Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Original Paper

OXFORD

Systems biology

SBML2Modelica: Integrating biochemical models within open-standard simulation ecosystems

F. Maggioli [1], T. Mancini [1]*, E. Tronci [1]

[1] Computer Science Department, Sapienza University of Rome, Italy

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: SBML is the most widespread language for the definition of biochemical models. Although dozens of SBML simulators are available, there is a general lack of support to the integration of SBML models within open-standard general-purpose simulation ecosystems. This hinders co-simulation and integration of SBML models within larger model networks, in order to, *e.g.,* enable *in-silico clinical trials* of drugs, pharmacological protocols, or engineering artefacts such as biomedical devices against Virtual Physiological Human models.
Modelica is one of the most popular existing open-standard general-purpose simulation languages, supported by many simulators. Modelica models are especially suited for the definition of complex networks of heterogeneous models from virtually all application domains. Models written in Modelica (and in 100+ other languages) can be readily exported into *black-box Functional Mock-Up Units (FMUs)*, and seamlessly co-simulated and integrated into larger model networks within *open-standard language-independent simulation ecosystems*.
Results: In order to enable SBML model integration within heterogeneous model networks, we present SBML2Modelica, a software system translating SBML models into well-structured, user-intelligible, easily modifiable Modelica models. SBML2Modelica is SBML Level 3 Version 2–compliant and succeeds on 96.47% of the SBML Test Suite Core (with a few rare, intricate, and easily avoidable combinations of constructs unsupported and cleanly signalled to the user). Our experimental campaign on 613 models from the BioModels database (with up to 5438 variables) shows that the major open-source (*general-purpose*) Modelica and FMU simulators achieve performance comparable to state-of-the-art *specialised* SBML simulators.
Availability and Implementation: SBML2Modelica is written in Java and is freely available for non-commercial use at `https://bitbucket.org/mclab/sbml2modelica`
Contact: T. Mancini, tmancini@di.uniroma1.it

1 Introduction

The mathematical modelling and simulation of biochemical systems is of paramount importance in several areas, *e.g.,* computational and systems biology, model-based pharmacology, chemistry. The current *de facto* standard for *modelling* biochemical systems is *Systems Biology Markup Language* (SBML, Hucka *et al.*, 2003, `http://www.sbml.org`), an XML-based markup language allowing the definition of biochemical models in terms of reactions, species, compartments, and parameters. SBML allows the quantitative modelling of various kinds of biological phenomena, including metabolic networks, cell signalling pathways, regulatory networks, infectious diseases, just to mention a few.

Simulation of SBML models of practical relevance is crucial for their analysis, as they are often too large or intricate for being analysed statically. Indeed, many third-party simulators of SBML models have been developed and are currently publicly available (see, *e.g.,* the SBML web-site).

1.1 Motivations

Available SBML simulators do not fully support the integration, within open-standard simulation ecosystems, of SBML models with models defined using *other* languages. This severely hinders the possibility to *co-simulate* and *integrate* SBML models within large *model networks* comprising biochemical as well as other kinds of models, possibly at different levels of abstraction (*multi-scale* model networks, see, *e.g.,* de Bono and Hunter, 2012), and applying standard systems engineering approaches for the model-based analysis of such heterogeneous model networks.

For example, the interconnection of quantitative models of the human physiology (*e.g.,* Physiomodel, Mateják and Kofránek, 2015), drugs pharmacokinetics/pharmacodynamics (*e.g.,* Open Systems Pharmacology Suite, Eissing *et al.*, 2011), (possibly semi-autonomous) biomedical devices, pharmacological protocol guidelines or treatment schemes, enables the set-up of *in silico clinical trials* for the (model-based) safety and efficacy pre-clinical assessment of such drugs, protocols, treatments, devices, using *standard system engineering approaches* to perform their simulation-based analysis at system level (see, *e.g.,* Kanade *et al.*, 2009; Mancini *et al.*, 2013, 2014; Zuliani *et al.*, 2013; Zuliani, 2015; Mancini *et al.*, 2016a, 2017). Works in this direction include, *e.g.,* (Schaller *et al.*, 2016; Messori *et al.*, 2018), where a model-based verification activity of a sensor-augmented insulin pump is conducted against a model of the human glucose metabolism in patients with diabetes mellitus, (Madec *et al.*, 2019), where a model of a penicillin bio-sensor (integrating biochemistry, electrochemistry, and electronics models) is simulated to compute a first dimensioning of the sensor, and (Tronci *et al.*, 2014; Mancini *et al.*, 2015), where representative populations of virtual patients are generated from parametric models of the human physiology, a key step to enable *in silico* clinical trials (see, e.g., Mancini *et al.*, 2018).

One of the most widely adopted open-standard languages for modelling dynamical systems is Modelica (`http://www.modelica.org`), a general-purpose fully-fledged language based on ordinary differential as well as algebraic equations plus procedural snippets. The language supports object-orientation and allows the definition of complex systems as networks of smaller subsystems.

Modelica is widespread in application domains as diverse as mechanical, electrical, electronic, hydraulic, thermal, control, electric engineering, but also physiology and pharmacology (see, *e.g.,* Mateják and Kofránek, 2015), and several efficient and highly-configurable simulators are currently available: proprietary (*e.g.,* Dymola and Wolfram System Modeler) as well as open-source (*e.g.,* OpenModelica and JModelica).

A Modelica model can also be easily exported into a Functional Mock-Up Unit (FMU), an executable *opaque* (binary) object implementing the Functional Mock-Up Interface (FMI, `http://fmi-standard.org`), one of the currently most widespread open standards for model exchange, integration and co-simulation. Being *black-box*, FMU models can be shared or integrated within larger model networks while protecting their *intellectual property* (see, *e.g.,* Mancini *et al.*, 2016b). This is crucial when sharing, integrating, or co-simulating models coming from different providers (*e.g.,* pharma companies, or manufacturers of novel biomedical devices). The FMI/FMU standard is currently supported by more than 100 simulators for *virtually all* application domains, making it the largest open-standard ecosystem for (language-independent) model exchange, integration, and co-simulation.

1.2 Contributions

In this paper we present SBML2Modelica, a software system that translates SBML models into well-modularised user-intelligible *Modelica* code, which preserves both the structure and the documentation of the input SBML models. The generated Modelica models can then be easily modified, integrated within other models, and can be readily run using *any* available Modelica simulator. Furthermore, the generated Modelica models can be easily exported into FMUs, thus allowing their seamless co-simulation and integration into model networks within open-standard language-independent simulation ecosystems (a helper tool is provided in the SBML2Modelica repository which generates a FMU directly from an SBML model by leveraging the JModelica API).

SBML2Modelica is compliant to the *latest* SBML standard (SBML Level 3 Version 2, Hucka *et al.*, 2018) and succeeds on 96.47% of the SBML Test Suite Core v3.3.0 (see Section 3.1), with a few rare, intricate, and easily avoidable combinations of constructs (see Section 4) unsupported and cleanly signalled to the user. Furthermore, our experimental campaign on 613 models from the BioModels database (with up to 5438 variables) shows that major open-source (*general-purpose*) Modelica and FMU simulators (OpenModelica and JModelica, with the latter that converts the input Modelica model into an FMU and then simulates such an FMU), when used in their default configurations achieve performance comparable to state-of-the-art *specialised* SBML simulators (see Section 3.2).

SBML2Modelica can be freely downloaded for non-commercial uses. The system has been implemented in Java and can be executed on all platforms for which a Java Virtual Machine is available. This includes most computer operating systems.

1.3 Available SBML simulators

A plethora of systems for the simulation of models written in SBML are currently available (most of them are listed in the SBML web-site), and a comprehensive review of them is out of the scope of this paper. We note, however, that both their functionalities and compliance to the SBML standard is highly variable.

In particular, only for six systems a certified report of their compliance to the official SBML Test Suite Core is (at the time of writing) publicly available (see the SBML web-site). Some of such systems (namely: libRoadRunner, Somogyi *et al.*, 2015; libSBMLSim, Takizawa *et al.*, 2013; and Simulation Core Library, Keller *et al.*, 2013) are pure SBML simulators, allowing the user to numerically simulate the SBML model given as input. All of them support a previous SBML standard (SBML Level 3 Version 1), while SBML2Modelica supports the *latest* standard (SBML Level 3 Version 2) with only a few minor limitations (see Section 4).

The other systems (namely: BioUML, Kolpakov *et al.*, 2019; iBioSim, Myers *et al.*, 2009; and COmplex PAthways SImulator –COPASI, Lee *et al.*, 2006) are more general platforms that, beyond model simulation, allow the user to modify, extend, and connect different SBML models together. Of them, only BioUML supports SBML Level 3 Version 2, as iBioSim and COPASI only support the older SBML Level 3 Version 1.

By translating SBML models into an open-standard general-purpose widely-adopted simulation language as Modelica (preserving both the structure and the documentation of the input models), SBML2Modelica not only allows simulation of the generated models (using any Modelica simulator), but also opens up a huge plethora of new possibilities to integrate SBML biochemical models with models of other kinds of systems (see Section 1.1) written in languages different than SBML.

Enabling interoperability and integration of biochemical models into cross-domain model networks has been strongly advocated. Attempts in this directions include, *e.g.,* (Madec *et al.*, 2017), where biochemical models are converted into Spice, a standard integrated electronic circuit simulator, for the model-based design of bio-sensors and labs-on-chip. Model conversion is performed exploiting clever analogies between the behaviour of biochemical systems and electronic circuits and between molecular diffusion and heat diffusion (Gendrault *et al.*, 2014; Madec

et al., 2019). SBML2Modelica acts at a higher level, by translating SBML models into a genuinely general-purpose cross-domain open system modelling language (Modelica), hence enabling seamless integration and co-simulation of SBML models with models of virtually *all* application domains, without the need to exploit cross-domain analogies, hence fully preserving model readability and extensibility. The possibility to export Modelica models into FMUs is one step further, allowing model integration and co-simulation in a language-independent way.

Previous approaches to translate SBML models into *general-purpose* simulation platforms include the SimBiology Matlab toolbox and Wolfram SystemModeler (a proprietary simulator accepting the Modelica language) with the BioChem plug-in (Larsdotter Nilsson and Fritzson, 2003; Fritzson *et al.*, 2007). In particular, by providing a user-friendly interface and high-level library abstractions, the BioChem SystemModeler plug-in allows the definition of visually appealing biochemical networks in some Modelica editors. Differently from SBML2Modelica, both SimBiology and SystemModeler+BioChem are based on commercial simulators and support only subsets of older SBML standards (SBML Level 3 Version 1 and SBML Level 2 Version 4 respectively), with several major limitations, including lack of support of delayed and prioritised events. Also, for none of them a compliance assessment to the SBML Test Suite core is available.

2 Materials and methods

In the following, we briefly outline the main structure of an SBML (Section 2.1) and of a Modelica model (Section 2.2), before sketching (Section 2.3) how SBML2Modelica generates a structured Modelica model from an input SBML model.

2.1 High-level view of an SBML model

Here we recall the main constructs of SBML, namely: parameters, compartments, species, and events. The reader interested to a more in-depth description is referred to the official SBML web-site (`http://www.sbml.org`) for the full language specification.

Parameters denote quantities with a symbolic name. Such quantities can be either constant or varying during model evolution.

Compartments denote containers of a particular type and positive *size* (possibly varying during model evolution).

Species represent model entities (*e.g.,* biochemical substances), whose *amount* may vary during model evolution. Each species belongs to a compartment. Species may take part to *reactions*. At any time, the *concentration* of a species in its compartment is defined as $\frac{\text{amount}}{\text{size}}$, where size is the size of the species compartment at that time.

Model parameters, species, and size of compartments are defined by means of *model variables* and can be assigned to values. An *initial assignment* defines the value of a model variable at time 0.

Reactions are statements describing any transformation, transport or binding process that changes the amount of one or more species. A reaction of the form $\alpha \rightarrow \beta$ (where α and β are *mixtures*, defined as linear combinations of species, *e.g.,* $\alpha = s_1 + 2s_2$, $\beta = s_3$, where s_1, s_2, s_3 are species) describes how (and how much of) certain species (those in α, called reactants) are transformed into certain other species (those in β, called products). Reactions have associated *kinetic rate expressions* that describe how quickly they take place.

According to the SBML specification, for any species s, the set of reactions $R_1, \ldots, R_n$ in which s occurs (together with their associated kinetic rate expressions $k_{R_1}, \ldots, k_{R_n}$) collectively define the time derivative of the available amount of s as:

$$\frac{\mathrm{d}s}{\mathrm{d}t} = \sum_{i=1}^{n} k_{R_i} \times \nu(s, R_i) \tag{1}$$

where $\nu(s, R_i)$ is the sum of the coefficients that multiply the occurrences of s in reaction R_i. Coefficients occurring on the left side of R_i are multiplied by -1 in order to model species consumption, while those occurring on the right side are taken as they are in order to model species production (see forthcoming Example 1).

Events represent *instantaneous* and *discontinuous* changes in the value of some quantities (*e.g.,* amounts of species, parameters, size of compartments) of the model. An event is defined in terms of a *trigger condition* (a Boolean formula), and a set of *assignments*, which update some model variables when the event *occurs* (*i.e.,* when the trigger condition switches from false to true). Optionally, events can be *delayed* by a certain time interval, whose length could change during model evolution. To avoid that two events occurring at the same time assign different values to the same variable, a *priority* expression (on the model variables) can be defined for events. Event priority expressions, evaluated when events occur, define the order in which concurrent events must be handled.

SBML events can be either persistent or non-persistent. Let e be an event, t be the time instant when the trigger condition of e becomes true, and d be the event delay. *Non-persistent* (respectively, *persistent*) event e must be executed at time instant $t + d$ *only if* (respectively, *regardless of whether*) the trigger condition remains true during the whole delay period (*i.e.,* from time t to time $t + d$).

Rules provide additional means to define the values of variables in a model in ways that cannot be expressed using reactions or initial assignments. The following three types of rules are provided (below, $\boldsymbol{V}$ is a set of –possibly all– model variables). *Algebraic rules* are of the form $f(\boldsymbol{V}) = 0$. *Assignment rules* are of the form: $x = f(\boldsymbol{V} - \{x\})$. Finally, *rate rules* define the rate of change of a model variable and are of the form: $\frac{\mathrm{d}x}{\mathrm{d}t} = f(\boldsymbol{V})$.

Example 1 shows a simple SBML model that will be used as a running example when outlining how SBML2Modelica works (Section 2.3). Although the model is clearly artificial and might not recall a known biochemical mechanism, it has the merit of compactly showing all the most important SBML constructs that we will address in the remainder of the paper.

Example 1 (Running example). *Our SBML model (whose code is available in the SBML2Modelica repository) consists of the following elements:*

Parameters: *p_1 with constant value 10^{-6} [$l\ sec^{-1}$]; p_2 whose value is initially set to 1 [$mol^{-1} sec^{-1}$]; p_3 with constant value 10^{-3} [mol]; p_4 whose initial value is set to 300 [sec].*

Species *(all in [mol]): s_1, s_2, s_3 (initially set to 10^{-3}), and s_4.*

Compartments: *One compartment* c *containing all the species.*

Reactions: *Reaction $R : s_1 + 2s_2 \longrightarrow s_3$ with kinetic rate expression $k_R = p_2 s_1 s_2$ [$mol\ sec^{-1}$].*

Events:

- *e_1 with trigger condition $s_1 s_2 \leq 10^{-7} \vee s_3 s_4 \leq 10^{-7}$ and priority equal to s_4. When the event is triggered, parameter p_2 is set to 0.*
- *e_2 with the same trigger condition as e_1, but with a delay of p_4 and priority equal to s_2. When triggered, parameter p_2 is set to -1 and parameter p_4 is set to 0.*

Rules:

- *Rate rule r_1, defining $\frac{\mathrm{d}p_2}{\mathrm{d}t} = 0$;*
- *Assignment rule r_2, which sets the* size *of compartment* c *to $1 + p_1 \times t$ [l], where* t *is the value of the current time-instant;*
- *Algebraic rule r_3, which imposes that constraint $\frac{s_2}{s_1} - \frac{s_3}{s_4} = 0$ holds at all time points.*

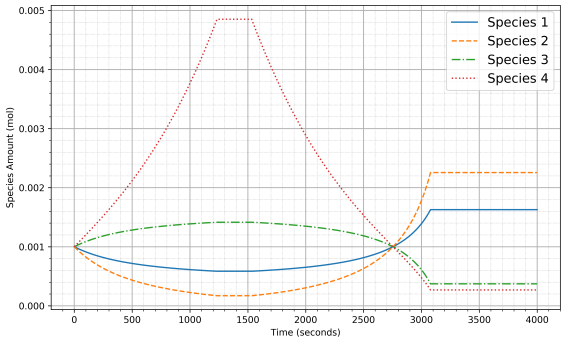

Fig. 1. Time evolution of the SBML model of Example 1.

Initial assignment *which sets the value of s_2 to p_3 at time zero.*

The model in Example 1 comprises 4 species $(s_1, \ldots, s_4)$ all belonging to a single compartment (whose size is constantly increasing as dictated by assignment rule r_2). The time-evolution of the available amount of s_1, s_2, and s_3 is governed by reaction R, while that of s_4 is determined by algebraic rule r_3 (hence, s_4 is always equal to $\frac{s_1 s_3}{s_2}$). The kinetic rate expression k_R of reaction R dictates how quickly the reaction takes place. In our example, reaction R defines the following time-derivatives for the available amounts of the involved species, according to Eq. (1): $\frac{\mathrm{d}s_1}{\mathrm{d}t} = -k_R$, $\frac{\mathrm{d}s_2}{\mathrm{d}t} = -2k_R$, $\frac{\mathrm{d}s_3}{\mathrm{d}t} = k_R$. However, k_R is not constant in time, and, moreover, depends on parameter p_2, whose value is affected by the two events e_1 and e_2. This makes our model not trivial.

Figure 1 shows the time evolution of the available amount of each species of our model, when starting from the initial state, where the available amount of each species is 10^{-3}, $p_1 = 10^{-6}$, $p_2 = 1$, $p_3 = 10^{-3}$, $p_4 = 300$. With $p_2 = 1$, k_R is positive, hence R defines a reaction where s_1 and s_2 are consumed in favour of the production of s_3. When, at time 1233.10 [sec], $s_1 s_2$ becomes $\leq 10^{-7}$ both events e_1 and e_2 are triggered. Given that e_2 has a *delay* of $p_4 = 300$ [sec], e_1 is processed immediately, while e_2 is processed only after 300 more seconds. This implies that p_2 is immediately set to 0 (as dictated by e_1). The kinetic rate expression k_R of reaction R is thus set to 0 and the system stabilises. When, at time 1533.10 [sec], also e_2 is processed, parameter p_2 is set to -1 and p_4 to 0. The new value for p_2 makes k_R negative, hence reaction R turns into modelling the consumption of s_3 in favour of the production of s_1 and s_2. This behaviour continues until time 3078.48 [sec], when $s_3 s_4$ becomes $\leq 10^{-7}$, thus triggering again both e_1 and e_2. This time, however, e_2 has a delay of $p_4 = 0$ [sec], hence e_1 and e_2 are now triggered and processed *simultaneously*. Since the priority of e_2 (*i.e.,* s_2) is higher than the priority of e_1 (*i.e.,* s_4), the SBML specification stipulates that the two events are processed in the order e_2, e_1. This implies that p_2 is first set to -1 and then (at the *same* time point) to 0. With p_2 being set to 0, also k_R becomes 0 and the system stabilises again.

2.2 High-level structure of a Modelica model

Modelica is an object-oriented language for the definition of systems of differential-algebraic equations. Below, we briefly recall the general structure of a Modelica model. The reader interested to a more in-depth description is referred to the official Modelica web-site (`http://www.modelica.org`) for the full language specification.

A Modelica model is a network of *objects*. Each object defines a set of *variables*, initial assignments as well as differential and algebraic *equations* for them, *events* and *algorithmic* sections. At any time point, the state of a model is the value of the variables belonging to all its objects.

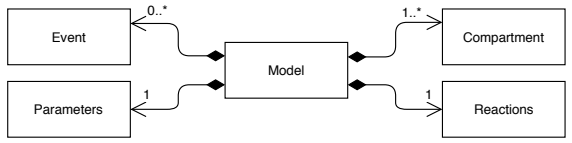

Fig. 2. UML class diagram of the Modelica models generated by SBML2Modelica.

Variables belonging to an object can be referenced from other objects via proper *connections*.

2.3 Modelica code generation

Differently from the BioChem plug-in of Wolfram SystemModeler (the only other Modelica-based SBML simulator available), SBML2Modelica does not rely on library abstractions, but generates stand-alone yet well-structured and human-intelligible Modelica code.

In particular, the Modelica model generated by SBML2Modelica starting from the SBML model given as input is a network of different objects of 5 different classes (whose code is stored in separate files), following the structure shown in Figure 2.

This structure ensures full portability and extensibility of the generated Modelica code (no plug-ins are required), and enables easy modifications at the level of each basic component (as no library classes are involved).

2.3.1 The Model object

The Model object acts as an orchestrator, by holding links to the other model objects and defining, via proper connections, the inter-object visibility of model variables.

Furthermore, the Model object defines the algebraic equations encoding the algebraic rules occurring in the input SBML model, as they may constrain variables belonging to different Modelica objects. For instance, the Model object generated from Example 1 would define an algebraic equation encoding algebraic rule r_3.

Finally, the Model object hosts a set of auxiliary functions (also generated by SBML2Modelica) needed to handle conflicting assignments from simultaneous events (see Section 2.3.4).

2.3.2 The Compartment objects

SBML2Modelica defines a Modelica *Compartment* object for each SBML model compartment. Such objects are then linked with the Model object.

The object variables define the compartment size and the amount and concentrations of all the species belonging to the compartment. For example, the object associated to compartment c of Example 1 would define variables c^{size}, s_1^{am}, s_1^{con}, ..., s_4^{am}, s_4^{con}. Initial assignments to the object variables (if defined within the input SBML model, as it happens in Example 1) and differential/algebraic equations for the species belonging to the compartment as well for the compartment size are encoded using information from SBML initial assignments, reactions, and rules.

For instance, the Modelica code generated from Example 1 would define the time-derivative of the amount of each species involved in reaction R (*i.e.,* of variables s_1^{am}, s_2^{am}, and s_3^{am}) as stipulated by Eq. (1), referencing variables (which store data on the reaction) belonging to the Reactions object (see below). Hence, we would have differential equations $\frac{\mathrm{d}s_i^{\text{am}}}{\mathrm{d}t} = \nu(s_i, R) \times k_R$, for $i \in [1, 3]$. Also, algebraic equation $c^{\text{size}} = 1 + p_1 \times t$ (where t refers to the current time instant) would be generated to encode assignment rule r_2. The time-derivative of the amount of any species involved in a rate rule (there are none in Example 1) would instead be encoded using its associated rule.

Conversely, equations for species whose amount is defined by means of an SBML algebraic rule (like s_4 in Example 1) are defined within the

Model object, as they represent constraints whose scope may span several Modelica objects.

Finally, variables representing the concentrations of all species are defined from their amounts. So, for compartment c of Example 1, we would have variable assignments $s_i^{\text{con}} \leftarrow \frac{s_i^{\text{am}}}{c^{\text{size}}}$ ($i \in [1, 4]$).

Suitable assertions are injected into the Modelica code to ensure that variables referring to compartment sizes are always strictly positive (as dictated by the SBML semantics), hence guaranteeing that species concentration variables are always defined.

2.3.3 The Reactions object

SBML2Modelica defines a single *Reactions* Modelica object storing data for all reactions defined in the input SBML model. Such an object is then linked with the Model object.

Object variables hold the kinetic rate expression for each reaction R in the model, as well as the coefficients $\nu(s, R)$ for each species s occurring in reaction R (see Eq. (1)). Thus, as for the single reaction R of Example 1, we would have variable k_R defining the kinetic rate expression of R via the algebraic equation $k_R = p_2 \times s_1^{\text{am}} \times s_2^{\text{am}}$, plus variables $\nu(s_1, R)$, $\nu(s_2, R)$, and $\nu(s_3, R)$ set to constant values -1, -2, and $+1$ respectively.

2.3.4 The Event objects

An *Event* Modelica object is defined for each event e defined in the input SBML model, in order to represent the event trigger condition, the event priority and delay (if any). All such objects are linked with the Model object.

In order to properly handle simultaneous events with conflicting assignments, the management of event assignments is split in three parts.

First, an auxiliary variable is defined in the Event object encoding e for each SBML model variable assigned by e. When e occurs (and after the event delay, if any), each such variable is set to the new value to be assigned to its associated model variable v, as stipulated by e.

Second, conflicting assignments stemming from simultaneous events are resolved (within the Model object, Section 2.3.1) by means of auxiliary functions, which also take into account the priorities of the competing events.

Third, the objects owning the variables to be assigned after the occurring event(s) are informed of the final required changes and take care of actually performing the assignments.

Depending on whether each event is persistent or non-persistent, SBML2Modelica generates different code. Specialised and more efficient code is also generated for the common case of events with no delay.

2.3.5 The Parameters object

A single *Parameters* Modelica object is defined to encode all the parameters of the input SBML model, together with their associated initial assignments and differential/algebraic equations stemming from SBML rate or assignments rules.

Hence, the Parameters object for Example 1 (which is linked with the Model object) would define and properly initialise variables p_1, p_2, p_3, p_4, and encode differential equations $\frac{\mathrm{d}p_2}{\mathrm{d}t} = 0$ (stemming from rate rule r_1) and $\frac{\mathrm{d}p_4}{\mathrm{d}t} = 0$ (stemming from the fact that value to p_4 changes only upon events).

3 Results

Section 3.1 below shows the results of our experiments aimed at assessing the correctness of SBML2Modelica against the SBML Test Suite Core, while Section 3.2 compares the performance of general-purpose Modelica and FMU simulators when running the Modelica code generated by SBML2Modelica against the other SBML simulators for which an SBML Test Suite Core report is publicly available.

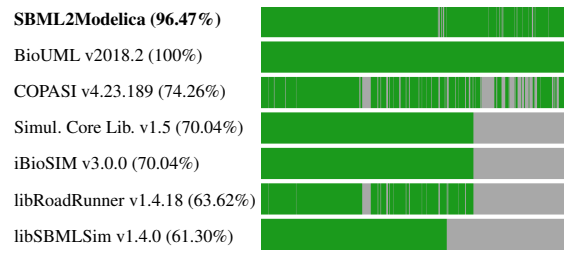

Fig. 3. Compliance of SBML2Modelica-generated code (simulated by OpenModelica and JModelica) to the SBML Test Suite Core v3.3.0, compared to other SBML simulators with certified public reports (test cases with deprecated SBML constructs are ignored).

3.1 Compliance to the SBML Test Suite Core

In order to assess the correctness of SBML2Modelica, we ran it against the test cases provided by SBML Test Suite Core v3.3.0, available in the SBML web-site.

As SBML2Modelica aims at supporting the *latest* SBML standard (SBML Level 3 Version 2), we ignored the test cases involving *deprecated* and *discouraged* constructs such as *fast reactions*. Hence, we ran SBML2Modelica against the remaining 1588 out of the overall 1623 test cases and simulated the generated Modelica code with the two major open-source Modelica implementations, namely OpenModelica (we used v1.32.2) and JModelica (we used v2.4), with the latter converting the input Modelica code into an FMU and then simulating such an FMU.

SBML2Modelica achieves very high marks: the output (always identical between OpenModelica and JModelica) is exactly as expected on 1532 out of 1588 test cases (96.47%). Figure 3 compares the test cases of the SBML Test Suite Core v3.3.0 successfully simulated by OpenModelica/JModelica (on input provided by SBML2Modelica) to the results *declared* by the other six systems for which a certified public report is available in the SBML web-site. Namely, the figure shows, for each system, a series of 1588 thin vertical bars, one per test case (sorted by their identifier in ascending order from left to right). Each vertical bar is coloured in green (respectively, grey) if the simulator output is (respectively, is not) exactly as expected by the SBML Test Suite Core upon numerical simulation of that test case.

The figure shows that SBML2Modelica+OpenModelica/JModelica rank among the top-compliant SBML simulators, being second only to BioUML. Note that libSBMLSim, iBioSim, Simulation Core Library and libRoadRunner fail over a large number of test cases located on the right part of their plot. This is due to the fact that the test cases containing the constructs introduced in the latest SBML standard (not supported by them) have the highest values of their identifiers.

As for the 56 test cases for which the output computed by SBML2Modelica+OpenModelica/JModelica differs from the output expected by the test suite, in 8 cases the difference is *semantically meaningless*. In particular, the time series for the model variables computed by our Modelica simulators contain one more line with respect to the SBML Test Suite Core expected output, returning the value of the model variables immediately before each event (even if such events do not occur at time-points multiple of the requested sampling time). This is an intended behaviour of OpenModelica and JModelica, aimed at better showing the discontinuities in the values of the model variables that arise when events occur. By ignoring such additional lines, our output is exactly as expected.

The other 48 test cases where the output of SBML2Modelica+OpenModelica/JModelica differs from the output expected from the

test suite are due to combinations of SBML constructs *unsupported* by SBML2Modelica. Such combinations are discussed in Section 4. However, we anticipate that they are very rare in practice, semantically intricate, and easily avoidable. Although some more involved/cryptic Modelica code could be in principle be generated to handle them, we chose to keep our output Modelica models as structured and human-intelligible as possible, in order to ease their extension and integration within larger model networks.

Anyway, there is no risk to accidentally generate flawed Modelica code, since any problematic combinations of constructs are *statically detected* by SBML2Modelica, which warns the user during model translation about possible issues. The user can then act directly on the generated Modelica code to fix any raised issue. Also, to further assist the user, suitable *assertions* are injected in the generated Modelica code that would raise proper exceptions during simulation in case the Modelica model (generated *with* warnings) actually behaves in a way not fully compliant to the SBML semantics.

3.2 Model simulation performance

In this section we aim at assessing to what extent translating SBML models into an open-standard general-purpose modelling language such as Modelica and into an open-standard general-purpose simulation ecosystem such as FMI/FMU introduces an overhead in simulation performance, when compared to simulation algorithms specialised to biochemical models. To this end, we consider the major open-source (*general-purpose*) Modelica and FMU simulators (OpenModelica and JModelica, respectively). Note that, while OpenModelica simulates the input model directly, JModelica works by translating the input Modelica model into an FMU and then by actually simulating such an FMU using the FMI API.

The performance of OpenModelica and JModelica/FMI in simulating SBML models translated by SBML2Modelica is compared against that of the *specialised* SBML simulators reported in Section 3.1 plus SystemModeler+BioChem, the only other available Modelica-based simulator for SBML models.

Although our results are by *no means* to be intended as the outcome of a competition among SBML simulators, they clearly show that the advantages of using SBML2Modelica+OpenModelica and SBML2Modelica+JModelica/FMI *do not generally come at any significant performance overhead*.

3.2.1 Benchmarks

We used the BioModels Database (Le Novère *et al.*, 2006), a well-known repository of mathematical models of biological and biomedical systems taken from the scientific literature. Models manually reviewed to guarantee reproducibility of results belong to the set of *manually curated* models. This set of models is widely used as a benchmark for SBML interpreters and simulators.

We selected the subset of manually curated models of the BioModels Database (as of December 2018) that are *accepted* by SBML2Modelica (*i.e.,* do not contain deprecated constructs or the unsupported combinations of constructs described in Section 4). As a result, our benchmark set consists of 613 models (out of the 641 manually curated models of the BioModels Database), which have from 6 to 5438 variables.

3.2.2 Experimental setting

The computational complexity of a model simulation is affected by many factors, which are way beyond the mere number of variables. For example: number and structure of the differential as well algebraic equations; number and frequency of occurrence of events and complexity of their trigger conditions; algorithm and parameters used by the simulation engine.

Setting up a methodologically-sound competition among OpenModelica, JModelica/FMI, and specialised SBML simulators is a complex task, which is clearly out of the scope of this paper. So is the choice of the optimal simulation algorithm and configuration for a given model (in particular both OpenModelica and JModelica/FMI offer a wide portfolio of highly-configurable integrators to choose from).

Given our goals (see Section 3.2), in our analysis we proceeded as follows.

First, for each system we measured the *core simulation time*, i.e., the time of simulating the given model from its (system-specific) *internal representation*. This is consistent with the most demanding use-cases, such as parameter identification or estimation procedures, where simulator initialisation *and* model preparation are performed only once, while a large number of simulations (with different parameter assignments) takes place, among which such initialisation costs are amortised. As for OpenModelica and JModelica/FMI, this means that we ignored SBML2-Modelica translation time, which, anyway, always takes *less than 6% of the simulation time*. For the same reason, as for JModelica/FMI we also ignored the time to generate the FMU (which is also *negligible*).

Second, we used all systems with their *default* integrators and settings.

Third, we fixed the simulation horizon and the maximum time step to, respectively, 100 and 0.01 (model) time units. This last choice allowed us to extrapolate a clear performance trend of each system on the basis of the number of model variables.

All simulations were performed on a commodity computer (AMD A12-9720P CPU, 12 GB RAM, SSD, standard Linux environment), with a time-out of 360 seconds.

3.2.3 Experimental results

The scatter (log-log) plot in Figure 4 shows a dot for each model in our benchmark set and each system (if that system terminated within our time-out). For each dot in position (x, y), x denotes the number of variables of the associated SBML model, while y denotes the time (in seconds) required by the system associated to the dot colour to terminate. SBML model import in Wolfram SystemModeler+BioChem and iBioSim requires manual user interaction via GUI and could not be automated. We overcame this issue by manually launching such systems on all models with *at least* 250 variables.

Figure 4 shows that OpenModelica and JModelica/FMI are *competitive* on most of the benchmark set, although, on the two largest models, there is a visible performance gap with respect to some of the other systems, with OpenModelica and JModelica going in time-out for, respectively, one and both of them. Also, OpenModelica appears generally faster than JModelica/FMI, even if it seems to slow down a bit when simulating models with frequently-occurring events (as shown by the bimodal behaviour on models having a similar number of variables). However, also in these cases, its overall performance remain aligned to that of several specialised SBML simulators. Conversely, performance of JModelica/FMI are more stable, regardless of the events occurrence frequency.

Besides the above peculiarities, the *overall trend* emerging from Figure 4 is that the benefits enabled by a translation into a *general-purpose* open-standard modelling language for dynamical systems such as Modelica and, as for JModelica/FMI, into a general-purpose open-standard simulation ecosystem such as FMI/FMU, generally come at *no significant overhead*, when compared against performance of *specialised* SBML simulators.

4 Discussion

Section 3 shows that, by translating SBML models into Modelica (and in turn, as for JModelica/FMI, into FMUs), SBML2Modelica

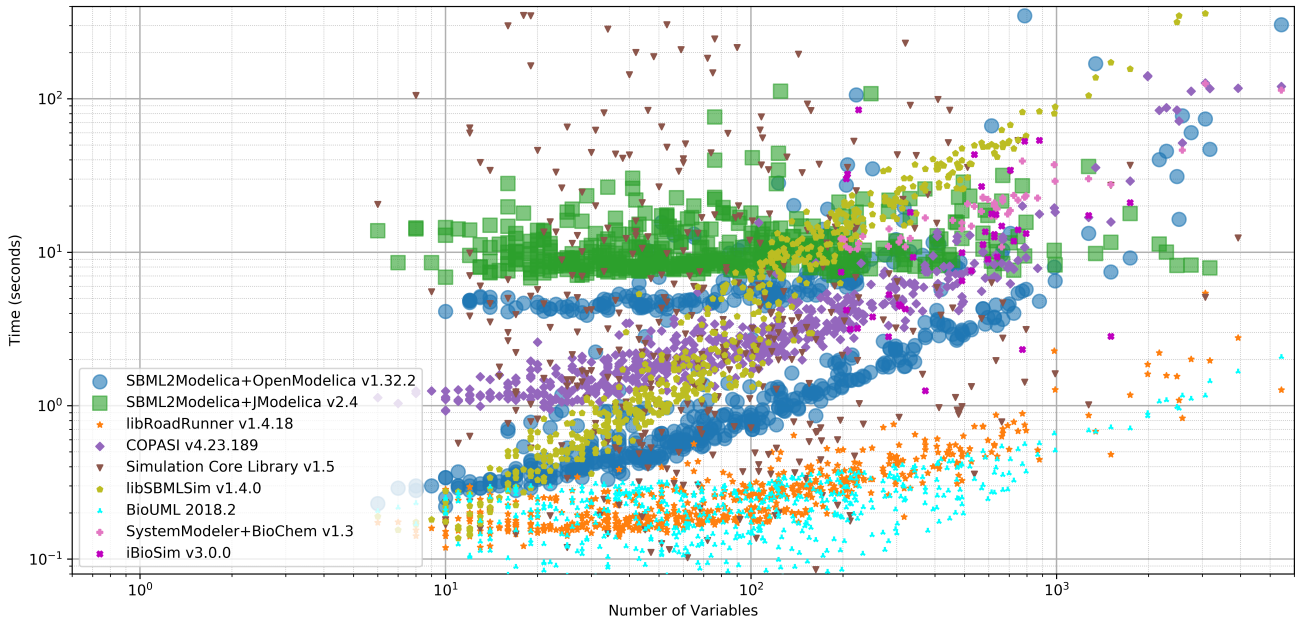

Fig. 4. Performance trend (log-log plot) of general-purpose OpenModelica (v1.32.2) and JModelica/FMI (v2.4) simulators on our benchmark models (from BioModels) translated with SBML2Modelica, compared with that of specialised SBML simulators. (Due to the need of manual GUI interaction, results of Wolfram SystemModeler+BioChem and iBioSim are reported only for models with at least 250 variables.)

effectively enables *seamless integration* of SBML models into larger heterogeneous model networks without (in most cases) sacrificing simulation performance.

Below we analyse the 48 test cases in the SBML Test Suite Core containing those combinations of SBML constructs not supported by SBML2Modelica (see Section 3.1). Note that such construct combinations are *rare* in real-world models, *semantically intricate*, and can be *easily circumvented*. SBML2Modelica always *detects* such cases, and *warns* the user accordingly, hence there is *no risk* to run flawed Modelica code.

4.1 Events recomputation

If, at a certain time instant t, *multiple* events are *simultaneously* triggered, such events are ordered by their priorities and the event e with the *highest* priority is executed first (Example 1 shows one such case). However, the execution of e might change the value of variables that occur in the priority expression or in the trigger condition of some of the other events waiting to be executed. In such situations (which can easily be circumvented by modelling the underlying mechanisms of such interfering events more explicitly), SBML2Modelica cannot generate correct Modelica code. To avoid to generate a flawed translation, SBML2Modelica statically detects whether such interferences might occur between events and warns the user.

4.2 Nested triggers

Assume that a persistent event e is triggered for the first time at time t_1 and that it is requested to be delayed by duration d_1. If, for some reason, event e is triggered again at time t_2 such that $t_1 < t_2 < t_1 + d_1$ and is requested to be delayed by duration d_2, the trigger must be recorded and e must execute at time $t_2 + d_2$. As we cannot statically detect how many times the trigger of a persistent event can be nested, SBML2Modelica raises a warning during model translation, informing the user that a second trigger for a persistent model event might potentially occur when a previous trigger for the same event is on hold because of a delay. Also, SBML2Modelica injects an assertion into the generated Modelica code which stops the simulation in case this situation *actually* occurs at run-time (making the model behaviour not compliant with the SBML semantics). Again, such situations can be easily circumvented by modelling the underlying causes of event delays in more explicit ways.

4.3 Negative time

SBML allows the definition of quantities also in *negative* time-points. This may cause issues. For example, the value of a model variable x at time 0 could be re-assigned by an event trigger (processed at time 0) using an expression such as $(\mathrm{pre}\,(x))\,(0)$. In this case, the SBML semantics stipulates that $(\mathrm{pre}\,(x))\,(0)$ must be determined *somewhere else* in the model (otherwise, a semantic error occurs). Another example of negative time is the occurrence of $f(t-d)$, where f is a function and $d > 0$ is the duration of a *delay*. According to the SBML specifications, the value of $f(t-d)$ is defined also when $t < d$, by assuming that f is defined also for negative time-points. Conversely, the Modelica language specification forbids negative time-points, as time-point 0 is assumed to be the *initial time-point* for simulation.

Although, in principle, additional Modelica code could be generated by SBML2Modelica in order to properly handle such cases (*e.g.,* by computing a suitable time-offset for all the model variables, and by artificially shifting in time the evolution of the entire model), we decided not to support this possibility, in order to keep the generated Modelica code well-structured and fully readable. Hence, when the above situations are detected, SBML2Modelica issues a warning. The user interested in supporting such cases, can take full responsibility by acting directly on the generated Modelica code.

4.4 Unsupported math

SBML allows users to explicitly assign value `NaN` to variables. When such values are found in the input SBML model, SBML2Modelica notices the user, since the numerical model simulation is clearly not possible.

5 Conclusions

In this paper we presented SBML2Modelica, an SBML Level 3 Version 2–compliant software system that translates SBML models into well-structured, user-intelligible, easily modifiable *Modelica* code, an open-standard general-purpose modelling language for which several efficient simulators (both commercial and open-source) are available. Modelica models can also be exported into black-box language-independent FMUs, an open standard supported by more than 100 simulators from virtually all application domains.

All this paves the way to the seamless integration (without lack of simulation performance) of SBML models within open-standard ecosystems, where biochemical models can be used as components of large heterogeneous model networks (together with models of, *e.g.,* human physiology, clinical protocol guidelines, treatment schemes, biomedical devices), and where standard system engineering approaches can be employed to perform their simulation-based analysis at system level.

Acknowledgements. This work was partially supported by: Italian Ministry of University and Research under grant "Dipartimenti di Eccellenza 2018–2022" of the Department of Computer Science of Sapienza University of Rome; EC FP7 project PAEON (Model Driven Computation of Treatments for Infertility Related Endocrinological Diseases, 600773); INdAM "GNCS Project 2019"; Sapienza University 2018 project RG11816436BD4F21 "Computing Complete Cohorts of Virtual Phenotypes for In Silico Clinical Trials and Model-Based Precision Medicine". Authors are very grateful to the anonymous reviewers for their valuable comments.

References

de Bono, B. and Hunter, P. (2012). Integrating knowledge representation and quantitative modelling in physiology. *Biotechnology Journal*, **7**(8), 958–972.

Eissing, T. *et al.* (2011). A computational systems biology software platform for multiscale modeling and simulation: Integrating whole-body physiology, disease biology, and molecular reaction networks. *Frontiers in physiology*, **2**, 4.

Fritzson, P., *et al.* (2007). Biochemical mathematical modeling with modelica and the biochem library. In *APLIMAT 2007*, pages 147–159.

Gendrault, Y. *et al.* (2014). Modeling biology with HDL languages: A first step toward a genetic design automation tool inspired from microelectronics. *IEEE Transactions on Biomedical Engineering*, **61**(4), 1231–1240.

Hucka, M. *et al.* (2003). The Systems Biology Markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, **19**(4), 524–531.

Hucka, M. *et al.* (2018). The Systems Biology Markup Language (SBML): Language specification for Level 3 Version 2 Core. *Journal of Integrative Bioinformatics*, **15**(1).

Kanade, A. *et al.* (2009). Generating and analyzing symbolic traces of Simulink/Stateflow models. In *CAV 2009*, LNCS volume 5643, pages 430–445. Springer.

Keller, R. *et al.* (2013). The systems biology simulation core algorithm. *BMC Systems Biology*, **7**, 55.

Kolpakov, F. *et al.* (2019). BioUML: an integrated environment for systems biology and collaborative analysis of biomedical data. *Nucleic Acids Research*, **47**(W1), W225–W233.

Larsdotter Nilsson, E. and Fritzson, P. (2003). BioChem - A biological and chemical library for Modelica. In *Modelica 2003*, pages 215–220.

Le Novère *et al.* (2006). BioModels Database: A free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research*, **34**(suppl_1), D689–D691.

Lee, C. *et al.* (2006). COPASI —- A COmplex PAthway SImulator. *Bioinformatics*, **22**(24), 3067–3074.

Madec, M. *et al.* (2017). Modeling and simulation of biological systems using SPICE language. *PLoS ONE*, **12**(8), 1–21.

Madec, M. *et al.* (2019). Environment for modeling and simulation of biosystems, biosensors, and lab-on-chips. *IEEE Transactions on Electron Devices*, **66**(1), 34–43.

Mancini, T. *et al.* (2013). System level formal verification via model checking driven simulation. In *CAV 2013*, LNCS volume 8044, pages 296–312. Springer.

Mancini, T. *et al.* (2014). System level formal verification via distributed multi-core hardware in the loop simulation. In *PDP 2014*, pages 734–742. IEEE.

Mancini, T. *et al.* (2015). Computing biological model parameters by parallel statistical model checking. In *IWBBIO 2015*, LNCS volume 9044, pages 542–554. Springer.

Mancini, T. *et al.* (2016a). Anytime system level verification via parallel random exhaustive hardware in the loop simulation. *Microprocessors and Microsystems*, **41**, 12–28.

Mancini, T. *et al.* (2016b). SyLVaaS: System level formal verification as a service. *Fundamenta Informaticae*, **1–2**, 101–132.

Mancini, T. *et al.* (2017). On minimising the maximum expected verification time. *Information Processing Letters*, **122**, 8–16.

Mancini, T. *et al.* (2018). Computing personalised treatments through in silico clinical trials. A case study on downregulation in assisted reproduction. In *RCRA 2018*.

Mateják, M. and Kofránek, J. (2015). Physiomodel – An integrative physiology in Modelica. In *EMBC 2015*, pages 1464–1467. IEEE.

Messori, M. *et al.* (2018). Individualized model predictive control for the artificial pancreas: In silico evaluation of closed-loop glucose control. *IEEE Control Systems Magazine*, **38**(1), 86–104.

Myers, C. *et al.* (2009). iBioSim: A tool for the analysis and design of genetic circuits. *Bioinformatics*, **25**(21), 2848–2849.

Schaller, S. *et al.* (2016). Robust PBPK/PD-based model predictive control of blood glucose. *IEEE Transactions on Biomedical Engineering*, **63**(7), 1492–1504.

Somogyi, E. *et al.* (2015). libRoadRunner: A high performance SBML simulation and analysis library. *Bioinformatics*, **31**(20), 3315–3321.

Takizawa, H. *et al.* (2013). LibSBMLSim: A reference implementation of fully functional SBML simulator. *Bioinformatics*, **29**(11), 1474–1476.

Tronci, E. *et al.* (2014). Patient-specific models from inter-patient biological models and clinical records. In *FMCAD 2014*, pages 207–214. IEEE.

Zuliani, P. (2015). Statistical model checking for biological applications. *International Journal on Software Tools for Technology Transfer*, **17**(4), 527–536.

Zuliani, P. *et al.* (2013). Bayesian statistical model checking with application to Stateflow/Simulink verification. *Formal Methods in System Design*, **43**(2), 338–367.