowledge about what constitutes valid and meaningful SBML models is tied up in non-machine-readable formats. Here we propose an approach complementary to the existing paradigm that will ultimately facilitate the development of automated strategies for model improvement and manipulation by drawing upon internal consistency and inference capabilities.

We have developed MFO, an OWL representation of the structure and structural constraints for SBML documents. We addressed two use-cases where constraints are documented in the SBML Manual, but not in the SBML XSD. We demonstrate how to capture constraints on SBO terms associated with SBML elements, and show examples of reasoner output validating this usage. We also present a method to encode more complex constraints on the Species element, which alter where the species can legally be referenced within a document.

The value of this reasoning is two-fold. Firstly, in the case where we have a complete model, the reasoner can automatically enforce constraints that currently are only described textually in the SBML Manual. Secondly, for an incomplete model, the reasoner can additionally deduce the range of legal values for missing portions. For example, given a Species that is referenced as a product or reactant, the reasoner can deduce that this Species element can not have both the constant attribute set to true and the boundary condition attribute set to false.

Libraries such as libSBML capture the process of validating constraints, while MFO explicitly captures the constraints themselves, thus identifying where the SBML Manual is not explicit or appears to contradict itself. This ongoing process will improve the quality of the SBML Manual. Once captured in MFO, these constraints can be investigated systematically by running a reasoner. This allows many aspects of models to be validated without the development of rule-specific code. Furthermore, more complete capture of constraints will not require changes to supporting software. Whilst these illustrative examples presented are fairly simple, they will become more complex and real-world oriented as MFO grows.

We envisage further longer-term benefits of this approach that will build upon the flexibility of the OWL representation. The use of MFO should enable a greater degree of automation in the process of adding value to existing models via a formal framework for the integration of additional data. This process will necessarily include such semantically complex operations as manipulating, annotating, merging, enhancing and validating models. Ultimately, the approach will be extended to move towards the automatic construction of skeleton models. A SBML - MFO - SMBL converter is under development which will allow development of models to occur in OWL. A user will then be able to create individuals in MFO and export them as SBML, employing the ontology to do all the “hard work” of ensuring consistency and correctness. This tool will be made available on the supporting web site shortly. MFO is in active development. The coverage of the ontology is increasing over time, and the ultimate aim is to incorporate all of SBO, the SBML structure, and as much of the SBML Manual as possible. We hope to work closely with the SBML community in this endeavour.

Information on and download of MFO is available from http://www.cisban.ac.uk/MFO/.

4 Software Versions Used

The following versions of documents and programs were used in this work:

- As input for MFO: SBML Level 2 Version 3 Release 1, http://www.sbml.org; "Structures and Facilities for Model Definition" document for SBML Level 2 Version 3 Release 1, http://www.sbml.org; SBO in OWL, Downloaded 15 June 2007 from http://www.ebi.ac.uk/sbo/.
- Ontology Editors: Protégé 3.3 Beta (also used to create the figures shown in this paper), http://protege.stanford.edu/; Protégé 4 Alpha, http://protege.stanford.edu/; SWOOP Version 2.2.2, http://code.google.com/p/swoop/.
- Reasoners: Pellet[7] Version 1.5, http://pellet.owldl.com/; FaCT++[8] Version 1.1.8, http://owl.man.ac.uk/factplusplus/.

5 Acknowledgements

ALL is funded by the BBSRC and EPSRC as part of CISBAN, and MP by ComparaGrid (BBSRC grant ref: BBS/B/1731). We gratefully acknowledge the support of the Newcastle Systems Biology Resource centre, the Newcastle Bioinformatics Support Unit, and the North East regional e-Science centre. We thank P. Lord for help with consistency checks of MFO and N. Le Novère, C. Whitaker, J. Hallinan, C. Wipat and F. Gibson for proofreading the manuscript.

6 References

[1] Hucka, M. *et al.*: The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics (Oxford, England) **19** (2003) 524-531

[2] Le Novere, N.: Model storage, exchange and integration. BMC Neurosci **7 Suppl 1** (2006) S11

[3] Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. J. of Web Semantics **1** (2003) 7-26

[4] Soldatova, L.N., King, R.D.: An ontology of scientific experiments. Journal of the Royal Society, Interface / the Royal Society **3** (2006) 795-803

[5] Heja, G., Varga, P., Pallinger, P., Surjan, G.: Restructuring the foundational model of anatomy. Studies in health technology and informatics **124** (2006) 755-760

[6] Heja, G., Surjan, G., Lukacsy, G., Pallinger, P., Gergely, M.: GALEN based formal representation of ICD10. International journal of medical informatics **76** (2007) 118-123

[7] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., Katz, Y. : Pellet: A practical OWL-DL reasoner. Journal of Web Semantics **5** (2007)

[8] Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. Lecture Notes in Artificial Intelligence **4130** (2006) 292-297

[9] Aranguren, M.E., Bechhofer, S., Lord, P., Sattler, U., Stevens, R.: Understanding and using the meaning of statements in a bio-ontology: recasting the Gene Ontology in OWL. BMC bioinformatics **8** (2007) 57